

## Introduction à Qt 4

---

Eric Lecolinet

Ecole Nationale Supérieure des Télécommunications  
(ENST/INFRES)

- <http://www.infres.enst.fr/~elc> -> cours -> vihm
- [\[premier transparent\]](#)
- [\[transparentes sur un seul fichier\]](#)
- [\[index\]](#)

## Liens utiles

---

### A l'ENST

- [Page Qt ENST](#)
- [Doc Qt \(local\)](#)
- [Tutoriel C++](#)

### Site Troll Tech

- [Site Qt](#)

## Toolkit Qt (TrollTech)

---

### Toolkit graphique complet en C++

- Relativement simple (ressemble un peu à Swing)
- Nombreux outils et extensions
- Gratuit (GPL) pour usage non commercial
- Base de l'environnement KDE

### Multi-plateformes

- Unix, Windows et Mac OS X (Qt version 3)
- Linux embarqué (avec Frame buffer virtuel)
- Look and feel simulé et paramétrable (comme Swing)
- Implémenté sur couches basses des systèmes graphiques

# Caractéristiques et références

---

## Caractéristiques, extensions, outils

- Internationalisation: QString et Unicode
- Open GL multiplateformes
- XML: parseurs SAX et DOM
- Bases de données SQL
- Générateur d'interfaces Qt Designer

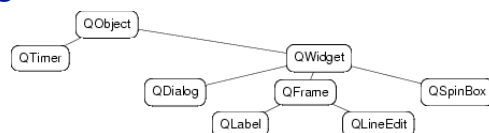
## Références

- Développé par Troll Tech: <http://www.trolltech.com/>
- Site KDE: <http://www.kde.org/>
- Documentation: <http://www.trolltech.com/developer/documentation/>
- A l'ENST: <http://www.infres.enst.fr/~elc/qt/doc-html/>

# Principaux widgets

---

## Arbre d'héritage



# Hello Word! : première version

---

```
#include <QApplication>
#include <QLabel>

int main(int argc, char **argv)
{
    QApplication* app = new QApplication(argc, argv);

    QLabel* hello = new QLabel("Hello Qt!");

    hello->show();
    return app->exec();
}
```



## Remarques

- attention aux -> et aux \*
- les pointeurs ne sont pas des références !
- parent passé en argument ou 0 (NULL) pour "top-level" widgets
- texte HTML simplifié (selon version de Qt)

# Deuxième version

---

```
#include <QApplication>
#include <QLabel>

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

    QLabel hello("Hello Qt!");
    hello.resize(100, 30);

    hello.show();
    return app.exec();
}
```



## Remarques

- objets créés dans la pile -> détruits en fin de fonction !
- notation pointée
- Qt3: il faudrait aussi: app.setMainWidget(&hello);

## 3e version avec conteneur intermédiaire

```
#include <QApplication>
#include <QPushButton>
#include <QWidget>
#include <QFont>

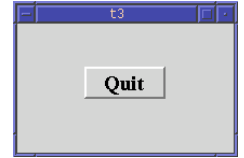
int main(int argc, char **argv)
{
    QApplication app(argc, argv);

    QWidget box;          // ne pas mettre les ()
    box.resize(200, 120);

    QPushButton quitBtn("Quit", &box);
    quitBtn.resize(100, 50);
    quitBtn.move(50, 35);
    quitBtn.setFont(QFont("Times", 18, QFont::Bold));

    QObject::connect(&quitBtn, SIGNAL(clicked()), &app, SLOT(quit()));

    box.show();
    return app.exec();
}
```



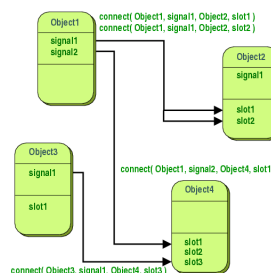
## Signaux et Slots

### Signaux

- les objets héritant de QObject (et donc, QWidget)
- émettent des **signaux** quand ils reçoivent des événements

