

KDE/Qt



Alexis Nédélec

Ecole Nationale d'Ingénieurs de Brest
Technopôle Brest-Iroise, Site de la Pointe du Diable
CP 15 29608 BREST Cedex (FRANCE)
e-mail : nedelec@enib.fr

Table des Matières

Introduction	3	Menus Surgissants	71
Hello World	18	Sauvegardes	74
Événements	26	Actions	79
Signaux / Slots	30	Affichage 2D	83
Gestionnaires de positionnement	36	Canevas d'affichage	96
Listes	47	Affichage 3D : OpenGL	83
Menus	58	Qt Designer	114
Zone Client	64	Bibliographie	117

Introduction

Qu'est Qt (<http://www.trolltech.com>):

- ▷ Environnement de développement d'applications graphiques de KDE
- ▷ une bibliothèque de classes C++
- ▷ orienté développement d'IHM graphiques (GUI)
- ▷ portable sous Windows et Unix

Qt est écrit C++:

- ▷ programmation objet, événementielle
- ▷ mécanismes de signaux et de slots (**moc**)
- ▷ binding **C**, **python**, **C#**

Devise:

- ▷ écrire une seule fois, compiler n'importe où !

Introduction

Qt n'est pas que dédié IHM

- ▷ graphique 2D/3D (Open GL)
- ▷ connexion aux Bases de Données
- ▷ gestion de fichiers
- ▷ communication inter-processus
- ▷ communication réseau
- ▷ XML,SAX,DOM
- ▷ multithreading

Environ 400 classes Qt disponibles

Classes Qt

Regroupement de classes suivant leurs fonctionnalités:

1. **Abstract Widgets:** exploitables par héritage
 - ▷ `QPushButton`: `QPushButton`, `QRadioButton`, `QToolButton`...
2. **Basic Widgets:** les widgets les plus simples de l'API
 - ▷ `QAction`: gestion de l'interaction utilisateur
 - ▷ `QLabel`: affichage de texte
 - ▷ `QTextEdit`: écriture de texte
 - ▷ ...
3. **Advanced Widgets:** contrôle plus complexe de l'interface
 - ▷ `QListView`: implémente la gestion des listes, arbres multicolones
 - ▷ `QMultiLineEdit`: hérite de `QTextEdit`
 - ▷ ...

Classes Qt

4. **MainWindow and Related Classes:** pour l'application principale
 - ▷ QApplication, QEventLoop, QMainWindow, QWorkspace...
5. **Layout Management:** gestion de la géométrie de widgets
 - ▷ QVBoxLayout, QGridLayout, QGroupBox...
6. **Standard Dialogs:** boîtes de dialogue usuelles
 - ▷ QDialog, QFileDialog, QWizard...
7. **Object Model:** classes de base du modèle objet de l'API
 - ▷ QMetaObject, QMetaProperty, QObject...
8. **Events:** création et gestion des événements
 - ▷ QEvent, QKeyEvent, QMouseEvent, QTimer...

Classes Qt

9. **Graphics and Printing:** primitive de tracé 2D/3D
 - ▷ QColor, QFont, QGL, QImage, QPainter...
10. **Multimedia:** graphique, son, animation
 - ▷ QImage, QMovie, QSound...
11. **Text Related Classes:** gestion de texte
 - ▷ QChar, QString, QTextEdit, QTextBrowser...
12. **Widget Appearance :** personnalisation de widgets
 - ▷ QMotifStyle, QStyle, QWindowsStyle...
13. **Miscellaneous Classes :** widgets inclassables !
 - ▷ QAccel, QRegExp, QUrl...

Classes Qt

14. **Template Library** : gestion des containers

▷ QMap, QMap, QList, QStack, QVector...

15. **Utility Classes** : gestion des collections

▷ QMap, QDict, QMap, QList, QStringList...

16. **Threading Classes** : gestion des threads

▷ QMutex, QSemaphore, QThread, QWaitCondition...

17. **XML Classes** : gestion de documents XML

▷ QDomElement, QDomNode, QDomAttributes, QDomReader...

18. **Database Classes** : connexion aux SGBD

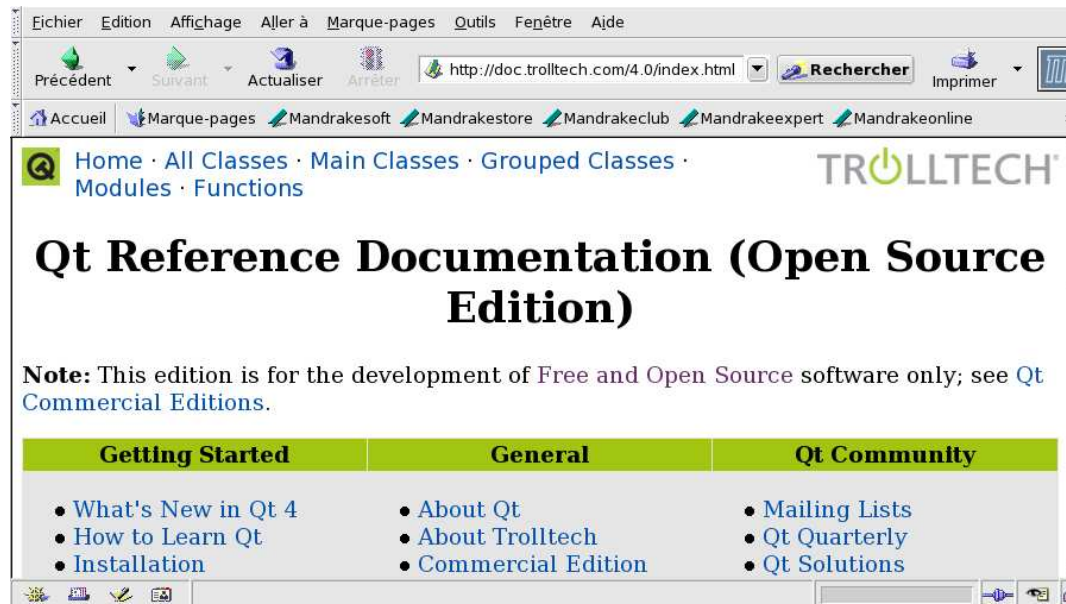
▷ QSqlTable, QSql, QSqlDriver, QSqlQuery, QSqlRecord...

Classes Qt

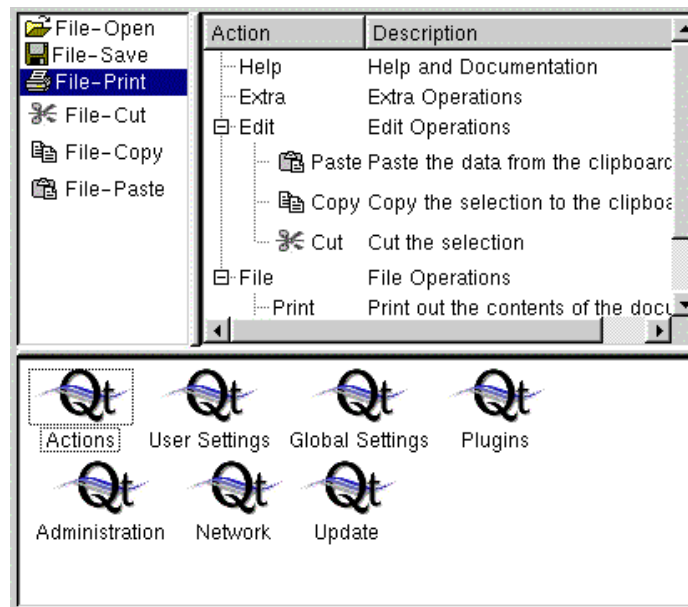
19. **Date and Time** : gestion du temps
 - ▷ QDate, QTime, QTimer...
20. **Environnement** : services généraux, internationalisation
 - ▷ QDesktopWidget, QSessionManager, QTranslator...
21. **Help system** : aide en ligne sur l'application
 - ▷ QStatusBar, QToolTip, QWhatsThis...
22. **Input/Output and Networking** : communication
 - ▷ QDataStream, QFile, QHttp, QNetworkProtocol...
23. **Shared Classes** : classes partagées (compteur de références)
 - ▷ QPixmap, QFont, QImage, QPixmap1, QValueStack...

Classes Qt

- 24. **Organizer:** organisation d'interfaces utilisateurs
 - ▷ QPushButtonGroup, QSplitter, QWidgetStack..
- 25. **Plugin Classes:** gestion des bibliothèques partagées
 - ▷ QLibrary, QSqlDriverPlugin, QWidgetPlugin...



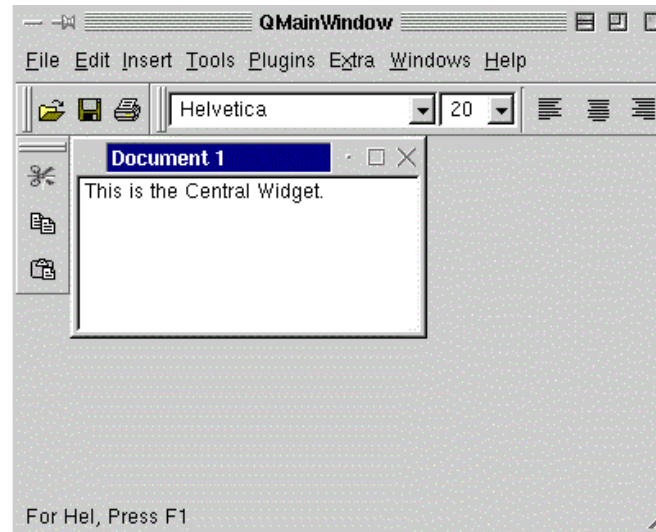
Exemples d'utilisations



Trois vues séparées par des QSplitter

- ▷ QListView: en haut à gauche
- ▷ QListView, QHeader, QScrollBar: en haut à droite
- ▷ QIconView: en bas

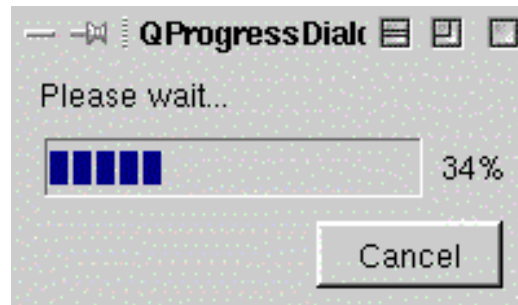
Exemples d'utilisations



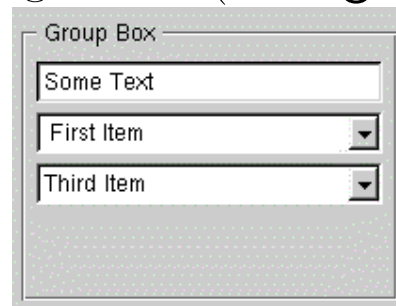
Représentation de fenêtre principale `QMainWindow` contenant

- ▷ `QMenuBar`: barre d'actions, de menu
- ▷ `QToolBar` avec `QToolButton`, `QComboBox`: les barres d'outils
- ▷ `QWorkspace`: zone client pour applications MDI
- ▷ `QMultiLineEdit`: Premier document MDI pour éditer
- ▷ `QStatusBar`, `QsizeGrip`: en bas à gauche, à droite

Exemples d'utilisations



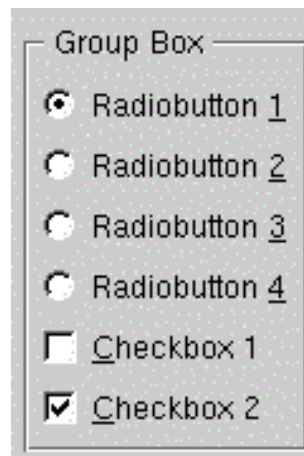
Représentation de barre de progression (QProgressDialog, QProgressBar)



