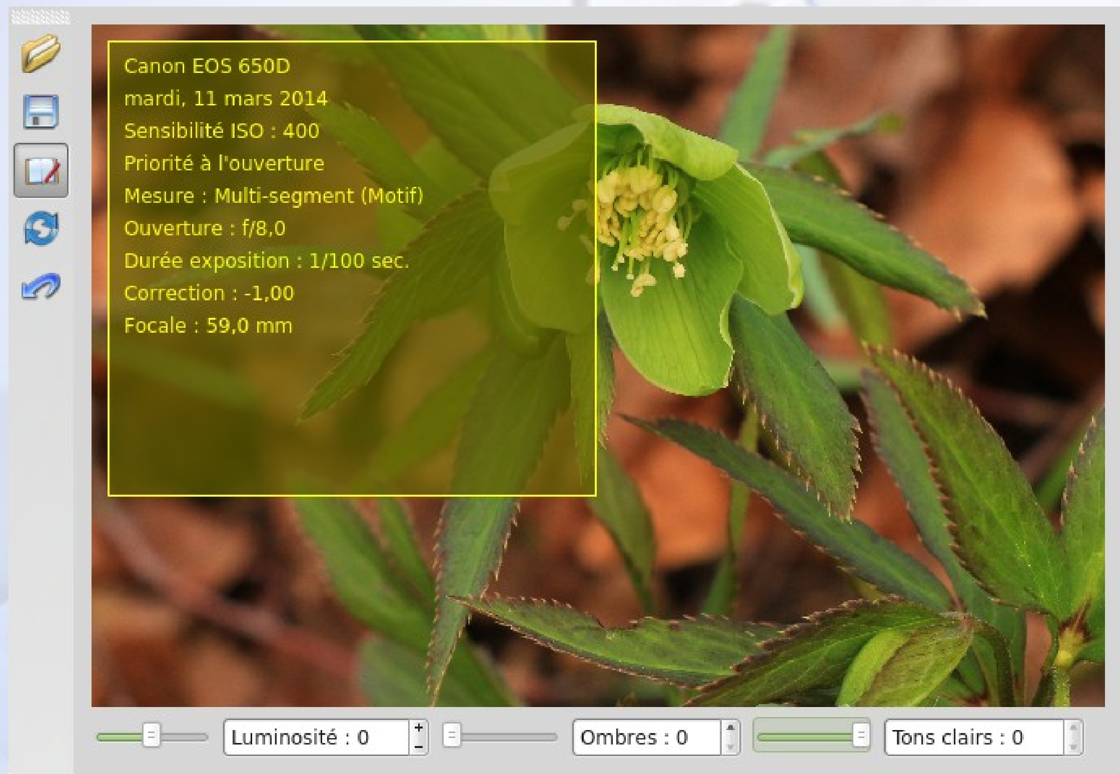
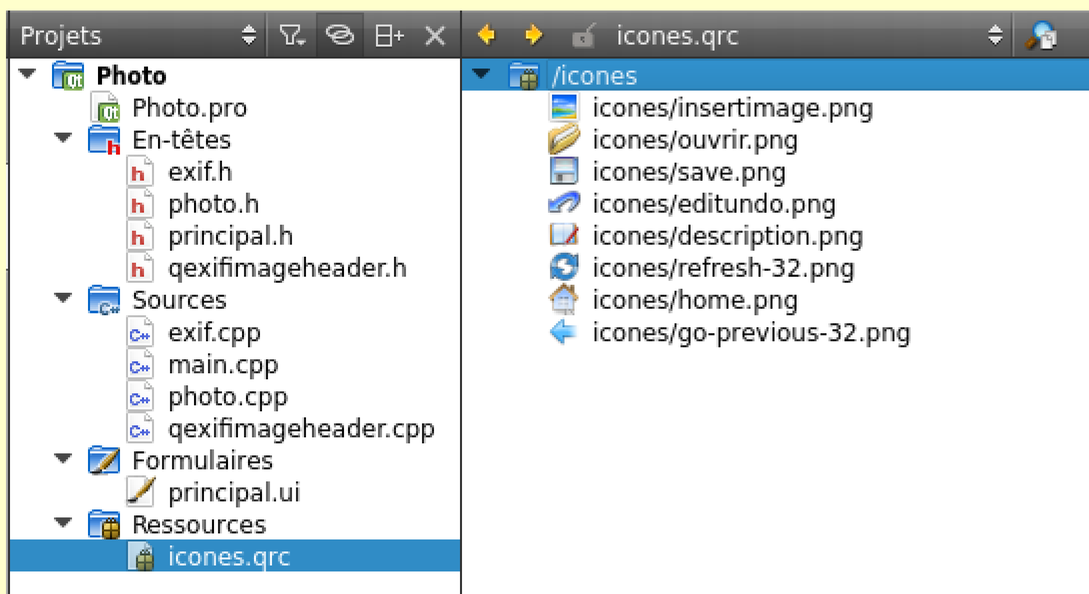


