

Qt

**et graphique
interactif**

Toolkit graphique Qt
Dessin interactif
Machines à états
Qt Designer

Eric Lecolinet - Télécom ParisTech
www.telecom-paristech.fr/~elc

Septembre/octobre 2016

Toolkit graphique Qt

Dessin interactif

Machines à états

Qt Designer

Qt

et graphique interactif

Eric Lecolinet - Télécom ParisTech

www.telecom-paristech.fr/~elc

Septembre/octobre 2016

Qt

Un environnement complet

- Dont une boîte à outils graphique **puissante et multiplateformes**
- Mais aussi tout un ensemble de **bibliothèques utilitaires et d'extensions**
- Utilisé dans de **nombreux produits** libres (KDE...) ou propriétaires

Développement et licences

- Développé par **TrollTech**, puis **Nokia**, puis **Digia**
 - <http://qt.digia.com>
- Licences **LGPL** (gratuite) et commerciales

- Qt WebEngine Widgets The Qt WebEngine Widgets module provides a web browser engine as well as C++ classes to render and interact with web content.
- Qt WebKit Widgets The Qt WebKit Widgets module provides a web browser engine as well as C++ classes to render and interact with web content.
- Qt3DCore Qt3D Core module contains functionality to support near-realtime simulation systems
- Qt3DInput Qt3D Input module provides classes for handling user input in applications using Qt3D
- Qt3DRenderer Qt3D Renderer module contains functionality to support 2D and 3D rendering using Qt3D
- QtBluetooth Enables basic Bluetooth operations like scanning for devices and connecting them
- QtConcurrent Qt Concurrent module contains functionality to support concurrent execution of program code
- QtCore Provides core non-GUI functionality
- QtDBus Qt D-Bus module is a Unix-only library that you can use to perform Inter-Process Communication using the D-Bus protocol
- QtDesigner Provides classes to create your own custom widget plugins for Qt Designer and classes to access Qt Designer components
- QtGui Qt GUI module provides the basic enablers for graphical applications written with Qt
- QtHelp Provides classes for integrating online documentation in applications
- QtLocation Provides C++ interfaces to retrieve location and navigational information
- QtMultimedia Qt Multimedia module provides audio, video, radio and camera functionality
- QtNetwork Provides classes to make network programming easier and portable
- QtNfc An API for accessing NFC Forum Tags
- QtOpenGL Qt OpenGL module offers classes that make it easy to use OpenGL in Qt applications
- QtPositioning Positioning module provides positioning information via QML and C++ interfaces
- QtPrintSupport Qt PrintSupport module provides classes to make printing easier and portable
- QtQml C++ API provided by the Qt QML module
- QtQuick Qt Quick module provides classes for embedding Qt Quick in Qt/C++ applications
- QtQuickWidgets C++ API provided by the Qt Quick Widgets module
- QtScript Qt Script module provides classes for making Qt applications scriptable
- QtScriptTools Provides additional components for applications that use Qt Script
- QtSensors Provides classes for reading sensor data
- QtSerialPort List of C++ classes that enable access to a serial port
- QtSql Provides a driver layer, SQL API layer, and a user interface layer for SQL databases
- QtSvg Qt SVG module provides functionality for handling SVG images
- QtTest Provides classes for unit testing Qt applications and libraries
- QtUiTools Provides classes to handle forms created with Qt Designer
- QtWebChannel List of C++ classes that provide the Qt WebChannel functionality
- QtWebSockets List of C++ classes that enable WebSocket-based communication
- QtWidgets Qt Widgets module extends Qt GUI with C++ widget functionality
- QtXml Qt XML module provides C++ implementations of the SAX and DOM standards for XML
- QtXmlPatterns Qt XML Patterns module provides support for XPath, XQuery, XSLT and XML Schema validation

3

Toolkit graphique Qt

Boîte à outils graphique

- en **C++** à la base
- mais également extension **Qt Quick** basée sur le langage déclaratif QML

Multi-plateformes

- principaux **OSs** : Windows, Mac OS X, Linux/X11
- et plate-formes mobiles : iOS, Android, WinRT

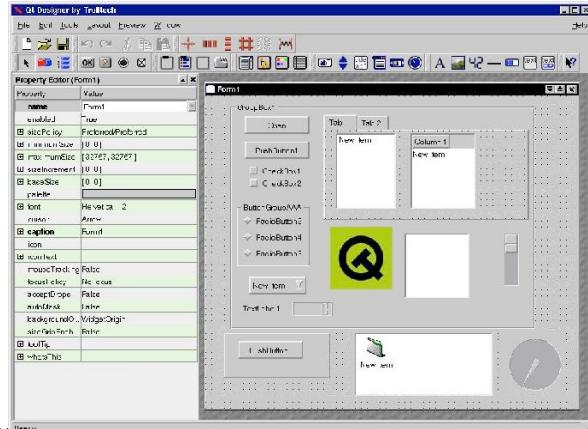
Outils et facilités

Outils de développement

- **QtCreator / QtDesigner**
- qmake
- Qt Linguist
- Qt Assistant
- etc.

Facilités

- internationalisation:
 - **QString et Unicode**
- sockets, **XML**, **SQL**, outils **Web**, **OpenGL**, etc.



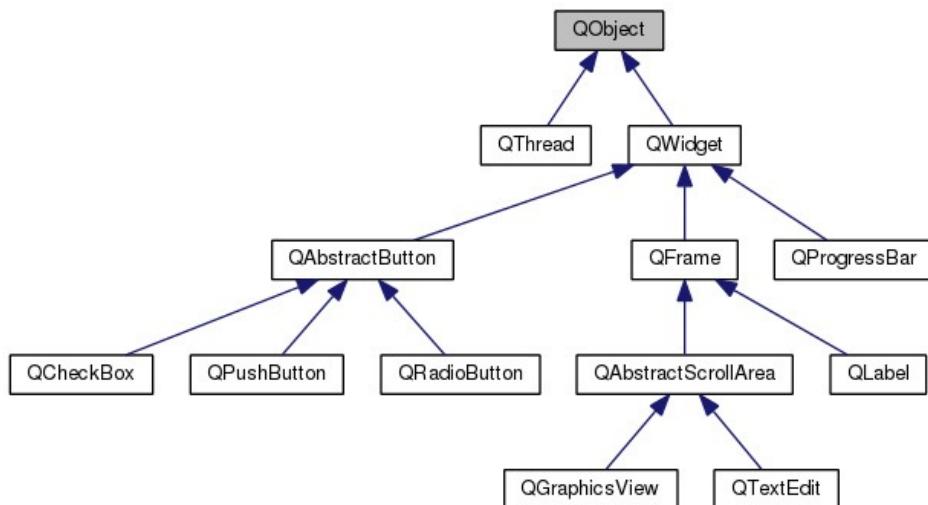
Liens

Liens utiles

- **Site Télécom** : <http://www.telecom-paristech.fr/~elc/qt>
- **Site général Qt** : <http://www.qt.io/>
- **Qt 5.7** :
 - Page principale : <http://doc.qt.io/qt-5/>
 - Toutes les classes (par modules) : <http://doc.qt.io/qt-5/modules-cpp.html>
- **Qt Widgets** :
 - Documentation : <http://doc.qt.io/qt-5/qtwidgets-index.html>
 - Liste des classes : <http://doc.qt.io/qt-5/qtwidgets-module.html>

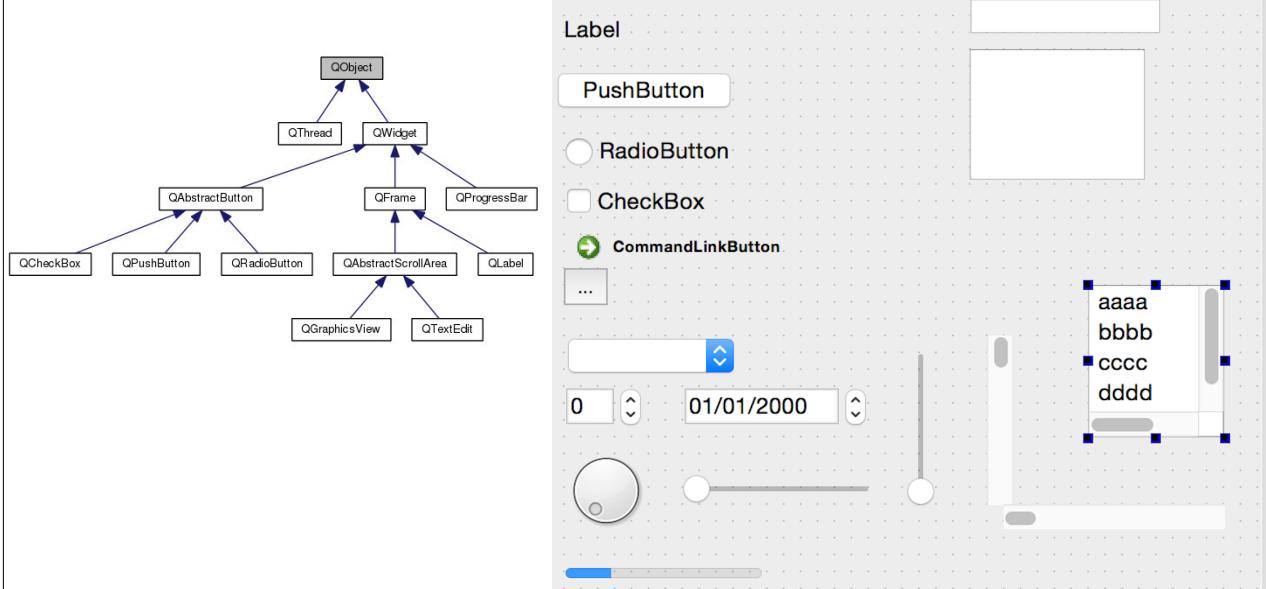
Principaux widgets

Arbre d'héritage (très réduit)



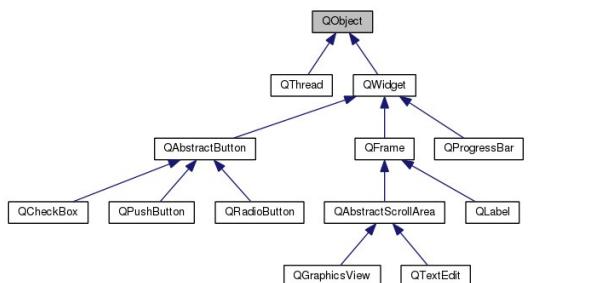
Principaux widgets

Interacteurs



Principaux widgets

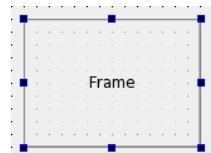
Conteneurs



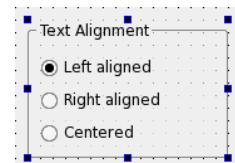
QWidget

QDockWidget

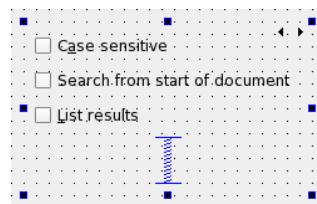
Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif



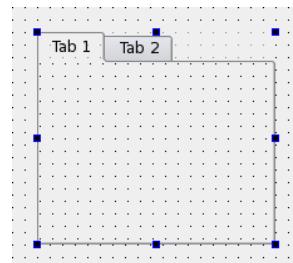
QFrame



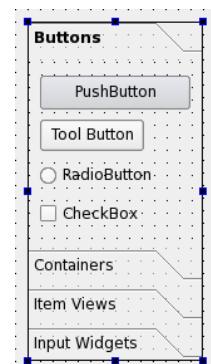
QGroupBox



QStackedWidget



QTabWidget

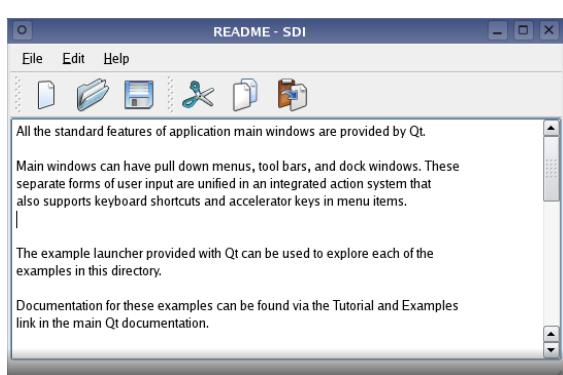


QToolbox

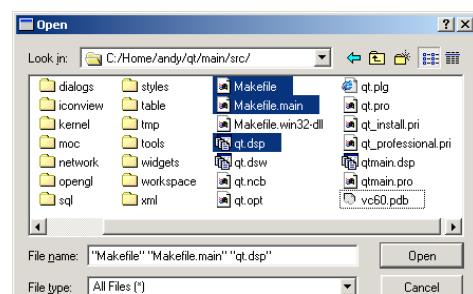
9

Principaux widgets

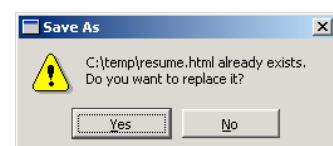
Fenêtre principale, boîtes de dialogue, menus



QMainWindow



QFileDialog



QMessageBox

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

10

Hello Word!

```
#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

- pointeurs C++ : attention aux `->` et aux `*`
- `hello->show()` => le `QLabel` devient le widget principal
- `exec()` lance la boucle de gestion des événements

Variante (objets dans la pile)

```
#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

- la variable **contient l'objet** => `.` pour accéder aux champs et non `->`
- attention : objets **créés dans la pile** => détruits en fin de fonction

Avec un conteneur

```
#include <QApplication>      // inclure headers adéquats !
#include <QPushButton>
#include <QWidget>
#include <QFont>

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

    QWidget * box = new QWidget();  
box->resize(200, 120);

    QPushButton * quitBtn = new QPushButton("Quit", box);    // box sera le parent de quitBtn
    quitBtn->resize(100, 50);
    quitBtn->move(50, 35);
    quitBtn->setFont( QFont("Times", 18, QFont::Bold) );

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



Le parent est passé en argument du constructeur

- pas d'argument pour les "top-level" widgets

Signaux et slots

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

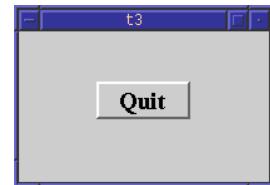
int main( int argc, char **argv ) {
    QApplication * app = new QApplication(argc, argv);

    QWidget * box = new QWidget();  
box->resize(200, 120);

    QPushButton * quitBtn = new QPushButton("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() ) );    // connexion

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



Signaux et slots

Un signal est émis vers le monde extérieur

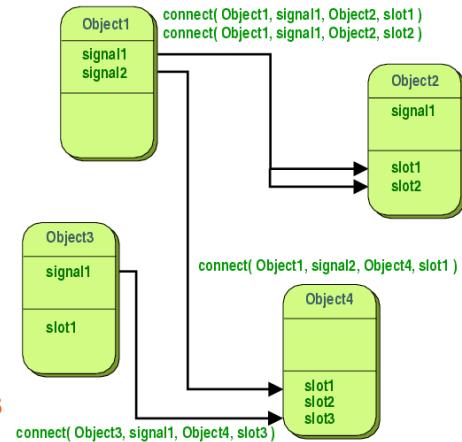
- par un objet quand il **change d'état**
- on **n'indique pas** à qui il s'adresse

Un slot est un récepteur

- en pratique c'est une **méthode**

Les signaux sont connectés à des slots

- les **slots** sont alors appelés automatiquement



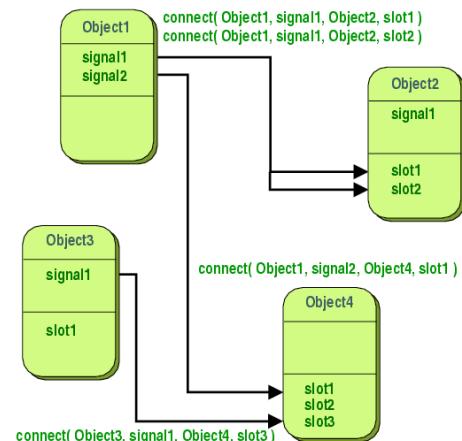
Signaux et slots

Modularité, flexibilité

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

Noter que

- l'émetteur **n'a pas besoin de connaître** le ou les récepteur(s)
- il **ne sait pas** si le signal est reçu
- le récepteur n'a **pas besoin de connaître** l'émetteur



Signaux et slots

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

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

    QWidget * box = new QWidget();
    box->resize(200, 120);

    QPushButton * quitBtn = new QPushButton("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() ) ); // connexion

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



Connexion

Règles

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