### Slots

- les signaux peuvent être connectés à des **slots** :
- un slot = fonction membre ("méthode") appelée automatiquement



### Connexion

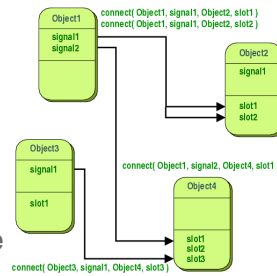
```
QApplication app(argc, argv);
....
QPushButton quitBtn("Quit", &box);
....
QObject::connect( &quitBtn, SIGNAL(clicked()), &app, SLOT(quit()) );
```

# Philosophie

---

## un Signal est émis vers le "monde extérieur"

- par un objet quand il change d'état
  - l'émetteur n'a pas besoin de connaître les récepteurs
  - il ne sait pas si le signal est reçu
  - le receveur n'a pas non plus besoin de connaître l'émetteur
- -> programmation par composants



# Propriétés

---

## Modularité, flexibilité

- le même signal peut être connecté à plusieurs slots
- plusieurs signaux peuvent être connectés à un même slot

## Sécurité, typage fort

- les types des paramètres des signaux et des slots doivent être les mêmes
- un slot peut avoir **moins** de paramètres qu'un signal

```
QObject::connect( &quitBtn, SIGNAL(clicked()), &app, SLOT(quit()) );
```

# Remarques

---

## Remarques

- aspect central de Qt
- diffère de l'habituel mécanisme des "callbacks" (ou Listeners)

## Avantages / inconvénients

- **SLOT** et **SIGNAL** sont des MACROS
- nécessite une phase de précompilation
- plus sur (typage) et plus modulaire

# Déclaration de slot

---

## Dans le fichier header (.h, .hh)

```
class BankAccount : public QObject {  
    Q_OBJECT  
private:  
    int curBalance;  
  
public:  
    BankAccount() { curBalance = 0; }  
    int getBalance() const { return curBalance; }  
  
public slots:  
    void setBalance( int newBalance );  
  
signals:  
    void balanceChanged( int newBalance );  
};
```

- sous classe de **QObject**
- mot-clés **Q\_OBJECT**, **slots** et **signals** pour précompilateur
- les **signaux** ne sont pas implémentés
- les **slots** doivent être implémentés

## Définition de slot

---

### Dans le fichier source de l'implémentation (.cpp, .cc)

```
void BankAccount::setBalance(int newBalance) {  
    if ( newBalance != curBalance ) {  
        curBalance = newBalance;  
        emit balanceChanged(curBalance);  
    }  
}
```

- mot-clé **emit** pour précompilateur
- provoque l'"émission"
  - du signal "balanceChanged"
  - avec la nouvelle valeur de "curBalance"
- Note: on vérifie que la valeur a effectivement changé
  - pour éviter boucles infinies !

## Connexion

---

```
BankAccount x, y;
```

```
connect( &x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)) );
```

```
x.setBalance( 2450 );
```

- x est mis à 2450
- le signal balanceChanged() est émis
- il est reçu par le slot setBalance() de y
- y est mis à 2450

# Compilation

---

## Meta Object Compiler (MOC)

- pré-processeur C++
- génère du code supplémentaire
  - tables de signaux /slots
- permet aussi de récupérer le nom de la classe et de faire des test d'héritage
- **attention:** ne pas oublier le mot-clé `Q_OBJECT`

## Utilisation de qmake

- dans le répertoire contenant les fichiers sources faire:

```
qmake -project // crée le fichier xxx.pro
qmake // crée le fichier Makefile
make // crée les fichiers moc (un par fichier
// définissant des slots),
// et les fichiers binaires (*.o et exécutable)
```

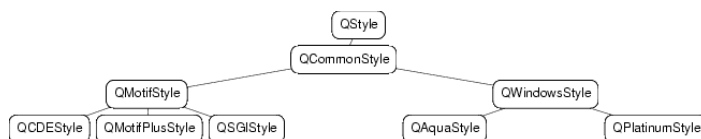
# Widgets

---

## Emulation du "Look and Feel"

- Qt émule (i.e. n'utilise pas) les widgets natifs Windows ou Motif
- rapidité, flexibilité, extensibilité,
- pas restreint à un "dénominateur commun"
- même principe que Swing mais le contraire d'AWT Components