Représentation d'un groupe de widgets QGroupBox contenant

- ▷ QLineEdit
- ▷ QComboBox

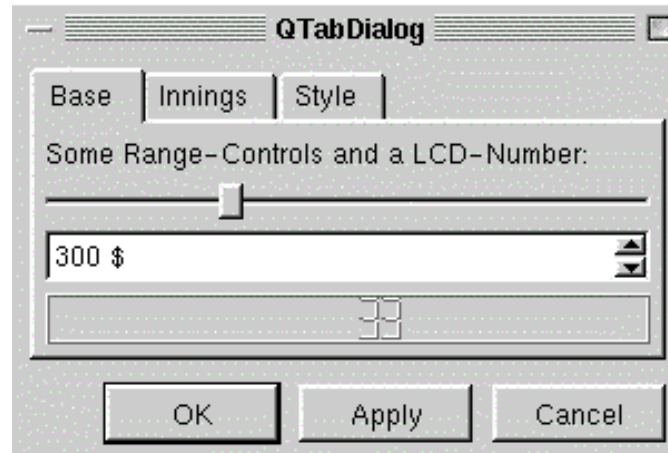
Exemples d'utilisations



Représentation d'un groupe de widgets `QButtonGroup` contenant

- ▷ `QRadioButton`
- ▷ `QCheckBox`

Exemples d'utilisations



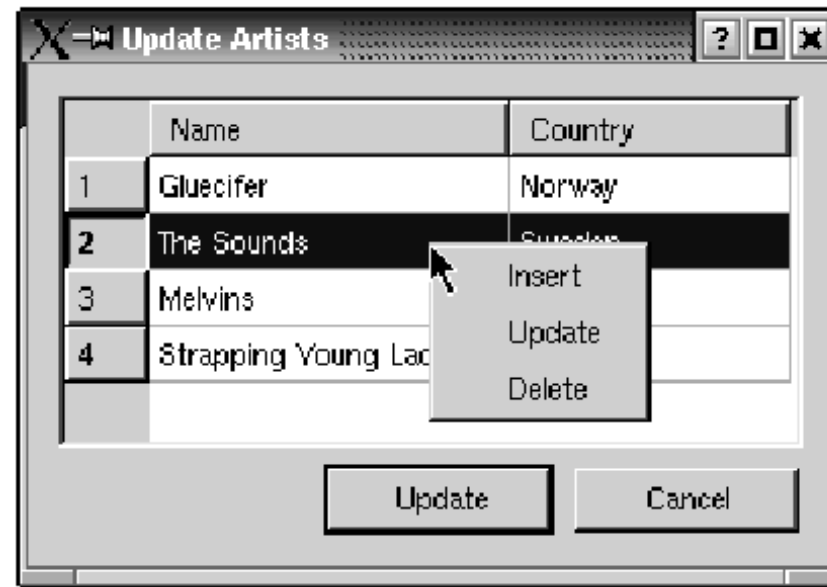
Représentation d'une table `QTabDialog` contenant des

- ▷ `QTabBar`, `QTabWidget`: onglets

La page représentée contient

- ▷ `QLabel`: affichage du texte
- ▷ `QSlider`: modification de valeurs
- ▷ `QSpinBox`: affichage et modification de valeurs
- ▷ `QLCDNumber`: affichage de valeurs

Exemples d'utilisations



- ▷ QSqlDatabase: connexion sur une base de données
- ▷ QSqlCursor: parcours des enregistrements
- ▷ QDataTable: affichage du parcours
- ▷ QSqlQuery: lancer des requêtes (Insert, Update, Delete)

Exemples d'utilisations



Représentation d'une page HTML: QTextBrowser, QTextView

Hello World

Interaction sur un composant graphique par :

- ▷ gestion d'événements (héritage `QWidget`)
- ▷ connexion de signaux et slots (héritage `QObject`)

Interaction par gestion d'événements (`main.cpp`) :

```
#include <qapplication.h>
#include <qwidget.h>
#include "event.h"
int main( int argc, char **argv ) {
    QApplication app( argc, argv );
    MyWidget*  helloWidget = new MyWidget();
    helloWidget->setGeometry(50, 500, 400, 400);
    app.setMainWidget( helloWidget );
    helloWidget->show();
    return app.exec();
}
```

Héritage QWidget

Surdéfinition de méthodes :

▷ `mousePressEvent (QMouseEvent *)`

▷ `keyPressEvent (QKeyEvent *)`

▷ ...

Définition de classe (`event.h`) :

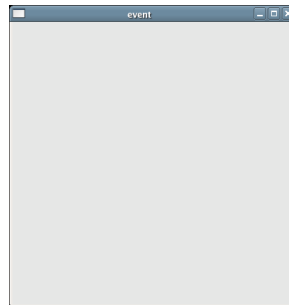
```
#ifndef HELLOWORLD_EVENT_H
#define HELLOWORLD_EVENT_H
#include <iostream>
#include <qwidget.h>
class MyWidget : public QWidget {
public:
    MyWidget();
protected:
    virtual void mousePressEvent ( QMouseEvent * );
private:
    QPoint _click;
};
#endif
```

Héritage QWidget

Implémentation de classe (`event.cpp`) :

```
#include "event.h"
MyWidget::MyWidget (void) { }
void MyWidget::mousePressEvent ( QMouseEvent *event) {
    _click = event->pos();
    std::cerr << "MyWidget::mousePressEvent" << std::endl;
    std::cerr << "X: " << _click.x() << " Y: " << _click.y() << std::endl;
}
```

Test du programme:



```
{logname@hostname} event
MyWidget::mousePressEvent
X: 136 Y: 146
```

Héritage QObject

Interaction par signaux et slots (`main.cpp`) :

```
#include <qapplication.h>
#include <qpushbutton.h>
#include "hello.h"

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

    QPushButton helloPushButton( "Hello world!", 0 );
    helloPushButton.resize( 100, 30 );
    Hello helloObject(0);
    QObject::connect( &helloPushButton , SIGNAL(clicked()),
                    &helloObject , SLOT(helloSlot()) );
    app.setMainWidget( &helloPushButton );
    helloPushButton.show();
    return app.exec();
}
```

Héritage QObject

Définition de classe (**hello.h**):

```
#include <iostream>
using namespace std;
#include <qdialog.h>
#include "hello.h"
class Hello : public QObject {
    Q_OBJECT
public:
    Hello( QObject* parent = 0, const char* name = 0);
public slots:
    void helloSlot();
};
```

Implémentation de classe (**hello.cpp**):

```
#include "hello.h"
Hello::Hello( QObject* parent, const char* name) { }
void
Hello::helloSlot() { cerr << "Hello::helloSlot()" << endl; }
```

Héritage QObject

Création d'un projet, compilation, édition de liens, création de l'exécutable:

```
{logname@hostname} ls  
hello.cpp hello.h main.cpp  
{logname@hostname} qmake -project; qmake hello.pro; make  
{logname@hostname} ls  
hello*      hello.h hello.pro main.o    moc_hello.cpp  
hello.cpp  hello.o main.cpp  Makefile  moc_hello.o
```

Test du programme :



```
{logname@hostname} hello  
Hello::helloSlot()
```

Gestion d'Événements

Classe de base pour gérer les événements:

▷ **QWidget**

Regroupement de méthodes suivant leurs fonctionnalités:

1. fonctionnalités de fenêtre racine (Toplevel):

▷ `windowTitle()`, `setWindowIcon()`, `activateWindow()`...

2. fonctionnalités de fenêtre:

▷ `show()`, `raise()`, `close()`...

3. contenu de fenêtre:

▷ `update()`, `repaint()`, `scroll()`...

4. géométrie de fenêtre:

▷ `pos()`, `rect()`, `x()`, `height()`, `move()` ...

Gestion d'Événements

5. Look and feel :

▷ `setStyle()`, `cursor()`, `fontInfo()`...

6. focus clavier :

▷ `setFocusPolicy()`, `hasFocus()`, `clearFocus()` ...

7. saisie souris, clavier :

▷ `grabMouse()`, `releaseKeyboard()`, `mouseGrabber()`...

8. gestion d'événements :

▷ `event()`, `mousePressEvent()`, `paintEvent()`, `dragMoveEvent()`
...

9.

Gestion d'Événements

```
// main.cpp
#include <qapplication.h>
#include "painter.h"

int main( int argc, char **argv ) {
    QApplication app( argc, argv );
    PainterWindow* paint = new PainterWindow();
    paint->setGeometry(50, 500, 400, 400);
    app.setMainWidget( paint );
    paint->show();
    return app.exec();
}
```



Gestion d'Événements

```
// painter.h
class PainterWindow : public QWidget
{
public:
    PainterWindow();
protected:
    virtual void mousePressEvent (QMouseEvent* event);
    virtual void mouseMoveEvent (QMouseEvent* event);
    virtual void paintEvent (QPaintEvent* event);
    virtual void resizeEvent (QResizeEvent* event);
private:
    QPoint    _last;
    QPixmap   _buffer;
};
```

Gestion d'Événements

```
// painterWindow.cpp
PainterWindow::PainterWindow(void) {
    setBackgroundMode( NoBackground );
}
void PainterWindow::mousePressEvent (QMouseEvent* event) {
    _last = event->pos();
}
void PainterWindow::mouseMoveEvent (QMouseEvent* event) {
    QPainter paintWindow;
    QPainter paintBuffer;

    paintWindow.begin(this);
    paintBuffer.begin(&_buffer);
    paintWindow.drawLine(_last, event->pos());
    paintBuffer.drawLine(_last, event->pos());
    paintWindow.end();
    paintBuffer.end();
    _last = event->pos();
}
```

Gestion d'Événements

```
void
PainterWindow::paintEvent (QPaintEvent* event)
{
    bitBlt(this, 0, 0, &_buffer);
    // bit Block transfer : copie d'une zone de pixel entre 2 QPaintDevice
    // bitBlt( destinationWidget,    destPositionInWidget,
    //        sourceWidgetOrPixmap, sourceRectangleAreaToCopy )
}
void
PainterWindow::resizeEvent (QResizeEvent* event)
{
    QPixmap save(_buffer);
    _buffer.resize(event->size());
    _buffer.fill( white );
    bitBlt(&_buffer, 0, 0, &save);
}
```

Connexion de Signal/Slot

Rôles des signaux et des slots:

- ▷ faire communiquer les objets entre eux

En programmation d'IHM il est souvent nécessaire de

- ▷ notifier à un composant qu'un changement s'est produit sur un autre

En programmation événementielle classique:

- ▷ mécanismes des fonctions réflexes (callbacks)
- ▷ pas de vérification de type des données clientes
- ▷ fonction réflexe fortement couplée à la fonction appelante

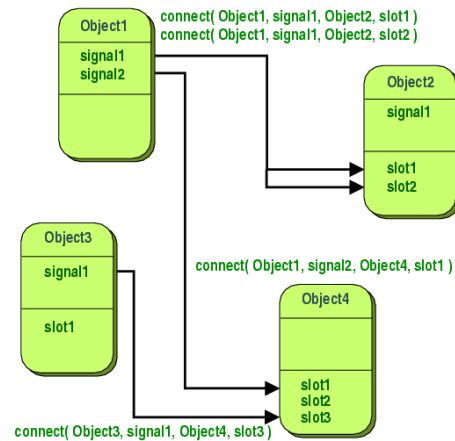
Alternative Qt:

- ▷ émission de signal par un widget lorsqu'un événement survient
- ▷ déclenchement d'un slot en réponse à l'émission d'un signal