```
QObject::connect( quitBtn, SIGNAL(clicked()), app, SLOT(quit() ) );
QObject::connect( x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)) );
QObject::connect( x, SIGNAL(balanceChanged(int)), app, SLOT(quit()) );
```

Aspect central de Qt

- diffère de l'habituel mécanisme des **callbacks** ou **listeners**

Avantages / inconvénients

- **SLOT** et **SIGNAL** sont des **macros** => phase de **précompilation**

Déclaration de slot

Dans le fichier header (.h)

```
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 le précompilateur
- **signals** pas implémentés, **slots doivent** être implémentés

Définition de slot

Dans le fichier d'implémentation (.cpp)

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

```
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 );
};
```

- mot-clé **emit** pour **précompilateur**
- provoque l'**émission** :
 - du signal **balanceChanged()**
 - avec la nouvelle valeur de curBalance

Connexion

Connexion simple

```
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 );  
};
```

```
BankAccount x, y;  
connect( &x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)) );  
x.setBalance( 2450 );
```

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

Connexion

Connexion simple

```
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 );  
};
```

```
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

Connexion

Connexion dans les deux sens

```
BankAccount x, y;  
connect( &x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)) );  
connect( &y, SIGNAL(balanceChanged(int)), &x, SLOT(setBalance(int)) );  
x.setBalance( 2450 );
```

- OK ?

Connexion

Connexion dans les deux sens

```
BankAccount x, y;  
connect( &x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)) );  
connect( &y, SIGNAL(balanceChanged(int)), &x, SLOT(setBalance(int)) );  
x.setBalance( 2450 );
```

- OK : car **test** dans `setBalance()`

vérifier que la valeur a effectivement changé pour éviter les **boucles infinies** !

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

Connexion

Nouvelle syntaxe

```
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 );  
};
```

```
BankAccount x, y;  
connect(&x, &BankAccount::balanceChanged, &y, &BankAccount::setBalance);  
x.setBalance( 2450 );
```

- utilise les **pointeurs de méthodes** de C++
- **avantage** : vérification de la **validité de la connexion** à la **compilation**
 - avec l'ancienne syntaxe c'est fait à **l'exécution**
- **inconvénient** : pas compatible avec **QtQuick**

Connexion

Propagation de signal

```
class Controller : public QObject {  
    Q_OBJECT  
signals:  
    void changed(int);  
....  
};
```

```
connect( &x, SIGNAL(balanceChanged(int)), &controller, SIGNAL(changed(int)) );
```

Déconnexion

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

Compilation

Meta Object Compiler (MOC)

- préprocesseur 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**

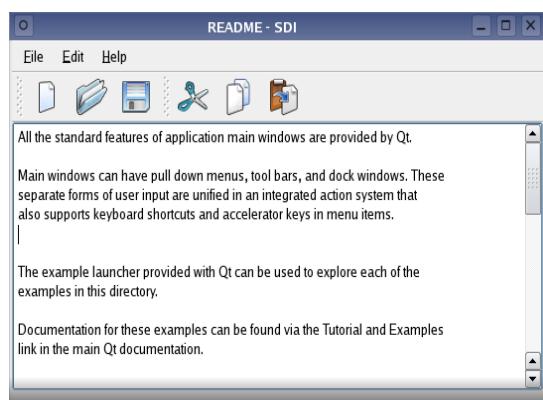
Utilisation de qmake

- dans le répertoire contenant les fichiers sources faire:
 - **qmake -project** // crée le fichier **xxx.pro** (décrit le projet)
 - **qmake** // crée le fichier **Makefile** (ou équivalent si IDE)
 - **make** // crée les fichiers **moc** (un par fichier ayant des **slots**),
// et les fichiers binaires (*.o et exécutable)

Fenêtre principale (QMainWindow)

Zones prédefinies pour:

- barre de menu
- barre d'outils
- barre de statut
- zone centrale
- (et d'autres fonctionnalités...)



Utilisation:

- créer une sous-classe de **QMainWindow**
- dont le constructeur crée / ajoute des objets graphiques à cette fenêtre

Fenêtre principale

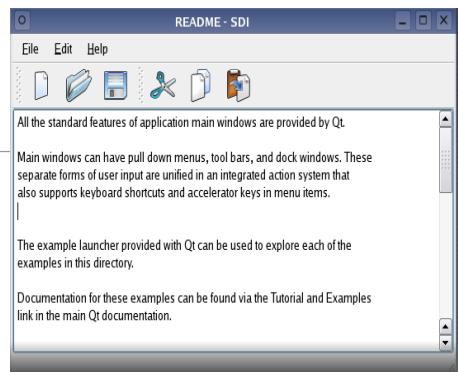
Dans le constructeur d'une classe dérivant de **QMainWindow**

```
// menuBar() est une méthode de QMainWindow
QMenuBar * menuBar = this->menuBar();
QMenu * fileMenu = menuBar->addMenu( tr ("&File") );

// new.png est un fichier qui sera spécifié dans un fichier de ressources .qrc
 QAction * newAction = new QAction( QIcon(":new.png"), tr("&New..."), this);

newAction->setShortcut( tr("Ctrl+N"));           // accélérateur clavier
newAction->setToolTip( tr("New file"));          // bulle d'aide
newAction->setStatusTip( tr("New file"));         // barre de statut

fileMenu->addAction(newAction);                  // rajouter l'action au menu déroulant
// connecter le signal à un slot de this
connect(newAction, SIGNAL(triggered()), this, SLOT(open()));
```



Fenêtre principale

Actions

- les actions peuvent s'ajouter **à la fois** dans les **menus** et les **toolbars**

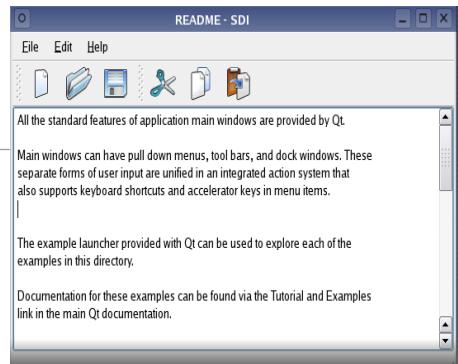
```
QToolBar * toolBar = this->addToolBar( tr("File"));
toolBar->addAction(newAction);
```

Zone centrale

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

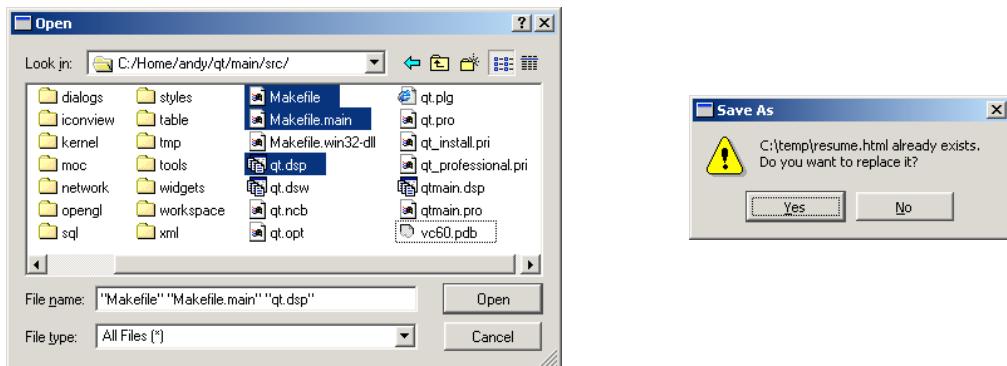
tr()

- permet de **traduire** le texte (localisation)
- (à suivre)



Boîtes de dialogue

QFileDialog, QMessageBox



Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

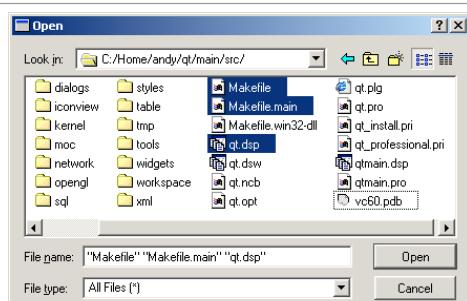
31

Boîte de dialogue modale

Solution générale

```
QFileDialog dialog (parent);
dialog.setFilter("Text files (*.txt)");
QStringList fileNames;

if (dialog.exec() == QDialog::Accepted) {
    fileNames = dialog.selectedFiles();
    QString firstName = fileNames[0];
    ...
}
```



dialog.exec() lance une boucle de gestion des événements secondaire

Solution simplifiée

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

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

32

QString

Codage Unicode 16 bits

- Suite de **QChars**
 - 1 caractère = 1 **QChar** de 16 bits (cas usuel)
 - 1 caractère = 2 **QChars** de 16 bits (pour valeurs > 65535)
- **Conversions** d'une **QString** :
 - **toAscii()** : ASCII 8 bits
 - **toLatin1()** : Latin-1 (ISO 8859-1) 8 bits
 - **toUtf8()** : UTF-8 Unicode multibyte (1 caractère = 1 à 4 octets)
 - **toLocal8Bit()** : codage local 8 bits
- **qPrintable (const QString & str)**
 - équivalent à : str.toLocal8Bit().constData()

QFile

QFile

- lecture, écriture de fichiers
- exemples :

```
QFile file( fileName );
if ( file.open( QIODevice::ReadOnly | QIODevice::Text ) ...;
if ( file.open( QIODevice::WriteOnly ) ) ...;
```

QTextStream

QTextStream

- lecture ou écriture de **texte** depuis un **QFile** :

```
QTextStream stream( &file );
```

- compatibles avec **QString** et **Unicode**

- opérateurs **<<** et **>>** :

- `output_stream << arg;`
- `input_stream >> arg;` // s'arrête au premier espace si arg est une string

- lire une ligne ou un fichier de texte :

- `QString readLine(taillemax = 0);` // pas de limite de taille si = 0
- `QString readAll();` // à n'utiliser que pour des petits fichiers

- codecs :

- `setCodec(codec), setAutoDetectUnicode(bool);`

Ressources

Fichier source

```
QAction * newAction =
    new QAction( QIcon(":new.png"),
                 tr("&New..."),
                 this );
newAction->setShortcut( tr("Ctrl+N") );
```

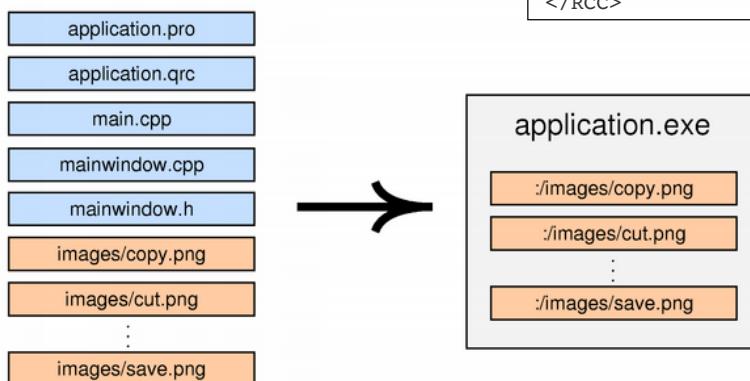
// : signifie: relatif au programme
// tr() pour éventuelle traduction
// l'accélérateur clavier peut être traduit

Ressources

Fichier .qrc

- créé à la main ou par **QtCreator**

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
  <file>images/copy.png</file>
  <file>images/cut.png</file>
  <file>images/new.png</file>
  <file>images/open.png</file>
  <file>images/paste.png</file>
  <file>images/save.png</file>
</qresource>
</RCC>
```



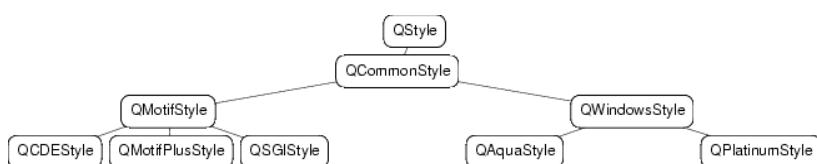
Styles

Emulation du "Look and Feel"

- Look and feel simulé et paramétrable (comme Swing)
- rapidité, flexibilité, extensibilité,
- pas restreint à un "dénominateur commun »

QStyle

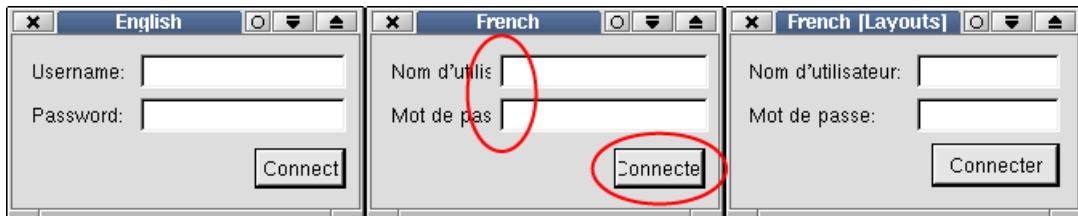
- QApplication::**setStyle**(new MyCustomStyle);



Layout

Buts

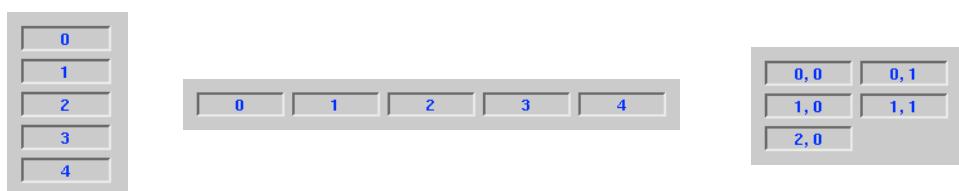
- internationalisation
- retaillage interactif
- éviter d'avoir à faire des calculs



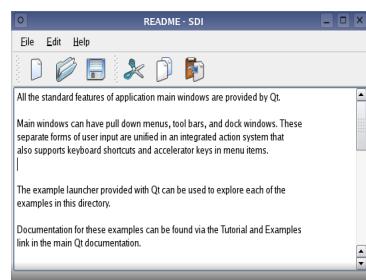
Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

39

Layout



QHBoxLayout, QVBoxLayout, QGridLayout



QFormLayout

(cf. aussi **QMainWindow**)

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

40

Layout : exemple

```
QVBoxLayout * v_layout = new QVBoxLayout();
v_layout->addWidget( new QPushButton( "OK" ) );
v_layout->addWidget( new QPushButton( "Cancel" ) );
v_layout->addStretch();
v_layout->addWidget( new QPushButton( "Help" ) );
```



Les Layouts

- peuvent être emboîtés
- ne sont pas liés à une hiérarchie de conteneurs comme en Java
- cf. le « **stretch** »



Layout : exemple

```
QVBoxLayout * v_layout = new QVBoxLayout();
v_layout->addWidget( new QPushButton( "OK" ) );
v_layout->addWidget( new QPushButton( "Cancel" ) );
v_layout->addStretch();
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 );
```



Layout : exemple

```
QVBoxLayout * v_layout = new QVBoxLayout();
v_layout->addWidget( new QPushButton( "OK" ) );
v_layout->addWidget( new QPushButton( "Cancel" ) );
v_layout->addStretch();
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();
top_layout->addWidget( new QLabel( "Select a country", this ) );
top_layout->addLayout( h_layout );
```

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



43

Retour sur les signaux et les slots

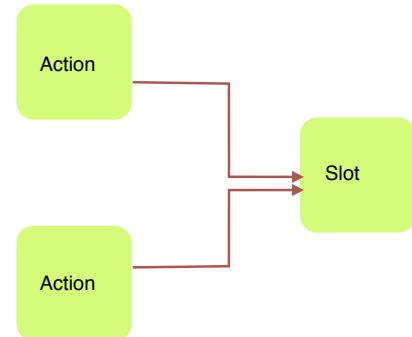
Comment différencier des actions dans un même slot ?