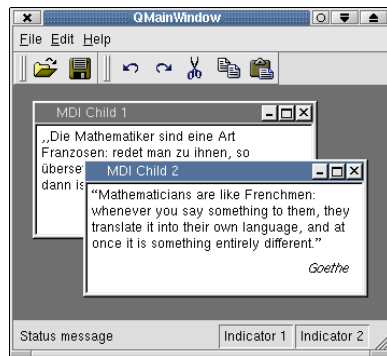
## QStyle



```
QApplication::setStyle( new MyCustomStyle );
```

## QMainWindow: menus et actions

---



```
QMenuBar* menubar = menuBar(); // menuBar() est une methode de QMainWindow

QMenu* filemenu = menubar->addMenu( tr("&File" ) );

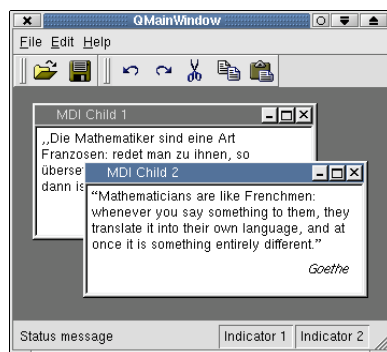
QAction* action = new QAction( QIcon(":/images/new.png"), tr("&New..."), this );
action->setShortcut( tr("Ctrl+N" ) );
action->setStatusTip( tr("New file" ) );

connect( action, SIGNAL(triggered()), this, SLOT(open()) );

filemenu->addAction( action );
```

## QMainWindow: toolbars, etc.

---



- QToolBar

```
QToolBar* toolbar = addToolBar( tr("File" ) );
toolbar->addAction(newAct);
```

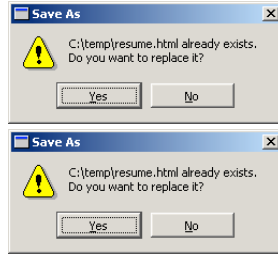
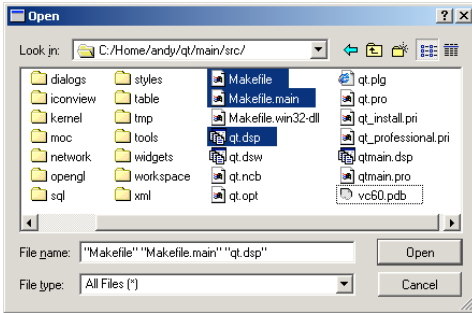
- Central widget

```
QTextEdit* text = new QTextEdit(this);
setCentralWidget(text);
```

- QToolTip, etc...

# Boîtes de dialogue

## QMessageBox, QFileDialog, QProgressDialog



- Et aussi: QWizard, QFontDialog, etc.

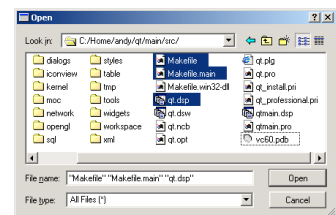
## Utilisation d'un dialogue modal

- Solution générale

```
QFileDialog dialog(parent);  
  
dialog.setFilter("Text files (*.txt)");  
QStringList file_names;  
  
if (dialog.exec() == QDialog::Accepted) {  
    file_names = dialog.selectedFiles();  
    QString first_name = file_names[0];  
    ...  
}
```

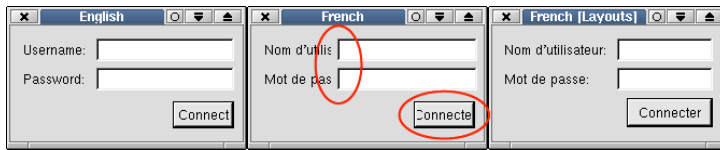
- Solution simplifiée

```
QString fileName = QFileDialog::getOpenFileName(this,  
tr("Open Image"),  
"/home/jana",  
tr("Image Files (*.png *.jpg *.bmp)"));
```

