Développement d'applications

version. 1.0 – 24/11/2019

Michel GRIMALDI
http://grimaldi.univ-tln.fr
grimaldi@univ-tln.fr

Organisation des TP

Applications Qt

Exercices sur les principales classes de Qt

20 %

Projet personnel

Application Qt de jeu en réseau avec interface graphique.

80 %

Projet

Un jeu de Poker en réseau

Règles : https://www.clubdejeux.com/poker-online/regles

Hypothèse :
Les deux joueurs sont connectés sur un même réseau local
Les deux joueurs utilisent le même programme.

Scenario :

1 - Quand le joueur local lance le programme, il vérifie si l'adversaire est prêt, c'est à dire si un serveur TCP est actif sur sa machine, sur un port prédéfini.

2 - S'il l'adversaire n'est pas prêt, il lance lui-même un serveur et attend qu'il vienne se connecter. On l'appellera alors "joueur/serveur".

3 - Le joueur/serveur se chargera :

- d'initialiser la partie
- de la gestion du jeu de 32 cartes. C'est lui qui distribuera les cartes (5 cartes par joueur), puis le nombre de cartes demandées à la seconde phase du jeu.
- de la gestion des mises. Il attribuera une somme de départ à chaque joueur.
- de la comparaison des deux jeux pour définir qui a gagné le pli.
- de garder un historique horodaté des coups dans un fichier de *log* au format csv.
- de la fin de la partie

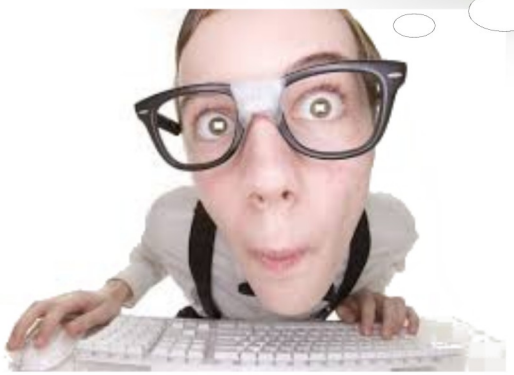
Ok pour Net Poker !

C'est parti !

Dans quel langage
voulez-vous
que je code ?

Geek de service

Quel langage de programmation ?



C'est parti pour Net Poker

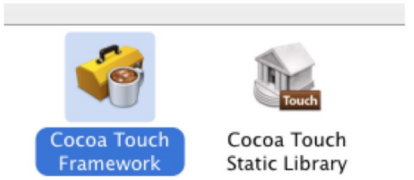
Sous quel OS ?

Avec quel API graphique?


android


Windows


Linux


Cocoa Touch
Framework
Touch
Cocoa Touch
Static Library


Java


ZOPE


ça recommence !


spring
by Pivotal


Qt


Microsoft®
.NET
Framework


Mithril
A Javascript Framework for Building Brilliant Applications


THE
APACHE
SOFTWARE FOUNDATION


JavaScript

C'est parti...

ça fait trois jours
que je n'ai pas dormi,

rien ne marche,
bououou….

Un mois après ...

Cycle de développement classique

client

€ acompte

développeurs

accord client

choix du langage de programmation (c++, java, .*etc*.)

identification des besoins

analyse fonctionnelle

génération automatique du code source

classes, attributs, méthodes, types, relations, critères de validation, .*etc*.

conception UML

implémentation des méthodes

tests

validation

paiement

Marche à suivre

- Concepts saillants
- Diagramme des classes primaire
- Génération d'un code intermédiaire, sans réseau ni graphique
- Concepts secondaires, réseau
- Diagramme des classes
- Génération d'un code intermédiaire, sans graphique
- Concepts secondaires, interface
- Génération d'un code intermédiaire
- tests et mise au point

Quels sont les concepts saillants

Joueur

Carte

Jeu

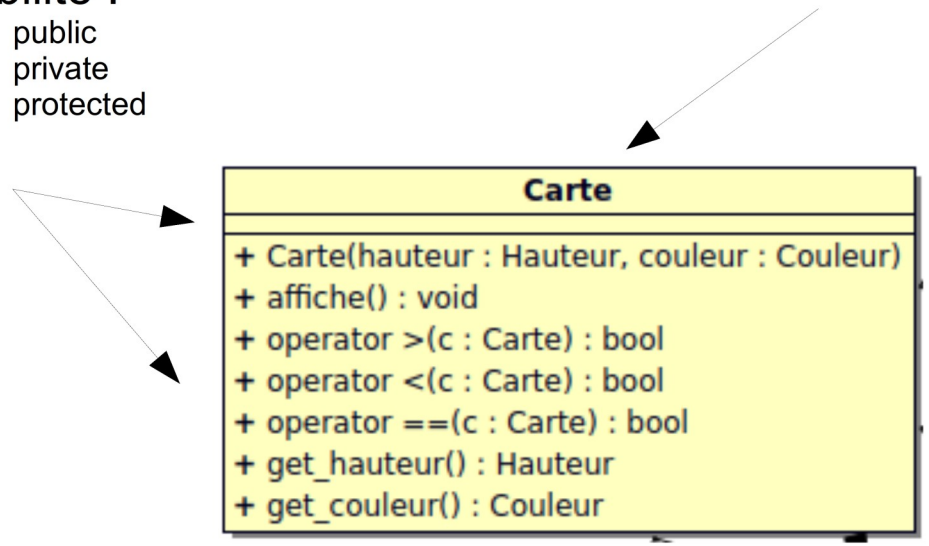
Partie

Main

Représentation graphique unifiée, UML

Visibilité :
+ public
- private
protected

Nom de la classe



attributs

méthodes

Analyse conception OO

Découpage du problème en classes, relations, objets, *.etc*.

Génération automatique du code dans le langage désiré

Génération de la documentation

.etc.

BOUML est une suite d'outils UML 2 gratuits comprenant un modeleur vous permettant de spécifier et générer du code C++, Java, Idl, Php, Python et MySQL.
https://www.bouml.fr

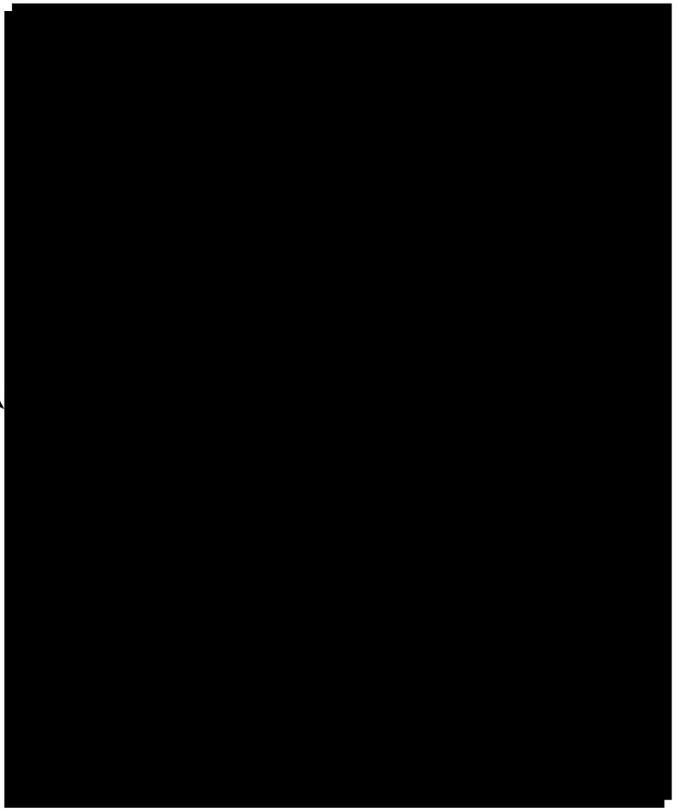
en UML, le type est spécifié après l'identifiant
x : double ~ x est un double

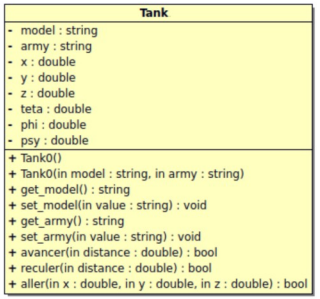
ArgoUML StarUML Modelio Umbrello Visual Paradigm