- l'émetteur n'a **pas besoin de connaître** le ou les récepteur(s)
- le récepteur n'a **pas besoin de connaître** l'émetteur ... **parfois si !**

Solution 1: QObject::sender()

```
// Dans le .h en variables d'instance de MaClasse :  
 QAction * action1, * action2, ...;  
  
// Dans le .cpp:  
void MaClasse::createGUI() {  
    action1 = new QAction(tr("Action 1"), this);  
    connect(action1, SIGNAL(triggered()), this, SLOT(dolt()));  
  
    action2 = new QAction(tr("Action 2"), this);  
    connect(action2, SIGNAL(triggered()), this, SLOT(dolt()));  
    ...  
}  
  
void MaClasse::dolt() {  
    QObject * sender = QObject::sender();           // un peu comme getSource() de Java/Swing  
    if (sender == action1) ....;  
    else if (sender == action2) .... ;  
    ...  
}
```

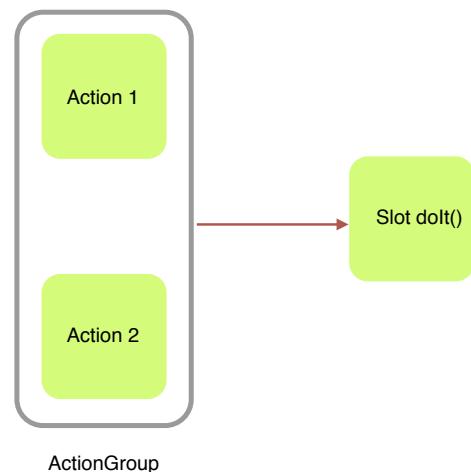


Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

45

Solution 2: QActionGroup

```
// Dans le .h en variables d'instance de MaClasse :  
 QAction * action1, * action2, ...;  
  
// Dans le .cpp:  
void MaClasse::createGUI() {  
    QActionGroup *group = new QActionGroup(this);  
    // un seul connect !  
    connect(group, SIGNAL(triggered(QAction *)),  
            this, SLOT(dolt(QAction *)));  
    action1 = group->addAction(tr("Action 1"));  
    action2 = group->addAction(tr("Action 2"));  
    ...  
}  
  
void MaClasse::dolt(QAction * sender) {           // l'action est récupérée via le paramètre  
    if (sender == action1) ....;  
    else if (sender == action2) .... ;  
    ...  
}
```



Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

46

Solution 2: QActionGroup

Par défaut le groupe est **exclusif**, sinon faire :

```
group->setExclusive(false);
```

On peut faire de même pour les boutons (QPushButton, QRadioButton, QCheckBox ...) :

```
QButtonGroup * group = new QButtonGroup(this);
```

En utilisant un des signaux suivants de QButtonGroup :

```
buttonClicked(QAbstractButton * button)
```

ou :

```
buttonClicked(int id)
```

Solution 3: QSignalMapper

```
void MaClasse::createGUI() {
    QSignalMapper* mapper = new QSignalMapper (this) ;
    connect(mapper, SIGNAL(mapped(int)), this, SLOT(dolt(int))) ;

    QPushButton * btn1 = new QPushButton("Action 1"), this);
    connect (btn1, SIGNAL(clicked( )), mapper, SLOT(map( )));
    mapper->setMapping (btn1, 1) ;

    QPushButton * btn2 = new QPushButton("Action 1"), this);
    connect (btn2, SIGNAL(clicked( )), mapper, SLOT(map( )));
    mapper->setMapping (btn2, 2) ;
    ...
}

void MaClasse::dolt(int value){      // l'action est récupérée via le paramètre
    ...
}
```

Possible pour les types : **int, QString&, QWidget*** et **QObject***

Graphique 2D illustré en Qt

Dessiner dans un widget

Un widget n'est repeint que lorsque c'est nécessaire

- l'application est **lancée** ou **déiconifiée**
- l'application est **déplacée** (selon l'OS)
- une fenêtre qui **cachait** une partie du widget est **déplacée**
- on le **demande explicitement** via la fonction **update()**

Dessiner dans un widget

Un widget n'est repeint que lorsque c'est nécessaire

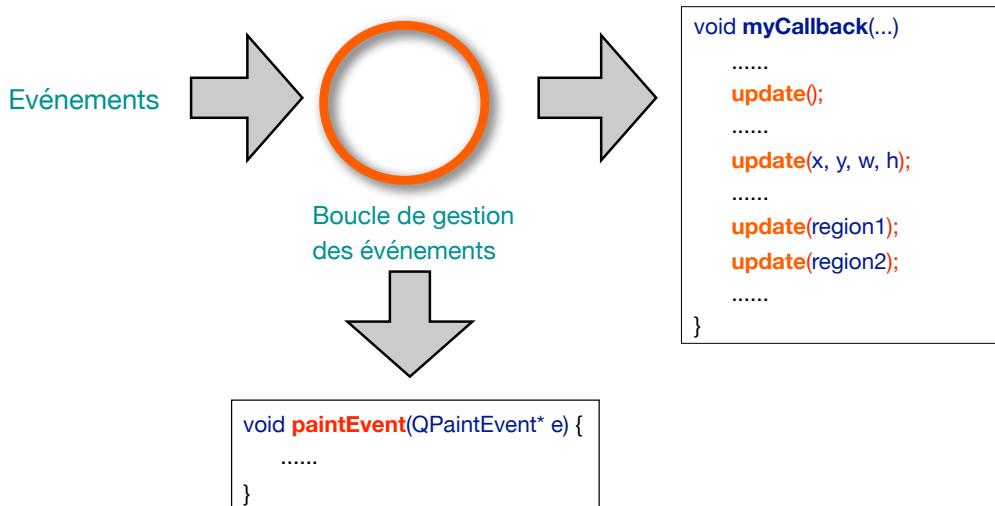
Modèle « damaged / repaint »

- contrairement aux applications 3D où, généralement, on réaffiche tout le temps

Dans tous les cas

- la méthode `void paintEvent (QPaintEvent *)` de **QWidget** est appelée

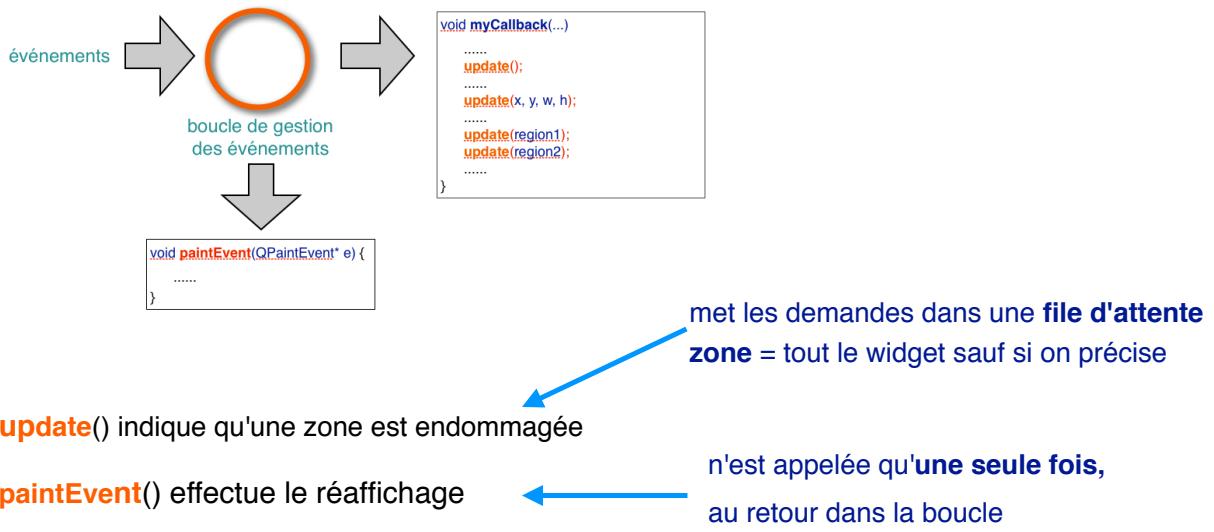
Modèle "damaged / repaint"



update() indique qu'une zone est endommagée

paintEvent() est automatiquement appelée quand il faut réafficher

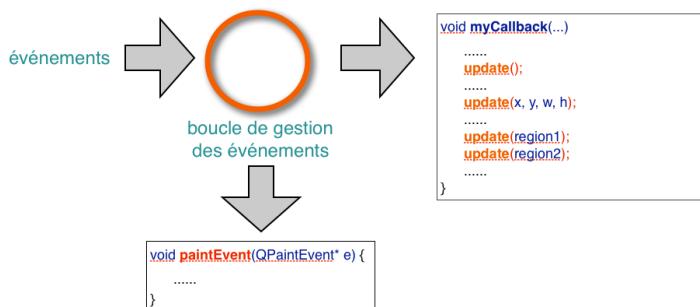
Modèle "damaged / repaint"



Attention

- afficher le dessin **dans** `paintEvent()`
- ne pas appeler `paintEvent()` directement

Modèle "damaged / repaint"



Compléments

repaint() entraîne un **réaffichage immédiat** (contrairement à **update()**)

- ne pas l'utiliser sauf cas particuliers (animation)

Java fonctionne de manière similaire mais les noms sont **différents** !

- **update()** Qt == **repaint()** Java
- **paintEvent()** Qt == **paint()** Java

Redessiner

Créer une sous-classe de QWidget qui redéfinit paintEvent()

```
#include <QWidget>                                         Header Canvas.h

class Canvas : public QWidget {
public:
    Canvas(QWidget* parent) : QWidget(parent) {}
protected:
    virtual void paintEvent(QPaintEvent*);
```

```
#include <QPainter>                                         Implémentation Canvas.cpp
#include "Canvas.h"

void Canvas::paintEvent(QPaintEvent* e) {
    QWidget::paintEvent(e);                                // comportement standard (afficher le fond, etc.)
    QPainter painter(this);                               // crée un Painter pour ce Canvas
    painter.drawLine(50, 10, 100, 20);
}
```

N'utiliser QPainter que dans paintEvent() (à cause du double buffering)

DéTECTER les événements

Méthodes de QWidget appelées quand :

- on presse/relâche un bouton
- on double-clique
- on déplace la souris
 - en **appuyant (ou pas)**
sur le bouton souris selon
mouseTracking()

Remarque

- pas de signals / slots !

- void **mousePressEvent(QMouseEvent*)**;
- void **mouseReleaseEvent(QMouseEvent*)**;
- void **mouseDoubleClickEvent(QMouseEvent*)**;
- void **mouseMoveEvent(QMouseEvent*)**;
- void **setMouseTracking(bool)**
- bool **hasMouseTracking()**

DéTECTER les événements

```
Canvas.h
#include <QWidget>
#include <QMouseEvent>

class Canvas : public QWidget {
public:
    Canvas(QWidget* p) : QWidget(p) {}

protected:
    void mousePressEvent(QMouseEvent*);
```

```
Canvas.cpp
#include "Canvas.h"

void Canvas::mousePressEvent(QMouseEvent* e) {
    if (e->button() == Qt::LeftButton) {
        .....
        .....
        update();           // demande de réaffichage
    }
}
```

QMouseEvent permet de récupérer :

- **button()** : **bouton** souris qui a déclenché l'événement. ex: `Qt::LeftButton`
- **buttons()** : **état** des autres boutons. ex: `Qt::LeftButton | Qt::MidButton`
- **modifiers()** : **modificateurs** clavier. ex: `Qt::ControlModifier | Qt::ShiftModifier`

Récupérer la position du curseur

QMouseEvent permet de récupérer :

- la **position locale** (relative au widget) : `pos()` ou `localPos()`
- la **position** relative à un référentiel : `globalPos()`, `screenPos()`, `windowPos()` => utile si on déplace le widget interactivement

Alternative

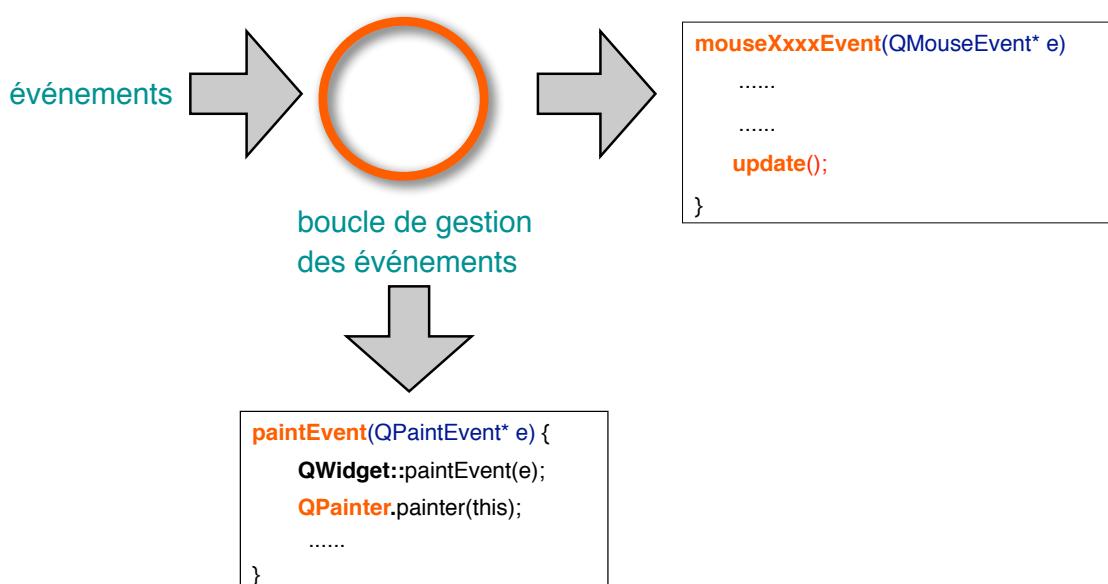
- `QCursor::pos()` : position du curseur sur l'écran
- `QWidget::mapToGlobal()`, `mapFromGlobal()`, etc. : conversion de coordonnées

Ignorer les événements

Ignorer les événements

- **QEvent::ignore()** : signifie que le receveur ne veut pas l'événement
 - ex : dans méthode **closeEvent()** de la MainWindow pour ne pas quitter l'appli

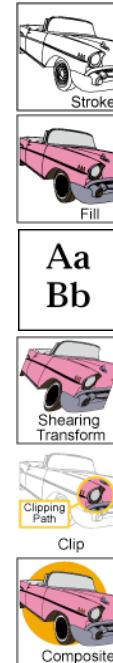
Synthèse



QPainter

Attributs

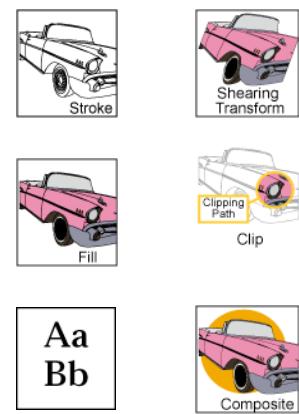
- **setPen()** : lignes et contours
- **setBrush()** : remplissage
- **setFont()** : texte
- **setTransform(), etc.** : transformations affines
- **setClipRect/Path/Region()** : clipping (découpage)
- **setCompositionMode()** : composition



QPainter

Lignes et contours

- **drawPoint()**, **drawPoints()**
- **drawLine()**, **drawLines()**
- **drawRect()**, **drawRects()**
- **drawArc()**, **drawEllipse()**
- **drawPolygon()**, **drawPolyline()**, etc...
- **drawPath()** : chemin complexe



Remplissage

- **fillRect()**, **fillPath()**



Divers

- **drawText()**
- **drawPixmap()**, **drawImage()**, **drawPicture()**
- etc.

QPainter

Classes utiles

- entiers: **QPoint, QLine, QRect, QPolygon**
- flottants: **QPointF, QLineF, ...**
- chemin complexe: **QPainterPath**
- zone d'affichage: **QRegion**

Pinceau: QPen



Attributs

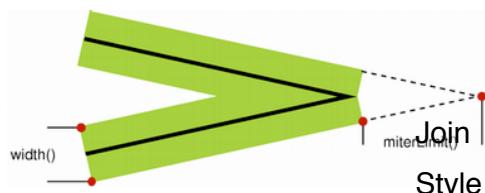
- **style** : type de ligne
- **width** : épaisseur (**0 = «cosmétique»**)
- **brush** : attributs du pinceau (couleur...)
- **capStyle** : terminaisons
- **joinStyle** : jointures



Qt::PenStyle



Cap Style



Join Style

Pinceau: QPen



Exemple

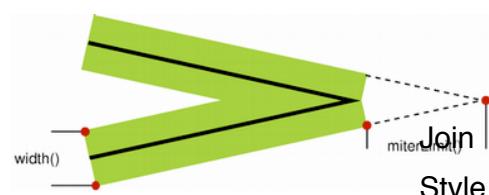
```
// dans méthode paintEvent()  
  
QPen pen;      // default pen  
pen.setStyle(Qt::DashDotLine);  
pen.setWidth(3);  
pen.setBrush(Qt::green);  
pen.setCapStyle(Qt::RoundCap);  
pen.setJoinStyle(Qt::RoundJoin);  
  
QPainter painter(this);  
painter.setPen(pen);
```



Qt::PenStyle



Cap Style



Style

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

65

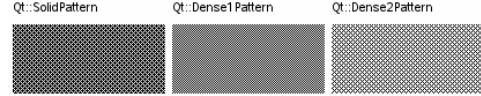
Remplissage: QBrush



Attributs

Qt::BrushStyle

- style
- color
- gradient
- texture



QBrush brush(...);

....