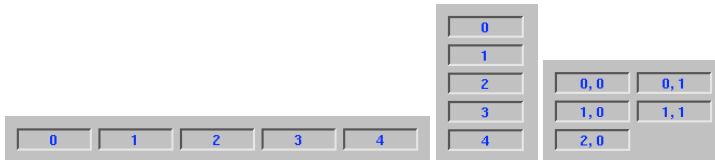


# Layout

---



**QHBoxLayout, QVBoxLayout, QGridLayout :**



# Exemple de spécification de Layout

---



```
QVBoxLayout *v_layout = new QVBoxLayout();
```

```
v_layout->addWidget( new QPushButton("OK") );  
v_layout->addWidget( new QPushButton("Cancel") );  
v_layout->addStretch( 1 );  
v_layout->addWidget( new QPushButton("Help") );
```

```
QListbox *country_list = new QListbox( this );  
countryList->insertItem( "Canada" );  
...etc...
```

```
QHBoxLayout *h_layout = new QHBoxLayout();  
h_layout->addWidget( country_list );  
h_layout->addLayout( v_layout );
```

```
QVBoxLayout *top_layout = new QVBoxLayout();  
topLevelBox->addWidget( new QLabel("Select a country", this) );  
topLevelBox->addLayout( h_layout );
```

```
window->setLayout(top_layout);  
window->show();
```

## Graphique 2D

---

### QIcon



```
QPushButton *button = new QPushButton( "&Find Address", parent );  
button->setIcon( QIcon(":/images/new.png") );
```

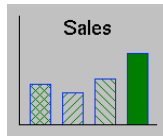
### QImage

### QPainter

- modèle graphique évolué (rotations, indépendance / pilote ...)
- équivalent de Graphics2D de Java
- exemples de méthodes :
  - painter.**setPen**( QPen(red, 2, DashLine) );
  - painter.**drawRect**( 25, 15, 120, 60 );

## Exemple avec QPainter

---



### Redéfinir la méthode paintEvent

```
void BarGraph::paintEvent(QPaintEvent *)
{
    QWidget::paintEvent(e);          // ! ne pas oublier: reaffiche le widget

    QPainter painter( this );

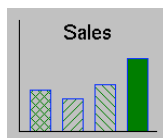
    draw_bar( &painter, 0, 39, Qt::DiagCrossPattern );
    draw_bar( &painter, 1, 31, Qt::BDiagPattern );
    draw_bar( &painter, 2, 44, Qt::FDiagPattern );
    draw_bar( &painter, 3, 68, Qt::SolidPattern );

    painter.setPen( black );
    painter.drawLine( 0, 0, 0, height() - 1 );
    painter.drawLine( 0, height() - 1, width() - 1, height() - 1 );

    painter.setFont( QFont("Helvetica", 18) );
    painter.drawText( rect(), AlignHCenter | AlignTop, "Sales" );
}
```

## Exemple avec QPainter (suite)

---



```
void BarGraph::paintEvent(QPaintEvent *)
{
    QWidget::paintEvent(e);
    QPainter painter( this );

    draw_bar( &painter, 0, 39, Qt::DiagCrossPattern );
    draw_bar( &painter, 1, 31, Qt::BDiagPattern );
    .....
}

void BarGraph::draw_bar(QPainter *painter, int month, int barHeight,
                        BrushStyle pattern)
{
    painter->setPen( blue );
    painter->setBrush( QBrush( darkGreen, pattern ) );
    painter->drawRect( 10 + 30 * month, height() - barHeight, 20, barHeight );
}
```

# Graphique Structuré

---

- Qt3 : QCanvas
  - graphe de scène (graphique 2D orienté objet)
  - contient des QCanvasItem
  - peut avoir plusieurs QCanvasView
- Qt4 : Graphics View
  - QGraphicsScene, QGraphicsView, QGraphicsItem...