Unified Modeling Language ~ Indépendance du langage utilisé

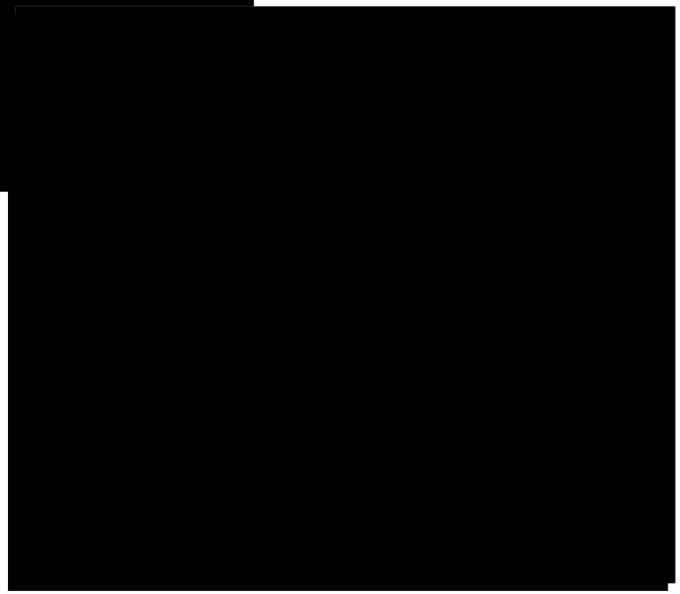
java

c++

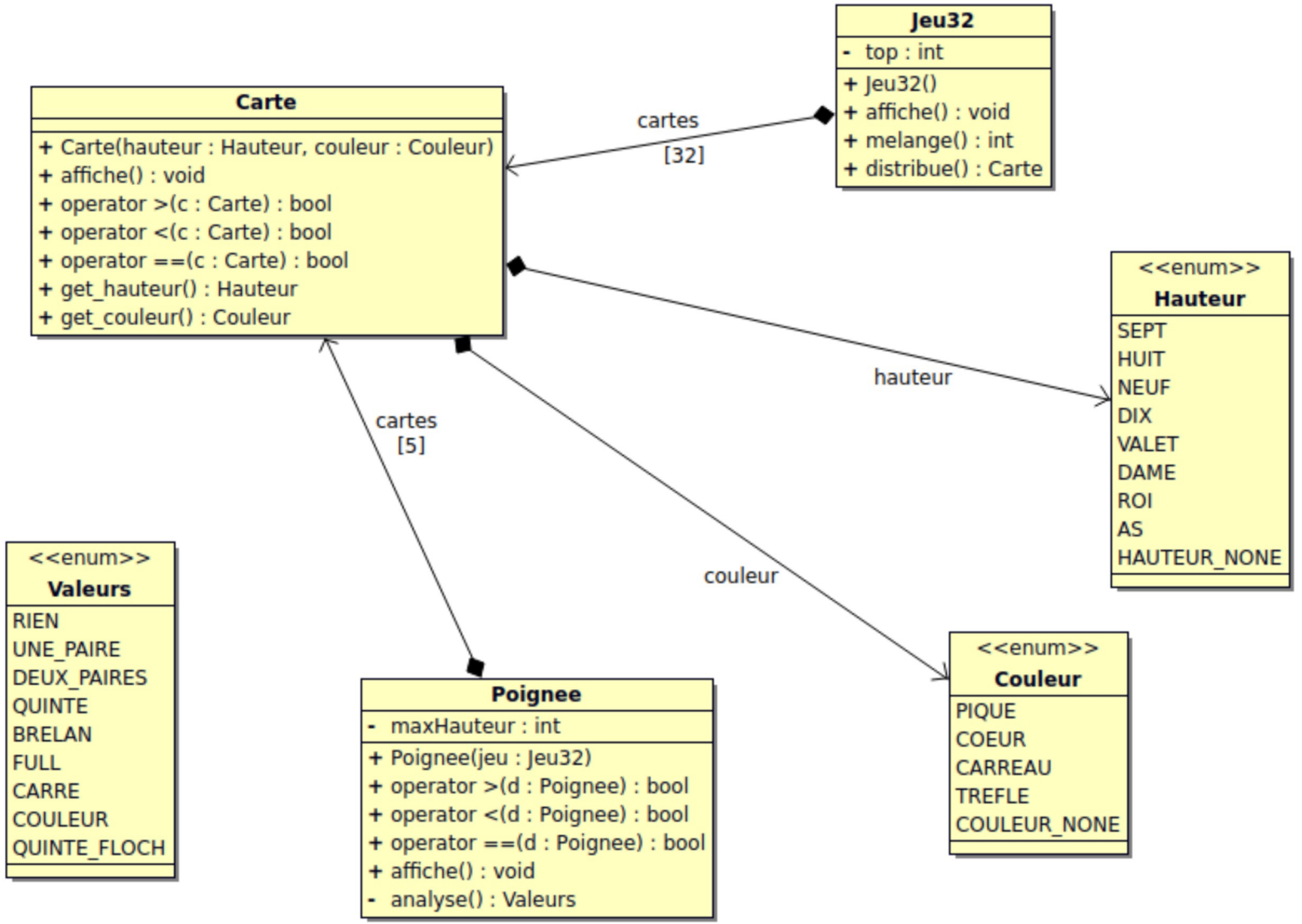




python

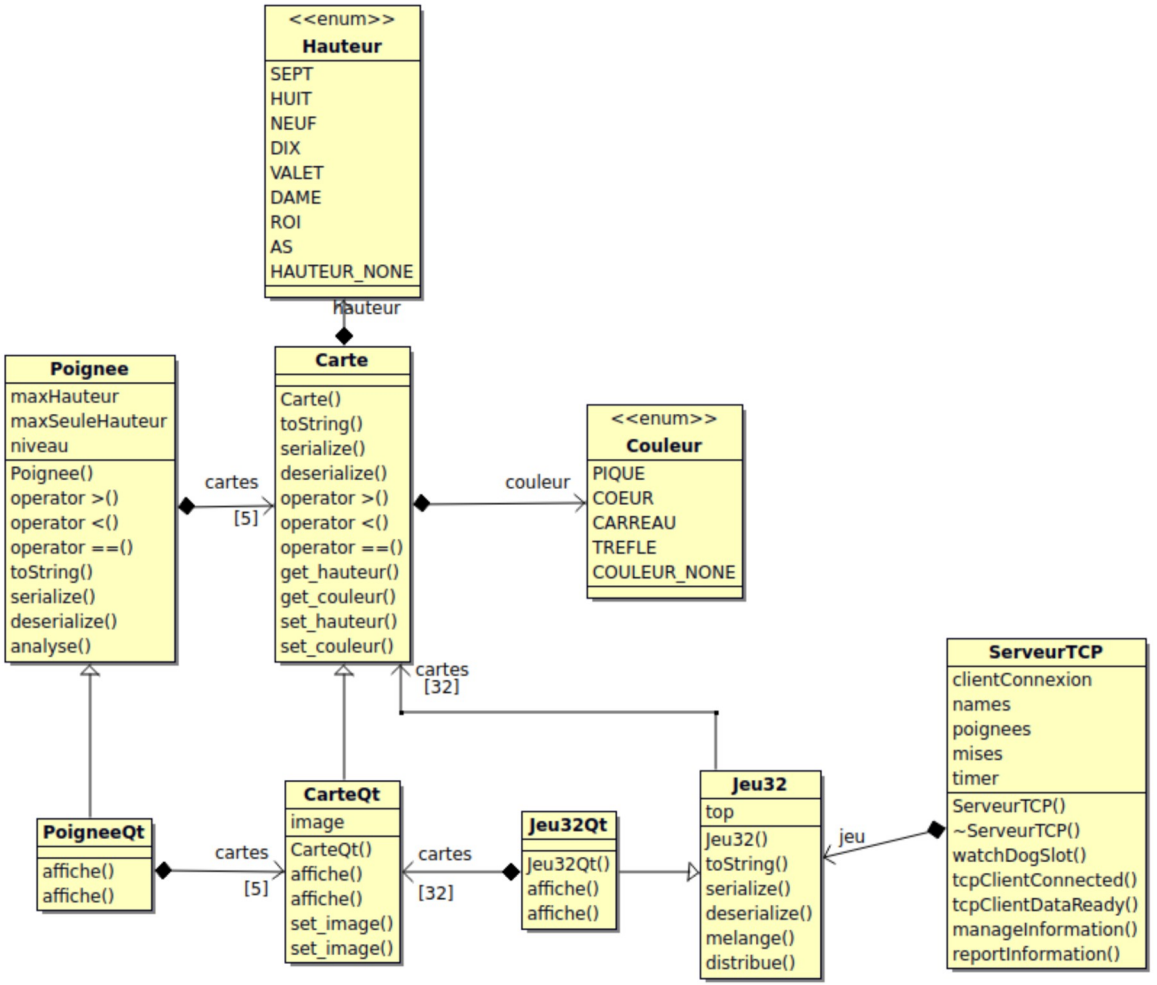


Relations entre classes : composition



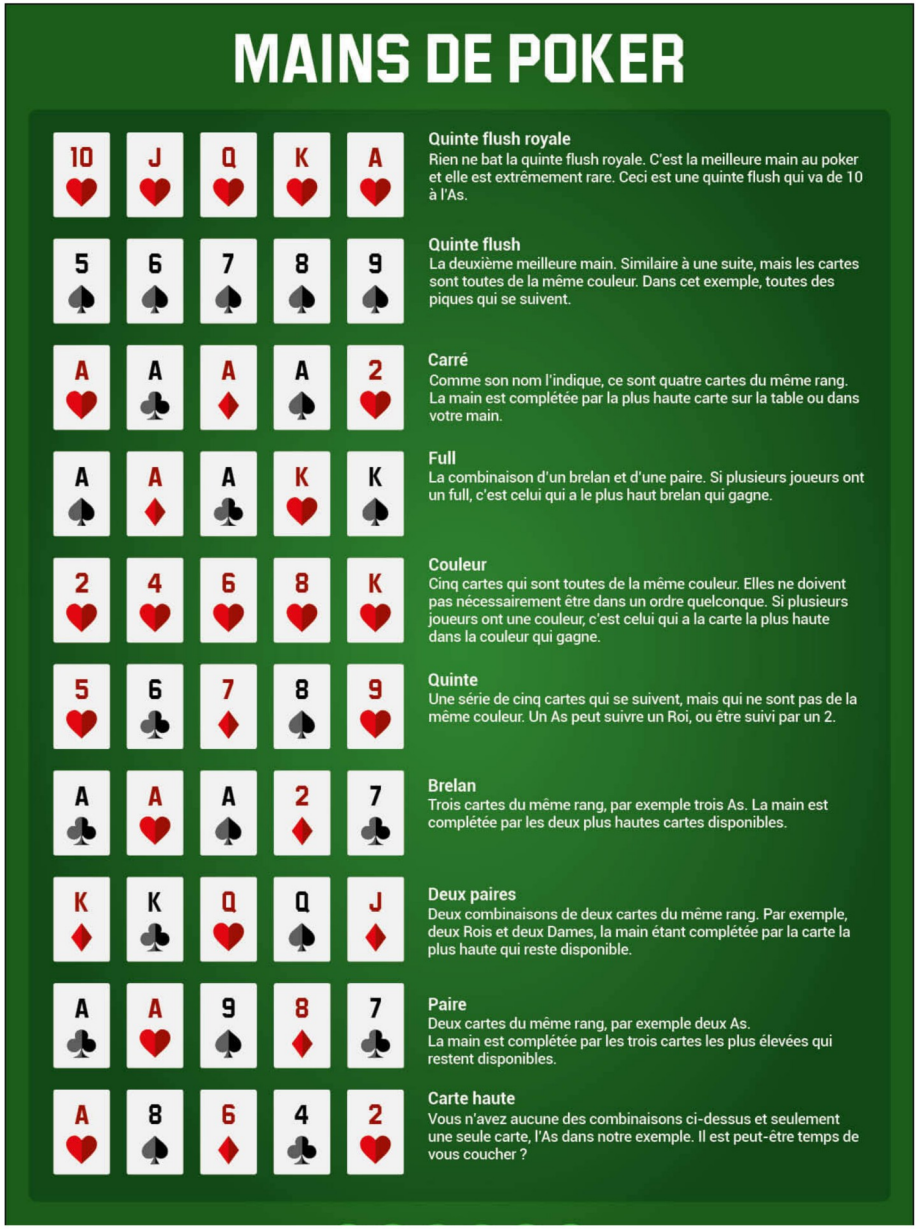
Règle : **Un seul concept par classe**, une classe pour chaque concept !

Relations entre classes : héritage



Quand on peut dire "*est un.e*"!

Valeur d'une main au Poker 32



Le framework Qt

Applications de haut niveau :

- graphiques
- en réseau
- bases de données
- mobiles
- .*etc*.