QPainter painter(this);

painter.setBrush(brush);

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

66

Remplissage: QBrush



Attributs

- style
- color
- gradient
- texture

white	black	cyan	darkCyan
red	darkRed	magenta	darkMagenta
green	darkGreen	yellow	darkYellow
blue	darkBlue	gray	darkGray
lightGray			

QColor

Qt::GlobalColor

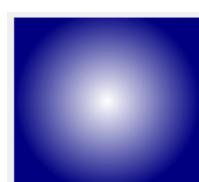
- modèles RGB, HSV or CMYK
- composante alpha (transparence) :
 - alpha blending
- couleurs prédefinies:
 - Qt::GlobalColor

Remplissage: gradients



Type de gradients

- linéaire,
- radial
- conique



QLinearGradient

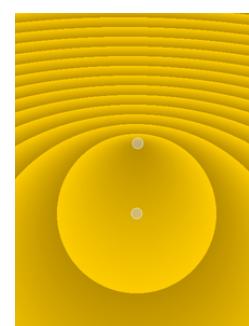
QRadialGradient

QLinearGradient

```
gradient( QPointF(0, 0), QPointF(100, 100) );
```

```
gradient.setColorAt(0, Qt::white);  
gradient.setColorAt(1, Qt::blue);
```

```
QBrush brush(gradient);
```



QConicalGradient

Type de coordonnées

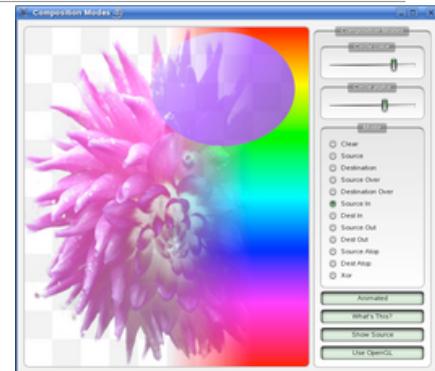
- défini par **QGradient::CoordinateMode**

répétition: setSpread()

Composition

Modes de composition

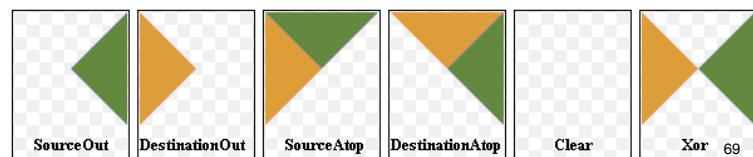
- opérateurs de **Porter Duff**:
- définissent : $F(\text{source}, \text{destination})$
- défaut : **SourceOver**
 - avec alpha blending
 - $\text{dst} \leq a_{\text{src}} * \text{src} + (1-a_{\text{src}}) * a_{\text{dst}} * \text{dst}$
- limitations
 - selon implémentation et Paint Device



Méthode:

`QPainter::setCompositionMode()`

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif



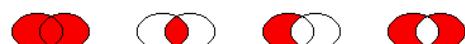
Découpage (clipping)

Découpage

- selon un rectangle, une région ou un path
- `QPainter::setClipping()`, `setClipRect()`, `setClipRegion()`, `setClipPath()`

QRegion

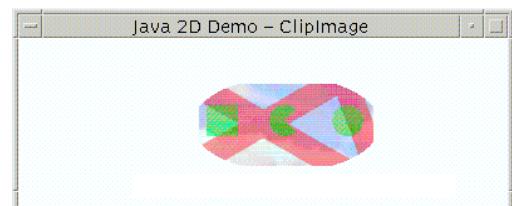
- `united()`, `intersected()`, `subtracted()`, `xored()`



```
QRegion r1(QRect(100, 100, 200, 80), QRegion::Ellipse); // r1: elliptic region
QRegion r2(QRect(100, 120, 90, 30)); // r2: rectangular region
QRegion r3 = r1.intersected(r2); // r3: intersection
```

```
QPainter painter(this);
painter.setClipRegion(r3);

...etc... // paint clipped graphics
```



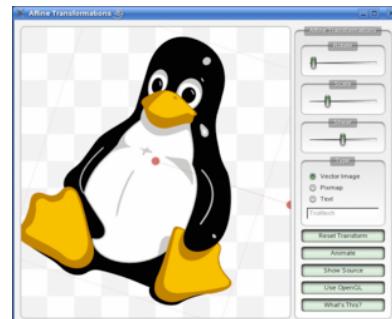
Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

Transformations affines

Transformations

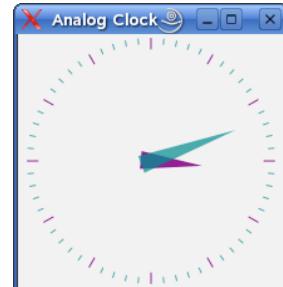
- `translate()`
- `rotate()`
- `scale()`
- `shear()`
- `setTransform()`

Démo !



```
QPainter painter( this );
painter.setRenderHint( QPainter::Antialiasing );
painter.translate( width() / 2, height() / 2 );
painter.scale( side / 200.0, side / 200.0 );

painter.save();           // empile l'état courant
painter.rotate( 30.0 * ((time.hour() + time.minute() / 60.0)) );
painter.drawConvexPolygon( hourHand, 3 );
painter.restore();        // dépile
```



Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

71

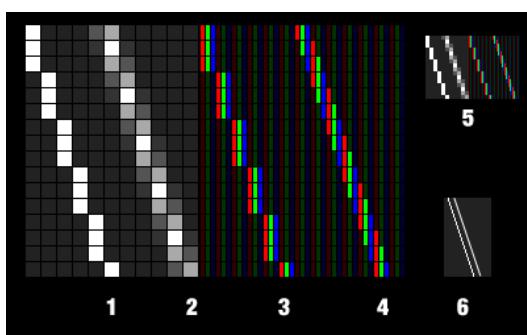
Antialiasing

Anti-aliasing

- éviter l'effet d'escalier
- particulièrement utile pour les polices de caractères

Subpixel rendering

- exemples : ClearType, texte sous MacOSX



ClearType
(Wikipedia)

oeil.jpeg

MacOSX

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

72

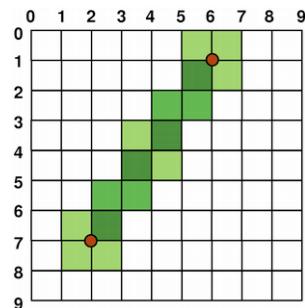
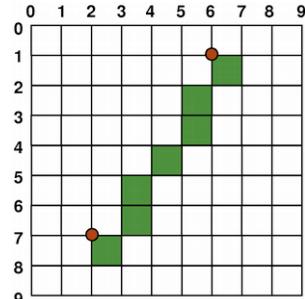
Antialiasing

Anti-aliasing sous Qt

```
QPainter painter(this);
painter.setRenderHint(QPainter::Antialiasing);
painter.setPen(Qt::darkGreen);
painter.drawLine(2, 7, 6, 1);
```

Rendering hints

- “hint” = option de rendu
 - effet non garanti
 - dépend de l’implémentation et du matériel
- **QPainter::setRenderingHints()**



Antialiasing et cordonnées

Epaisseurs impaire

- pixels dessinés à droite et en dessous

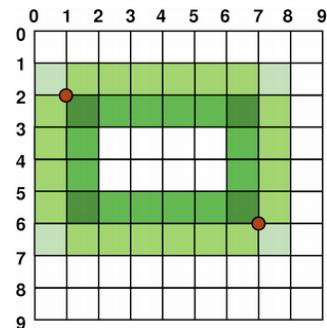
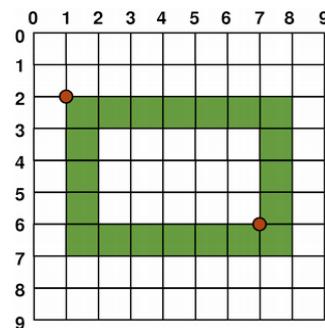
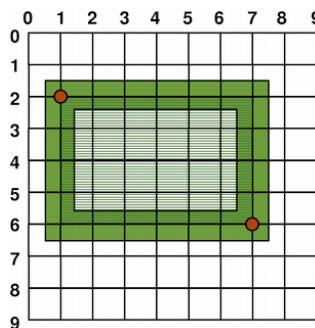
Note

QRect::right() = left() + width() -1

QRect::bottom() = top() + height() -1

Dessin anti-aliasé

- pixels répartis autour de la ligne idéale



Paths

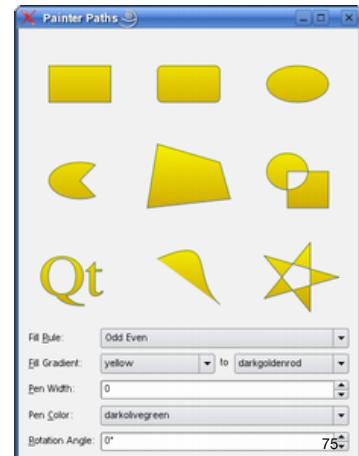
QPainterPath

- figure composée d'une suite arbitraire de lignes et courbes
 - affichée par: **QPainter::drawPath()**
 - peut aussi servir pour remplissage, profilage, découpage



Méthodes

- **déplacements:** `moveTo()`, `arcMoveTo()`
- **dessin:** `lineTo()`, `arcTo()`
- **courbes** de Bezier: `quadTo()`, `cubicTo()`
- `addRect()`, `addEllipse()`, `addPolygon()`, `addPath()` ...
- `addText()`
- `translate()`, union, addition, soustraction...
- et d'autres encore ...



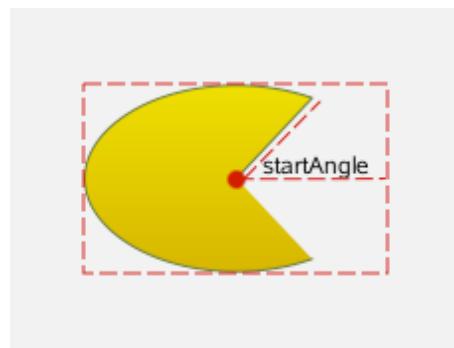
Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

Paths

```
QPointF center, startPoint;  
QPainterPath myPath;
```

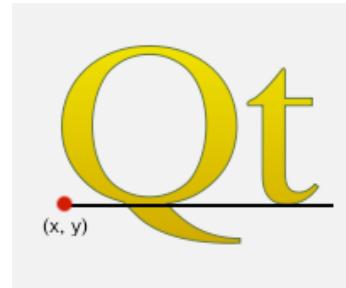
```
myPath.moveTo( center );  
myPath.arcTo( boundingRect, startAngle, sweepAngle );
```

```
QPainter painter( this );  
painter.setBrush( myGradient );  
painter.setPen( myPen );  
painter.drawPath( myPath );
```



Paths

```
QPointF baseline(x, y);  
QPainterPath myPath;  
  
myPath.addText( baseline, myFont, "Qt" );  
  
QPainter painter( this );  
... etc....  
painter.drawPath( myPath );
```



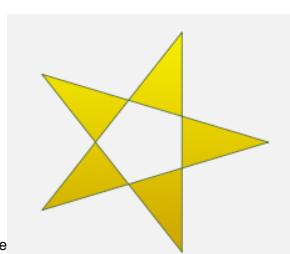
Paths

```
QPainterPath path;  
path.addRect(20, 20, 60, 60);  
path.moveTo(0, 0);  
path.cubicTo(99, 0, 50, 50, 99, 99);  
path.cubicTo(0, 99, 50, 50, 0, 0);
```

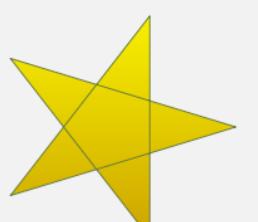
```
QPainter painter(this);  
painter.fillRect(0, 0, 100, 100, Qt::white);  
painter.setPen(QPen(QColor(79, 106, 25), 1, Qt::SolidLine, Qt::FlatCap, Qt::MiterJoin));  
painter.setBrush(QColor(122, 163, 39));  
painter.drawPath(path);
```



Qt::OddEvenFill
(défaut)



Qt::WindingFill



Images

Types d'images

- **QImage**: optimisé pour E/S et accès/manipulation des pixels
 - **QPixmap, QBitmap** : optimisés pour affichage l'écran
- **QPicture**: pour enregistrer et rejouer les commandes d'un **QPainter**
- dans tous les cas : on peut dessiner dedans avec un **QPainter**

Entrées/sorties

- **load()** / **save()** : depuis/vers un fichier, principaux formats supportés
- **loadFromData()** : depuis la mémoire

Images

Format RGB

```
QImage image(3, 3,  
QImage::Format_RGB32);  
QRgb value;  
  
value = qRgb(122, 163, 39); // 0xff7aa327  
image.setPixel(0, 1, value);  
image.setPixel(1, 0, value)
```

	0xff7aa327	
0xff7aa327	0xffbd9527	0xffedba31

Images

Format indexé

```
QImage image(3, 3, QImage::Format_Indexed8);
QRgb value;

value = qRgb(122, 163, 39); // 0xff7aa327
image.setColor(0, value);

value = qRgb(237, 187, 51); // 0xffffedba31
image.setColor(1, value);

image.setPixel(0, 1, 0);
image.setPixel(1, 0, 1);
```

		0	
0		2	1
1			
2			

QPicture

Enregistrer et rejouer les commandes d'un QPainter

- Enregistrer :

```
QPicture picture;
QPainter painter;
painter.begin( &picture );           // paint in picture
painter.drawEllipse( 10,20, 80,70 );   // draw an ellipse
painter.end();                      // painting done
picture.save( "drawing.pic" );       // save picture
```

- Rejouer :

```
QPicture picture;
picture.load( "drawing.pic" );        // load picture
QPainter painter;
painter.begin( &myImage );            // paint in myImage
painter.drawPicture( 0, 0, picture ); // draw the picture at (0,0)
painter.end();                      // painting done
```

Picking

Picking avec QRect, QRectF

- `intersects()`
- `contains()`

Picking avec QPainterPath

- `intersects(const QRectF & rectangle)`
- `intersects(const QPainterPath & path)`
- `contains(const QPointF & point)`
- `contains(const QRectF & rectangle)`
- `contains(const QPainterPath & path)`

Retourne l'intersection

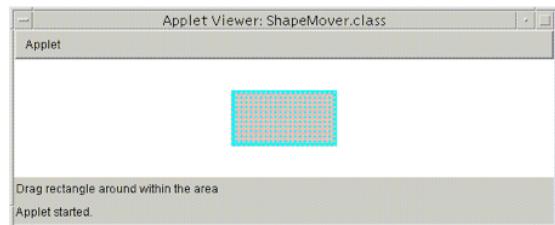
- `QPainterPath intersected(const QPainterPath & path)`

Picking / Interaction

Exemple

- teste si la souris est dans le rectangle quand on appuie sur le bouton de la souris
- met à jour la position du rectangle pour qu'il soit centré

```
QRect rect; // variable d'instance de mon Widget de dessin
void mousePressEvent(QMouseEvent* e) {
    if (rect.contains( e->pos() )) {
        rect.moveCenter( e->pos() );
        update();
    }
    ...
}
void paintEvent(QPaintEvent* e ) {
    QPainter painter( this );
    ..... // specifier attributs graphiques
    painter.drawRect( rect );
}
```



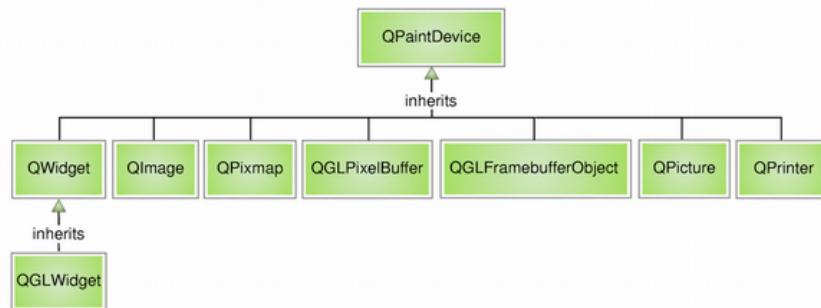
Surfaces d'affichage



- **QPainter** : outil de dessin
- **QPaintDevice** : objet sur lequel on peut dessiner
- **QPaintEngine** : moteur de rendu

On peut dessiner dans tout ce qui hérite de **QPaintDevice**, en particulier **QWidget**

- mais aussi: QPrinter, QPixmap, QImage, QPicture, QGLPixelBuffer, etc.