Lors de cette étude, nous allons voir comment récupérer et modifier des données **EXIFs** présentes dans la plupart des photos. La plupart du temps, ces données sont perdues lorsque nous fabriquons notre propre logiciel de traitement d'images. Nous allons voir comment les restituer dans notre nouvelle photo travaillée.

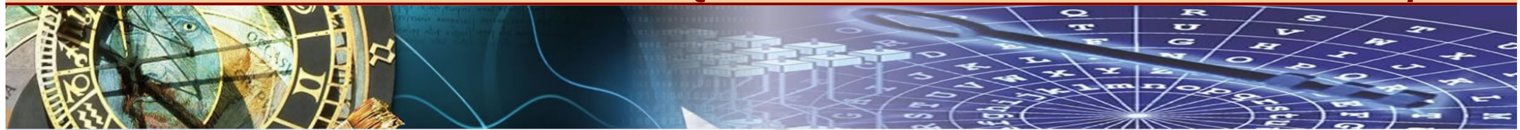
✗ CONSTITUTION DU PROJET

Afin d'élaborer correctement ce logiciel de traitement d'images, nous avons besoin d'un certain nombres de fichiers, notamment les sources concernant la récupération des données EXIFs dans un environnement QT, savoir :

- ✗ ***qexifimaheheader.h***
- ✗ ***qexifimageheader.cpp***



✗



✗ PHOTO.PRO

```
#-------------------------------------------------
#
# Project created by QtCreator 2014-02-12T17:20:01
#
#-------------------------------------------------
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = Photo
TEMPLATE = app
CONFIG += c++11
SOURCES  += main.cpp photo.cpp qexifimageheader.cpp exif.cpp
HEADERS  += principal.h photo.h qexifimageheader.h exif.h
FORMS    += principal.ui
RESOURCES += icones.qrc
```

✗ EXIF.H

Afin de bien gérer les données *Exifs* avec des méthodes adaptées, j'ai mis en œuvre une nouvelle classe **Exif** qui hérite de la classe **QExifImageHeader** :