Le framework Qt

Qt est un **framework** orienté objet et développé en C++ par Qt Development Frameworks, filiale de Nokia.
Elle offre des composants d'interface graphique (**widgets**), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc.
Qt est par certains aspects un framework lorsqu'on l'utilise pour concevoir des i**nterfaces graphiques** ou que l'on architecture son application en utilisant les mécanismes des **signaux et slots.**

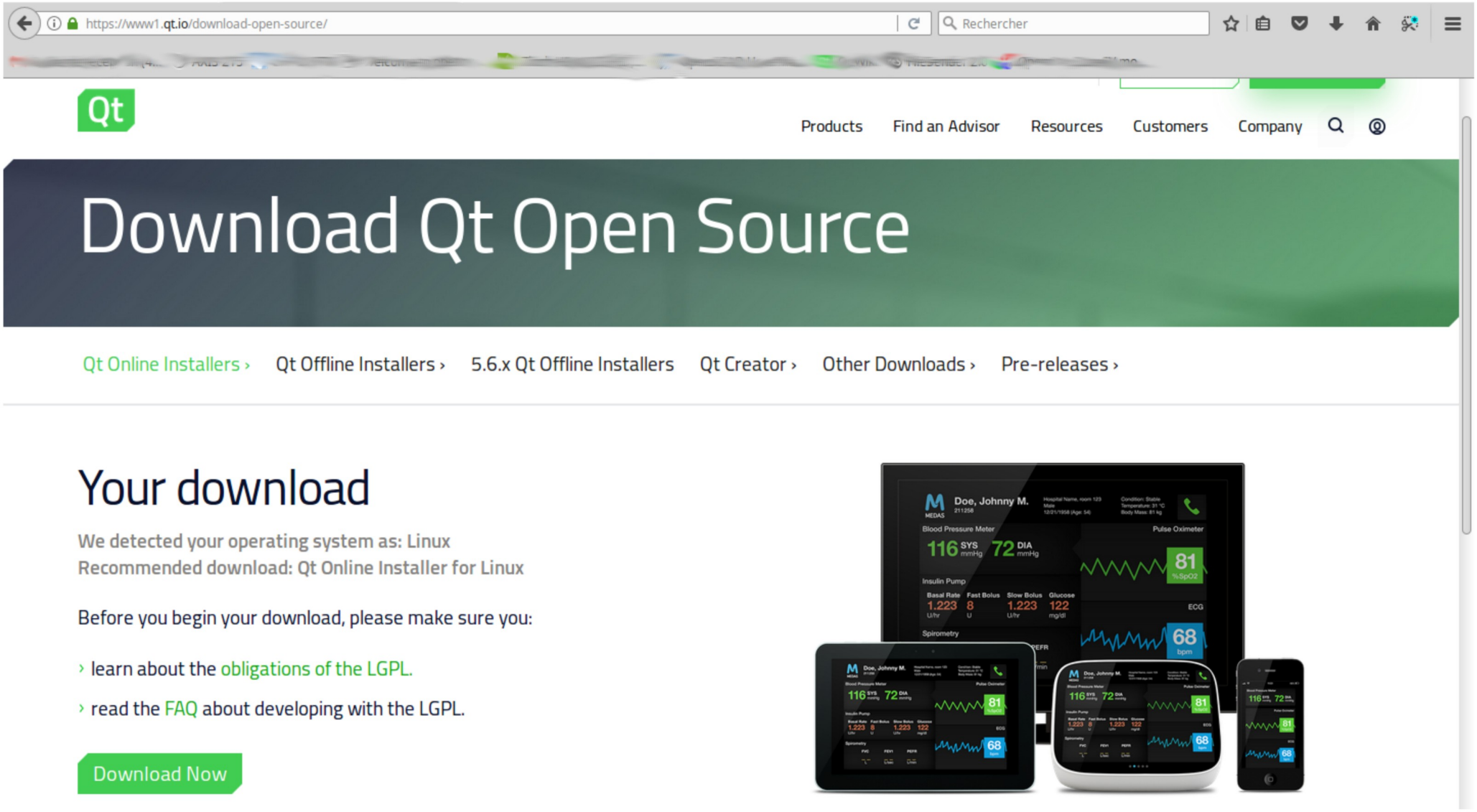
Qt **permet la portabilité des applications** qui n'utilisent que ses composants par simple recompilation du code source. Les environnements supportés sont les **Unix** (dont Linux) qui utilisent le système graphique X Window System, **Windows** et **Mac OS X**.

source : http://fr.wikipedia.org/wiki/Qt

Obtenir le SDK de Qt

https://www1.qt.io/download-open-source/

https://www1.qt.io/download-open-source/#section-2



Environnement de développement : Qt Creator

Qt Creator

Qt Creator est un environnement de développement intégré multiplate-forme faisant partie du framework Qt. Il est donc orienté pour la programmation en C++.

Il intègre directement dans l'interface un **débogueur**, un **outil de création d'interfaces graphiques**, des **outils pour la publication de code sur Git et Mercurial** ainsi que la **documentation Qt.**

L'éditeur de texte intégré permet l'autocomplétion ainsi que la coloration syntaxique.

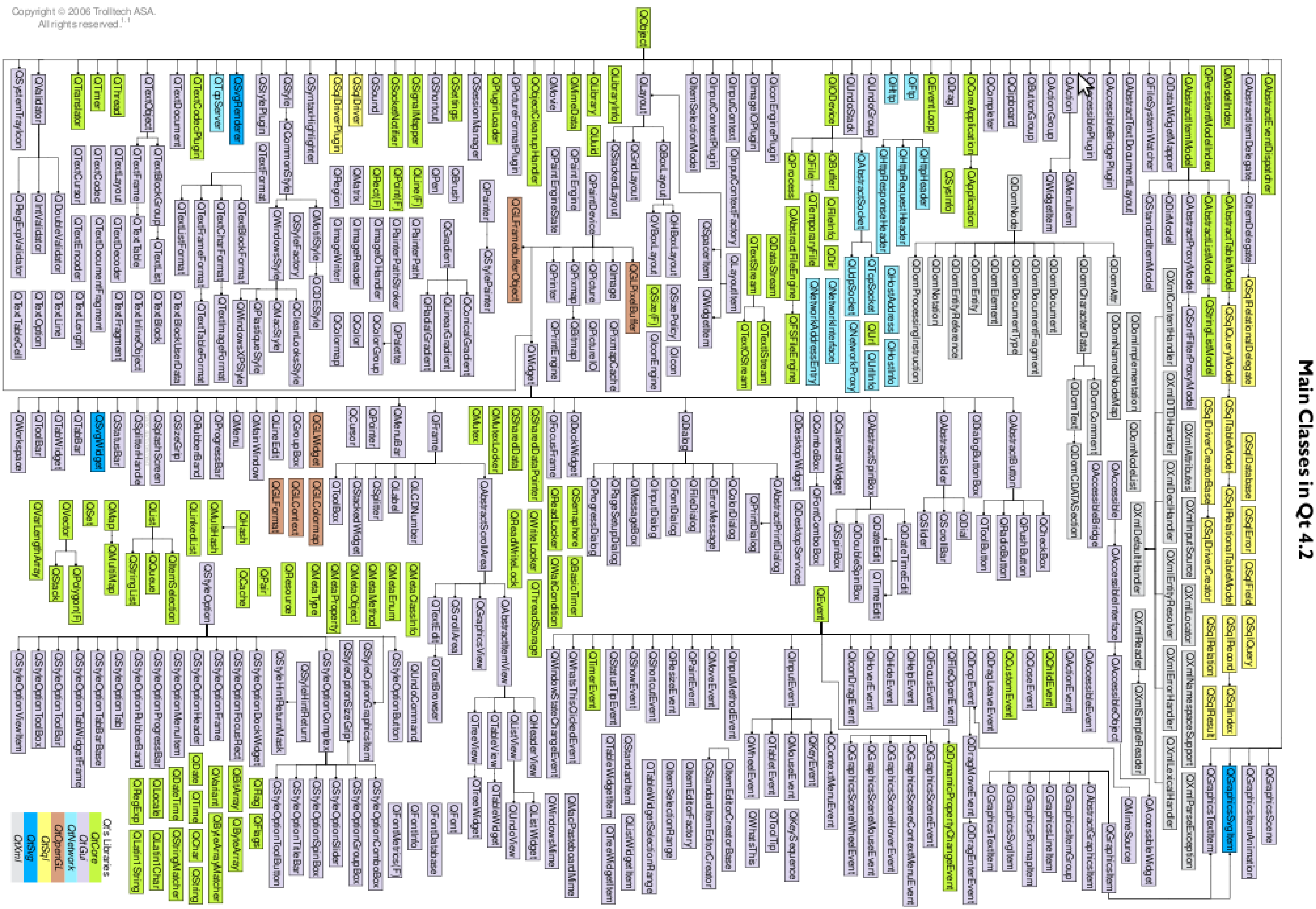
Qt Creator utilise sous Linux le compilateur gcc et MinGW par défaut sous Windows.

Les classes du framework Qt

Le framework Qt est basé sur une hiérarchie de plusieurs milliers de classes toutes héritées d'une classe QObject, et regroupées par module :

Module	Description
Qt Core	Core non-graphical classes used by other modules.
Qt GUI	Base classes for graphical user interface (GUI) components. Includes OpenGL.
Qt Multimedia	Classes for audio, video, radio and camera functionality.
Qt Multimedia Widgets	Widget-based classes for implementing multimedia functionality.
Qt Network	Classes to make network programming easier and more portable.
Qt QML	Classes for QML and JavaScript languages.
Qt Quick	A declarative framework for building highly dynamic applications with custom user interfaces.
Qt Quick Controls	Reusable Qt Quick based UI controls to create classic desktop-style user interfaces.
Qt Quick Dialogs	Types for creating and interacting with system dialogs from a Qt Quick application.
Qt Quick Layouts	Layouts are items that are used to arrange Qt Quick 2 based items in the user interface.
Qt SQL	Classes for database integration using SQL.
Qt Test	Classes for unit testing Qt applications and libraries.
Qt Widgets	Classes to extend Qt GUI with C++ widgets.

http://doc.qt.io/qt-5/classes.html


Main Classes in Qt 4.2
Copyright © 2006 Trolltech ASA. All rights reserved.
Qt's Libraries
QtCore
QtGui
QtNetwork
QtOpenGL
QtSql
QtSvg
QtXml
QObject

Quelques classes de base prises au hasard...