Eric Lecolinet - Téléc

85

Surfaces d'affichage

SVG

- QSvgWidget
 - QSvgRenderer

OpenGL

- QGLWidget
 - QGLPixelBuffer
 - QGLFramebufferObject

Impression

- QPrinter



QSvgWidget

Graphique structuré 2D

Graphics View

- graphique structuré
 - représentation objet du dessin
 - réaffichage, picking automatiques
- vues d'un **graphe de scène** :
 - QGraphicsScene
 - QGraphicsView
 - QGraphicsItem, etc.

```
QGraphicsScene scene;
QGraphicsRectItem * rect =
    scene.addRect( QRectF(0,0,100,100) );
QGraphicsItem *item = scene.itemAt(50, 50);
.....
QGraphicsView view( &scene );
view.show();
```

Performance de l'affichage

Problèmes classiques :

- **Flickering** et **Tearing** (scintillement et déchirement)
- **Lag** (latence)

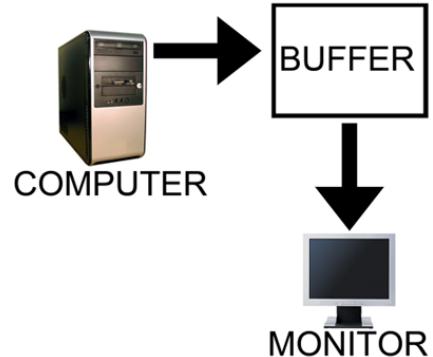
Flicker

Diagnostic

- scintillement de l'affichage
- exemple :
<https://www.youtube.com/watch?v=QoZ8L1quWnc>

Problème

- l'œil perçoit les étapes intermédiaires
- se produit (normalement) en « simple buffering »
 - dessin directement sur le **buffer** de l'écran

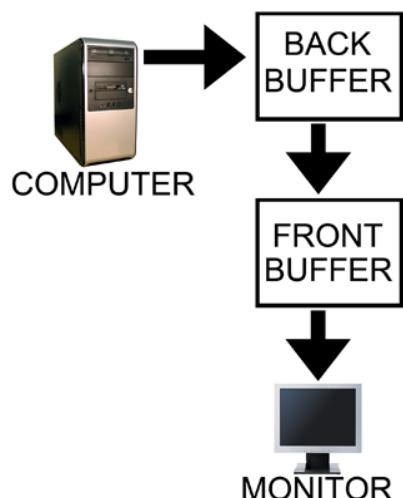


source: AnandTech

Double buffering

Solution

- solution au flickering :
 - dessin dans le **back buffer** (caché)
 - recopie dans le **front buffer** (écran)
- par défaut avec Qt4 ou Java Swing

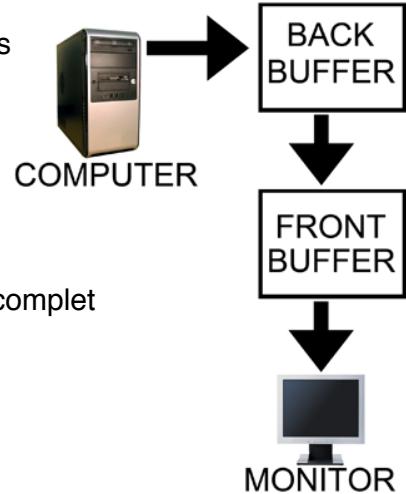


source: AnandTech

Tearing

Diagnostic

- l'image apparaît en 2 (ou 3...) parties horizontales
- exemple :
 - <https://www.youtube.com/watch?v=1nFBtTq0DHo>



Problème

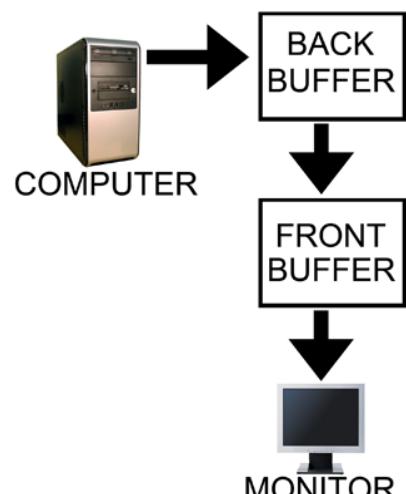
- recopie du **back buffer** avant que le dessin soit complet
 - mélange de plusieurs "frames" vidéo
 - en particulier avec jeux vidéo

source: AnandTech

Tearing

Solution

- **VSync** (Vertical synchronization)
- rendu synchronisé avec l'affichage
- inconvénient : ralentit l'affichage

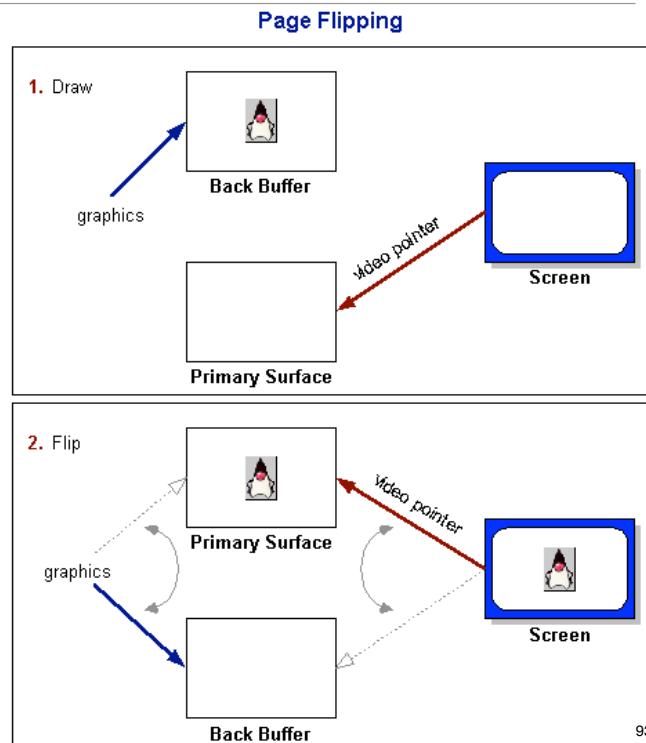


source: AnandTech

Page flipping

Page flipping

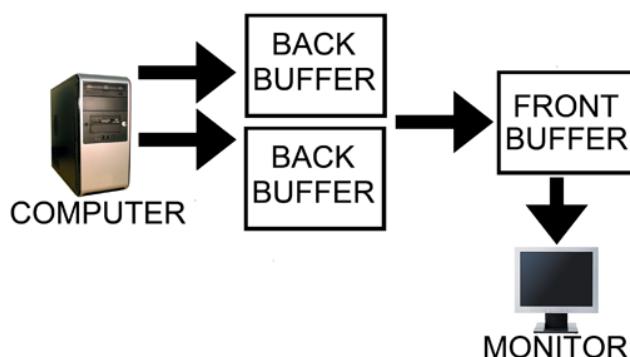
- alternative au double buffering
- pas de recopie des pixels, on change juste de buffer
- plus rapide (mais tout de même constraint par VSync)



Triple buffering

Triple buffering

- évite le ralentissement dû à la **VSync**
 - un des **deux back buffers** est toujours complet
 - le **front buffer** (écran) peut être mis à jour sans attendre

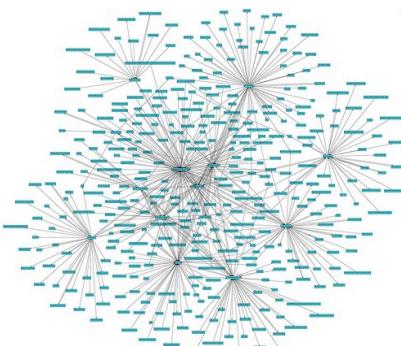


source: AnandTech

Latence (dessin interactif)

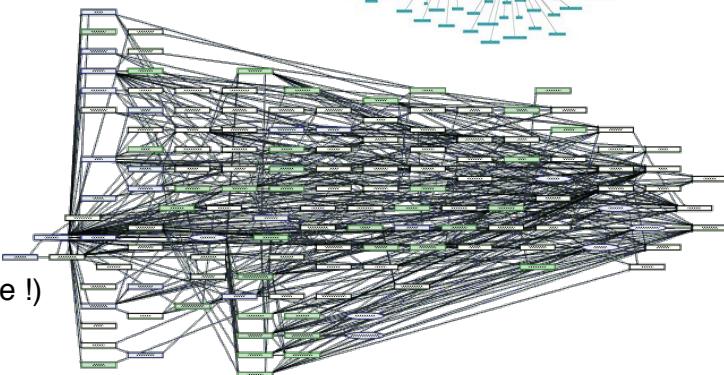
Diagnostic

- l'affichage «ne suit pas» l'interaction
 - ex : gros graphe dont on veut déplacer un noeud



Problème

- trop d'éléments à réafficher à chaque interaction



Solution

- réafficher moins d'éléments
- (ou changer de carte graphique !)

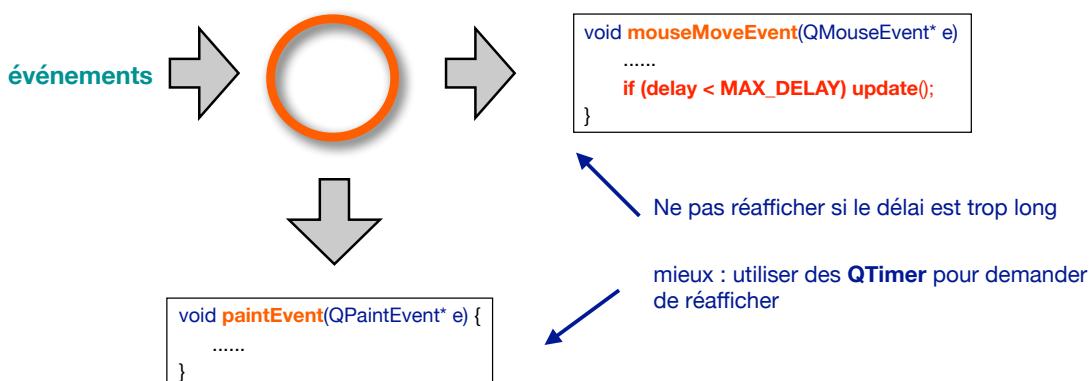
Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

95

Performance du dessin interactif

1) Afficher moins de choses dans le temps

- sauter les images intermédiaires : réafficher une fois sur N ou selon un délai
- inconvénient : affichage moins fluide, plus saccadé



Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

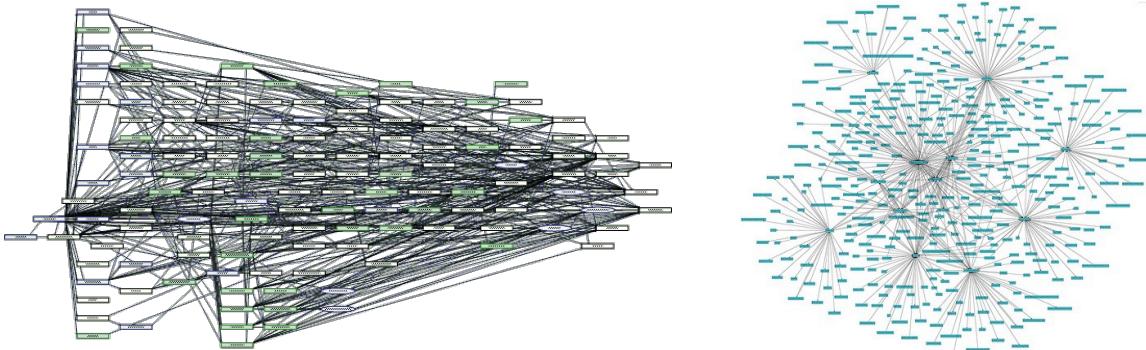
96

Performance du dessin interactif

2) Afficher moins de choses dans l'espace

- Ne réafficher que la partie qui change

- sauver ce qui ne change pas dans une image avant l'interaction
- à chaque interaction réafficher l'image puis ce qui change par dessus



Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

97

XOR

Principe

- affichage en mode XOR
 - très efficace pour déplacements interactifs

Afficher 2 fois pour effacer

- 1er affichage: $\text{pixel} = \text{bg} \wedge \text{fg}$
- 2eme affichage: $\text{pixel} = (\text{bg} \wedge \text{fg}) \wedge \text{fg} = \text{bg}$

Problèmes

- couleurs « aléatoires »

Exemple Java Graphics :

- `setColor(Color c1)`
- `setPaintMode() : dessine avec c1`
- `setXORMode(Color c2) : échange c1 et c2`

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

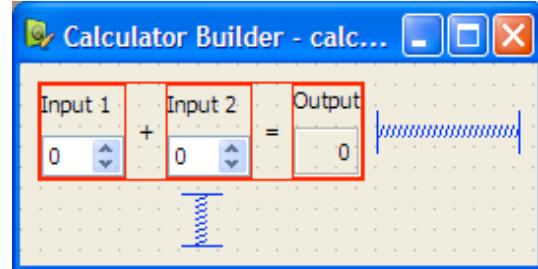
98

Qt Designer



- **Fichiers**

- calculator.pro
- **calculator.ui**
- calculator.h
- calculator.cpp
- main.cpp



- **calculator.ui**

- fichier XML

calculator.pro

```
TEMPLATE = app
FORMS = calculator.ui
SOURCES = main.cpp
HEADERS = calculator.h
```

- **Exemple**

- vidéo et code .ui

99

Qt Designer

main.cpp

```
#include "calculator.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    Calculator widget;
    widget.show();
    return app.exec();
}
```

100

Qt Designer

calculator.h

```
#include "ui_calculator.h"

class Calculator : public QWidget {
    Q_OBJECT
public:
    Calculator(QWidget * parent = 0);
private :
    Ui::Calculator * ui;
};
```

calculator.cpp

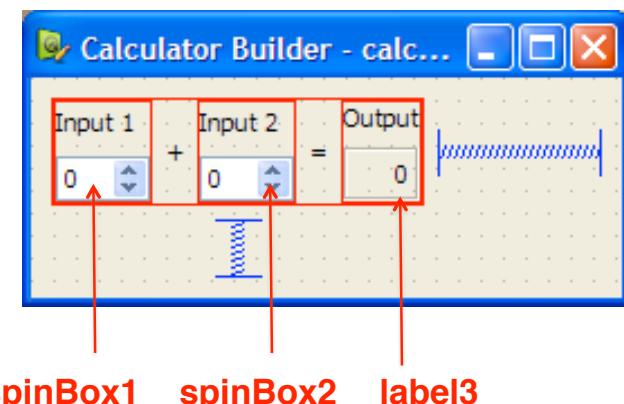
```
#include "calculator.h"

Calculator::Calculator(QWidget * parent) :
    QWidget(parent),
    ui(new Ui::Calculator)
{
    ui->setupUi(this);
    .....
}
```

101

Lien avec le code source

- Comment afficher le résultat du calcul dans le **Qlabel** ?