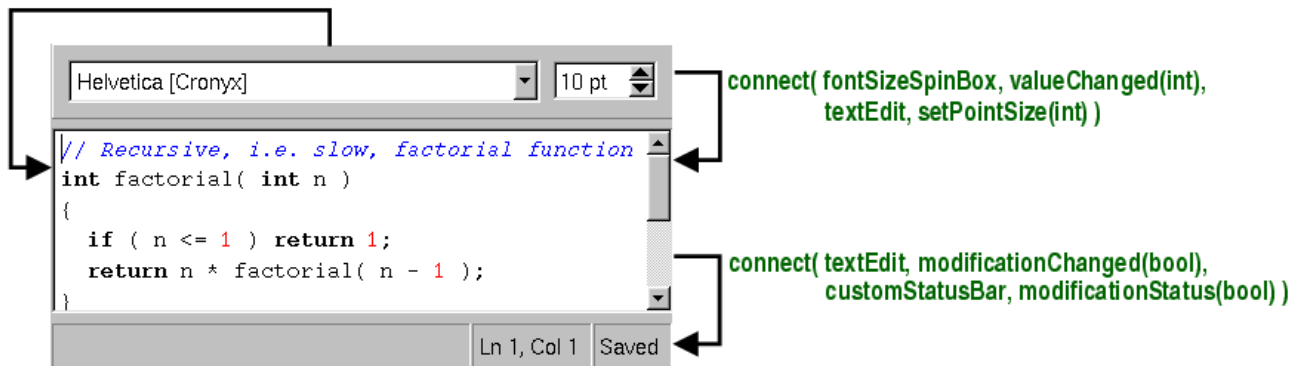
Avec le mécanisme Qt de gestion des signaux et des slots:

- ▷ garantie de types (signature de méthodes)
- ▷ couplage faible (connexions 1 signal/des slots et réciproquement)

Connexion de Signal/Slot



```
connect( fontFamilyComboBox, activated(QString),
        textEdit, setFamily(QString) )
```



Connexion de Signal/Slot

Une classe C++ avec signal et slots (`slotsignal.h`)

```
#include <qobject.h>

class SlotSignal : public QObject {
Q_OBJECT
public:
    SlotSignal(): _val(0) {}
    int value() const { return _val; }
public slots:
    void setValue( int );
signals:
    void valueChanged( int );
private:
    int _val;
};
```

Connexion de Signal/Slot

Emission de signal par un composant (`slotsignal.cpp`):

```
void MyObject::setValue( int v ) {  
    if ( v != _val ) {  
        _val = v;  
        emit valueChanged(v);  
    }  
}
```

Test d'émission de signal (`main.cpp`):

```
int main(int argc, char* argv[]) {  
    SlotSignal a, b;  
  
    QObject::connect(&a, SIGNAL(valueChanged(int)), &b, SLOT(setValue(int)));  
    b.setValue( 10 );  
    cerr << a.value() << endl ; // 0 or 10 ?  
    a.setValue( 100 );  
    cerr << b.value() << endl ; // 10 or 100 ?  
}
```

Connexion de Signal/Slot

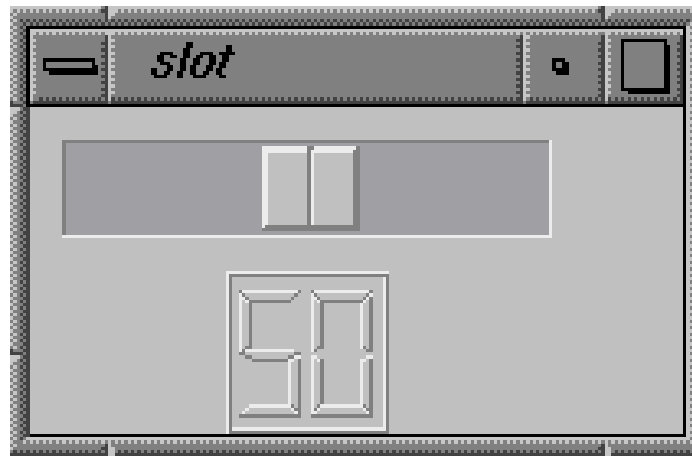
```
#include <qapplication.h>
#include <qslider.h>
#include <qlcdnumber.h>

int main( int argc, char **argv ) {
    QApplication myApp( argc, argv );
    QWidget* myWindow = new QWidget();
    myWindow->setGeometry(10, 10, 200, 100);
    QSlider* mySlider = new QSlider( 0, 99,           // min,max
                                    1, 50,           // increment, valeur initiale
                                    QSlider::Horizontal,
                                    myWindow);       // le parent

    mySlider->setGeometry(10, 10, 150, 30);
    QLCDNumber* myLCDNumber = new QLCDNumber( 2, myWindow );
    myLCDNumber->setGeometry(60, 50, 50, 50);
    myLCDNumber->display(50);
}
```

Connexion de Signal/Slot

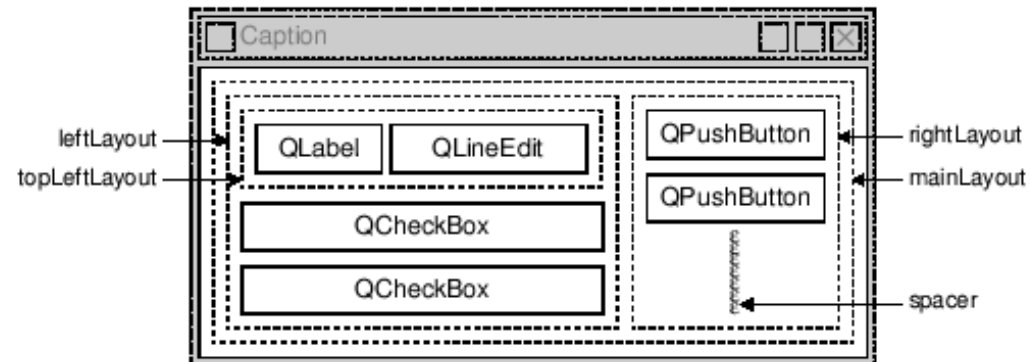
```
QObject::connect( mySlider, SIGNAL(sliding()),  
                 myLCDNumber, SLOT(display(int)) );  
myApp.setMainWidget( myWindow );  
myWindow->show();  
return myApp.exec();  
}
```



Layout Manager

Qt fournit trois manières de gérer le positionnement de widgets:

- ▷ définir les valeurs (x, y, w, h) de positionnement
 - ◇ `setGeometry(x,y,w,h)`
- ▷ en paramétrant la taille (w, h) de positionnement
 - ◇ `setMinimumSize()`, `width()`, `minimumWidth()`
- ▷ en utilisant des gestionnaires de positionnement (Layout managers)
 - ◇ `QLayout`, `QHBoxLayout`, `QVBoxLayout`, `QGridLayout`...



Layout Manager

Utilité des gestionnaires de disposition :

- ▷ positionnement automatique des widgets insérés
- ▷ modification de la disposition:
 - ◇ retaillage de fenêtre
 - ◇ changement de police de caractère
 - ◇ visibilité, destruction des widgets insérés

Gestionnaire de positionnement :

- ▷ QBox, QGrid
- ▷ QHBoxLayout, QVBoxLayout
- ▷ QBoxLayout, QGridLayout
- ▷ QHBoxLayout, QVBoxLayout
- ▷ QGroupBox, QButtonGroup
- ▷ ...

Examples



```
#include <qapplication.h>
#include <qhbox.h>
#include <qpushbutton.h>
```

```
QHBoxLayout* makeBox(void)
{
    QHBoxLayout* box = new QHBoxLayout( );

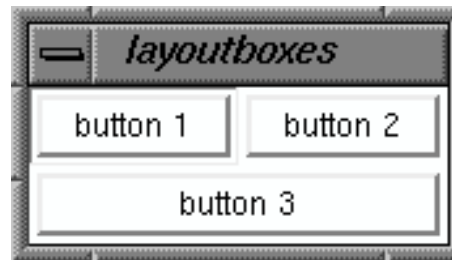
    box->setSpacing(10);
    QPushButton* button = new QPushButton("button1", box);
    box->setStretchFactor ( button, 1);
    button = new QPushButton("button2", box);
    box->setStretchFactor ( button, 2);
    return box;
}
```

Exemples



```
QGrid* makeBox(void)
{
    QGrid* box = new QGrid( 2 ); // une grille a 2 colonnes
    new QLabel( "LastName", box );
    new QLineEdit( box, "lastname" );
    new QLabel( "FirstName", box );
    new QLineEdit( box, "firstname" );
    new QPushButton( "OK !", box ); // seul sur la 3ieme ligne
    return box;
}
```

Exemples



```
#include 'mydialog.h'  
int main( int argc, char **argv ) {  
    QApplication myApp ( argc, argv );  
    MyDialog* dlg = new MyDialog();  
    dlg->show();  
    myApp.setMainWidget( dlg );  
    return myApp.exec();  
}
```

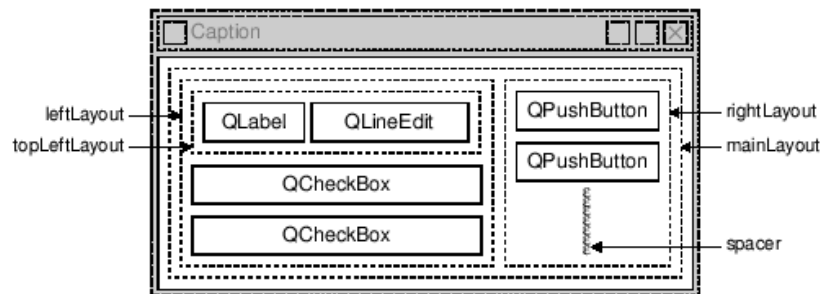
Exemples

Exemple:

```
class MyDialog : public QDialog {
public:
    MyDialog(void);
};
MyDialog::MyDialog( void ) {
    QVBoxLayout* vlayout = new QVBoxLayout ( this );
    QHBoxLayout* hlayout = new QHBoxLayout ();
    vlayout->addLayout(hlayout);
    QPushButton* b = new QPushButton("button 1", this );
    hlayout->addWidget (b);
    b = new QPushButton("button 2", this );
    hlayout->addWidget (b);
    hlayout = new QHBoxLayout ();
    vlayout->addLayout(hlayout);
    b = new QPushButton("button 3", this );
    hlayout->addWidget (b);
}
```

Exemples

Boîte de dialogue de recherche dans un texte:



```
#include <qapplication.h>
#include "findDialog.h"
int main( int argc, char **argv )
{
    QApplication a( argc, argv );
    FindDialog *fileDialog = new FindDialog();
    a.setMainWidget(fileDialog);
    fileDialog->show();
    return a.exec();
}
```

Examples

```
#include <qdialog.h>
class QLabel;
class QLineEdit;
// others useful Qt classes
class FindDialog : public QDialog {
    Q_OBJECT
public:
    FindDialog(QWidget *parent = 0, const char *name = 0);
signals:
    void findNext(const QString &str, bool caseSensitive);
    void findPrev(const QString &str, bool caseSensitive);
private slots:
    void findClicked();
    void enableFindButton(const QString &text);
private:
    QLabel *_label;
    QLineEdit *_lineEdit;
    QCheckBox *_caseCheckBox, *_backwardCheckBox; ;
    QPushButton *_findButton, *_closeButton;
};
```

Exemples

Implémentation (`findDialog.cpp`) :

```
FindDialog::FindDialog(QWidget *parent, const char *name): QDialog(parent, name)
{
    setCaption(tr("Find"));
    _label = new QLabel(tr("Find &what:"), this);
    _lineEdit = new QLineEdit(this);
    _label->setBuddy(lineEdit);
    _caseCheckBox = new QCheckBox(tr("Match &case"), this);
    _backwardCheckBox = new QCheckBox(tr("Search &backward"), this);
    _findButton = new QPushButton(tr("&Find"), this);
    _findButton->setDefault(true);
    _findButton->setEnabled(false);
    _closeButton = new QPushButton(tr("Close"), this);
    connect(lineEdit, SIGNAL(textChanged(const QString &)),
           this,      SLOT(enableFindButton(const QString &)));
    connect(findButton, SIGNAL(clicked()),
           this,      SLOT(findClicked()));
    connect(closeButton, SIGNAL(clicked()),
           this,      SLOT(close()));
}
```

Examples

```
QHBoxLayout *topLeftLayout = new QHBoxLayout;
topLeftLayout->addWidget(_label);
topLeftLayout->addWidget(_lineEdit);
QVBoxLayout *leftLayout = new QVBoxLayout;
leftLayout->addLayout(topLeftLayout);
leftLayout->addWidget(_caseCheckBox);
leftLayout->addWidget(_backwardCheckBox);
QVBoxLayout *rightLayout = new QVBoxLayout;
rightLayout->addWidget(_findButton);
rightLayout->addWidget(_closeButton);
rightLayout->addStretch(1);
QHBoxLayout *mainLayout = new QHBoxLayout(this);
mainLayout->setMargin(11);
mainLayout->setSpacing(6);
mainLayout->addLayout(leftLayout);
mainLayout->addLayout(rightLayout);
}
```