La classe QString

Une chaîne de caractère internationale UNICODE

```
#include "QString.h"

const QString dept = "GEII";
QString message;

message = "Vous avez réussi !";
message = dept;
message = 'W';

QString phrase = "Il était une fois une jolie princesse";
phrase.insert(20, " très");
//phrase contient donc "Il était une fois une très jolie princesse"

phrase.remove(21, 5);
//On revient donc à "Il était une fois une jolie princesse"

phrase.replace(21, 5, "affreuse");
//phrase contient maintenant "Il était une fois une affreuse princesse"

QString suite;
suite = phrase + " qui attendait son prince" ;
//suite contient maintenant "Il était une fois une affreuse princesse qui
attendait son prince"
```

Source : http://sites.univ-provence.fr/wcpp/V2/index.htm

QString : conversions numériques

Conversion d'une chaîne en nombre
les fonctions toShort(), toInt(), toLong(), toFloat() et toDouble()

```
QString chiffres = "123";
int n = chiffres.toInt();          // n contient maintenant 123
chiffres.replace(1, 1, ".");       //chiffres contient maintenant "1.3"
double k = chiffres.toDouble();    //k contient maintenant 1.3
n = chiffres.toInt();              //n contient maintenant 0

bool ok ;
chiffre = "1SS23";
n = chiffres.toInt(&ok);           // on tente la conversion

// ok devient faux si impossible
```

Conversion d'un nombre en chaine
la fonction setNum(x)

```
QString resultat;
double pi = 3.14159 ;
resultat.setNum(pi) ;    //resultat  contient maintenant 3.14159
```

Source : http://sites.univ-provence.fr/wcpp/V2/index.htm

La classe QPoint et QPointF

Un point du plan en précision entière (QPoint) ou réelle (QPointF)

	QPoint()
	QPoint(int *xpos*, int *ypos*)
bool	**isNull**() const
int	**manhattanLength**() const
int &	**rx**()
int &	**ry**()
void	**setX**(int *x*)
void	**setY**(int *y*)
int	**x**() const
int	**y**() const
QPoint &	**operator*=**(float *factor*)
QPoint &	**operator*=**(double *factor*)
QPoint &	**operator*=**(int *factor*)
QPoint &	**operator+=**(const QPoint & *point*)
QPoint &	**operator-=**(const QPoint & *point*)
QPoint &	**operator/=**(qreal *divisor*)

bool	**operator!=**(const QPoint & *p1*, const QPoint & *p2*)
const QPoint	**operator***(const QPoint & *point*, float *factor*)
const QPoint	**operator***(const QPoint & *point*, double *factor*)
const QPoint	**operator***(const QPoint & *point*, int *factor*)
const QPoint	**operator***(float *factor*, const QPoint & *point*)
const QPoint	**operator***(double *factor*, const QPoint & *point*)
const QPoint	**operator***(int *factor*, const QPoint & *point*)
const QPoint	**operator+**(const QPoint & *p1*, const QPoint & *p2*)
const QPoint	**operator+**(const QPoint & *point*)
const QPoint	**operator-**(const QPoint & *p1*, const QPoint & *p2*)
const QPoint	**operator-**(const QPoint & *point*)
const QPoint	**operator/**(const QPoint & *point*, qreal *divisor*)
QDataStream &	**operator<<**(QDataStream & *stream*, const QPoint & *point*)
bool	**operator==**(const QPoint & *p1*, const QPoint & *p2*)
QDataStream &	**operator>>**(QDataStream & *stream*, QPoint & *point*)

```
p1=(0,0)
p2=(100,230)
distance de Mahanattann de p1=330
p1=(1000,2300)
p1=(100,230)
p1=(80,280)
```

```
#include <QPoint>
#include <iostream>
using namespace std;

int main()
{
QPoint p1, p2(100, 230);

cout<<"p1=("<<p1.x()<<','<<p1.y()<<')'<<endl;
cout<<"p2=("<<p2.x()<<','<<p2.y()<<')'<<endl;

cout<<"distance de Mahattann de
p1="<<p2.manhattanLength()<<endl;

p1 = p2*10;
cout<<"p1=("<<p1.x()<<','<<p1.y()<<')'<<endl;

p1 = p1/10;
cout<<"p1=("<<p1.x()<<','<<p1.y()<<')'<<endl;

p1 = p2+QPoint(-20, 50);
cout<<"p1=("<<p1.x()<<','<<p1.y()<<')'<<endl;

return 0;
}
```

getters

Les classes QRect et QSize

Un rectangle du plan, une taille (largeur, hauteur)

	QRect()
	QRect(const QPoint & *topLeft*, const QPoint & *bottomRight*)
	QRect(const QPoint & *topLeft*, const QSize & *size*)
	QRect(int *x*, int *y*, int *width*, int *height*)
void	**adjust**(int *dx1*, int *dy1*, int *dx2*, int *dy2*)
QRect	**adjusted**(int *dx1*, int *dy1*, int *dx2*, int *dy2*) const
int	**bottom**() const
QPoint	**bottomLeft**() const
QPoint	**bottomRight**() const
QPoint	**center**() const
bool	**contains**(const QPoint & *point*, bool *proper* = false) const
bool	**contains**(int *x*, int *y*, bool *proper*) const
bool	**contains**(int *x*, int *y*) const
bool	**contains**(const QRect & *rectangle*, bool *proper* = false) const
void	**getCoords**(int * *x1*, int * *y1*, int * *x2*, int * *y2*) const
void	**getRect**(int * *x*, int * *y*, int * *width*, int * *height*) const
int	**height**() const
QRect	**intersected**(const QRect & *rectangle*) const
bool	**intersects**(const QRect & *rectangle*) const
bool	**isEmpty**() const
bool	**isNull**() const
bool	**isValid**() const
int	**left**() const
void	**moveBottom**(int *y*)
void	**moveBottomLeft**(const QPoint & *position*)
void	**moveBottomRight**(const QPoint & *position*)
void	**moveCenter**(const QPoint & *position*)
void	**moveLeft**(int *x*)
void	**moveRight**(int *x*)
void	**moveTo**(int *x*, int *y*)
void	**moveTo**(const QPoint & *position*)
void	**moveTop**(int *y*)
void	**moveTopLeft**(const QPoint & *position*)
void	**moveTopRight**(const QPoint & *position*)
QRect	**normalized**() const
int	**right**() const
void	**setBottom**(int *y*)
void	**setBottomLeft**(const QPoint & *position*)

```
#include <QPoint>
#include <QRect>
#include <iostream>
using namespace std;
// affichage console de données d'un QRect et d'un QPoint
void affiche_rect(string titre, QRect r){
cout<<titre<<":("<<r.x()<<','<<r.y()<<",largeur="<<r.width()
     <<",hauteur="<<r.height()<<')'<<endl;
}
void affiche_point(string titre, QPoint p){
    cout<<titre<<"=("<<p.x()<<','<<p.y()<<')'<<endl;
    }

int main()
{
QPoint p1(100, 230), p2(200, 300), p3, p4;
QRect r1, r2(p1, p2), r3(p1, QSize(20, 30));

affiche_rect("R1", r1);
affiche_rect("R2", r2);
affiche_rect("R3", r3);

p3=r3.bottomRight();              // point en bas à droite
affiche_point("P3", p3);
p4 = r3.bottomLeft();            // point en bas à gauche
affiche_point("P4", p4);

r3.translate(20, -20);  // translaton du rectangle
affiche_rect("R3", r3);

if (r2.intersects(r3)){
        r1=r2.intersected(r3);    // intersection de rectangles
        affiche_rect("R1", r1);
}
else cout<<"pas d'intersection R2, R3"<<endl;

if (r1.contains(p3))cout<<"P3 dans R1"; else cout<<"P3 n'est pas dans R1";
cout<<endl;
return 0;
}
```

```
R1:(0,0,largeur=0,hauteur=0)
R2:(100,230,largeur=101,hauteur=71)
R3:(100,230,largeur=20,hauteur=30)
P3=(119,259)
P4=(100,259)
R3:(120,210,largeur=20,hauteur=30)
R1:(120,230,largeur=20,hauteur=10)
P3 n'est pas dans R1
```

La classe QDate

Une date sous toutes ses formes

	QDate()
	QDate(int *y*, int *m*, int
QDate	addDays(qint64 *nday*
QDate	addMonths(int *nmon*
QDate	addYears(int *nyears*)
int	day() const
int	dayOfWeek() const
int	dayOfYear() const
int	daysInMonth() const
int	daysInYear() const
qint64	daysTo(const QDate
void	getDate(int * *year*, int
bool	isNull() const
bool	isValid() const
int	month() const
bool	setDate(int *year*, int
qint64	toJulianDay() const
QString	toString(const QStrin
QString	toString(Qt::DateFor
int	weekNumber(int * *ye*
int	year() const
bool	operator!=(const QD
bool	operator<(const QDa
bool	operator<=(const QD
bool	operator==(const QD
bool	operator>(const QDa
bool	operator>=(const QD

```
date ? :31/8/2021
d1: 15.08.2019, 227 ème jour de l'année 2019
d2: 31.08.2021, mardi, dans 747 jours/aujourd'hui'
d3: 01.08.2021 dimanche, 717 jours/aujourd'hui, dimanche, 2021 a 365 jours
```

```
 d2(

QDate::currentDate()

   toString("dd.MM.yyyy")
                                        d1.dayOfYear()


        d2.toString
      QDate::longDayName d2.dayOfWeek
          d1.daysTo

d2.addDays



 d3.year          d3.daysInYear
```

La classe QTime

L'heure sous toutes ses formes

	QTime()
	QTime(int *h*, int *m*, int *s* = 0, int *ms* = 0)
QTime	**addMSecs**(int *ms*) const
QTime	**addSecs**(int *s*) const
int	**elapsed**() const
int	**hour**() const
bool	**isNull**() const
bool	**isValid**() const
int	**minute**() const
int	**msec**() const
int	**msecsSinceStartOfDay**() const
int	**msecsTo**(const QTime & *t*) const
int	**restart**()
int	**second**() const
int	**secsTo**(const QTime & *t*) const
bool	**setHMS**(int *h*, int *m*, int *s*, int *ms* = 0)
void	**start**()
QString	**toString**(const QString & *format*) const
QString	**toString**(Qt::DateFormat *format* = Qt::Te
bool	**operator!=**(const QTime & *t*) const
bool	**operator<**(const QTime & *t*) const
bool	**operator<=**(const QTime & *t*) const
bool	**operator==**(const QTime & *t*) const
bool	**operator>**(const QTime & *t*) const
bool	**operator>=**(const QTime & *t*) const

```
t1: 09:39:27
t2: 15:12:03
t1 est avant t2
il y a 19956 secondes d'écart
```

```
void affiche_time(string titre, QTime t){
cout<<titre<<": "<<t.toString().toUtf8().constData()<<endl;
}

        t2(

    QTime::currentTime

   t1>
        t1==

                t1.secsTo(
```

La classe QTime

L'heure sous toutes ses formes

	QTime()
	QTime(int *h*, int *m*, int *s* = 0, int *ms* = 0)
QTime	**addMSecs**(int *ms*) const
QTime	**addSecs**(int *s*) const
int	**elapsed**() const
int	**hour**() const
bool	**isNull**() const
bool	**isValid**() const
int	**minute**() const
int	**msec**() const
int	**msecsSinceStartOfDay**() const
int	**msecsTo**(const QTime & *t*) const
int	**restart**()
int	**second**() const
int	**secsTo**(const QTime & *t*) const
bool	**setHMS**(int *h*, int *m*, int *s*, int *ms* = 0)
void	**start**()
QString	**toString**(const QString & *format*) cons
QString	**toString**(Qt::DateFormat *format* = Qt::T
bool	**operator!=**(const QTime & *t*) const
bool	**operator<**(const QTime & *t*) const
bool	**operator<=**(const QTime & *t*) const
bool	**operator==**(const QTime & *t*) const
bool	**operator>**(const QTime & *t*) const
bool	**operator>=**(const QTime & *t*) const

```
les boucles seules ont pris 24 ms
la fonction rand toute seule prend 8.6 ns
```

```
void affiche_time(string titre, QTime t){
cout<<titre<<": "<<t.toString().toUtf8().constData()<<endl;
}
```