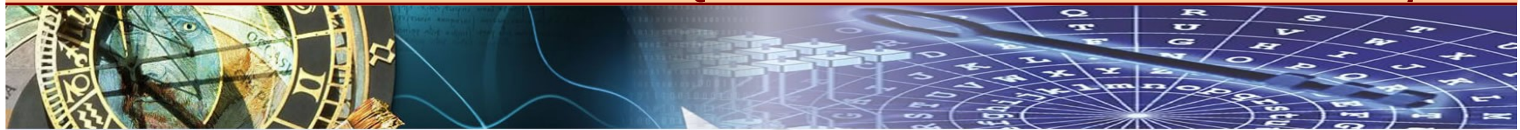
```
#ifndef EXIF_H
#define EXIF_H
#include "qexifimageheader.h"
struct Exif : public QExifImageHeader
{
    enum OrientationImage {Normal=1, Bas=3, Gauche=8, Droite=6};
private:
    QString modele;
    QString ouverture;
    QString focale;
    QString iso;
    QString tempsExposition;
    QString date;
    QString modeExposition;
    QString correctionExposition;
    QString calculExposition;
    int type;
    bool existe;
    bool GPSExiste;
    QString longitude;
    QString longitudeRef;
    QString latitude;
    QString latitudeRef;
    QString altitude;
    QString direction;
    QStringList informations;
    OrientationImage orientation;
public:
    Exif() = default;
    void chargerFichierJPEG(const QString &nom);
private:
    void setCarateristiques();
    void setGPS();
    void setInformations();
public:
    QString getModele() const               { return modele; }
    QString getOuverture() const            { return ouverture; }
    QString getFocale() const               { return focale; }
    QString getIso() const                  { return iso; }
    QString getTempsExposition() const      { return tempsExposition; }
    QString getDate() const                 { return date; }
    QString getModeExposition() const       { return modeExposition; }
    QString getCorrectionExposition() const { return correctionExposition; }
    QString getCalculExposition() const     { return calculExposition; }
    int getType() const                     { return type; }
    bool isExiste() const                   { return existe; }
    bool isGPSExiste() const                { return GPSExiste; }
    QString getLongitude() const            { return longitude; }
    QString getLongitudeRef() const         { return longitudeRef; }
    QString getLatitude() const             { return latitude; }
    QString getLatitudeRef() const          { return latitudeRef; }
    QString getAltitude() const             { return altitude; }
    QString getDirection() const            { return direction; }
    QStringList getInformations() const     { return informations; }
    OrientationImage getOrientation() const { return orientation; }
    void setOrientationNormal()             { setValue(Orientation, Normal); }
```

```
    void setDescription(const QString &description) { setValue(ImageDescription, description); }
    void setAuteur(const QString &auteur)           { setValue(Artist, auteur); }
};
#endif // EXIF_H
```