Examples

```
void FindDialog::findClicked()
{
    QString text = _lineEdit->text();
    bool caseSensitive = _caseCheckBox->isOn();
    if (_backwardCheckBox->isOn())
        emit findPrev(text, caseSensitive);
    else
        emit findNext(text, caseSensitive);
}

void FindDialog::enableFindButton(const QString &text)
{
    _findButton->setEnabled(!text.isEmpty());
}
```

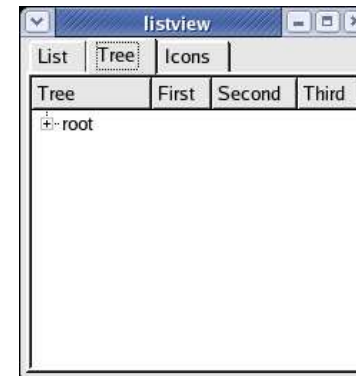
Gestion de listes

Un grand nombre d'applications IHM nécessitent de visualiser:

- ▷ des listes
- ▷ des tableaux (plusieurs colonnes)
- ▷ des arborescences

Classes Qt disponibles

- ▷ QListBox, QListView



QListBox

Forme la plus simple de gestion de liste

- ▷ Colonne d'éléments de texte

Méthodes d'insertion, de destruction d'items:

- ▷ `insertItem()`
- ▷ `clear()`

```
class ListBoxExample : public QVBox
{
    Q_OBJECT
public:
    ListBoxExample( QWidget *parent=0, char *name=0 );
public slots:
    void addItem();
    void newSelection();
private:
    QListBox *_listBox;
    QLineEdit *_lineEdit;
    QLabel *_label;
};
```

QListBox

```
ListBoxExample::ListBoxExample( QWidget *parent, char *name ) : QVBox( parent, name )
{
    _listBox = new QListBox( this );
    QHBoxLayout *hb = new QHBoxLayout( this );
    QLineEdit *lineEdit = new QLineEdit( hb );
    QPushButton *pbAdd = new QPushButton( "Add", hb );
    QPushButton *pbClear = new QPushButton( "Clear", hb );
    hb = new QHBoxLayout( this );
    new QLabel( "Selection: ", hb );
    _label = new QLabel( "nothing", hb );
    connect( pbAdd, SIGNAL(clicked()), this, SLOT(addItem()) );
    connect( pbClear, SIGNAL(clicked()), _listBox, SLOT(clear()) );
    connect( _listBox, SIGNAL(selectionChanged()), this, SLOT(newSelection()) );
}
```

Trois comportements (slots)

- ▷ rajouter un item: `addItem()`
- ▷ vider la liste: `clear()`
- ▷ sélection d'un item: `newSelection()`

QListBox

Rajout d'un item avec le contenu de l'entrée de texte:

```
void ListBoxExample::addItem()
{
    _listBox->insertItem(_lineEdit->text() );
    _lineEdit->setText( "" );
}
```

Sélection d'un item et affichage de l'item sélectionné dans un **QLabel**

```
void ListBoxExample::newSelection()
{
    if( !_listBox->selectedItem() )
        _label->setText( "nothing" );
    else
        _label->setText( _listBox->selectedItem()->text() );
}
```

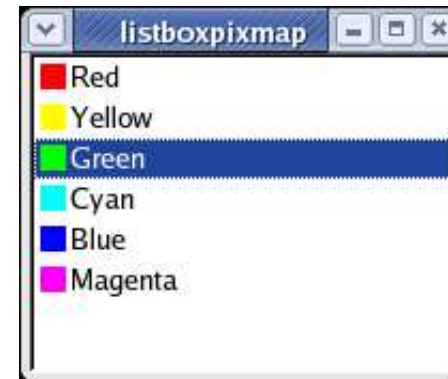
QListBox

Il existe différentes méthodes, classes d'insertion dans une liste :

▷ `QListBox::insertItem (const QPixmap & pixmap, const QString & text, int index = -1)`

▷ `QListBoxPixmap`

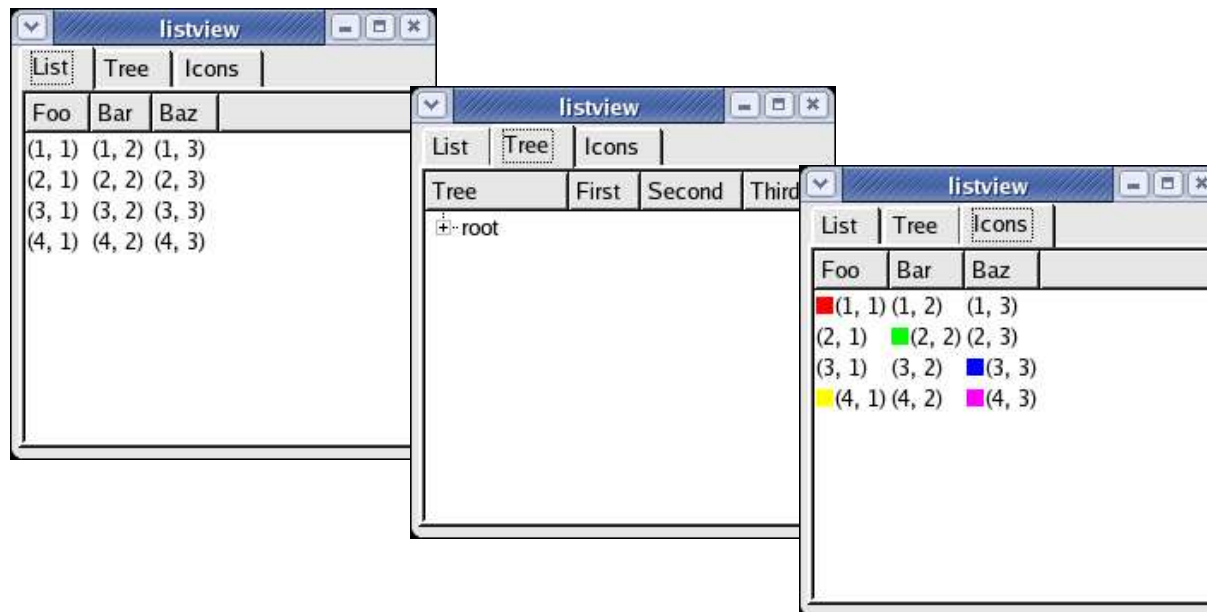
```
int main( int argc, char **argv ) {  
    ...  
    QListBox *lb = new QListBox();  
    QPixmap pm( 12, 12 );  
    pm.fill( Qt::red );  
    new QListBoxPixmap( lb, pm, "Red" );  
    pm.fill( Qt::yellow );  
    new QListBoxPixmap( lb, pm, "Yellow" );  
    ...  
}
```



QListView

Pour gérer des listes:

- ▷ sur plusieurs colonnes
- ▷ sous formes d'arbres
- ▷ avec textes, pixmaps



QListView

Gestion des listes `QListView` dans des onglets `QTabWidget`

```
ListViewExample::ListViewExample(QWidget *parent, char *name) : QTabWidget(parent,name)
{
    addTab( setupListTab(), "List" );
    addTab( setupTreeTab(), "Tree" );
    addTab( setupIconTab(), "Icons" );
}
```

Premier onglet: Insertion de colonne dans une liste

```
QWidget* ListViewExample::setupListTab()
{
    _listView = new QListView();
    _listView->addColumn( "Foo" );
    _listView->addColumn( "Bar" );
    _listView->addColumn( "Baz" );
    _listView->setAllColumnsShowFocus( true );
// setupListTab(): To be Cntd
```

La sélection se fera sur la ligne entière et non sur la première colonne

QListView

Insertion de quatres lignes sur les trois colonnes.

```
// setupListTab() : end
new QListViewItem( _listView, "(1, 1)", "(1, 2)", "(1, 3)" );
new QListViewItem( _listView, "(2, 1)", "(2, 2)", "(2, 3)" );
new QListViewItem( _listView, "(3, 1)", "(3, 2)", "(3, 3)" );
new QListViewItem( _listView, "(4, 1)", "(4, 2)", "(4, 3)" );

return _listView;
}
```

Deuxième onglet: Insertion dans une arborescence:

```
QWidget *ListViewExample::setupTreeTab() {
    _treeView = new QListView();
    _treeView->addColumn( "Tree" );
    _treeView->addColumn( "First" );
    _treeView->addColumn( "Second" );
    _treeView->addColumn( "Third" );
    _treeView->setRootIsDecorated( true );
// setupTreeTab() : To be Cntd
```

QListView

```
// setupTreeTab() : end
QListViewItem *root = new QListViewItem( m_treeView, "root" );
QListViewItem *a = new QListViewItem( root, "A" );
QListViewItem *b = new QListViewItem( root, "B" );
QListViewItem *c = new QListViewItem( root, "C" );

new QListViewItem( a, "foo", "1", "2", "3" );
new QListViewItem( a, "bar", "i", "ii", "iii" );
new QListViewItem( a, "baz", "a", "b", "c" );
new QListViewItem( b, "foo", "1", "2", "3" );
new QListViewItem( b, "bar", "i", "ii", "iii" );
new QListViewItem( b, "baz", "a", "b", "c" );
new QListViewItem( c, "foo", "1", "2", "3" );
new QListViewItem( c, "bar", "i", "ii", "iii" );
new QListViewItem( c, "baz", "a", "b", "c" );
return m_treeView;
}
```

Différence avec la gestion de listes multicolonnees:

▷ les items de listes ont pour parent un item de liste.

QListView

Troisième onglet: Insertion de texte, pixmap dans une liste multicolonne

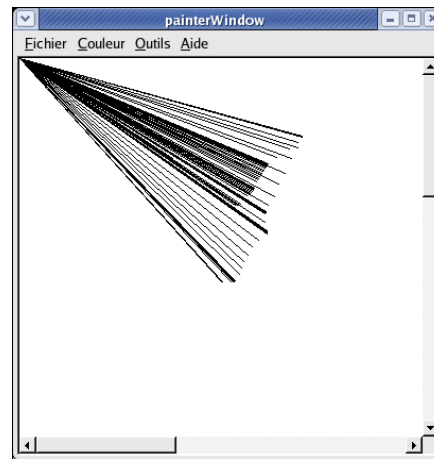
```
QWidget *ListViewExample::setupIconTab() {
    _iconView = new QListView();
    _iconView->addColumn( "Foo" );
    ...
    _iconView->setAllColumnsShowFocus( true );

    QPixmap pm( 10, 10 );
    QListViewItem *lvi = new QListViewItem( _iconView, "(1, 1)", "(1, 2)", "(1, 3)" );
    pm.fill( Qt::red );
    lvi->setPixmap( 0, pm );
    lvi = new QListViewItem( _iconView, "(2, 1)", "(2, 2)", "(2, 3)" );
    pm.fill( Qt::green );
    lvi->setPixmap( 1, pm );
    lvi = new QListViewItem( _iconView, "(3, 1)", "(3, 2)", "(3, 3)" );
    pm.fill( Qt::blue );
    lvi->setPixmap( 2, pm );
    ...
}
```

Editeur Graphique

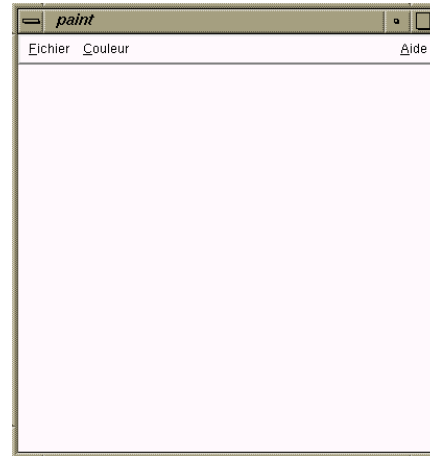
Création de fenêtre principale d'un éditeur graphique:

- ▷ barre d'action (de menu) `PainterWindow`
- ▷ zone client `PainterWindowArea`



- ▷ `QMenuBar`, `QPopupMenu`
- ▷ `QFileDialog`, `QPixmap`, `QPicture`
- ▷ `QFrame`, `QScrollView`
- ▷ ...