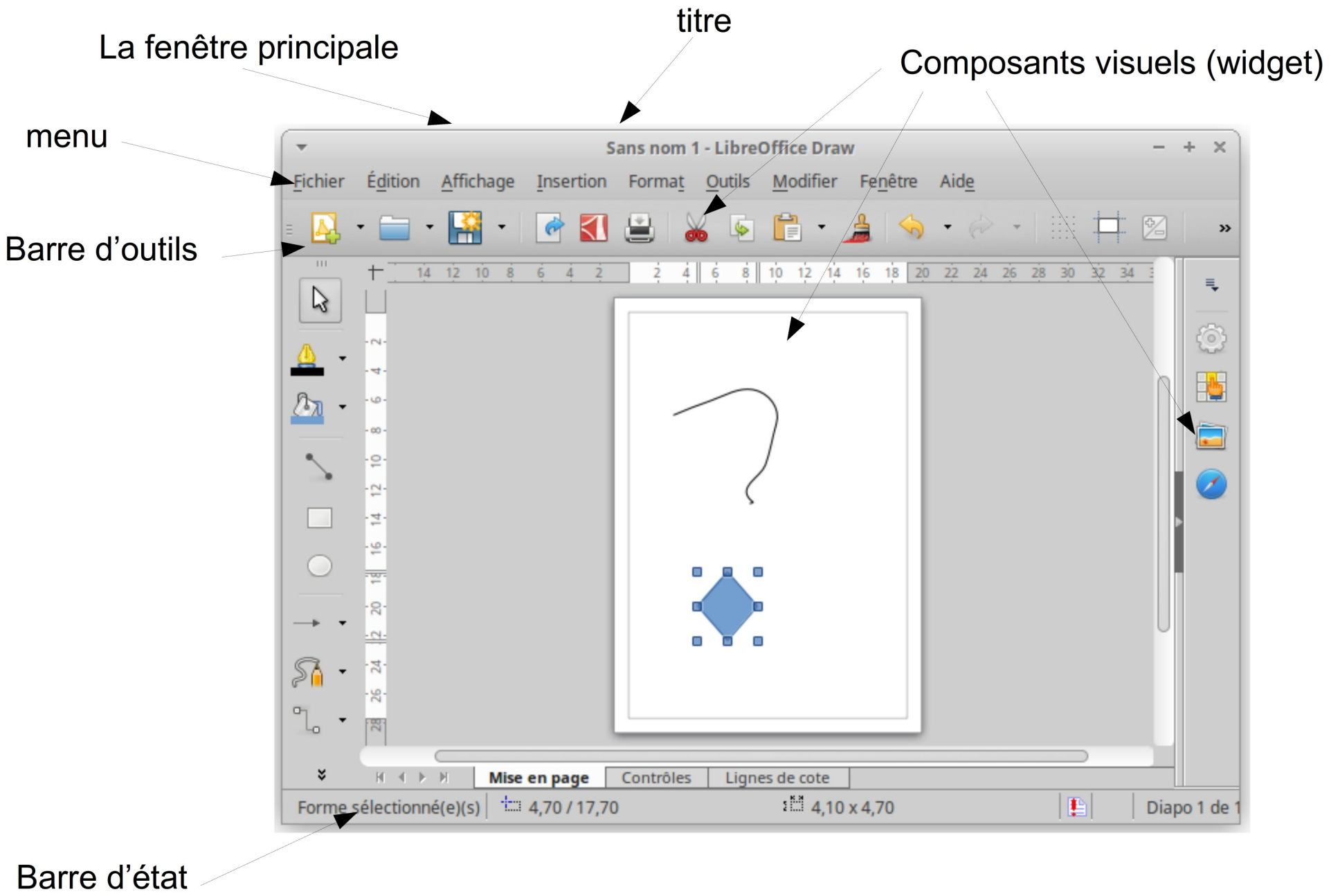
```
t3.start(
```

```
    t3.restart
```

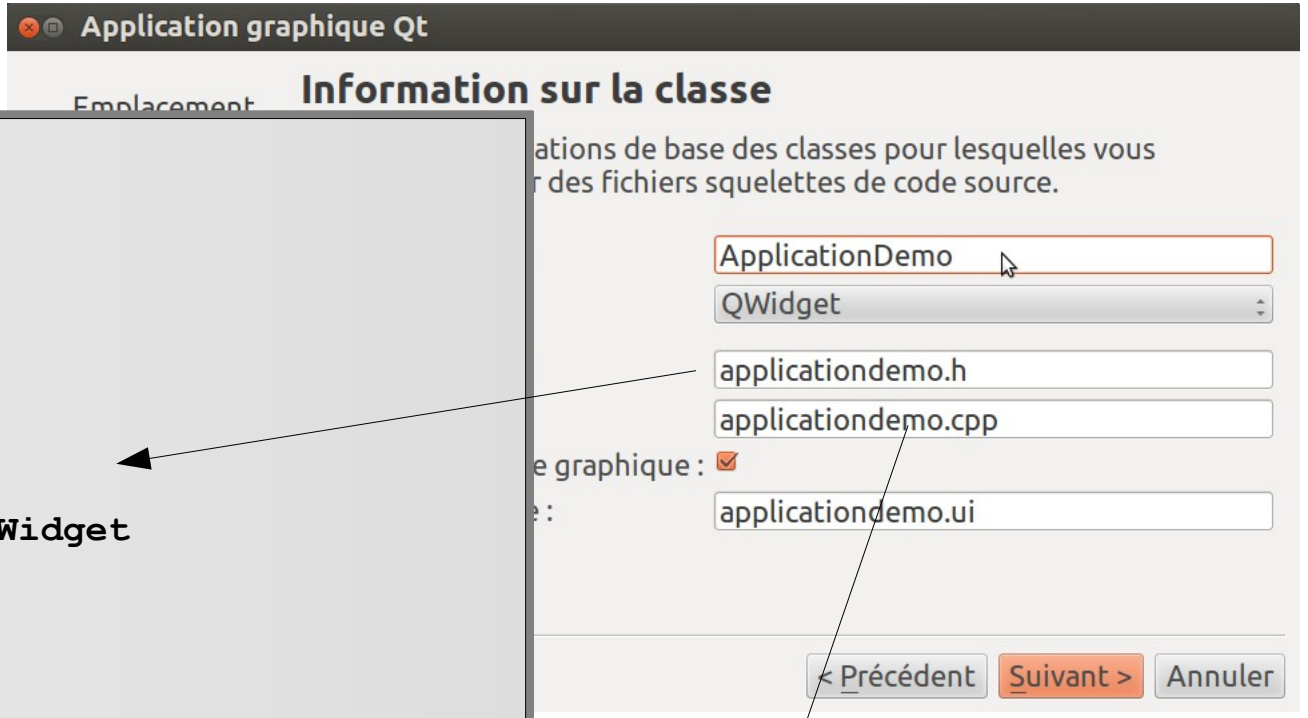
```
    t3.elapsed
```

Application graphique Qt

Application graphique : exemple



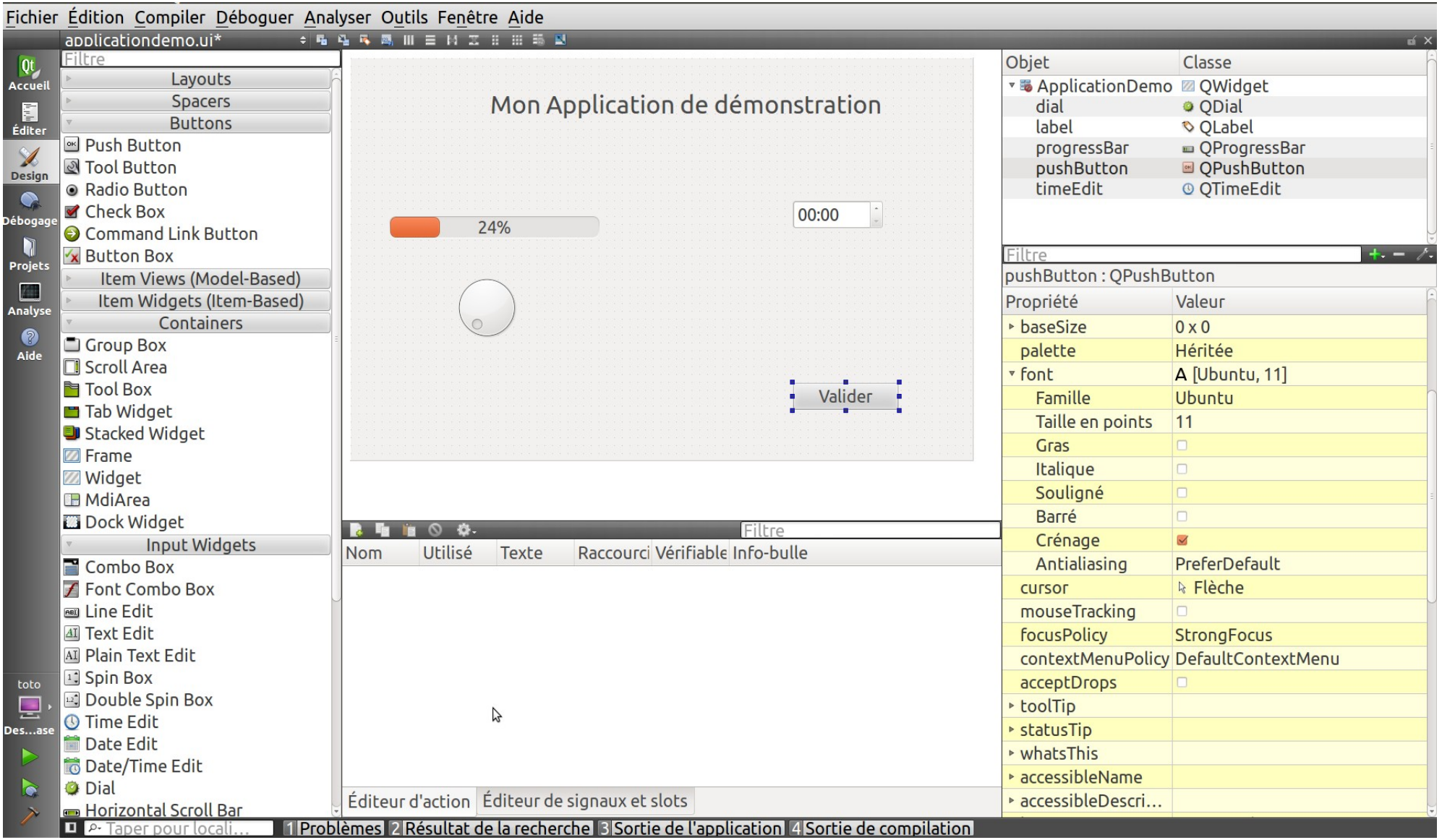
Création d'une application graphique Qt




```
#ifndef APPLICATIONDEMO_H
#define APPLICATIONDEMO_H

#include <QWidget>

namespace Ui {
class ApplicationDemo;
}

class ApplicationDemo : public QWidget
{
    Q_OBJECT

public:
    explicit ApplicationDemo(QWidget *parent = 0);
    ~ApplicationDemo();

private:
    Ui::ApplicationDemo *ui;
};

#endif // APPLICATIONDEMO_H
```

```
#include "applicationdemo.h"
#include "ui_applicationdemo.h"

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

ApplicationDemo::~ApplicationDemo()
{
    delete ui;
}
```