✗ EXIF.CPP

```
#include "exif.h"
#include <QDateTime>
#include <QFile>
#include <Qlocale>

void Exif::chargerFichierJPEG(const QString &nom)
{
    loadFromJpeg(nom);
    setCarateristiques();
    setGPS();
    setInformations();
}

void Exif::setCarateristiques()
{
    QLocale france;
    if (existe = contains(Model)) modele = value(Model).toString();
    if (contains(FNumber))
    {
        int numerateur = value(FNumber).toRational().first;
        int denominateur = value(FNumber).toRational().second;
        int rapport = numerateur / denominateur;
        ouverture = denominateur==1
                  ? QString("Ouverture : f/%1,0").arg(numerateur)
                  : QString("Ouverture : f/%1,%2").arg(rapport).arg(numerateur-rapport*denominateur);
    }
    if (contains(FocalLength))
    {
        int numerateur = value(FocalLength).toRational().first;
        int denominateur = value(FocalLength).toRational().second;
        int rapport = numerateur / denominateur;
        focale = denominateur==1
               ? QString("Focale : %1,0 mm").arg(numerateur)
               : QString("Focale : %1,%2 mm").arg(rapport).arg(numerateur-rapport*denominateur);
    }
    if (contains(ISOSpeedRatings)) iso = QString("Sensibilité ISO : %1").arg(value(ISOSpeedRatings).toShort());
    if (contains(ExposureTime))
    {
        int numerateur = value(ExposureTime).toRational().first;
        int denominateur = value(ExposureTime).toRational().second;
        int rapport = numerateur / denominateur;
        if (numerateur==1 && denominateur==1) tempsExposition = QString("1 sec.");
        else if (numerateur == 1) tempsExposition = QString("1/%1 sec.").arg(denominateur);
        else if (numerateur < 10) tempsExposition = QString("0\"%1").arg(numerateur);
        else if (rapport*denominateur == numerateur) tempsExposition = QString("%1").arg(rapport);
        else tempsExposition = QString("%1\"%2").arg(rapport).arg(numerateur-rapport*denominateur);
        tempsExposition = "Durée exposition : " + tempsExposition;
    }
    if (contains(DateTime))
    {
        QDateTime valeur = value(DateTime).toDateTime();
        date = valeur.toString("dddd, d MMMM yyyy");
    }
    if (contains(ExposureProgram))
    {
        switch (value(ExposureProgram).toShort())
        {
            case 0 : modeExposition = "Indéterminé"; break;
            case 1 : modeExposition = "Contrôle manuel"; break;
            case 2 : modeExposition = "Tout automatique"; break;
            case 3 : modeExposition = "Priorité à l'ouverture"; break;
            case 4 : modeExposition = "Priorité à la vitesse"; break;
            case 5 : modeExposition = "Programme créatif (Grande profondeur de champ)"; break;
            case 6 : modeExposition = "Programme d'action (vitesse élevée)"; break;
            case 7 : modeExposition = "Mode portrait"; break;
            case 8 : modeExposition = "Mode paysage"; break;
        }
    }
    if (contains(ExposureBiasValue))
    {
```

```
        double numerateur = value(ExposureBiasValue).toSignedRational().first;
        double denominateur = value(ExposureBiasValue).toSignedRational().second;
        correctionExposition = "Correction : " + france.toString(numerateur/denominateur, 'f', 2);
    }
    if (contains(MeteringMode))
    {
        switch (value(MeteringMode).toShort())
        {
            case 0 : calculExposition = "Indéterminé"; break;
            case 1 : calculExposition = "Moyenne"; break;
            case 2 : calculExposition = "Moyenne pondérée au centre"; break;
            case 3 : calculExposition = "Spot"; break;
            case 4 : calculExposition = "Multi-spot"; break;
            case 5 : calculExposition = "Multi-segment (Motif)"; break;
            case 6 : calculExposition = "Partielle"; break;
            case 255 : calculExposition = "Autre"; break;
        }
        calculExposition = "Mesure : "+calculExposition;
    }
    if (contains(Orientation)) orientation = (OrientationImage) value(Orientation).toShort();
}

void Exif::setGPS()
{
    QLocale france;
    if (GPSExiste = contains(GpsLongitude))
    {
        int degre = value(GpsLongitude).toRationalVector().at(0).first;
        int minute = value(GpsLongitude).toRationalVector().at(1).first;
        double numerateur = value(GpsLongitude).toRationalVector().at(2).first;
        double denominateur = value(GpsLongitude).toRationalVector().at(2).second;
        QString seconde =  france.toString(numerateur/denominateur, 'f', 2);
        longitude = QString("%1° %2\' %3\"").arg(degre).arg(minute).arg(seconde);
    }
    if (contains(GpsLongitudeRef)) longitudeRef = value(GpsLongitudeRef).toString() == "E" ? "Est" : "Ouest";
    if (contains(GpsLatitude))
    {
        int degre = value(GpsLatitude).toRationalVector().at(0).first;
        int minute = value(GpsLatitude).toRationalVector().at(1).first;
        double numerateur = value(GpsLatitude).toRationalVector().at(2).first;
        double denominateur = value(GpsLatitude).toRationalVector().at(2).second;
        QString seconde =  france.toString(numerateur/denominateur, 'f', 2);
        latitude = QString("%1° %2\' %3\"").arg(degre).arg(minute).arg(seconde);
    }
    if (contains(GpsLatitudeRef)) latitudeRef = value(GpsLatitudeRef).toString() == "N" ? "Nord" : "Sud";
    if (contains(GpsAltitude))
    {
        double numerateur = value(GpsAltitude).toRational().first;
        double denominateur = value(GpsAltitude).toRational().second;
        altitude = france.toString(numerateur/denominateur, 'f', 2);
    }
    if (contains(GpsImageDirection))
    {
        double numerateur = value(GpsImageDirection).toRational().first;
        double denominateur = value(GpsImageDirection).toRational().second;
        direction = france.toString(numerateur/denominateur, 'f', 2);
    }
}

void Exif::setInformations()
{
    informations.clear();
    informations.append(modele);
    informations.append(date);
    informations.append(iso);
    informations.append(modeExposition);
    informations.append(calculExposition);
    informations.append(ouverture);
    informations.append(tempsExposition);
    informations.append(correctionExposition);
    informations.append(focale);
    if (GPSExiste) {
        informations.append(longitude+" "+longitudeRef);
        informations.append(latitude+" "+latitudeRef);
        informations.append("Altitude : "+altitude);
        informations.append(QString("Direction : %1°").arg(direction));
    }
}
```