Les Menus



Classe de base

- ▷ **QmenuData**: gestion de liste d'items, d'accélérateurs...
- ▷ méthodes utilisées par les classes dérivées
 - ◇ `insertItem()`, `insertSeparator()`

Classes héritières

- ▷ **QmenuBar**: conteneur de menus
- ▷ **QPopupMenu**: menus, sous-menus et menus surgissants

painterWindow.h

```
#include <qapplication.h>
#include <qpainter.h>
#include <qpixmap.h>
#include <qwidget.h>

#include <qmenubar.h>
#include <qmessagebox.h>
#include <qpopupmenu.h>

enum menuIDs {
    COLOR_MENU_ID_BLACK,
    COLOR_MENU_ID_RED,
    COLOR_MENU_ID_BLUE,
    COLOR_MENU_ID_GREEN
};
```

painterWindow.h

```
class PainterWindow : public QWidget {
    Q_OBJECT
public:
    PainterWindow();
    virtual ~PainterWindow();
protected:
    virtual void mousePressEvent (QMouseEvent* event);
    ...
private slots:
    void slotAbout();
    void slotAboutQt();
    void slotColorMenu ( int );
private:
    ...
    QColor _currentcolor;
    QMenuBar* _menubar;
    QPopupMenu *_filemenu, *_colormenu, *_helpmenu;
};
```

painterWindow.cpp

```
PainterWindow::PainterWindow(void) {
...
    _filemenu = new QPopupMenu;
    _filemenu->insertItem("&Quitter", qApp, SLOT(quit()) );
    _colormenu = new QPopupMenu;
    _colormenu->insertItem("&Noir", COLOR_MENU_ID_BLACK );
    _colormenu->insertItem("&Rouge", COLOR_MENU_ID_RED );
    _colormenu->insertItem("&Bleu", COLOR_MENU_ID_BLUE );
    _colormenu->insertItem("&Vert", COLOR_MENU_ID_GREEN );
    QObject::connect( _colormenu, SIGNAL(activated(int)),
                      this,      SLOT(slotColorMenu(int)) );
    _helpmenu = new QPopupMenu;
    _helpmenu->insertItem("&A propos de Qt Painter", this, SLOT(slotAbout()) );
    _helpmenu->insertItem("&A propos de Qt", this,  SLOT(slotAboutQt()) );
    _menubar = new QMenuBar ( this );
    _menubar->insertItem( "&Fichier", _filemenu );
    _menubar->insertItem( "&Couleur", _colormenu );
    _menubar->insertSeparator();
    _menubar->insertItem( "&Aide", _helpmenu );
}
```

painterWindow.cpp

```
void PainterWindow::mouseMoveEvent (QMouseEvent* event) {
    ...
    paintWindow.begin(this);
    paintBuffer.begin(&_buffer);

    paintWindow.setPen(_currentcolor);
    paintBuffer.setPen(_currentcolor);

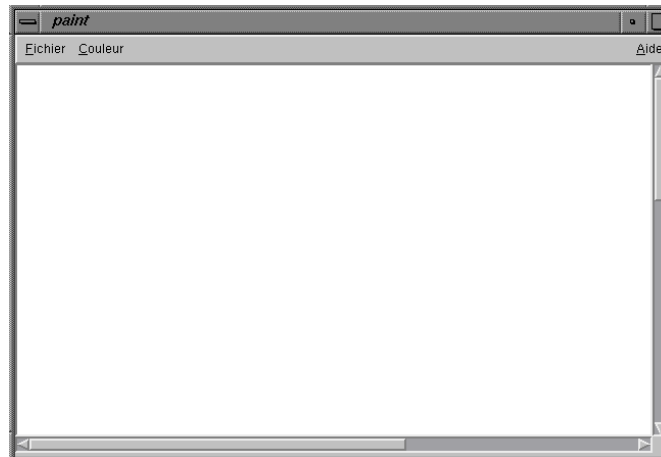
    paintWindow.drawLine(_last, event->pos());
    paintBuffer.drawLine(_last, event->pos());
    ...
}

void PainterWindow::slotAbout (void) {
    QMessageBox::information( this,
                              "Qt Painter Version 0.1",
                              "Copyright 2002 - IHM 1");
}
```

painterWindow.cpp

```
void PainterWindow::slotAboutQt (void) {
    QMessageBox::aboutQt( this, "A propos de Qt");
}
void PainterWindow::slotColorMenu (int item ) {
    switch (item) {
        case COLOR_MENU_ID_BLACK:
            _currentcolor = black;
            break;
        case COLOR_MENU_ID_RED:
            _currentcolor = darkRed;
            break;
        case COLOR_MENU_ID_BLUE:
            _currentcolor = darkBlue;
            break;
        case COLOR_MENU_ID_GREEN:
            _currentcolor = darkGreen;
            break;
    }
}
```

Zone Client



Classe de base

- ▷ **QFrame**: classe de base pour scroll, popup, menubar...
- ▷ méthodes surdéfinies par les classes dérivées
 - ◇ `drawContents()`

Zone Client

Méthodes de `QScrollView` utilisées par notre exemple

- ▷ `addChild()`

Classes héritières utilisant une fenêtre défilante

- ▷ `QListView`, `QGridView`, `QIconView`,
- ▷ `QListBox`, `QTextEdit`, `QTable...`

Deux classes pour notre exemple

- ▷ `PainterWindow` : gestion de la barre d'action, fenêtre défilante
- ▷ `PainterWindowArea` : zone de tracé,

painterWindowArea.h

```
...
class PainterWindowArea : public QWidget
{
    Q_OBJECT
public:
    PainterWindowArea();
    ...
public slots:
    void setColor( QColor );
protected:
    virtual void mousePressEvent (QMouseEvent* event);
    ...
private:
    QPoint _last;
    QPixmap _buffer;
    QColor _currentcolor;
} ;
```

painterWindowArea.cpp

```
void PainterWindowArea::setColor(QColor color) {
    _currentcolor = color;
}
void
PainterWindowArea::mouseMoveEvent (QMouseEvent* event)
{
    QPainter paintWindow;
    QPainter paintBuffer;
    // ----- Tests -----//
    QPoint p(0,0);
    paintWindow.begin(this);
    paintBuffer.begin(&_buffer);

    paintWindow.drawLine( p, event->pos() );
    paintBuffer.drawLine( p, event->pos() );
    // -----//
    paintWindow.end();
    paintBuffer.end();
}
```

painterWindow.h

```
class PainterWindow : public QWidget{
Q_OBJECT
public:
    PainterWindow();
    ...
private slots:
    void slotAbout();
    void slotAboutQt();
    void slotColorMenu ( int );
signals:
    void colorChanged( QColor );
protected:
    virtual void resizeEvent( QResizeEvent* );
private:
    QMenuBar*    _menubar;
    QPopupMenu *_filemenu, *_colormenu, *_helpmenu;
    QScrollView* _scrollview;
    PainterWindowArea* _area;
};
```

painterWindow.cpp

```
PainterWindow::PainterWindow(void) {
...

    _scrollview = new QScrollView( this );
    _scrollview->setGeometry( 0, _menubar->height(),
                             width(), height() - _menubar->height() );
    _area = new PainterWindowArea();
    _area->setGeometry( 0, 0, 1000, 1000 );
    _scrollview->addChild( _area );
    QObject::connect( _colormenu, SIGNAL(activated(int)), this, SLOT(slotColorMenu(int)) );
    QObject::connect( this, SIGNAL(colorChanged(QColor)), _area, SLOT(setColor(QColor)) );
}
void
PainterWindow::resizeEvent (QResizeEvent* event) {
    _scrollview->setGeometry( 0, _menubar->height(),
                             width(), height() - _menubar->height());
}
}
```

painterWindow.cpp

```
void PainterWindow::slotColorMenu (int item ) {
    switch (item)
    {
        case COLOR_MENU_ID_BLACK:
            emit colorChanged(black);
            break;
        case COLOR_MENU_ID_RED:
            emit colorChanged(darkRed);
            break;
        case COLOR_MENU_ID_BLUE:
            emit colorChanged(darkBlue);
            break;
        case COLOR_MENU_ID_GREEN:
            emit colorChanged( darkGreen);
            break;
    }
}
```

Menu Surgissant

Un menu surgissant

- ▷ `QPopupMenu`

n'est pas associé à une barre de menu

- ▷ `QMenuBar::insertItem()`

Mais à la zone de travail :

- ▷ `PainterWindowArea`, dans notre cas

On affiche un menu surgissant par l'appel de

- ▷ `exec()`: appel bloquant jusqu'à sélection d'item, fermeture du popup

- ▷ `popup()`: appel non-bloquant

Positionnement du menu sur le pointeur de souris:

- ▷ `exec(QCursor::pos())`

- ▷ `popup(QCursor::pos())`

painterWindowArea.h

```
class PainterWindowArea : public QWidget
{
    Q_OBJECT
public:
    PainterWindowArea();
    ...
public slots:
    void setColor( QColor );

private slots:
    void slotClearArea ();

protected:
    virtual void mousePressEvent (QMouseEvent* event);
    ...
private:
    QPoint _last;
    ...
    QPopupMenu* _popupmenu;
} ;
```

painterWindowArea.cpp

```
PainterWindowArea::PainterWindowArea(void) {
    ...
    _popupmenu = new QPopupMenu();
    _popupmenu->insertItem( "&Effacer", this, SLOT (slotClearArea()) );
}
PainterWindowArea::~PainterWindowArea(void) {
    delete _popupmenu;
}
void
PainterWindowArea::slotClearArea (void) {
    _buffer.fill( white );
    bitBlt( this, 0, 0, &_buffer);
}
void
PainterWindowArea::mousePressEvent (QMouseEvent* event) {
    if ( event->button() == RightButton )
        _popupmenu->exec ( QCursor::pos() );
    else _last = event->pos();
}
```

Sauvegardes

Classe de base pour lire, enregistrer un fichier

- ▷ `QFileDialog`

Les méthodes les plus importantes

- ▷ `getOpenFileName()`

- ▷ `getSaveFileName()`

Format d'enregistrement

- ▷ `QPixmap::save()`: matrice de pixel (portabilité)

- ▷ `QPicture::save()`: mode vectoriel (économique)

Das cet exemple:

- ▷ slots de chargement sur la fenêtre principale (`PainterWindow`)

- ▷ transfert de signal sur la zone de tracé (`PainterWindowArea`)

painterWindowArea.h

```
#include <qfiledialog.h>
...
class PainterWindowArea : public QWidget {
    Q_OBJECT
public:
    PainterWindowArea();
    ...
public slots:
    void setColor( QColor );
    void slotLoad( const char* );
    void slotSave( const char* );
private slots:
    void slotClearArea ();
protected:
    virtual void mousePressEvent (QMouseEvent* event);
    ...
private:
    QPoint _last;
    ...
} ;
```

painterWindowArea.cpp

```
void
PainterWindowArea::slotLoad( const char* filename )
{
    if ( !_buffer.load( filename) )
        QMessageBox::warning( 0,
                               "Erreur de chargement",
                               "Le fichier ne peut etre charge");

    repaint();
}

void
PainterWindowArea::slotSave(const char* filename )
{
    if ( !_buffer.save( filename, "BMP") )
        QMessageBox::warning( 0,
                               "Erreur d'enregistrement",
                               "Sauvegarde de fichier impossible!");
}
```

painterWindow.h

```
class PainterWindow : public QWidget {
Q_OBJECT
public:
    PainterWindow();
    virtual ~PainterWindow();
private slots:
    ...
    void slotLoad();
    void slotSave();
signals:
    ...
    void load( const char* );
    void save( const char* );
protected:
    virtual void resizeEvent( QResizeEvent* );
private:
    QMenuBar*    _menubar;
    ...
};
```

painterWindow.cpp

```
PainterWindow::PainterWindow(void) {
...
    _filemenu->insertItem("&Charger", this, SLOT(slotLoad()) );
    _filemenu->insertItem("&Enregistrer", this, SLOT(slotSave()) );
    _filemenu->insertSeparator();
...
    QObject::connect( this, SIGNAL(save(const char*)),
                      _area, SLOT(slotSave(const char*)) );
    QObject::connect( this, SIGNAL(load(const char*)),
                      _area, SLOT(slotLoad(const char*)) );
...
}
void PainterWindow::slotLoad(void) {
    QString filename = QFileDialog::getOpenFileName(".", "*.bmp", this);
    if ( !filename.isEmpty() ) emit load(filename);
}
void PainterWindow::slotSave(void) {
    QString filename = QFileDialog::getSaveFileName(".", "*.bmp", this);
    if ( !filename.isEmpty() ) emit save(filename);
}
```

Gestion des Actions

QAction permet d'associer les actions

▷ dans les barre de menu, barre d'outils, bare d'états...