L'interface utilisateur (*user interface*)



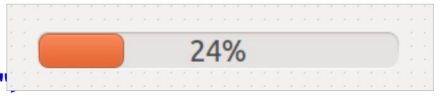
Qt Designer : dessin de l'interface (fichier : applicationdemo.ui)

Le fichier de design de l'UI

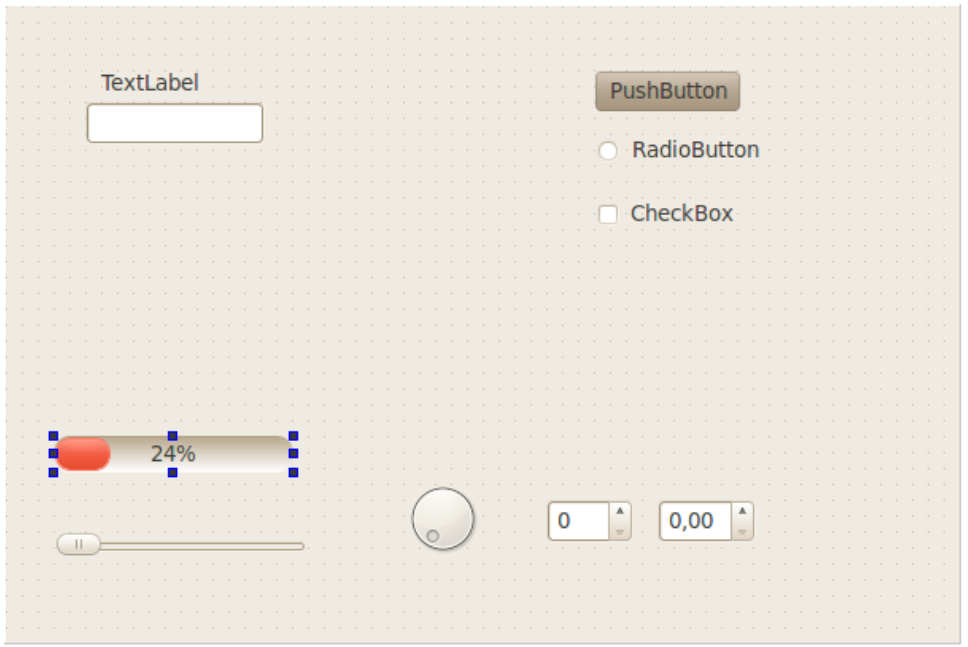
```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>ApplicationDemo</class>
 <widget class="QWidget" name="ApplicationDemo">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>803</width>
    <height>509</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>ApplicationDemo</string>
  </property>
  <widget class="QLabel" name="label">
   <property name="geometry">
    <rect>
     <x>180</x>
     <y>40</y>
     <width>505</width>
     <height>37</height>
    </rect>
   </property>
   <property name="font">
    <font>
     <pointsize>16</pointsize>
    </font>
   </property>
   <property name="text">
    <string>Mon Application de démonstration</string>
```

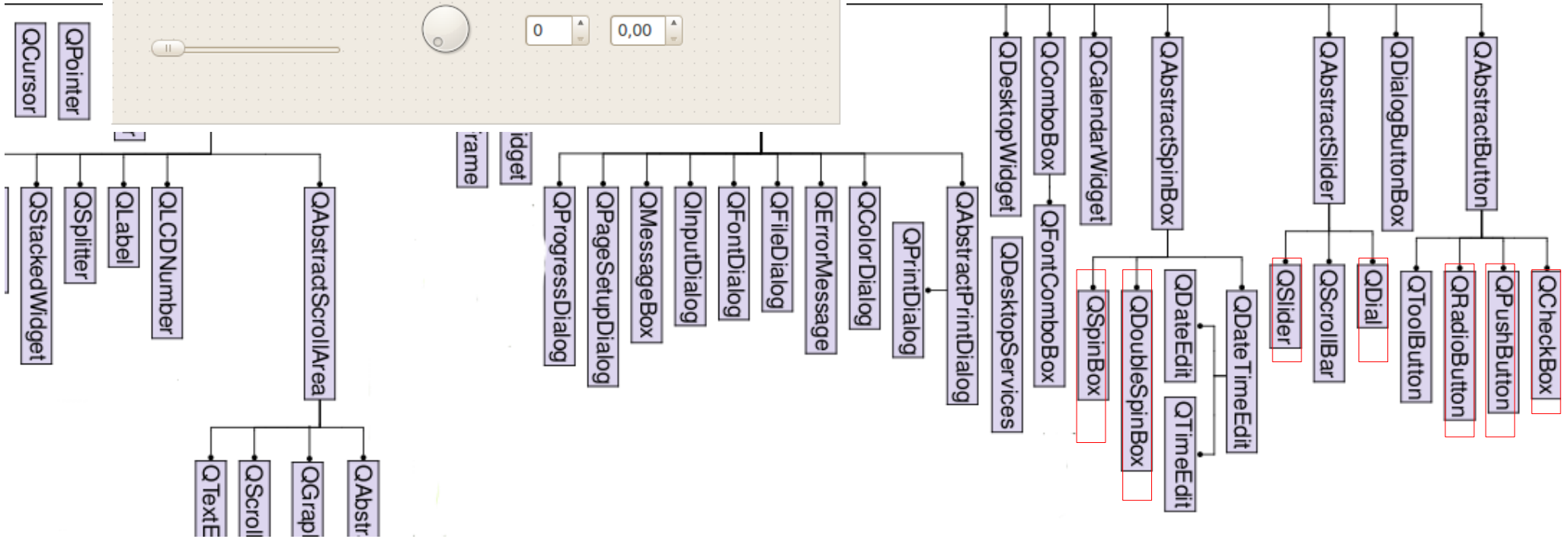
Mon Application de démonstration

```
<?xml version="1.0" encoding="UTF-8"?
   </property>
  </widget>
  <widget class="QDial" name="dial">
   <property name="geometry">
    <rect>
     <x>130</x>
     <y>270</y>
     <width>91</width>
     <height>91</height>
    </rect>
   </property>
  </widget>
  <widget class="QProgressBar" name="progressBar">
   <property name="geometry">
    <rect>
     <x>50</x>
     <y>200</y>
     <width>271</width>
     <height>28</height>
    </rect>
   </property>
   <property name="value">
    <number>24</number>
   </property>
  </widget>
  <widget class="QTimeEdit" name="timeEdit">
   <property name="geometry">
    <rect . . . . . .
…....
```



Quelques classes remarquables...

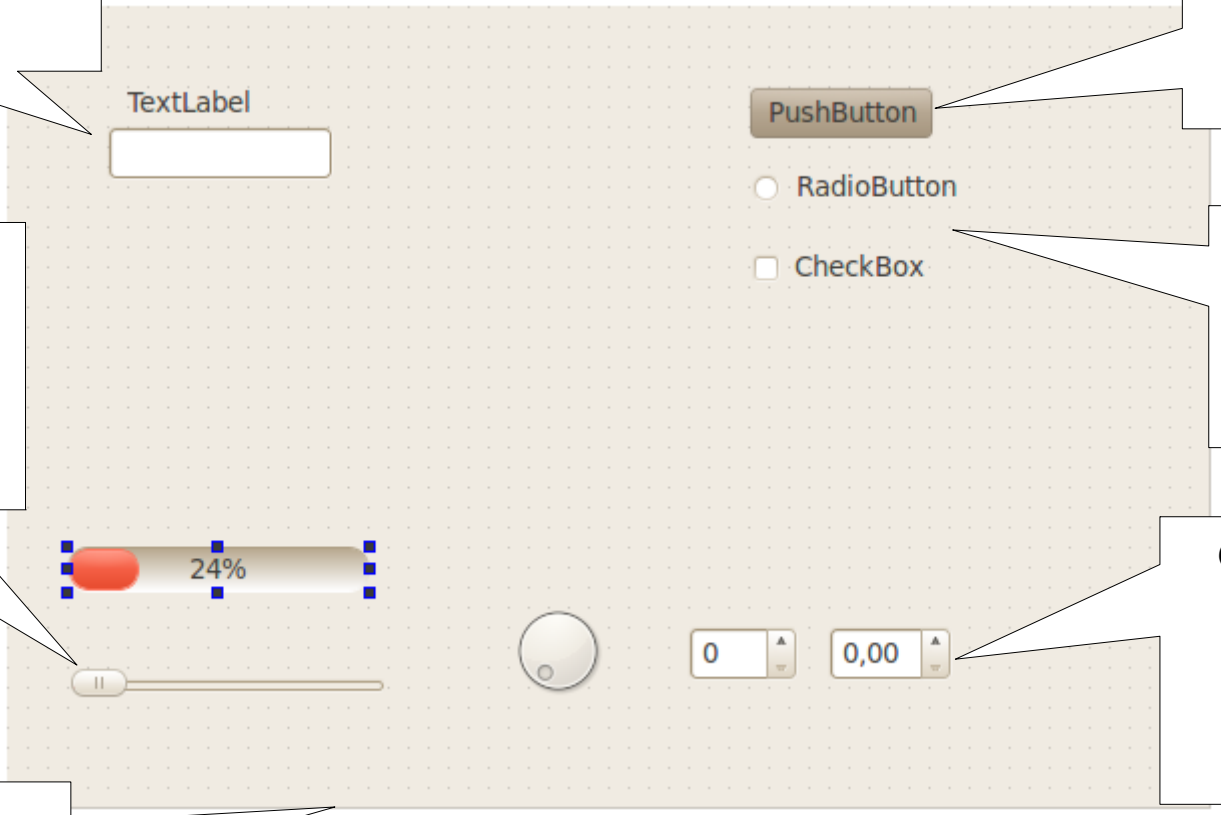




Événements signal/slots

On m’a modifié mon contenu
On a fait « entrée » sur moi
On m’a détruit
.etc.

On m’a appuyé dessus
On m’a relâché
On m’a double-cliqué
On m’a détruit
.etc.



On a modifié ma valeur
On m’a déplacé
On m’a cliqué
On m’a relâché
On m’a détruit
.etc.

On m’a appuyé dessus
On m’a relâché
On m’a double-cliqué
On m’a détruit
.etc.