✗ PRINCIPAL.H

```
#ifndef PRINCIPAL_H
#define PRINCIPAL_H
#include <QMainWindow>
#include "ui_principal.h"
class Principal : public QMainWindow, public Ui::Principal
{
    Q_OBJECT
public:
    explicit Principal(QWidget *parent = 0) : QMainWindow(parent) { setupUi(this); }
};
#endif // PRINCIPAL_H
```

✗ PHOTO.H

La grosse partie du projet se situe dans la classe **Photo** qui hérite de la classe **QWidget**. Le logiciel permet d'utiliser la molette de la souris pour agrandir l'image, le bouton gauche sert à déplacer l'image et le double-click permet d'afficher l'image dans sa totalité. Enfin, trois retouches sont possibles sans détérioration (pixels saturés), la luminosité, l'éclaircissement des ombres et l'atténuation des tons clairs (hautes lumières).

```
#ifndef PHOTO_H
#define PHOTO_H
#include <QWidget>
#include <QImage>
#include "exif.h"
class Photo : public QWidget
{
    Q_OBJECT
public:
    explicit Photo(QWidget *parent = 0) : QWidget(parent) { }
signals:
    void raz(int);
private slots:
    void chargerPhoto();
    void sauverPhoto();
    void donneesExif(bool activer)  { visualiserExif = activer; update(); }
    void changerLuminosite(int valeur);
    void changerOmbres(int valeur);
    void changerTonsClairs(int valeur);
    void choisirOriginal(bool choix);
    void annuler() { raz(0); }
protected:
    void paintEvent(QPaintEvent *) override;
    void wheelEvent(QWheelEvent *evt) override;
    void resizeEvent(QResizeEvent *) override;
    void mousePressEvent(QMouseEvent *evt) override;
    void mouseMoveEvent(QMouseEvent *evt) override;
    void mouseDoubleClickEvent(QMouseEvent *) override { setCadrage(); }
private:
    QImage tailleReelle, photo, original;
    Exif exif;
    bool visualiserExif=false, avant=false;
    double zoom;
    QRect cadrage;
    QPoint contact;
    int luminosite=0, ombres=0, clairs=0;
private:
    void setCadrage();
    void traitement(QImage &photo, QImage &original);
};
#endif // IMAGE_H
```