# OpenGL

---

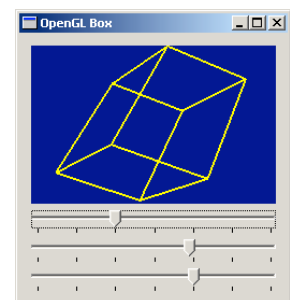
```
#include <QGLWidget>

class Box3D : public QGLWidget {
    Q_OBJECT
public:
    Box3D( QWidget *parent = 0);
    ~Box3D();

public slots:
    void setRotationX( int deg ) { rotX = deg; updateGL(); }
    void setRotationY( int deg ) { rotY = deg; updateGL(); }
    void setRotationZ( int deg ) { rotZ = deg; updateGL(); }

protected:
    virtual void initializeGL();
    virtual void paintGL();
    virtual void resizeGL( int w, int h );
    virtual GLuint makeObject();

private:
    GLuint object;
    GLfloat rotX, rotY, rotZ;
};
```



## OpenGL 3(suite)

---

```
#include "box3d.h"

Box3D::Box3D( QWidget *parent ) : QGLWidget( parent ) {
    object = 0;
    rotX = rotY = rotZ = 0.0;
}

Box3D::~Box3D() {
    makeCurrent();
    glDeleteLists( object, 1 );
}

void Box3D::initializeGL() {
    qglClearColor( darkBlue );
    object = makeObject();
    glShadeModel( GL_FLAT );
}

void Box3D::paintGL() {
    glClear( GL_COLOR_BUFFER_BIT );
    glLoadIdentity();
    glTranslatef( 0.0, 0.0, -10.0 );
    glRotatef( rotX, 1.0, 0.0, 0.0 );
    glRotatef( rotY, 0.0, 1.0, 0.0 );
    glRotatef( rotZ, 0.0, 0.0, 1.0 );
    glCallList( object );
}

void Box3D::resizeGL( int w, int h ) {
```

```
    glViewport( 0, 0, w, h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glFrustum( -1.0, 1.0, -1.0, 1.0, 5.0, 15.0 );
    glMatrixMode( GL_MODELVIEW );
}

GLuint Box3D::makeObject() {
    GLuint list = glGenLists( 1 );
    glNewList( list, GL_COMPILE );
    glColor( yellow );
    glLineWidth( 2.0 );

    glBegin( GL_LINE_LOOP );
    glVertex3f( +1.5, +1.0, +0.8 );
    glVertex3f( +1.5, +1.0, -0.8 );
    /* ... */
    glEnd();

    glEndList();
    return list;
}
```

## OpenGL (suite)

---

```
#include <qapplication.h>
#include <qslider.h>
#include <qvbox.h>
#include "box3d.h"

void create_slider( QWidget *parent, Box3D *box3d, const char *slot )
{
    QSlider *slider = new QSlider( 0, 360, 60, 0, QSlider::Horizontal, parent );
    slider->setTickmarks( QSlider::Below );
    QObject::connect( slider, SIGNAL(valueChanged(int)), box3d, slot );
}

int main( int argc, char **argv )
{
    QApplication::setColorSpec( QApplication::CustomColor );
    QApplication app( argc, argv );
    if ( !QGLFormat::hasOpenGL() )
        qFatal( "This system has no OpenGL support" );

    QVBox *parent = new QVBox;
    parent->setCaption( "OpenGL Box" );
    parent->setMargin( 11 );
    parent->setSpacing( 6 );

    Box3D *box3d = new Box3D( parent );
    create_slider( parent, box3d, SLOT(setRotationX(int)) );
    create_slider( parent, box3d, SLOT(setRotationY(int)) );
    create_slider( parent, box3d, SLOT(setRotationZ(int)) );

    app.setMainWidget( parent );
    parent->resize( 250, 250 );
    parent->show();
    return app.exec();
}
```