102

```
#include "ui_calculator.h"

class Calculator : public QWidget {
    Q_OBJECT
public:
    Calculator(QWidget *parent = 0);
private :
    Ui::Calculator * ui;
private slots :
    void on_spinBox1_valueChanged(int value);
    void on_spinBox2_valueChanged(int value);
};
```

Auto-connect



```
#include "calculator.h"

void Calculator::on_spinBox1_valueChanged(int val) {
    QString res = QString::number(val);
    // ou: QString res = QString::number(ui->spinBox1->value());

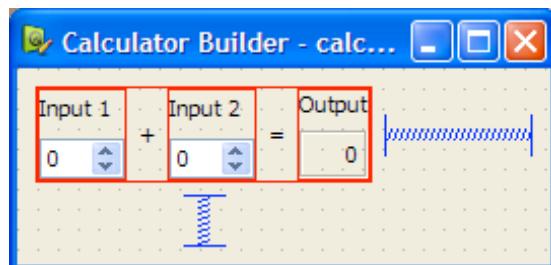
    ui->label3->setText(res);      // la variable label3 a le nom donné au widget dans Qt Designer
}
```

103

Lien avec le code source

On peut lier les signaux des objets créés interactivement avec n'importe quel **slot** :

- par **auto-connexion** :
 - void **onSpinBox1_valueChanged(int)**
- ou via le mode **Edition Signaux/Slots** de QtCreator



104

Variante

- Même principe mais en utilisant l'héritage multiple

```
#include "ui_calculatorform.h"

Class Calculator : public QWidget, public Ui::CalculatorForm {
    Q_OBJECT
public:
    Calculator(QWidget * parent = 0);
};
```

```
Calculator::Calculator(Qwidget *parent) : Qwidget(parent) {
    setupUi(this);
}

void Calculator::on_spinBox1_valueChanged(int val) {
    QString res = QString::number(val);
    label3->setText(res); // au lieu de : ui->label3->setText(res);
}
```

105

Chargement dynamique

```
#include <QtUiTools>

Calculator::Calculator(QWidget *parent) : QWidget(parent) {
    QUiLoader loader;
    QFile file(":/forms/calculator.ui");
    file.open(QFile::ReadOnly);
    QWidget * widget = loader.load(&file, this);
    file.close();
}
```

106

Chargement dynamique

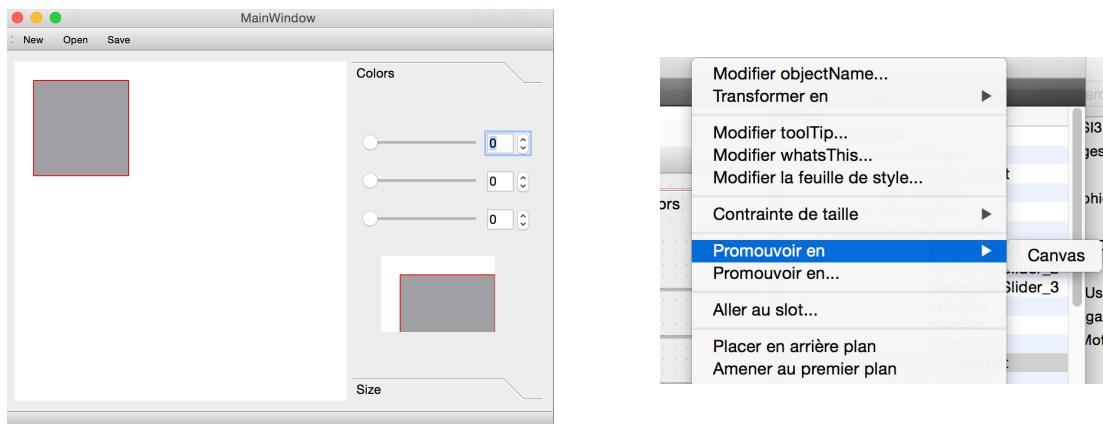
```
class Calculator : public QWidget {  
    Q_OBJECT  
public:  
    Calculator(QWidget *parent = 0);  
private:  
    QSpinBox *ui_spinBox1;  
    QSpinBox *ui_spinBox2;  
    QLabel *ui_label1;  
};  
.....  
ui_spinBox1 = qFindChild<QSpinBox*>(this, "spinBox1");  
ui_spinBox2 = qFindChild<QSpinBox*>(this, "spinBox2");  
ui_label1 = qFindChild<QLabel*>(this, "label1");
```

107

« Promotion »

Promote

- permet d'inclure des instances de ses propres classes dans Designer



Contrôle du layout

Pour chaque QWidget

- setMinimumSize(), setMaximumSize()
- setSizePolicy(QSizePolicy)
- setSizePolicy(QSizePolicy::Policy horizontal, QSizePolicy::Policy vertical)
- setHorizontalPolicy()
- setVerticalPolicy()

canvas : Canvas	
Propriété	Valeur
Hauteur	373
▼ sizePolicy	[Expanding,...]
Politique hori...	Expanding
Politique verti...	Expanding
Étirement hori...	0
Étirement vert...	0
▼ minimumSize	300 x 0
Largeur	300
Hauteur	0
▼ maximumSize	16777215 x ..

settingBox : QToolBox	
Propriété	Valeur
▼ sizePolicy	[Fixed, Pref...]
Politique hori...	Fixed
Politique verti...	Preferred
Étirement hori...	0
Étirement vert...	0
▼ minimumSize	200 x 0
Largeur	200
Hauteur	0
▼ maximumSize	16777215 x ...
Largeur	16777215

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

Contrôle du layout

QSizePolicy

- **Fixed:** = sizeHint() (taille recommandée)
- **Minimum:** >= sizeHint()
- **Maximum:** <= sizeHint()
- **Preferred:** ~ sizeHint(), can be shrunk, can be expanded
- **Expanding:** get as much space as possible, can be shrunk
- **MinimumExpanding:** idem, but can't be shrunk
- **Ignored:** idem, ignoring sizeHint()

canvas : Canvas	
Propriété	Valeur
Hauteur	373
▼ sizePolicy	[Expanding,...]
Politique hori...	Expanding
Politique verti...	Expanding
Étirement hori...	0
Étirement vert...	0
▼ minimumSize	300 x 0
Largeur	300
Hauteur	0
▼ maximumSize	16777215 x ..

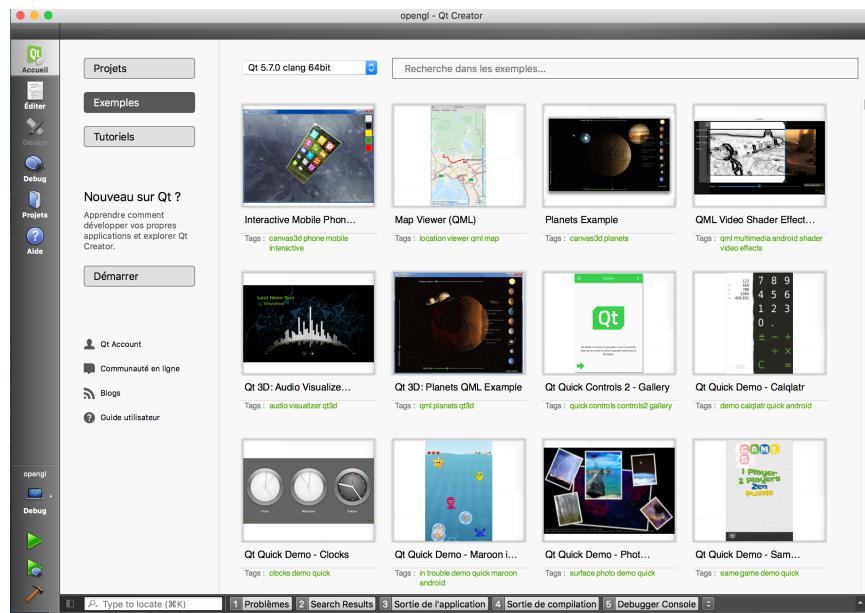
settingBox : QToolBox	
Propriété	Valeur
▼ sizePolicy	[Fixed, Pref...]
Politique hori...	Fixed
Politique verti...	Preferred
Étirement hori...	0
Étirement vert...	0
▼ minimumSize	200 x 0
Largeur	200
Hauteur	0
▼ maximumSize	16777215 x ...
Largeur	16777215

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

110

Exemples et tutoriels

Dans QtCreator > Accueil



Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

111

Graphique 3D

Dessin 3D

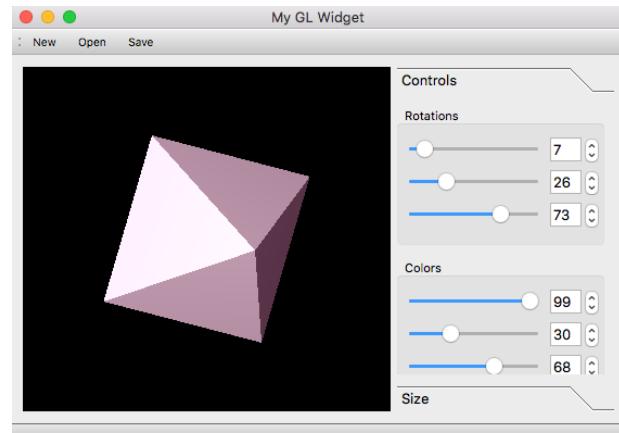
- via **OpenGL**
- principe : créer une sous-classe de **QOpenGLWidget** et redéfinir :
 - `virtual void initializeGL()`
 - `virtual void resizeGL(int w, int h)`
 - `virtual void paintGL()`
- pour demander un **réaffichage** il faut appeler :
 - `update()`
 - (effectué après terminaison d'une fonction de callback)
- Note: **QtGraphicsView** est également compatible avec **OpenGL**

OpenGL : Box3D.h

```
#include <QOpenGLWidget>
#include <QOpenGLFunctions>

class Box3D : public QOpenGLWidget, protected QOpenGLFunctions {
    Q_OBJECT
public:
    Box3D(QWidget * parent);
public slots:
    void setXRot(int angle);
    void setYRot(int angle);
    void setZRot(int angle);
signals:
    void xRotChanged(int angle);
    void yRotChanged(int angle);
    void zRotChanged(int angle);
private:
    void initializeGL();
    void paintGL();
    void resizeGL(int w, int h);
    int xRot, yRot, zRot;
    GLfloat red, green, blue;
```

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif



113

OpenGL : Box3D.cpp

```
#include "Box3D.h"

Box3D::Box3D(QWidget * parent) : QOpenGLWidget(parent) {
    rotX = rotY = rotZ = 0.;
    red = green = blue = 0.;
}

void Box3D::initializeGL() {
    initializeOpenGLFunctions(); // ne pas oublier !
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    GLfloat lightPosition[4] = {0, 0, 10, 1};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
    GLfloat lightColor[4] = {red, green, blue, 1};
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightColor);
}

void Box3D::setXRot(int angle) {
    if (angle != xRot) {
        xRot = angle;
        emit xRotChanged(angle);
        update();
    }
}

void Box3D::setYRot(int angle) {
    if (angle != yRot) {
        yRot = angle;
        emit yRotChanged(angle);
        update();
    }
}

void Box3D::setZRot(int angle) {
    if (angle != zRot) {
        zRot = angle;
        emit zRotChanged(angle);
        update();
    }
}
```

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

114

OpenGL : Box3D.cpp

```
void Box3D::paintGL() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLoadIdentity();
    glTranslatef(0.0, 0.0, -10.0);
    glRotatef(xRot, 1.0, 0.0, 0.0);
    glRotatef(yRot, 0.0, 1.0, 0.0);
    glRotatef(zRot, 0.0, 0.0, 1.0);

    glBegin(GL_QUADS);
    glNormal3f(0,0,-1);
    glVertex3f(-1,-1,0);
    glVertex3f(-1,1,0);
    glVertex3f(1,1,0);
    glVertex3f(1,-1,0);
    glEnd();

    glBegin(GL_TRIANGLES);
    glNormal3f(0,-1,0.707);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(-1,-1,0);
    glColor3f(1.0f, 1.0f, 0.0f);
    glVertex3f(1,-1,0);
    glColor3f(1.0f, 0.0f, 1.0f);
    glVertex3f(0,0,1.2);
    glEnd();
}

***
```

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

```
glBegin(GL_TRIANGLES);
glNormal3f(1,0, 0.707);
glVertex3f(1,-1,0);
glVertex3f(1,1,0);
glVertex3f(0,0,1.2);
glEnd();

glBegin(GL_TRIANGLES);
glNormal3f(0,1,0.707);
glVertex3f(1,1,0);
glVertex3f(-1,1,0);
glVertex3f(0,0,1.2);
glEnd();

glBegin(GL_TRIANGLES);
glNormal3f(-1,0,0.707);
glVertex3f(-1,1,0);
glVertex3f(-1,-1,0);
glVertex3f(0,0,1.2);
glEnd();

void Box3D::resizeGL(int w, int h) {
    int side = qMin(w, h);
    glViewport((w - side) / 2, (h - side) / 2, side, side);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2, +2, -2, +2, 1.0, 15.0);
    glMatrixMode(GL_MODELVIEW);
}
```

115

OpenGL : main et MainWindow

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent)
{
    ...
    Box3D box3d = new Box3D(parent);
    setCentralWidget(box3d);

    createSlider(box3d, SLOT(setXRot(int)));
    createSlider(box3d, SLOT(setYRot(int)));
    createSlider(box3d, SLOT(setZRot(int)));
    ...
}

void MainWindow::createSlider(Box3D * box3d,
    const char * slot)
{
    QSlider * slider = new QSlider(QSlider::Horizontal, this);
    connect(slider, SIGNAL(valueChanged(int)), box3d, slot);
}
```

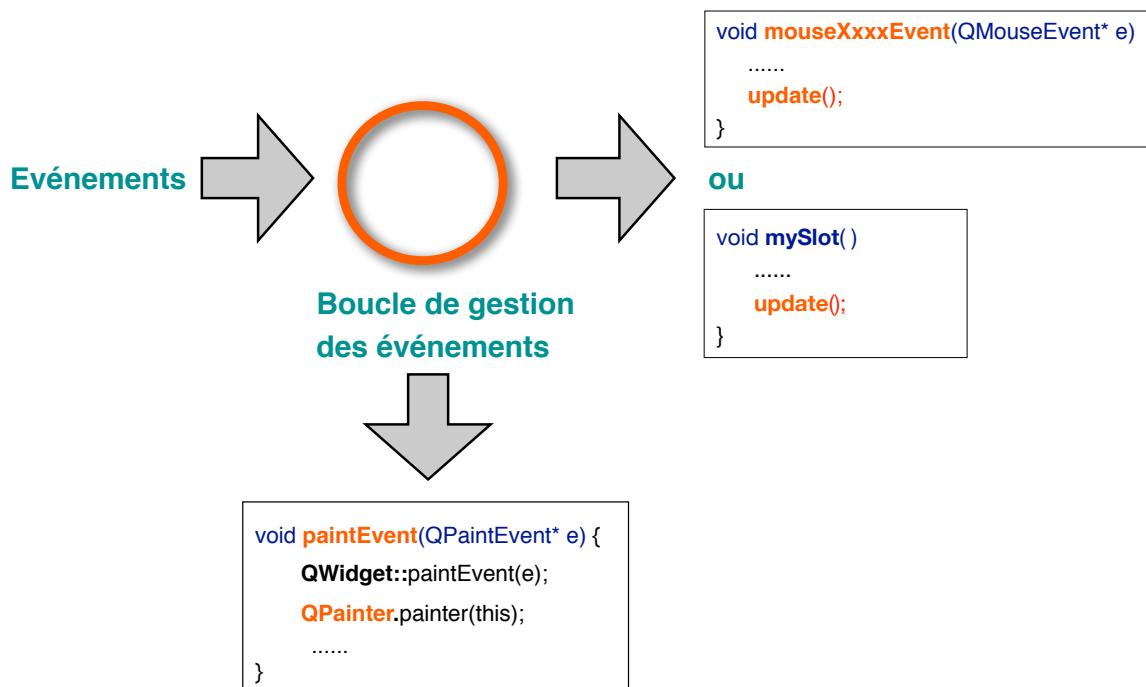
```
int main(int argc, char **argv) {
    QApplication app(argc, argv);

    QSurfaceFormat format;
    format.setDepthBufferSize(24);
    format.setStencilBufferSize(8);
    format.setProfile(QSurfaceFormat::CoreProfile);
    QSurfaceFormat::setDefaultFormat(format);