✗ PHOTO.CPP

```
#include "photo.h"
#include <QFileDialog>
#include <QPainter>
#include <QRect>
#include <QFile>
#include <QTransform>
#include <QwheelEvent>

void Photo::chargerPhoto()
{
    QString nom = QFileDialog::getOpenFileName(this, "Choisissez votre photo", "", "Images (*.jpeg *.jpg)");
```

```
    if (!nom.isEmpty())
    {
        QTransform transformation;
        tailleReelle.load(nom);
        exif.chargerFichierJPEG(nom);
        switch (exif.getOrientation()) {
        case Exif::Gauche :
            transformation.rotate(-90);
            tailleReelle = tailleReelle.transformed(transformation);
            exif.setOrientationNormal();
            break;
        case Exif::Droite :
            transformation.rotate(90);
            tailleReelle = tailleReelle.transformed(transformation);
            exif.setOrientationNormal();
            break;
        }
//        exif.setAuteur("Emmanuel REMY");
//        exif.setDescription("Harpe");
        original = photo = tailleReelle.scaledToWidth(width());
        raz(0);
        setCadrage();
        setMouseTracking(true);
    }
}

void Photo::sauverPhoto()
{
    if (!photo.isNull())
    {
        QString nom = QFileDialog::getSaveFileName(this, "Sauvegardez la photo", "", "Images (*.jpeg *.jpg)");
        if (!nom.isEmpty())
        {
            QImage definitive = tailleReelle;
            traitement(definitive, tailleReelle);
            definitive.save(nom, "JPEG", 96);
            exif.saveToJpeg(nom);
        }
    }
}

void Photo::paintEvent(QPaintEvent *)
{
    if (!photo.isNull())
    {
        QPainter dessin(this);
        dessin.drawImage(cadrage, avant ? original : photo, photo.rect());
        if (visualiserExif && exif.isExiste())
        {
            QBrush fond(QColor(96, 96, 0, 180), Qt::SolidPattern);
            QPen crayon(QColor(255, 255, 0));
            dessin.setBrush(fond);
            dessin.setPen(crayon);
            dessin.drawRect(10, 10, 300, 280);
            QStringList liste = exif.getInformations();
            for (int i=0; i<liste.size(); i++)
                dessin.drawText(20, 30+20*i, liste.at(i));
        }
    }
}

void Photo::wheelEvent(QWheelEvent *evt)
{
    if (!photo.isNull())
    {
        if (evt->angleDelta().y() > 0) { if (zoom >= 1.5) zoom -= 0.2; }
        else { if (width() < tailleReelle.width()/zoom) zoom += 0.2; }
        double largeur = tailleReelle.width() / zoom;
        double hauteur = tailleReelle.height() / zoom;
        int x = evt->x() - (evt->x() - cadrage.x()) * largeur / cadrage.width();
        int y = evt->y() - (evt->y() - cadrage.y()) * hauteur / cadrage.height();
        cadrage.setRect(x, y, largeur, hauteur);
        original = photo = tailleReelle.scaledToWidth(largeur);
        traitement(photo, original);
        update();
    }
}
```

```
void Photo::resizeEvent(QResizeEvent *)
{
    if (width() > cadrage.width())
    {
        zoom = tailleReelle.width() / (double) width();
        double largeur = tailleReelle.width() / zoom;
        double hauteur = tailleReelle.height() / zoom;
        cadrage.setRect(0, 0, largeur, hauteur);
    }
}

void Photo::setCadrage()
{
    zoom = tailleReelle.width() / (double) width();
    double largeur = tailleReelle.width() / zoom;
    double hauteur = tailleReelle.height() / zoom;
    cadrage.setRect(0, 0, largeur, hauteur);
    update();
}

void Photo::mousePressEvent(QMouseEvent *evt)
{
    contact = {cadrage.x()-evt->x(), cadrage.y()-evt->y()};
}

void Photo::mouseMoveEvent(QMouseEvent *evt)
{
    if (evt->buttons() == Qt::LeftButton)
    {
        int x = contact.x() + evt->x();
        int y = contact.y() + evt->y();
        cadrage.moveTo(x, y);
        update();
    }
}

void Photo::traitement(QImage &photo, QImage &original)
{
  int table[256];
  for (int x=0; x<256; x++)
    table[x] =  x + luminosite*sin(x*M_PI/255) + clairs*(x/128.0)*(x/128.0)*(1.0 - x/255.0) + ombres*x*(1
-x/255.0)*(1 -x/255.0)/ 64;

  for (int x=0; x<256; x++)
    if (table[x]>255) table[x] = 255;
    else if (table[x]<0) table[x] = 0;
  for (int ligne=0; ligne < photo.height(); ligne++)
    for (int colonne=0; colonne < photo.width(); colonne++) {
      int rouge = table[qRed(original.pixel(colonne, ligne))];
      int vert = table[qGreen(original.pixel(colonne, ligne))];
      int bleu = table[qBlue(original.pixel(colonne, ligne))];
      photo.setPixel(colonne, ligne, qRgb(rouge, vert, bleu));
    }
}

void Photo::changerLuminosite(int valeur)
{
  luminosite = valeur;
  traitement(photo, original);
  update();
}

void Photo::changerOmbres(int valeur)
{
  ombres = valeur;
  traitement(photo, original);
  update();
}

void Photo::changerTonsClairs(int valeur)
{
  clairs = valeur;
  traitement(photo, original);
  update();
}

void Photo::choisirOriginal(bool choix) { avant = choix; update(); }
```