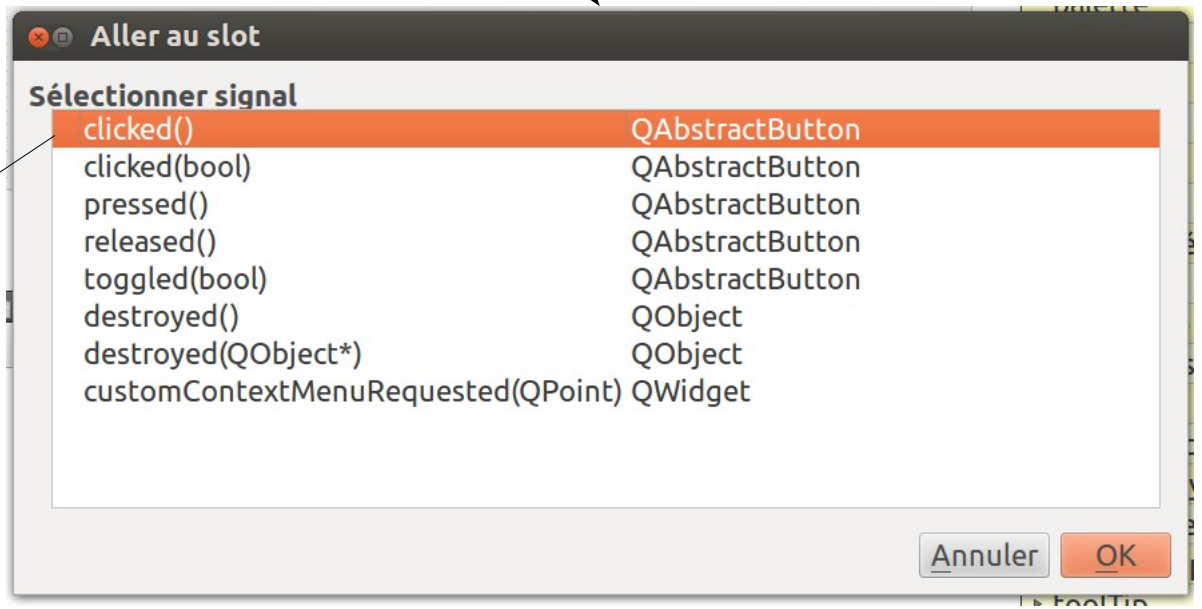
On a modifié ma valeur
On m’a déplacé
On m’a cliqué
On m’a relâché
On m’a détruit
.etc.

On m’a déplacé
On m’a retaillé
On veut me fermer
On m’a détruit
.etc.

Chaque composant Qt (représenté par une classe) peut émettre des **signaux,** correspondant à des actions, au noyau de l’application
On peut connecter un de ces signaux à une méthode d'une classe quelconque (**slot**)

connexion d'un signal à un slot dans le designer

Click droit



On choisit le signal que l'on veut gérer,

Fichier d'entête : une nouvelle méthode est crée

```
private slots:
    void on_toolButton_clicked();
```

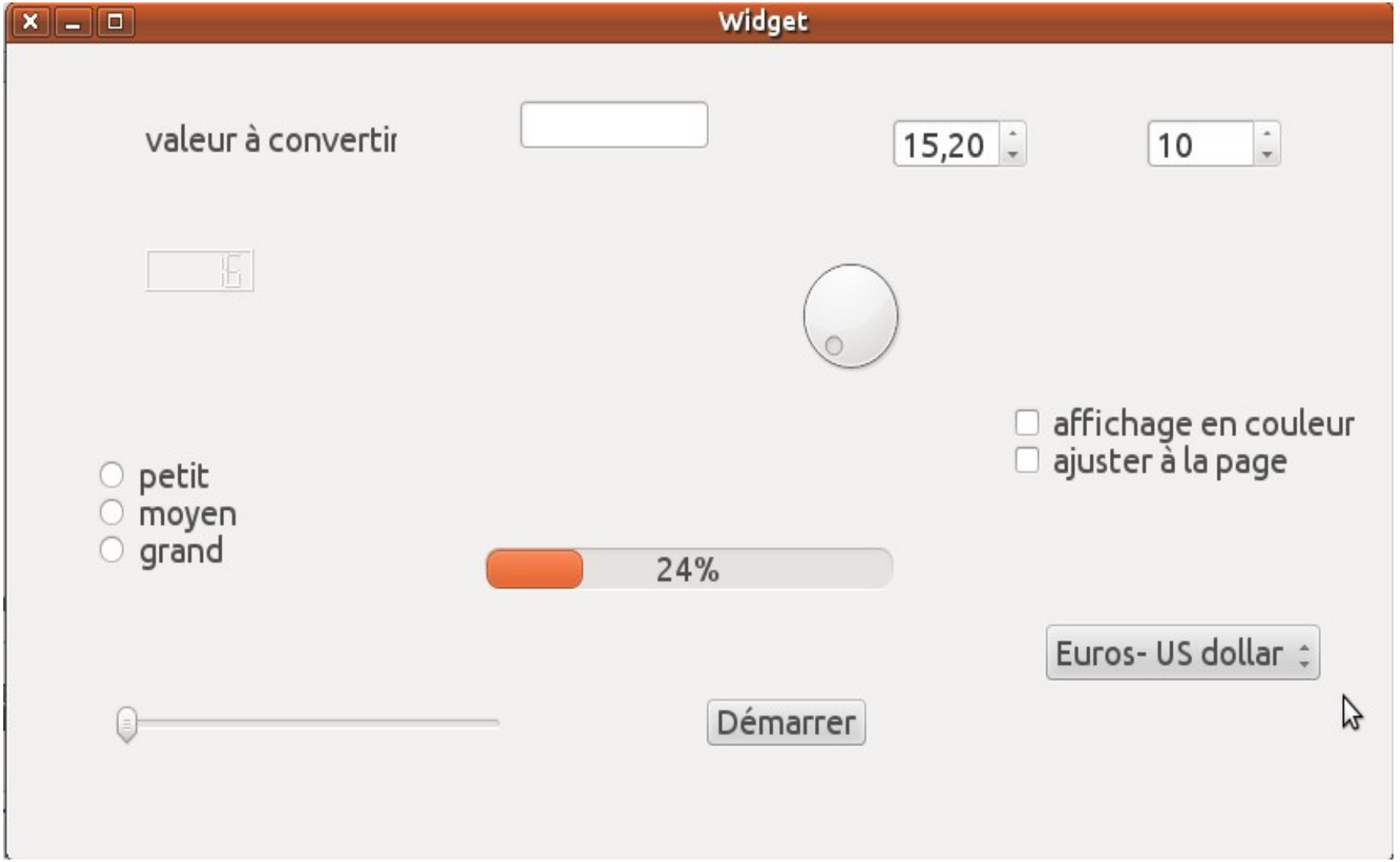
Fichier source : implémentation de la méthode

```
void ApplicationDemo::on_toolButton_clicked()
{

// Écrire ici le code qui se déroulera lorsqu'on
// aura clické sur le bouton « Valider »
}
```

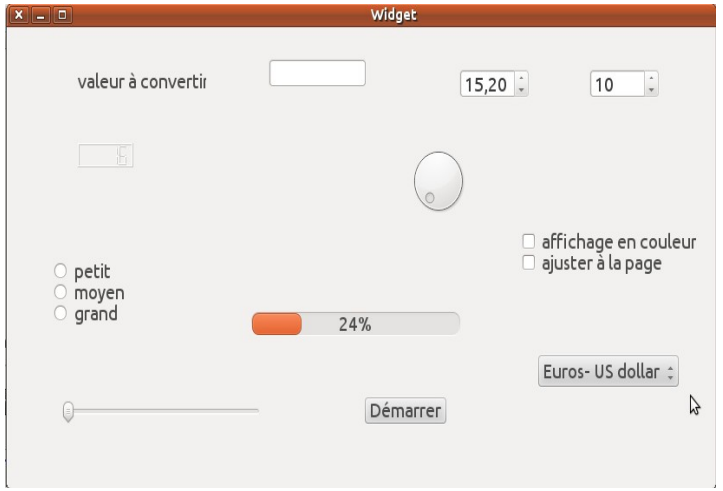
On pourrait également le faire par programme :

`connect (objet_ émetteur, signal, objet_récepteur, slot)`

Actions courantes sur les composants de base




```
this->ui->composant->methode(arg1, arg2, ...);
```



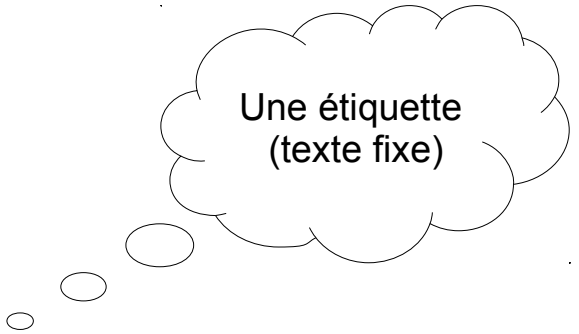
QWidget

La fenêtre principale de l'application

```
this->setVisible(true);                 // rend visible
this->setVisible(false);                // rend invisible
…
QFont serifFont("Times", 10, QFont::Bold);
this->setFont(serifFont);
…

this->setCursor(Qt::WaitCursor);        // curseur sablier
…
    // la taille et la position par rapport au coin haut gauche de l'écran
QRect r;                                // un rectangle
r = this->geometry();
r.setWidth(r.width()+30);               // augmente la largeur de 30 pixels
this->setGeometry(r);
…
this->setWindowTitle("Essai - version 1.00");
…
this->close();   // ferme le widget
…
```

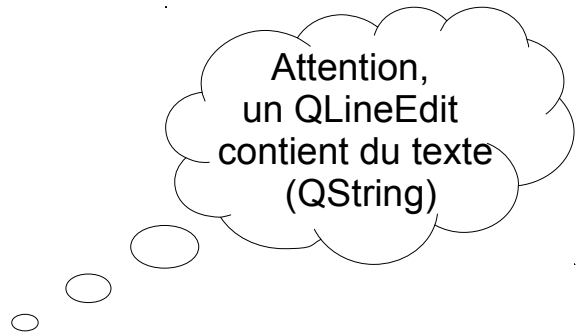
QLabel

valeur à convertir




```
ui->label->setText("salut");    // modifie le texte


ui->label->setVisible(true);    // visible
ui->label->setVisible(false);    // invisible


QString contenu = ui->label->text();    // le texte
contenu = contenu + " les GEII";        // concaténation
ui->label->setText(contenu);            // salut les GEII


if (ui->label>isVisible()){       // est il visible?
                           }
      else {  // pas visible
                 }
```

QLineEdit




```
ui->lineEdit->setVisible(true);

QRect r = ui->lineEdit->geometry();

ui->lineEdit->setCursor(Qt::CrossCursor);

double valeur;
QString chaine;
bool conversionOk;
chaine = ui->lineEdit->text();

valeur = chaine.toDouble(&conversionOk);

if (conversionOk && valeur>=0){
        valeur = sqrt(valeur);
        chaine.setNum(valeur);
        ui->lineEdit->setText(chaine);
        }
    else ui->lineEdit->setText("impossible");
```