    MainWindow mainwin;
    mainwin.setWindowTitle("OpenGL Box");
    mainwin.show();
    return app.exec();
}
```

Type d'un slot

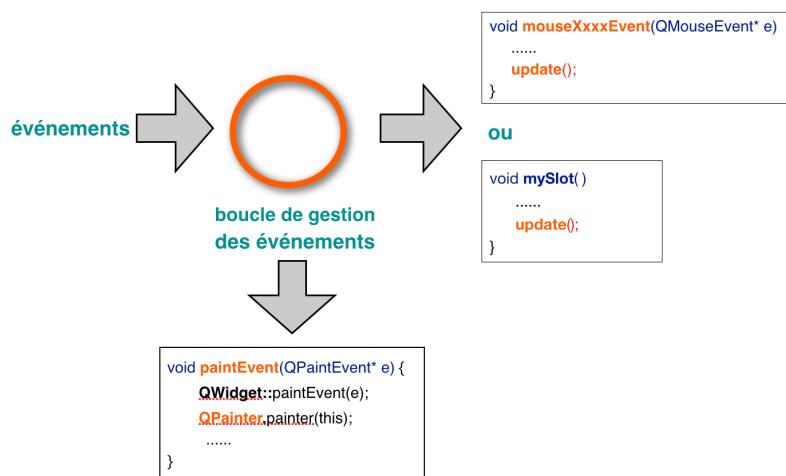
Gestion des événements



Gestion des événements

Boucle de gestion des événements

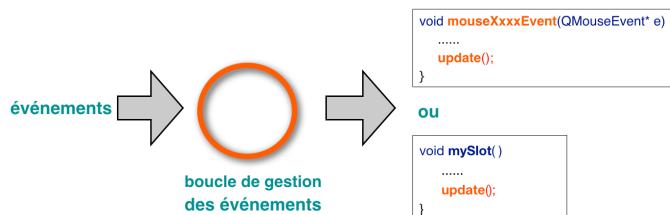
- les événements sont traités **séquentiellement**
- **problème ?**



Gestion des événements

Boucle de gestion des événements

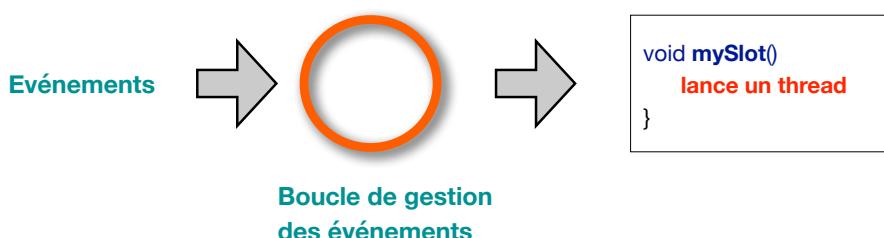
- les événements sont traités **séquentiellement**



Problème

- boucle **bloquée** si un slot **attend longtemps** des données :
 - les événements sont mis en attente dans une **pile**
 - typique avec les applications réseau (sockets, etc.)
 - => **blockage de l'interface** (plus de rafraîchissement)
- solution ?**

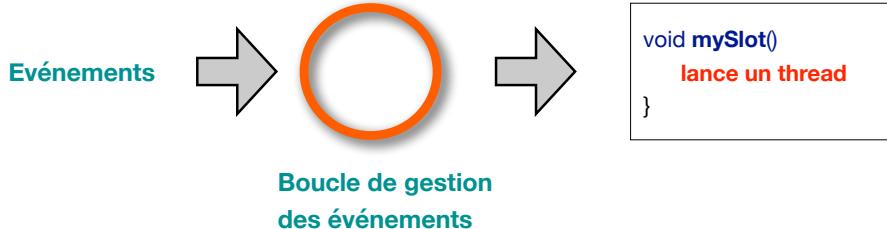
Threads



Solutions

- 1) le slot lance un **thread** et retourne **immédiatement**
 - le **thread** se termine une fois l'action réalisée
- 2) un **thread** tourne en **permanence**
 - et **notify** régulièrement l'application

Threads

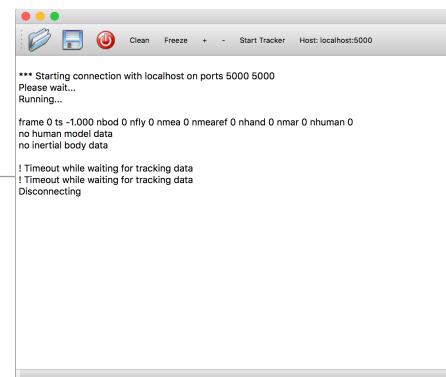


QThread

- on peut s'en servir de différentes façons (ici la plus simple)
- à utiliser avec **prudence** (voir la doc !)
- **principe** : sa méthode **run()** s'exécute dans un **nouveau thread**
 - elle lance **sa propre boucle** de gestion des événements
 - on peut la **redéfinir**

Threads

```
class DTracker : public QThread {  
    Q_OBJECT  
public:  
    DTracker(const std::string& hostname, int port);  
signals:  
    void dataReady(const QString& data);  
private:  
    /// run() is executed in a new thread and gets data from a socket.  
    void run() {  
        bool ok = true;  
        QString data;  
  
        while (ok) {  
            data = getData();  
            emit dataReady(data);  
        }  
    }  
};
```



Pour lancer le traitement

- appeler la méthode **start()**

Threads

Problème

- l'interface graphique « vit » dans le **thread principal**
 - **on doit pas** modifier les widgets depuis un autre thread !

Threads

Problème

- l'interface graphique « vit » dans le **thread principal**
 - **on doit pas** modifier les widgets depuis un autre thread !

Solution : connect()

- dans le thread principal (par exemple dans le constructeur de la **MainWindow**)

```
dtracker = new DTracker(hostname, port);
connect( dtracker, SIGNAL(dataReady(QString)), this, SLOT(showData(QString)) );
```

- **connect()** gère automatiquement une « **queue de connexions** »
 - => le slot **showData()** est bien exécuté dans le **thread principal**

Threads

Alternatives

- **QTimer**
- **QSocketNotifier** etc.
- **QEventLoop::processEvents()**
- **QtConcurrent** framework

Alternative : Timers

```
QTimer * timer = new QTimer();
connect(timer, SIGNAL(timeout()), widget, SLOT(doSomething()));

timer->start(delay)
...
timer->stop();
```

Le signal **timeout()** sera activé **indéfiniment**, chaque fois **après ce délai**

- ce **signal** (comme tous les autres) est compatible avec les **machines à états**
- **setSingleShot(true)** : une seule activation
- **délai nul => slot activé dès que possible**
(= dès que la boucle de gestion des événements est inactive)
- Peut également servir pour les animations

Propriétés et animation

Propriétés

```
s1->assignProperty( label, "text", "In state s1" );
s1->assignProperty( button, "geometry", QRectF(0, 0, 50, 50) );
```

Base des animations

```
s1->assignProperty( button, "geometry", QRectF(0, 0, 50, 50) );
s2->assignProperty( button, "geometry", QRectF(0, 0, 100, 100) );
addTrans( s1, s2, button, SIGNAL(clicked()) )
    ->addAnimation( new QPropertyAnimation( button, "geometry" ) );
```

Voir aussi QML

- langage déclaratif en **JavaScript**
- permet davantage de possibilités

Touch

Initialisation

```
setAttribute(Qt::WA_AcceptTouchEvents);
```

Redéfinir la méthode **event()** de QWidget

```
bool MyWidget::event(QEvent * e) {
    switch (e->type( )) {
        case QEvent::TouchBegin:
        case QEvent::TouchUpdate:
        case QEvent::TouchEnd:
            return myTouchFunc( static_cast<QTouchEvent *>(e) ); // ma méthode pour le touch
        default:
            return QWidget::event(e); // appeler la méthode standard dans les autres cas
    }
}
```

Touch

```
bool MyWidget::event(QEvent * e) {
    switch (e->type( )) {
        case QEvent::TouchBegin:
        case QEvent::TouchUpdate:
        case QEvent::TouchEnd:
            return myTouchFunc( static_cast<QTouchEvent *>(e) );
        default:
            return QWidget::event(e);
    }
}
```

Implémentation

La méthode **myTouchFunc(QTouchEvent * e)** :

- doit renvoyer **true** si l'événement est accepté
- a accès aux **points** (et leurs caractéristiques : ex : la pression si disponible) et leurs **états** (Pressed, Moved, Stationary, Released) via :

```
e->touchPoints()
e->touchPointStates()
```

Gestes

Gestes prédéfinis

```
grabGesture(Qt::PanGesture);
grabGesture(Qt::PinchGesture);
grabGesture(Qt::SwipeGesture);
```

Redéfinir la méthode **event()** de **QWidget**

```
bool MyWidget::event(QEvent *e) {
    switch (e->type( )) {
        case QEvent::Gesture:
            return myGestureFunc(static_cast<QGestureEvent *>(e)); // ma méthode
        default:
            return QWidget::event(e); // traitement standard dans les autres cas
    }
}
```

Gestes

```
bool MyWidget::event(QEvent *e) {
    switch (e->type()) {
        case QEvent::Gesture:
            return myGestureFunc(static_cast<QGestureEvent *>(e));
        default:
            return QWidget::event(e);
    }
}
```

Implémentation

La méthode **myGestureFunc(QGestureEvent * e)**

- **e->gesture()** permet de récupérer les gestes d'un certain type
- exemple :

```
bool MyWidget::gestureEvent(QGestureEvent *e) {
    if ((QGesture* gest = e->gesture(Qt::SwipeGesture)))
        return mySwipeFunc(static_cast<QSwipeGesture*>(gest));
    else return true;
}
```

Gestes

Créer de nouveaux gestes et les reconnaître

C'est possible en sous-classant :

- **QGesture**
- **QGestureRecognizer**

Voir cours et TP suivants

Machines à états et Statecharts

Machines à états finis

Example : rubber banding (repris de Scott Hudson - CMU)

```
Accept the press for endpoint p1;  
P2 = P1;  
Draw line P1-P2;  
Repeat  
    Erase line P1-P2;  
    P2 = current_position();  
    Draw line P1-P2;  
Until release event;  
Act on line input;
```



Quel est le problème ?

Machines à états finis

Rubber banding

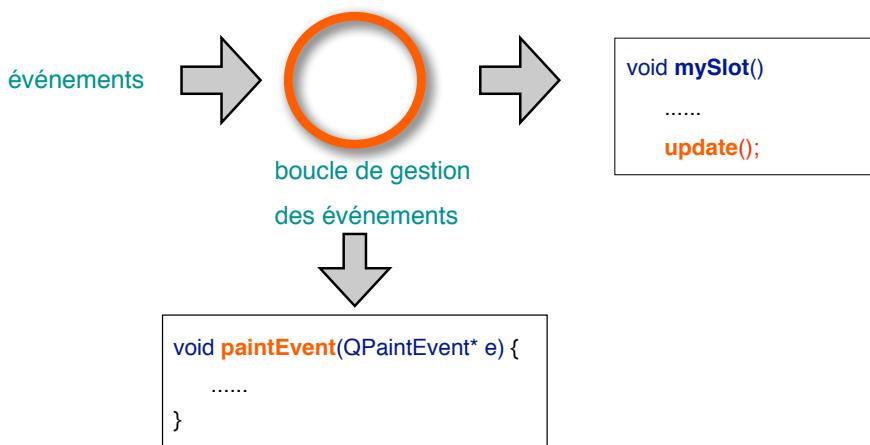
```
Accept the press for endpoint p1;  
P2 = P1;  
Draw line P1-P2;  
Repeat  
    Erase line P1-P2;  
    P2 = current_position();  
    Draw line P1-P2;  
Until release event;  
Act on line input;
```



Problème

- pas compatible avec la **gestion événementielle** !

Rappel : gestion des événements



Cycle événement / réaffichage

- on ne doit pas bloquer ce cycle ni ignorer les (autres) événements

Machines à états

Une solution appropriée pour « maintenir l'état » !

- simple et efficace pour modéliser les comportements
- évite les «spaghettis de callbacks», la multiplication des variables d'état et les erreurs !
- passage facile d'une **représentation visuelle** au **code source**
- divers outils, UML, QState



Etat



Etat de départ



Etat final

NB: en IHM généralement pas d'état final : on revient à l'état initial

Machines à états

Transitions

- Représentées par des **arcs** avec un label : **Événement / Action**
- Si on est dans l'état A et cet événement se produit
 - cette action est effectuée
 - puis on passe dans l'état B



- **Remarque :** les actions sont parfois sur les **états**
 - i.e. quand on entre / sort de l'état
 - au lieu d'être sur les transitions

Machines à états

Retour à notre «bande élastique»

Accept the press for endpoint p1;

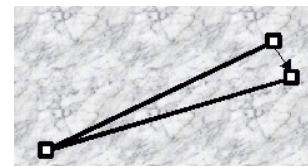
A P2 = P1;
 Draw line P1-P2;

Repeat

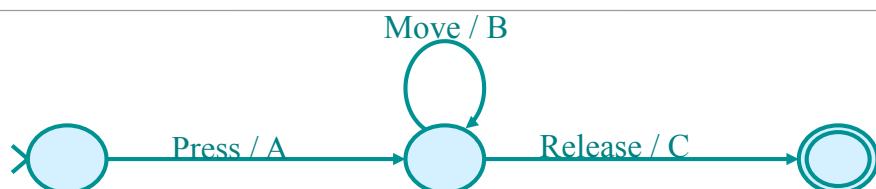
B Erase line P1-P2;
 P2 = current_position();
 Draw line P1-P2;

Until release event;

C Act on line input;



Machines à états



Accept the press for endpoint p1;

A P2 = P1;
 Draw line P1-P2;

Repeat

B Erase line P1-P2;
 P2 = current_position();
 Draw line P1-P2;

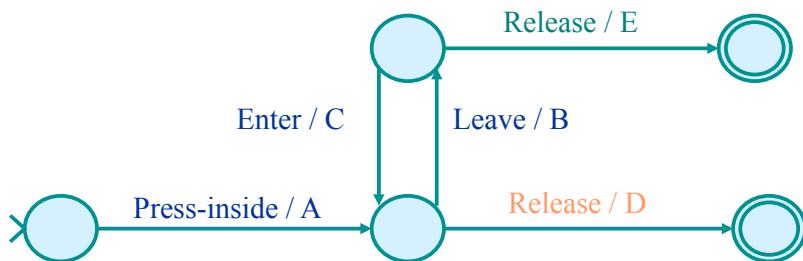
Until release event;

C Act on line input;



Machines à états

Autre exemple : bouton



- A: highlight button
- B: unhighlight button
- C: highlight button
- D: do button action
- E: do nothing

Machines à états

Remarques

- **Événements** : de plus ou moins haut niveau
 - peuvent aussi être des timeouts (cf. **QTimer**)
- **Gardes**
 - condition supplémentaire
 - notation : **prédictat** : événement / action
 - exemple: **button.enabled**: press-inside / A

Machines à états

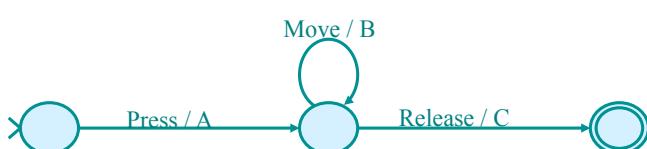
Implémentation

- doit être compatible avec la **boucle de gestion des événements**
- peut être faire « en dur » (cf. exemple page suivante)
- ou via des **tables d'états et de transitions**
- préférer l'utilisation d'**outils existants** (e.g. **QStateMachine** pour **Qt**)

```
state = start_state;  
for ( ; ; ) {  
    raw_evt = wait_for_event();  
    evt = transform_event(raw_evt);  
    state = fsm_transition(state, evt);  
}
```

Machines à états

Implémentation «en dur»



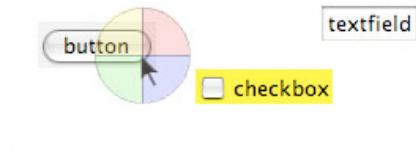
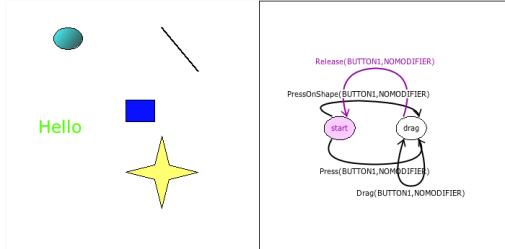
```
state = start_state;  
for ( ; ; ) {  
    raw_evt = wait_for_event();  
    evt = transform_event(raw_evt);  
    state = fsm_transition(state, evt);  
}
```