Un object de type **QAction** peut contenir

▷ icone **setIconSet()**

▷ texte de menu **setMenuText()**

▷ texte de statut **setStatusTip()**

▷ texte d'information **setWhatsThis()**

▷ accélérateur **setAccel()**

▷ bulle d'aide **setTooltip()**

Ajout/retrait dans une barre de menus, d'outils

▷ **QAction::addTo()**, **QAction::removeFrom()**

Une fois créé l'action, la relier à un slot

▷ `connect(_newAct, SIGNAL(activated()), this, SLOT(newFile()));`

Gestion des Actions

```
class PainterWindow : public QMainWindow {
Q_OBJECT
public:
    PainterWindow(QWidget *parent = 0, const char *name = 0);
    ...
private slots:
    void newFile();
    ...
private:
    QPopupMenu *_fileMenu, *_colorMenu, *_toolMenu, *_helpMenu;
    QAction *_newAct;
    QToolBar *_fileToolBar;
    void _createActions();
    void _createMenus();
    void _createToolBars();
    ...
};
```

Gestion des Actions

```
void PainterWindow::_createActions(void)
{
    _newAct = new QAction(tr("&New"), tr("Ctrl+N"), this);
    _newAct->setIconSet(QPixmap::fromMimeSource("new.png"));
    _newAct->setStatusTip(tr("Create a new file"));
    connect(_newAct, SIGNAL(activated()), this, SLOT(newFile()));
    ....
}
void PainterWindow::_createMenus()
{
    _fileMenu = new QPopupMenu(this);
    newAct->addTo(_fileMenu);
    ...
}
void MainWindow::_createToolBars()
{
    _fileToolBar = new QToolBar(tr("File"), this);
    newAct->addTo(_fileToolBar);
    ...
}
```

Gestion des Actions

Chargement d'images avec Qt

1. stockage dans des fichiers chargés à l'exécution
2. intégrés dans le code source (XPM)
3. mécanisme de collection d'images Qt

Insertion de collection d'objets dans le projet (`nom_de_projet.pro`)

```
TEMPLATE = app
INCLUDEPATH += .
```

```
# Input
HEADERS += painterWindow.h painterWindowArea.h
SOURCES += main.cpp painterWindow.cpp painterWindowArea.cpp
IMAGES      = new.png
```

Avantages :

- ▷ temps de chargement accéléré
- ▷ pas de problème d'accès aux fichiers

Affichage 2D

Possibilités graphiques de Qt:

- ▷ **QPainter**: dessiner sur un dispositif (device) d'affichage:
 - ◊ widget, pixmap, imprimante...
- ▷ **QCanvas**: dessiner avec une approche "élément" (items) de dessin
- ▷ **QGL**: dessiner en 3D en utilisant la librairie OpenGL

Utilisation typique de la classe **QPainter**

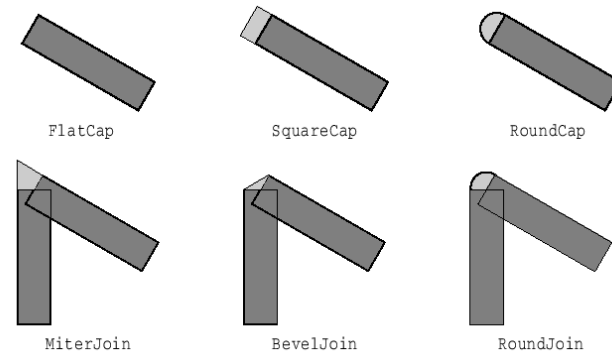
- ▷ Instancier un Painter
- ▷ lui associer un stylo (pen) un pinceau (brush) ...
- ▷ dessiner
- ▷ détruire le Painter

QPainter

Le stylo est utilisé pour :

- ▷ tracer des lignes et des contours
- ▷ avec une couleur, largeur de trait, styles de lignes
- ▷ des extrémités (cap) et jointure (joint) de lignes

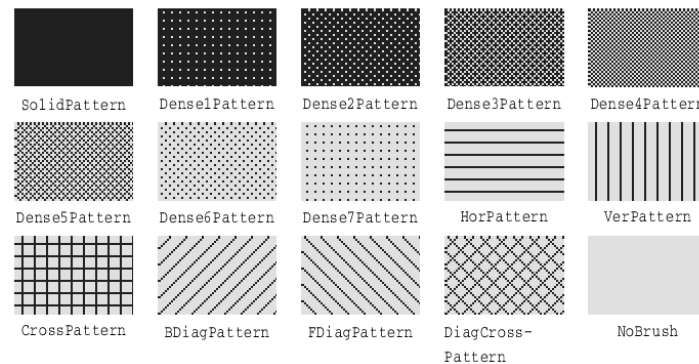
	line width			
	1	2	3	4
NoPen				
SolidLine				
DashLine				
DotLine				
DashDotLine				
DashDotDotLine				



QPainter

Le pinceau est utilisé pour le remplissage:

- ▷ une couleur, un style de remplissage



Une police de caractères (font) :

- ▷ pour tracer du texte

Méthodes (`QPainter`) pour modifier ces caractéristiques de tracé

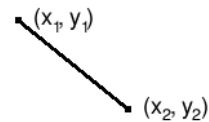
- ▷ `setPen()`, `setBrush()`, `setFont()`

En transemettant

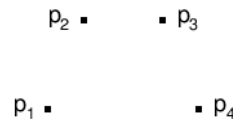
- ▷ `QPen`, `QBrush`, `QFont`

QPainter

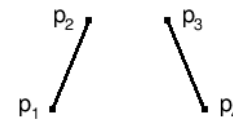
Méthodes de tracé de la classe QPainter :



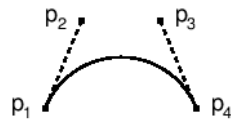
drawLine()



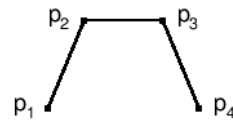
drawPoints()



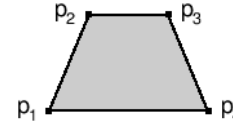
drawLineSegments()



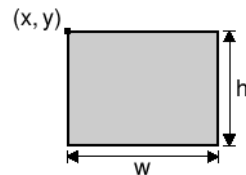
drawCubicBezier()



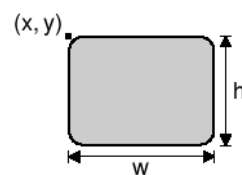
drawPolyline()



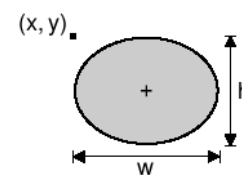
drawPolygon()



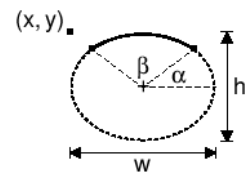
drawRect()



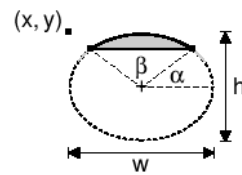
drawRoundRect()



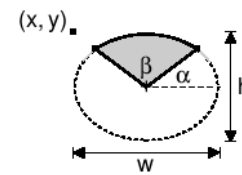
drawEllipse()



drawArc()



drawChord()



drawPie()

QPainter

En pratique tout, ou presque, se fait dans la méthode de classe:

▷ `paintEvent()`

Exemple :

```
void MyWidget::paintEvent(){
    QPainter paint( this );
    QBrush brush( Qt::yellow, Qt::SolidPattern );
    paint.setBrush( brush );
    paint.drawRect(5, 50, 200, 100);
    ...
}
```

La classe **QPainter** permet de :

- ▷ gérer les informations de tracé
- ▷ dessiner des objets graphiques
- ▷ transformer les objets graphiques

QPainter

Informations de tracé :

- ▷ polices de caractères : `font()`, `setFont()`, `fontInfo()`
- ▷ crayons, pinceaux : `pen()`, `setPen()`, `brush()`, `brushOrigin()`
- ▷ fond de dessin : `backgroundMode()`, `backgroundColor()`
- ▷ modes de tracé : `rasterOp()`: `CopyROP`, `XorROP`, ...
- ▷ système de coordonnées : `viewport()`, `window()`, `worldMatrix()`
- ▷ visibilité : `clip()`, `clipRegion()`
- ▷ position courante : `pos()`, `moveTo()`, `lineTo()`
- ▷ sauvegarde de l'état : `save()`, `restore()`

QPainter

Fonctionnalités de tracé :

- ▷ `drawPoint()`, `drawPoints()`
- ▷ `drawLine()`, `drawLineSegments()`
- ▷ `drawRect()`, `drawRoundRect()`
- ▷ `drawPolyline()`, `drawPolygon()`, `drawConvexPolygon()`
- ▷ `drawEllipse()`, `drawArc()`...
- ▷ `drawPie()`, `drawChord()`, `drawCubicBezier()`

Pour afficher de texte, images, pixmaps

- ▷ `drawText()`, `drawPicture()`
- ▷ `drawPixmap()`, `drawImage()`, `drawTiledPixmap()`

QPainter

Mécanismes d'affichage :

- ▷ **window** : coordonnées logique
- ▷ **viewport** : coordonnées physique

Pour dessiner indépendemment de:

- ▷ la taille, la résolution du dispositif d'affichage

Par défaut :

- ▷ **window**, **viewport** même dispositif rectangulaire

Exemple de widget de taille (320x200) :

- ▷ Par défaut origine (0,0), extrémité (320,200)

Si on veut dessiner avec

- ▷ l'origine au centre (0,0)
- ▷ les valeurs min (-50,-50) et max (50,50)

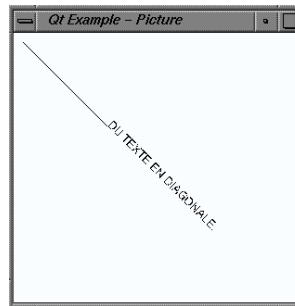
On modife le systèmes de coordonnées logiques :

- ▷ `painter.setWindow(QRect(-50,-50,100,100))`

QPainter

Transformations 2D dans une fenêtre :

▷ **QWMatrix** : translations, rotations, mise à l'échelle



```
QWMatrix old = paint.worldMatrix();
QWMatrix current = paint.worldMatrix();
current.translate(100,100);
current.rotate(45);
paint.setWorldMatrix(current);
paint.drawText(0, 0, "DU TEXTE EN DIAGONALE...");
paint.setWorldMatrix(old);
paint.drawLine(10,10,100,100); // une ligne diagonale avant le texte
```

QPainter

Méthodes de transformations de la classe `QPainter`

- ▷ `save()`, `restore()`
- ▷ `translate()`, `rotate()`, `scale()`

```
paint.save();  
paint.translate(100,100);  
paint.rotate (45);  
paint.drawText(0, 0, "DU TEXTE EN DIAGONALE...");  
paint.restore();  
paint.drawLine(10,10,100,100); // une ligne diagonale avant le texte
```