PRINCIPAL.UI

Luminosité : 0 | Ombres : 0 | Tons clairs : 0

Objet	Classe
Principal	QMainWindow
centralWidget	QWidget
hautesLumieres	QSlider
lum	QSpinBox
luminosite	QSlider
noir	QSpinBox
ombres	QSlider
photo	Photo
tonsClairs	QSpinBox
barreOutils	QToolBar
actionOuvrir	QAction
actionSauver	QAction
actionExif	QAction
actionPhotoOriginale	QAction
actionAnnuler	QAction

Filter

lum : QSpinBox

Propriété	Valeur
Vertical	AlignementCentreV
readOnly	☐
buttonSymbols	PlusMinus
specialValueText	
accelerated	☐
correctionMode	CorrectToPreviousValue
keyboardTracking	☑
QSpinBox	
suffix	
prefix	Luminosité :
minimum	-70
maximum	70
singleStep	5
value	0
displayIntegerBase	10

Filter

Nom	Utilisé	Texte	Raccourci	Vérifiable	Info-bulle
actionOuvrir	☑	Ouvrir	Ctrl+O	☐	Sélectioner une nouvelle photo
actionSauver	☑	Sauver	Ctrl+S	☐	Enregistrer la photo
actionExif	☑	Exif	Ctrl+E	☑	Toutes les caractéristiques de l'image
actionPhotoOriginale	☑	Photo Originale		☑	Revoir la photo originale
actionAnnuler	☑	Annuler	Ctrl+A	☐	Annuler tous les réglages de l'image

Éditeur d'action | Éditeur de signaux et slots

Luminosité : 0 | Ombres : 0 | Tons clairs : 0

Objet	Classe
Principal	QMainWindow
centralWidget	QWidget
hautesLumieres	QSlider
lum	QSpinBox
luminosite	QSlider
noir	QSpinBox
ombres	QSlider
photo	Photo
tonsClairs	QSpinBox
barreOutils	QToolBar
actionOuvrir	QAction
actionSauver	QAction
actionExif	QAction
actionPhotoOriginale	QAction
actionAnnuler	QAction

Filter

noir : QSpinBox

Propriété	Valeur
frame	☑
alignment	AlignementGauche, Alig...
Horizontal	AlignementGauche
Vertical	AlignementCentreV
readOnly	☐
buttonSymbols	UpDownArrows
specialValueText	
accelerated	☐
correctionMode	CorrectToPreviousValue
keyboardTracking	☑
QSpinBox	
suffix	
prefix	Ombres :
minimum	0
maximum	70
singleStep	5
value	0
displayIntegerBase	10

Émetteur	Signal	Receveur	Slot
tonsClairs	valueChanged(int)	photo	changerTonsClairs(int)
tonsClairs	valueChanged(int)	hautesLumieres	setValue(int)
photo	raz(int)	lum	setValue(int)
photo	raz(int)	noir	setValue(int)
photo	raz(int)	tonsClairs	setValue(int)
ombres	valueChanged(int)	noir	setValue(int)
noir	valueChanged(int)	photo	changerOmbres(int)
noir	valueChanged(int)	ombres	setValue(int)
luminosite	valueChanged(int)	lum	setValue(int)
lum	valueChanged(int)	photo	changerLuminosite(int)
lum	valueChanged(int)	luminosite	setValue(int)
hautesLumieres	valueChanged(int)	tonsClairs	setValue(int)
actionSauver	triggered()	photo	sauverPhoto()
actionPhotoOriginale	toggled(bool)	photo	choisirOriginal(bool)
actionOuvrir	triggered()	photo	chargerPhoto()
actionExif	toggled(bool)	photo	donneesExif(bool)
actionAnnuler	triggered()	photo	annuler()

Éditeur d'action | Éditeur de signaux et slots