```
State fsm_transition(state, event) {  
    switch(state) {  
        case 1:  
            switch(event.kind) {  
                case MouseMove:  
                    action_B();  
                    state = 1;  
  
                case MouseRelease:  
                    action_C()  
                    state = 2;  
            }  
            break;  
  
        case 0:  
            switch(event.kind) {  
                case ...  
            }  
    }  
    return state;  
}
```

SwingStates

Implementation Java

- Caroline Appert, LRI
- <http://swingstates.sourceforge.net/>
- classe **Canvas** qui facilite le dessin interactif
- exemples d'interactions avancées



Qt State Machine

Qt : State Machine Framework

- modèle hiérarchique : **Statecharts**
- basés sur **SCXML**
- permet entre autres :
 - **groupes** d'états (hiérarchies)
 - états **parallèles** (pour éviter l'explosion combinatoire)
 - états «**historiques**» (pour sauver et restaurer l'état courant)
 - d'injecter ses **propres** événements
- sert aussi pour les **animations**

Bouton à trois états

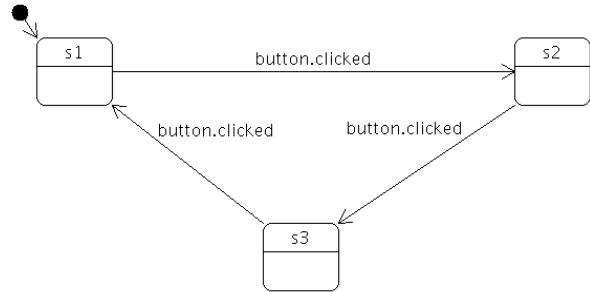
```
// créer la machine à états
QStateMachine * mac = new QStateMachine();

QState *s1 = new QState();
QState *s2 = new QState();
QState *s3 = new QState();
mac->addState(s1);
mac->addState(s2);
mac->addState(s3);

s1->addTransition(button, SIGNAL(clicked()), s2);           // transition de s1 à s2 si on clique le bouton
s2->addTransition(button, SIGNAL(clicked()), s3);           // les actions sont sur les états
s3->addTransition(button, SIGNAL(clicked()), s1);

QObject::connect(s3, SIGNAL(entered()), window, SLOT(showMaximized()));
QObject::connect(s3, SIGNAL(exited()), window, SLOT(showMinimized()));

mac->setInitialState(s1);          // ne pas oublier !
mac->start();                   // lance la machine
```



Hiérarchies

Avantages

- permettent de simplifier les schémas
- les états enfants **héritent des transitions** des états parents

Exemple (page suivante)

- on rajoute à l'exemple précédent un bouton **quitButton** qui ferme l'application
- les états **héritent** de la transition correspondante

Hiérarchies

```
// s11, s12, s13 sont groupés dans s1
QState *s1 = new QState();
mac->addState(s1);
QState *s11 = new QState(s1);
QState *s12 = new QState(s1);
QState *s13 = new QState(s1);
```

```
s11->addTransition(button, SIGNAL(clicked()), s12);
// et les 2 autres transitions...
s1->setInitialState(s11);
```

```
QFinalState *s2 = new QFinalState(); // s2 est un état final
mac->addState(s2);
```

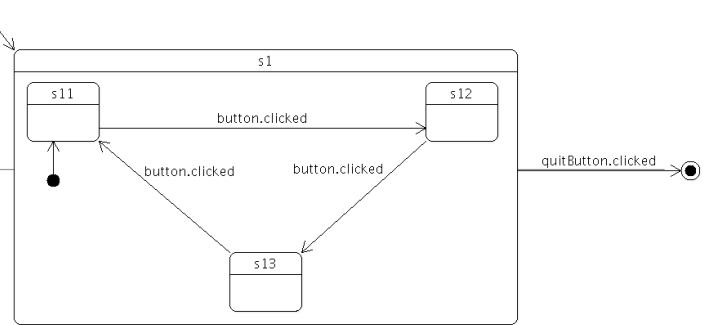
```
s1->addTransition(quitButton, SIGNAL(clicked()), s2);
// les enfants de s1 héritent de la transition s1 -> s2
```

```
// le signal finished() est émis quand on atteint l'état final
connect(mac, SIGNAL(finished()), QApplication::instance(), SLOT(quit()));
```

```
mac->setInitialState(s1);
mac->start();
```

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

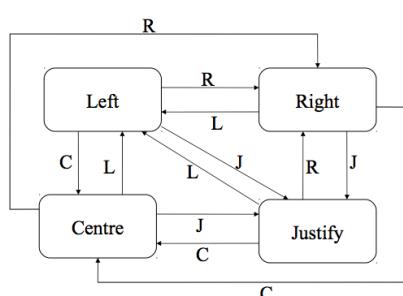
149



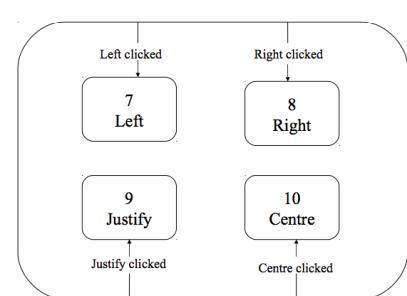
Hiérarchies

Second exemple

- 4 radio boutons (= modes) exclusifs
- héritage => schéma bien plus simple et intuitif !
- noter :
 - les transitions de l'état parent vers les enfants



solution « plate »



solution hiérarchique

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

150

Hiérarchies avec états parallèles

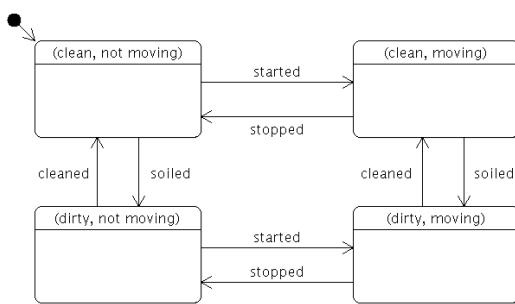
Pour éviter l'explosion combinatoire

- Exemple : **2 boutons à 2 états** (typiquement, 2 **check boxes indépendantes**)

```
QState *s1 = new QState(QState::ParallelStates);
```

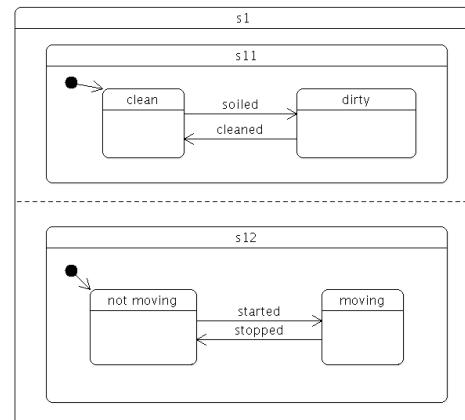
```
// on peut à la fois entrer dans s11 et dans s12
```

```
QState *s11 = new QState(s1);
```



solution « plate »

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif



solution hiérarchique

151

Savoir si un état est actif

Solution 1

- Méthode **active()** de chaque état

```
void MainWindow::setClock(int value) {  
    if (hours_state->active()) {  
        setHours(value);  
    }  
    else if (minutes_state->active()) {  
        setMinutes(value);  
    }  
    else if (seconds_state->active()) {  
        setSecond(value);  
    }  
}
```

Solution 2

- Méthode **configuration()**

```
QSet<QAbstractState*>  
config = machine->configuration();  
  
if (config.contains(hours_state)) {  
    ...  
}  
else if (config.contains(minutes_state)) {  
    ...  
}
```

réglage de l'heure d'une horloge

00:00:00

Eric Lecolinet - Télécom ParisTech - Qt et graphique interactif

152

Types de transitions

Pour les Signaux

- **QSignalTransition** : c'est ce qu'on a utilisé jusqu'à présent via :
`s1->addTransition(button, SIGNAL(clicked()), s2);`

Pour les Événements

- **QEventTranslation**
- **QKeyEventTranslation**
- **QMouseEventTransition**

On peut aussi créer ses propres transitions

- en dérivant **QAbstractTransition**

Transitions.h

Ce header « maison »

- définit des fonctions simplifiées pour les cas courants
- <http://www.telecom-paristech.fr/~elc/qt/Transitions.h>

Signaux

```
QSignalTransition* addTrans( QState* from, QState* to,
                             QObject* sender, const char* signal );
QSignalTransition* addTrans( QState* from, QState* to,
                             QObject* sender, const char* signal,
                             QObject* receiver, const char* slot );
```

- la 2e forme permet d'ajouter une **action** (= un slot)
- l'action est sur la **transition** (pas sur l'état)



Transitions sur les événements

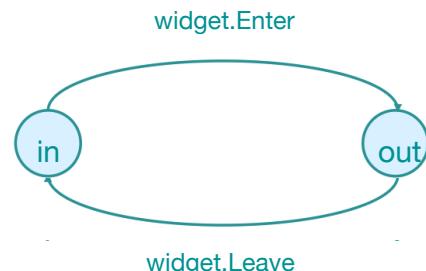
Événements quelconques

```
QEventTransition* addEventTrans( QState* from, QState* to,  
                                QObject* source, QEvent::Type eventType );
```

```
QEventTransition* addEventTrans( QState* from, QState* to,  
                                QObject* source, QEvent::Type eventType,  
                                QObject* receiver, const char* slot );
```

Exemple

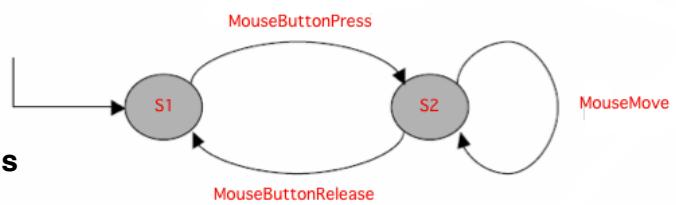
```
addEventTrans( out, in, widget, QEvent::Enter );  
addEventTrans( in, out, widget, QEvent::Leave );
```



Transitions sur les événements

Événements souris

- permet de spécifier les **types** d'événements souris et les **boutons**

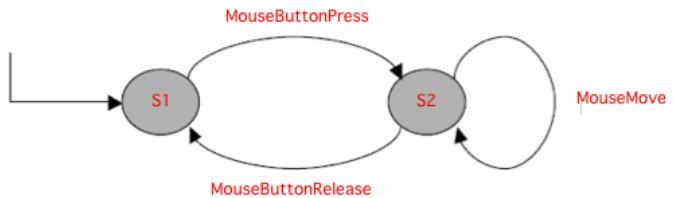


```
// transition de s1 vers s2 quand le bouton gauche de la souris est pressé sur «canvas»  
addMouseTrans( s1, s2,  
                canvas, QEvent::MouseButtonPress, Qt::LeftButton,  
                this, SLOT(newLine( )) );  
  
addMouseTrans( s2, s2,  
                canvas, QEvent::MouseMove, Qt::NoButton, // ATT: NoButton pour MouseMove !  
                this, SLOT(adjustLine( )) );  
  
addMouseTrans( s2, s1,  
                canvas, QEvent::MouseButtonRelease, Qt::LeftButton );
```

Transitions sur les événements

Evénements souris (2)

- plusieurs boutons
- modifieurs clavier



```
// il faut appuyer LeftButton ou RightButton  
addMouseTrans( s1, s2, canvas, QEvent::MouseButtonPress, Qt::LeftButton | Qt::RightButton );  
  
// double clic  
addMouseTrans( s1, s2, canvas, QEvent::MouseButtonDblClick, Qt::LeftButton );  
  
// il faut appuyer SHIFT et LeftButton  
addMouseTrans( s1, s2, canvas, QEvent::MouseButtonPress, Qt::LeftButton )  
    -> setModifierMask( Qt::ShiftModifier );
```

Position du curseur

```
addMouseTrans( s1, s2, canvas, QEvent::MouseButtonPress, Qt::LeftButton,  
    this, SLOT(newLine( ));  
  
addMouseTrans( s2, s2, canvas, QEvent::MouseMove, Qt::NoButton,  
    this, SLOT(adjustLine( ));
```

Problème

Les slots `newLine()` et `adjustLine()` n'ont pas de paramètre `QEvent` :
=> comment peuvent-ils accéder à la **position du curseur** ?

Position du curseur

```
addMouseTrans( s1, s2, canvas, QEvent::MouseButtonPress, Qt::LeftButton,
    this, SLOT(newLine( )) );

addMouseTrans( s2, s2, canvas, QEvent::MouseMove, Qt::NoButton,
    this, SLOT(adjustLine( )) );
```

Solution 1

```
QPoint cursorPos(QWidget* widget) { return widget->mapFromGlobal(QCursor::pos()); }
```

- renvoie la position du curseur **relativement à un widget**
- petit décalage éventuel avec les systèmes **asynchrones** (X11 en réseau)

Solution 2

```
QPoint pos;
addMouseTrans( s2, s2, canvas, QEvent::MouseMove, Qt::NoButton, pos);
• pos contiendra la position du curseur mise à jour à chaque transition
```

Implémentation : filtres de transitions

Principe

- 1) lorsqu'une transition **peut** être activée sa méthode **eventTest()** est appelée
- 2) la transition n'est franchie **que si eventTest()** renvoit **true**

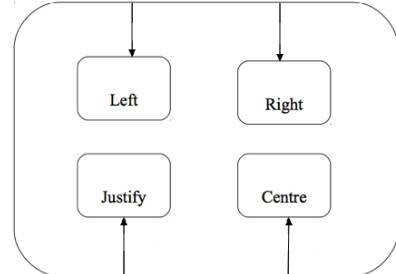
```
class MouseTransition : public QMouseEventTransition {
    QPoint& pos;

    bool eventTest(QEvent * e) {
        if ( ! QMouseEventTransition::eventTest(e)) return false;           // ne pas oublier cette ligne !
        QEvent* realEvent = static_cast<QStateMachine::WrappedEvent*>(e)->event(); // vrai événement
        switch (realEvent->type( )) {
            case QEvent::MouseMove:
            case QEvent::MouseButtonPress:
            case QEvent::MouseButtonRelease:
                pos = static_cast<QMouseEvent*>(realEvent)->pos();      // sauver la position de la souris
            }
        return true;           // true signifie que l'événement déclenche la transition (sinon il est ignoré)
    }
}
```

Activer les transitions programmatiquement

Exemple

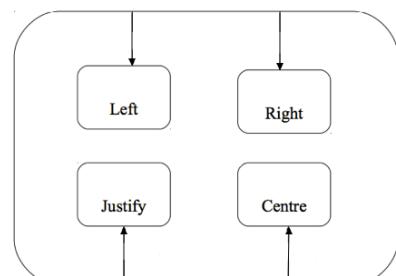
- 4 états correspondant à des modes exclusifs de **MaClasse**
- comment les activer via la fonction ?
`void MaClasse::setMode(int mode)`



Activer les transitions programmatiquement

Exemple

- 4 états correspondant à des modes exclusifs de **MaClasse**
- comment les activer via la fonction ?
`void MaClasse::setMode(int mode)`



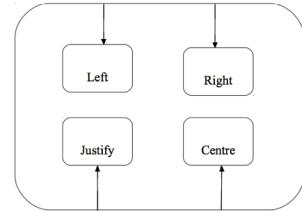
Solutions

- **setMode()** émet des **signaux**
- **setMode()** envoie des « **user events** » (définis par le programmeur)

User Events

```
// parent_state est le parent des autres états
addUserEventTrans( parent_state, left_state, LEFT_MODE );
addUserEventTrans( parent_state, right_state, RIGHT_MODE );
addUserEventTrans( parent_state, justify_state, JUSTIFY_MODE );
addUserEventTrans( parent_state, centre_state, CENTRE_MODE );

void MaClasse::setMode(int mode) {
    mac.postEvent( new QEvent(mode) ); // mac est la machine à états
}
```



setMode() envoie des « user events »

- dérivent de **QEvent** avec un **type** entre **QEvent::User** et **QEvent::MaxUser**
- peuvent être envoyés directement sur la **machine à états**

User Events avec délai temporel

```
void MaClasse::setMode(int mode) {
    mac.postDelayedEvent( new QEvent(mode) delay ); // en millisecondes
}
```

L'événement sera activé **après ce délai**

- peut aussi servir pour les **animations**