Si transformations fréquentes ne nécessitant pas d'affichage:

- ▷ `QWMatrix`

QPainter

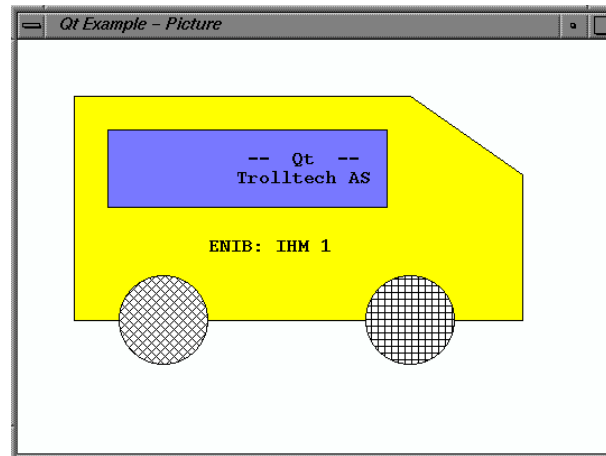
Méthodes de transformations de la classe `QPainter`

▷ `xForm()`, `xFormDev()`, `setWorldXForm()`

Activer/désactiver les transformations de système de coordonnées

```
paint.rotate (45);  
p.drawLine(10,10,100,100); // une ligne verticale !  
QPoint pos (100,100);  
pos = paint.xFormDev ( pos ); // position avant rotation du plan  
paint.drawText(pos, "DU TEXTE EN DIAGONALE...");  
paint.rotate (-45.0);  
paint.drawLine(10,10,100,100); // une ligne diagonale avant le texte  
...
```

QPainter



```
QPainter paint( this );  
QPointArray carray;  
QBrush brush( Qt::yellow, Qt::SolidPattern );  
paint.setBrush( brush );  
  
carray.setPoints( 5, 50,50, 350,50, 450,120, 450,250, 50,250 );  
paint.drawPolygon( carray );
```

QPainter

```
QFont f( "courier", 12, QFont::Bold );
paint.setFont( f );

QColor windowColor( 120, 120, 255 );           // a light blue color
brush.setColor( windowColor );
paint.setBrush( brush );
paint.drawRect( 80, 80, 250, 70 );           // car window
paint.drawText( 180, 80, 150, 70, Qt::AlignCenter, "-- Qt --\nTrolltech AS" );
paint.drawText( 150, 150, 150, 70, Qt::AlignCenter, "ENIB: IHM 1" );

paint.setBackgroundMode( Qt::OpaqueMode );
paint.setBrush( Qt::DiagCrossPattern );
paint.drawEllipse( 90, 210, 80, 80 );         // back wheel
paint.setBrush( Qt::CrossPattern );
paint.drawEllipse( 310, 210, 80, 80 );       // front wheel
...
}
```

Canevas d’Affichage

Affichage “manuel” avec les méthodes

- ▷ `QPainter::paintEvent()`
- ▷ `QScrollView::drawContents()`

Affichage d’ “éléments” manipulables par l’utilisateur :

- ▷ `QCanvas`, `QCanvasItem`
- ▷ `QCanvasLine`, `QCanvasRectangle`, ..., `QCanvasText`

Visualisation des ‘éléments’ qui sont les (modèle) données à afficher (vue) :

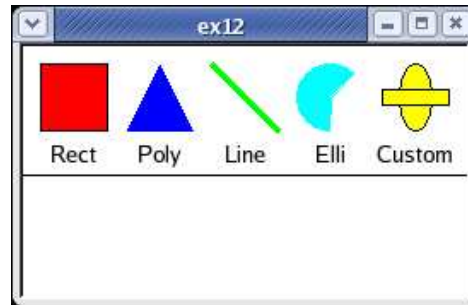
- ▷ `QCanvasView`

On peut donc avoir plusieurs vues sur des même données.

QCanvas

- ▷ réaffichage seulement des données modifiées
- ▷ mécanismes de détection/collision

Canevas d’Affichage



```
QCanvas *c = new QCanvas( 210, 75 );
QCanvasView *cv = new QCanvasView( c );

QCanvasRectangle *rect = new QCanvasRectangle( 10, 10, 40, 40, c );
rect->setPen( Qt::black );
rect->setBrush( Qt::red );
rect->show();
QCanvasText *t = new QCanvasText( "Rect", c );
t->setX( 30 );
t->setY( 55 );
t->setTextFlags( Qt::AlignHCenter );
t->show();
```

Canevas d’Affichage

Pour manipuler les `QCanvasItem`

▷ surdéfinir les méthodes de classes `QCanvasView`

```
class MyCanvasView : public QCanvasView {
public:
    MyCanvasView( QCanvas *c, QWidget *parent=0,
                  const char *name=0, WFlags f=0 ) : QCanvasView( c, parent, name, f )
    {
        _dragging = 0;
    }
protected:
    void contentsMouseEvent( QMouseEvent *event );
    void contentsMousePressEvent( QMouseEvent *event );
    void contentsMouseMoveEvent( QMouseEvent *event );
private:
    QCanvasItem *_dragging;
    int          _xoffset, _yoffset;
};
```

Canevas d’Affichage

```
void MyCanvasView::contentsMouseEvent( QMouseEvent *event )
{
    QCanvasItemList il = canvas()->collisions( event->pos() );
    for( QCanvasItemList::Iterator it=il.begin(); it!=il.end(); ++it )
    {
        if( (*it)->rtti() != QCanvasText::RTTI )
        {
            _dragging = (*it);
            _xoffset = (int)(event->x() - _dragging->x());
            _yoffset = (int)(event->y() - _dragging->y());
            return;
        }
    }
}
```

Méthode d’identification de type à l’exécution :

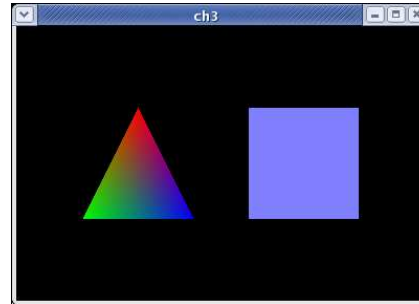
▷ `QCanvasItem::rtti()` : run-time identification

Equivalent de la méthode C++ : `dynamic_cast<T>()`

Canevas d’Affichage

```
void MyCanvasView::contentsMouseReleaseEvent( QMouseEvent *event )
{
    if( _dragging )
    {
        _dragging->setX( event->x() - _xoffset );
        _dragging->setY( event->y() - _yoffset );
        _dragging = 0;
        canvas()->update();
    }
}
void MyCanvasView::contentsMouseMoveEvent( QMouseEvent *event )
{
    if( _dragging )
    {
        _dragging->setX( event->x() - _xoffset );
        _dragging->setY( event->y() - _yoffset );
        canvas()->update();
    }
}
```

Affichage 3D: OpenGL



```
#include <qgl.h>
...
class NeHeWidget : public QGLWidget {           // http://nehe.gamedev.net
...
public:
    NeHeWidget( int timerInterval=0, QWidget *parent=0, char *name=0 );
protected:
    virtual void initializeGL() = 0;
    virtual void resizeGL( int width, int height ) = 0;
    virtual void paintGL() = 0;
    virtual void keyPressEvent( QKeyEvent *e );
...
};
```

Affichage 3D: OpenGL

```
void paintGL() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glTranslatef(-1.5f,0.0f,-6.0f);
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f,0.0f,0.0f);
    glVertex3f( 0.0f, 1.0f, 0.0f);
    glColor3f(0.0f,1.0f,0.0f);
    glVertex3f(-1.0f,-1.0f, 0.0f);
    glColor3f(0.0f,0.0f,1.0f);
    glVertex3f( 1.0f,-1.0f, 0.0f);
    glEnd();
}
```

Affichage 3D: OpenGL

```
glTranslatef(3.0f,0.0f,0.0f);  
glColor3f(0.5f,0.5f,1.0f);  
glBegin(GL_QUADS);  
glVertex3f(-1.0f, 1.0f, 0.0f);  
glVertex3f( 1.0f, 1.0f, 0.0f);  
glVertex3f( 1.0f,-1.0f, 0.0f);  
glVertex3f(-1.0f,-1.0f, 0.0f);  
glEnd();  
}
```



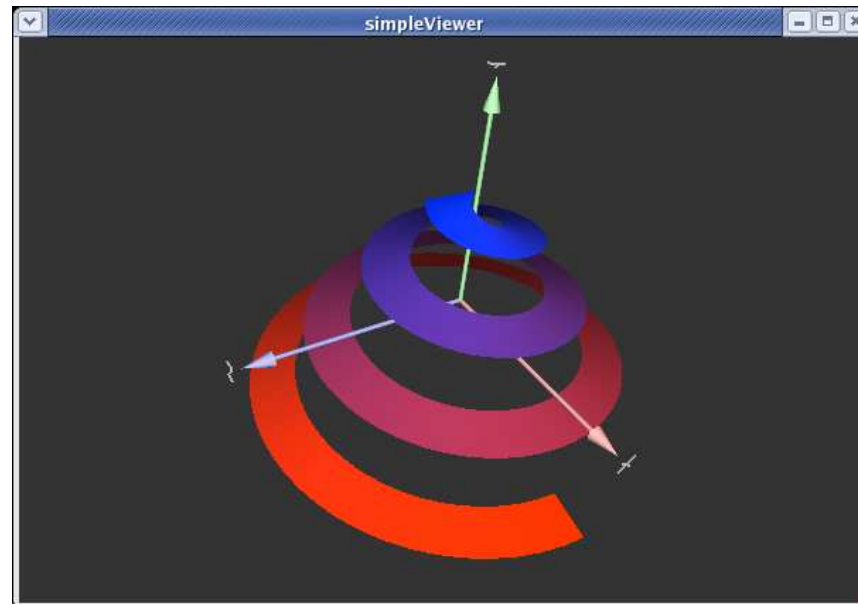
extension 3D

Extension : QGLViewer

▷ <http://artis.imag.fr/Membres/Gilles.Debunne/QGLViewer>

Objectifs

▷ développement rapide d'applications 3D sous OpenGL en Qt



QGLViewer

Fonctionnalités principales

- ▷ `init()`: initialisation de la scène
- ▷ `draw()`: affichage dans le repère du monde
- ▷ `camera()`: paramétrages OpenGL (matrice de projection...)

```
// simpleViewer.h
#include <QGLViewer/qglviewer.h>
class Viewer : public QGLViewer {
protected :
    virtual void draw();
    virtual void init();
};
```

```
// simpleViewer.cpp
#include "simpleViewer.h"
using namespace std;
void Viewer::draw() {
    const float nbSteps = 200.0;
    glBegin(GL_QUAD_STRIP);
```

QGLViewer

```
for (float i=0; i<nbSteps; ++i) {
    float ratio = i/nbSteps;
    float angle = 21.0*ratio;
    float c = cos(angle);
    float s = sin(angle);
    float r1 = 1.0 - 0.8*ratio;
    float r2 = 0.8 - 0.8*ratio;
    float alt = ratio - 0.5;
    const float nor = .5;
    const float up = sqrt(1.0-nor*nor);
    glColor3f(1.0-ratio, 0.2 , ratio);
    glNormal3f(nor*c, up, nor*s);
    glVertex3f(r1*c, alt, r1*s);
    glVertex3f(r2*c, alt+0.05, r2*s);
}
glEnd();
}
void Viewer::init(){
    restoreStateFromFile();
}
```

QGLViewer

```
#include <qapplication.h>

#include "simpleViewer.h"

int main(int argc, char** argv)
{
    QApplication application(argc,argv);
    Viewer v;
    v.show();

    #if QT_VERSION < 0x040000
        application.setMainWidget(&v);
    #endif

    return application.exec();
}
```