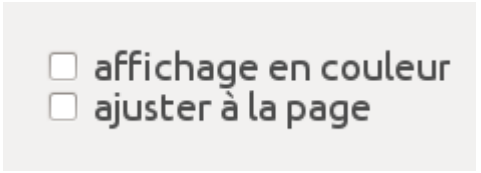
QPushButton


```
ui->pushButton->setText("start");

ui->pushButton->setVisible(true);   // ou true

ui->pushButton->setEnabled(false);  // ou true

QRect r = ui->pushButton->geometry();


if (ui->pushButton->text()=="start")
                            ui->pushButton ->setText("stop");
                      else  ui->pushButton->setText("start");
```

QRadioButton


```
ui->radioButton->setVisible(true);

ui->radioButton->setText("trés grand");

ui->radioButton->setChecked(true); // ou false


if (ui->radioButton->isChecked()){  // il est enfoncé
                                    …
                                  }
else                              { // il n'est pas enfoncé
                                  …
                                                  }
```

QCheckBox



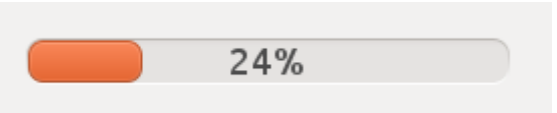

```
ui->checkBox->setVisible(true);


ui->checkBox->setText("affichage en couleur");


ui->checkBox->setChecked(true); // ou false


if (ui->checkBox->isChecked()){  // il est enfoncé
                                 }
else                             { // il n'est pas enfoncé
                                 }
```

QSpinBox


```
ui->spinBox->setVisible(true);  // ou false

// intervalle de définition
ui->spinBox->setMaximum(360);
ui->spinBox->setMinimum(-360);

// récupération de la valeur
int valeur = ui->spinBox->value();

valeur *=2;

ui->spinBox->setValue(valeur);
```

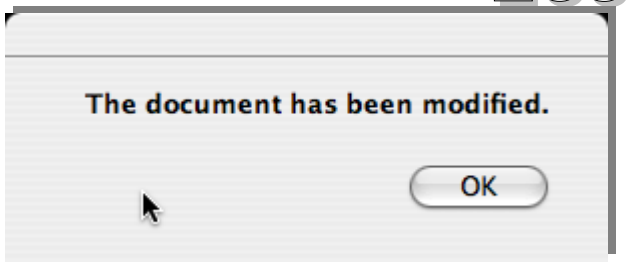
QDoubleSpinBox idem mais en double

QSlider- QDial - QProgressBar

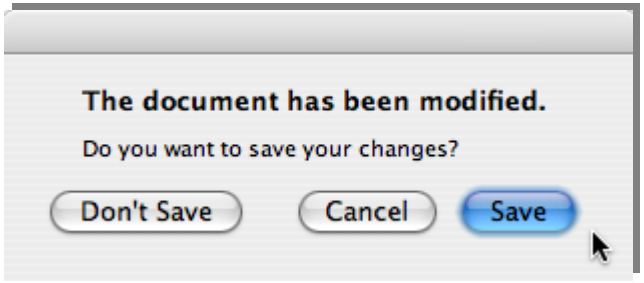



```
ui->horizontalSlider->setMaximum(200);  // on minimum
ui->progressBar->setMaximum(200);

int valeur = ui->horizontalSlider->value();

ui->progressBar->setValue(valeur);

ui->dial->setValue(200-valeur);

// mais aussi :
// la géométrie
// visible ou pas
// le cursor
// la font
// .etc.
```

Les MessagesBox prédéfinis




```
QMessageBox msgBox;
 msgBox.setText("The document has been modified.");
 msgBox.exec();
```



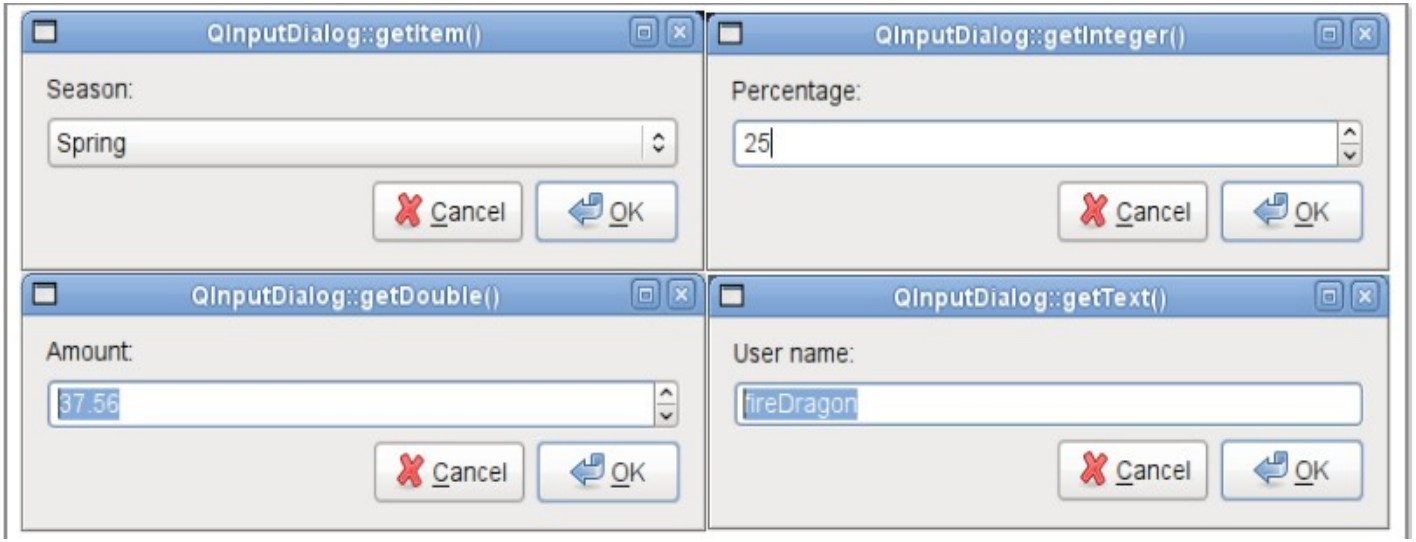

```
QMessageBox msgBox;
 msgBox.setText("The document has been modified.");
 msgBox.setInformativeText("Do you want to save your changes?");
 msgBox.setStandardButtons(QMessageBox::Save |
                           QMessageBox::Discard |
                           QMessageBox::Cancel);
 msgBox.setDefaultButton(QMessageBox::Save);
 int ret = msgBox.exec();
```

```
int ret = QMessageBox::warning(this, tr("My Application"),
                      tr("The document has been modified.\n",
                      "Do you want to save your changes?"),
                      QMessageBox::Save |
                      QmessageBox::Discard |
                      QMessageBox::Cancel,
                      QMessageBox::Save);
```

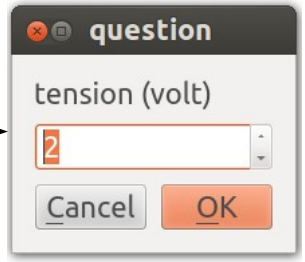
Les dialogues d'entrée prédéfinis

Static Public Members

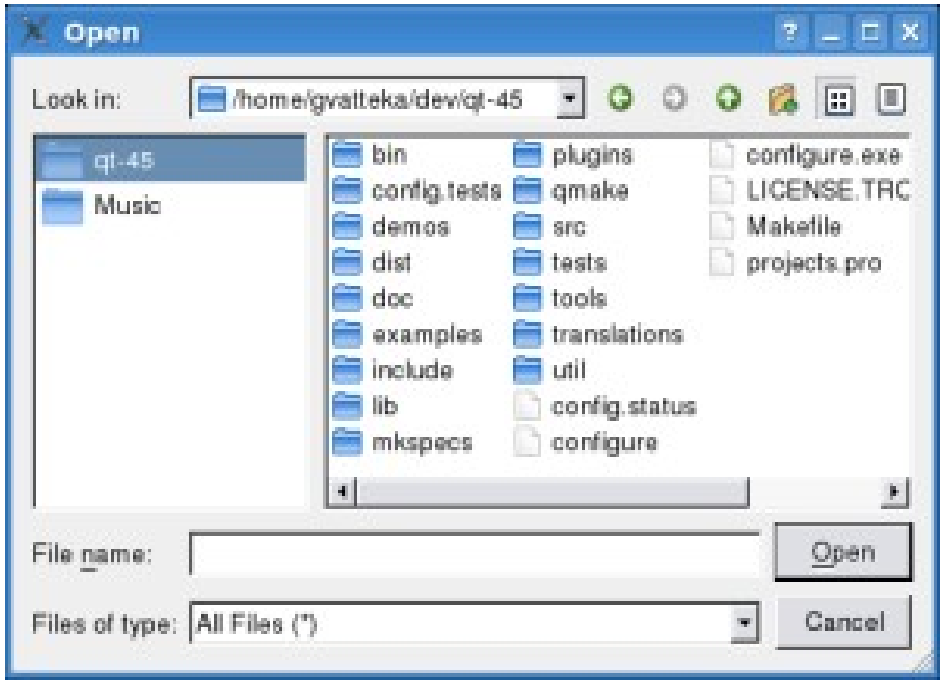
double	**getDouble**(QWidget * *parent*, const QString & *title*, const QString & *label*, double *value* = 0, double *min* = -2147483647, double *max* = 2147483647, int *decimals* = 1, bool * *ok* = 0, Qt::WindowFlags *flags* = 0)
int	**getInt**(QWidget * *parent*, const QString & *title*, const QString & *label*, int *value* = 0, int *min* = -2147483647, int *max* = 2147483647, int *step* = 1, bool * *ok* = 0, Qt::WindowFlags *flags* = 0)
int	**getInteger**(QWidget * *parent*, const QString & *title*, const QString & *label*, int *value* = 0, int *min* = -2147483647, int *max* = 2147483647, int *step* = 1, bool * *ok* = 0, Qt::WindowFlags *flags* = 0) (deprecated)
QString	**getItem**(QWidget * *parent*, const QString & *title*, const QString & *label*, const QStringList & *items*, int *current* = 0, bool *editable* = true, bool * *ok* = 0, Qt::WindowFlags *flags* = 0, Qt::InputMethodHints *inputMethodHints* = Qt::ImhNone)
QString	**getText**(QWidget * *parent*, const QString & *title*, const QString & *label*, QLineEdit::EchoMode *mode* = QLineEdit::Normal, const QString & *text* = QString(), bool * *ok* = 0, Qt::WindowFlags *flags* = 0, Qt::InputMethodHints *inputMethodHints* = Qt::ImhNone)




```
#include <QinputDialog>

…

int x = QInputDialog::getInt(this, "question","tension (volt)" );
```



Les dialogues de fichiers prédéfinis




```
QList<QUrl> urls;
urls << QUrl::fromLocalFile("/home/grimaldi/qt-45")
     << QUrl::fromLocalFile(QDesktopServices::storageLocation(QDesktopServices::MusicLocation));

 QFileDialog dialog;
 dialog.setSidebarUrls(urls);
 dialog.setFileMode(QFileDialog::AnyFile);
 if(dialog.exec()) {
      // ...
 }
```

Pour le reste voir l'aide de Qt .....