QGLViewer

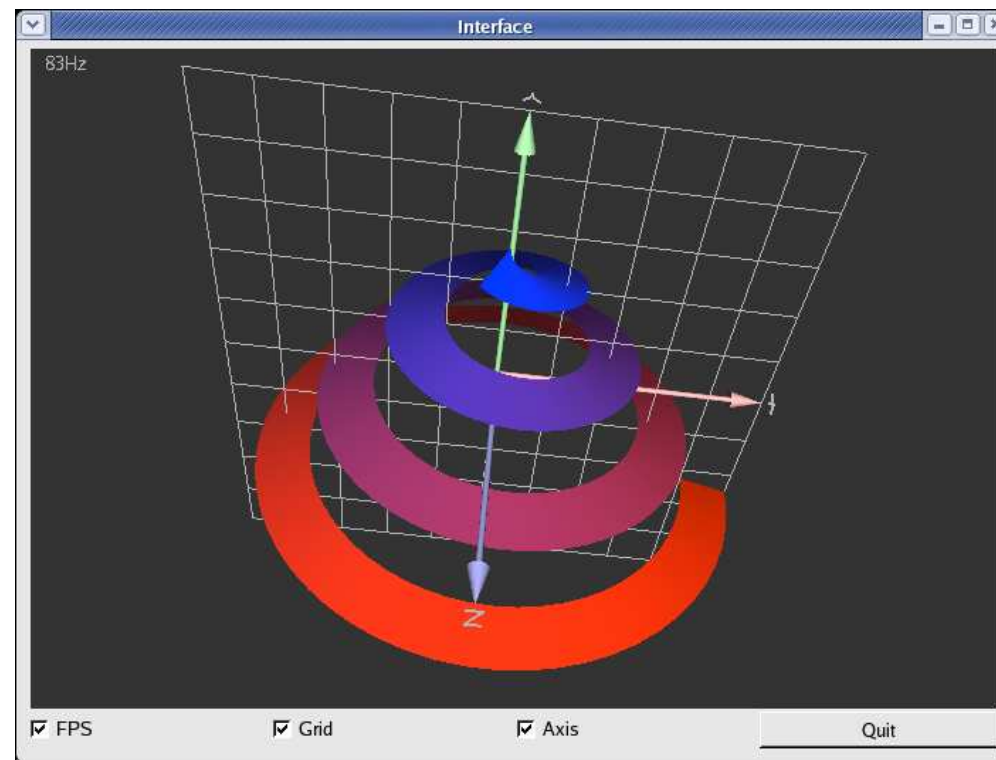
Même application à la mode Qt:

```
class Scene : public QObject {
    Q_OBJECT
public :
    Scene(const QGLViewer* const v);
public slots:
    void drawScene();
};

Scene::Scene(const QGLViewer* const v)
{
    // Connect the viewer signal to our draw function slot
    connect(v, SIGNAL(drawNeeded()), this, SLOT(drawScene()));
}
void Scene::drawScene()
{
    // idem SimpleViewer::draw()
    ...
}
```

QGLViewer

Intégration IHM 2D/3D sous Qt :



QGLViewer

```
class ViewerInterface : public QWidget {
    Q_OBJECT
public:
    ViewerInterface( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
    ~ViewerInterface();

    QCheckBox* FPSCheckBox;
    QCheckBox* AxisCheckBox;
    QPushButton* QuitButton;
    QCheckBox* GridCheckBox;
    Viewer* viewer;
protected:
    QGridLayout* InterfaceLayout;
protected slots:
    virtual void languageChange();
private:
    QPixmap image0;
};
```

QGLViewer

```
ViewerInterface::ViewerInterface( QWidget* parent, const char* name, WFlags fl )
    : QWidget( parent, name, fl ) {

// Qt GUI
    if ( !name ) setName( "Interface" );
    InterfaceLayout = new QGridLayout( this, 1, 1, 6, 2, "InterfaceLayout");
    FPSCheckBox = new QCheckBox( this, "FPSCheckBox" );
    InterfaceLayout->addWidget( FPSCheckBox, 1, 0 );
    AxisCheckBox = new QCheckBox( this, "AxisCheckBox" );
    InterfaceLayout->addWidget( AxisCheckBox, 1, 2 );
    QuitButton = new QPushButton( this, "QuitButton" );
    InterfaceLayout->addWidget( QuitButton, 1, 3 );
    GridCheckBox = new QCheckBox( this, "GridCheckBox" );
    InterfaceLayout->addWidget( GridCheckBox, 1, 1 );
```

QGLViewer

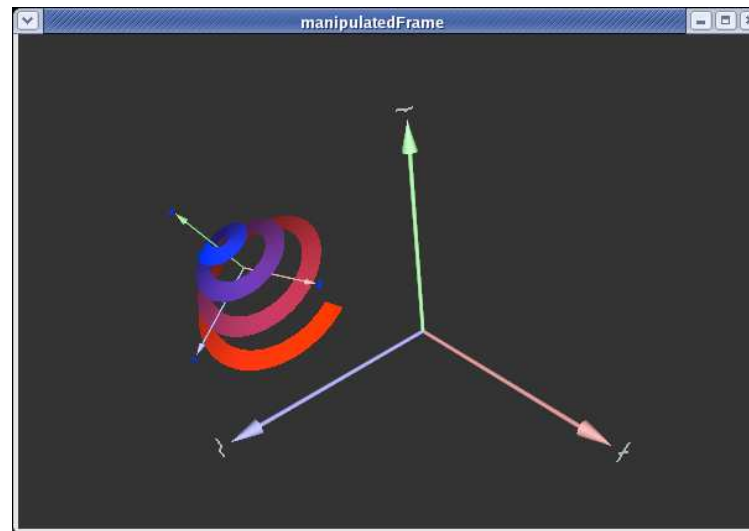
```
// QGLViewer in QGridLayout
viewer = new Viewer( this, "viewer" );
InterfaceLayout->addMultiCellWidget( viewer, 0, 0, 0, 3 );
languageChange();
resize( QSize(673, 438).expandedTo(minimumSizeHint()) );
clearWState( WState_Polished );

// signals and slots connections
connect( QuitButton, SIGNAL(released()), this, SLOT(close()));
connect( FPSCheckBox, SIGNAL(toggled(bool)), viewer, SLOT( setFPSIsDisplayed(bool)));
connect( AxisCheckBox, SIGNAL(toggled(bool)), viewer, SLOT( setAxisIsDrawn(bool)));
connect( GridCheckBox, SIGNAL(toggled(bool)), viewer, SLOT( setGridIsDrawn(bool)));
connect( viewer, SIGNAL(FPSIsDisplayedChanged(bool) ),
        FPSCheckBox, SLOT(setChecked(bool)));
connect( viewer, SIGNAL(axisIsDrawnChanged(bool)),
        AxisCheckBox, SLOT(setChecked(bool)));
connect( viewer, SIGNAL(gridIsDrawnChanged(bool)),
        GridCheckBox, SLOT(setChecked(bool)));
}
```

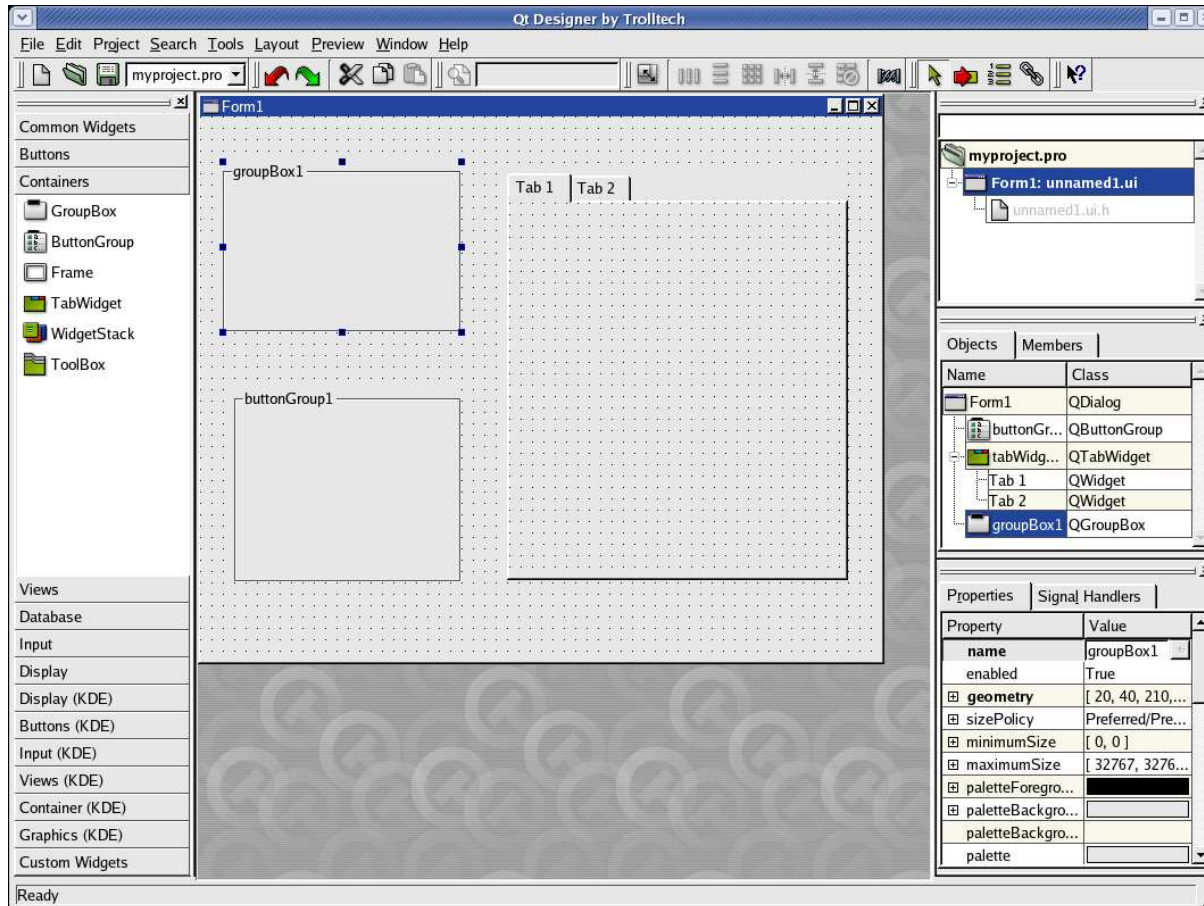
QGLViewer

Interaction avec l'environnement 3D :

```
void Viewer::init()
{
  ...
  setHandlerKeyboardModifiers(QGLViewer::FRAME, Qt::NoButton);
  setHandlerKeyboardModifiers(QGLViewer::CAMERA, Qt::ControlButton);
  ...
}
```



Qt Designer v2.0



{logname@hostname} designer

Qt Designer v2.0

Sauvegarde du fichier sous format XML (<filename>.ui)

```
<!DOCTYPE UI><UI>
<class>Form1</class>
<widget>
  <class>QDialog</class>
  <property stdset="1">
    <name>name</name>
    <cstring>Form1</cstring>
  </property>
  ...
  <widget>
    <class>QButtonGroup</class>
  ...
</widget>
</UI>
```

Qt Designer v2.0

Utilisation dans une application (`main.cpp`)

```
#include <qapplication.h>
#include "essai.h"
int main(int argc, char* argv[]) {
    QApplication app (argc,argv);
    Form1 form;
    app.setMainWidget(&form);
    form.show();
    return app.exec();
}
```

Génération du code C++, compilation, édition de liens

```
uic -o ihm1.h ihm1.ui
uic -i ihm1.h -o ihm1.cpp ihm1.ui
g++ -o ihm1 -I$QTDIR/include ihm1.cpp main.cpp -L$QTDIR/lib -lqt
```

Si connexion de signaux sous Qt Designer

```
moc -o moc_ihm1.cpp ihm1.h
g++ -o ihm1 -I$QTDIR/include ihm1.cpp main.cpp moc_ihm1.cpp
-L$QTDIR/lib -lqt
```

Bibliographie

Pour plus d'informations:

- ▷ Mathias Kalle DALHEIMER: "Programmer avec Qt"
Edition O'Reilly 2000,

La bonne adresse

- ▷ sur Qt: <http://www.trolltech.com>



A Suivre...