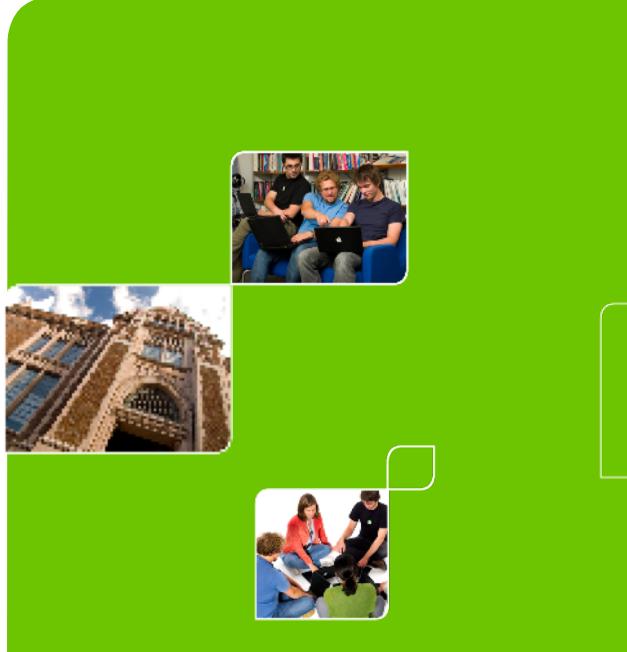




Qt dans l'Enseignement

## Widgets et Agencements



**digia**



## Traduction française et Adaptation des supports pour l'enseignement

© 2012 Digia Plc.

The enclosed Qt Materials are provided under the Creative Commons Attribution-Share Alike 2.5 License Agreement.



The full license text is available here:  
<http://creativecommons.org/licenses/by-sa/2.5/legalcode>.

Digia, Qt and the Digia and Qt logos are the registered trademarks of Digia Plc. in Finland and other countries worldwide.

**digia**

Qt Essentials - Widgets Module Training Course  
Visit us at <http://qt.digia.com>  
Produced by Digia Plc.  
Materialized Q1 2012 version on September 27, 2012

<http://www.qt.io/training-materials/>  
<http://www.qt.io/qt-essentials-widget-edition/>

**digia** 2

## Qt Les composants d'interface

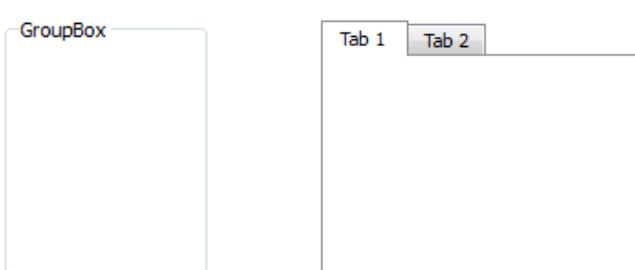


- Interfaces utilisateur ⇒ des widgets
- ~50 widgets proposées dans le concepteur
- + de 59 descendants directs de `QWidget`

**digia** 3

## Qt Widgets dans Widgets

- Widgets ⇒ placés hiérarchiquement



- Des classes conteneurs pour structuration visuelle
- ...mais aussi fonctionnelle (ex. `QRadioButton`)

**digia** 4



## Widgets : caractéristiques

- Occupe une zone rectangulaire de l'écran
- Reçoit des événements de périphériques d'entrée
- Émet des signaux quand il change
- Structuré hiérarchiquement entre eux
- Peut contenir d'autres widgets

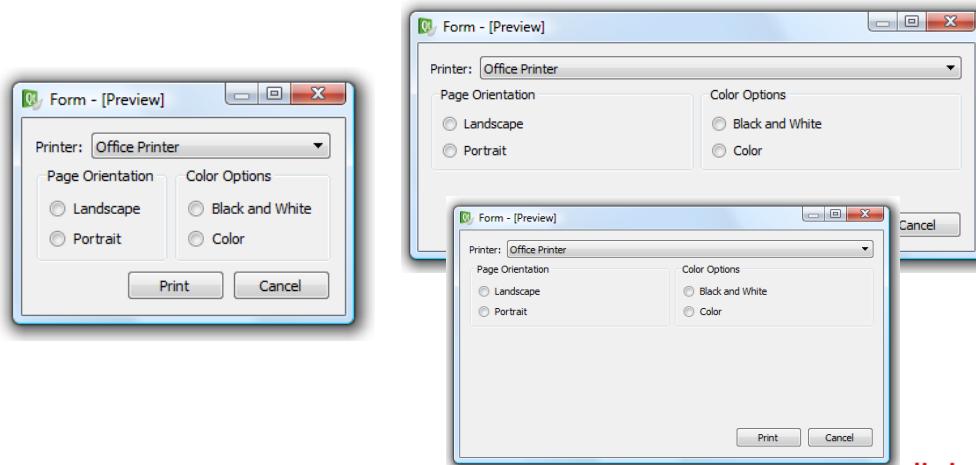
digia 5



## Un exemple de dialogue



- Les widgets sont placés dans des agenceurs
  - Rendre l'interface “élastique”



digia 6

**Qt Pourquoi vouloir l'élasticité?**

- Adaptation au contenu

```
\home\john\Documents\Work\Projects\Base
\home\john\Documents\Work\Projects\Brainstorming
\home\john\Documents\Work\Projects\Design
\home\john\Documents\Work\Projects\Hardware
```

- Adaptation aux traductions

- Adaptation aux configurations de l'utilisateur

News

digia 7

**Qt Agencements**

- Il y a plusieurs agenceurs disponibles

PushButton	PushButton	PushButton	PushButton
PushButton	QHBoxLayout		
PushButton			

QVBoxLayout		PushButton	PushButton	QGridLayout	
		PushButton	PushButton		

TextLabel	<input type="text"/>
TextLabel	<input type="text"/>
TextLabel	<input type="text"/>

QFormLayout

QStackedLayout

- Agenceurs et widgets “négocient” leurs tailles et positions
- Espaces ⇒ pour définir des zones vides

digia 8

# Qt Exemple de dialogue

- Plusieurs niveaux d'agenceurs et widgets

Object

- Form
- horizontalLayout
  - label
  - printerBox
- horizontalLayout\_2
  - cancelButton
  - horizontalSpacer
  - printButton
- horizontalLayout\_3
  - groupBox
    - landscapeButton
    - portraitButton
  - groupBox\_2
    - bwButton
    - colorButton
- verticalSpacer

Class

- QWidget
- QHBoxLayout
- QLabel
- QComboBox
- QHBoxLayout
- QPushButton
- Spacer
- QPushButton
- QHBoxLayout
- QGroupBox
- QRadioButton
- QRadioButton
- QGroupBox
- QRadioButton
- QRadioButton
- Spacer

digia 9

# Qt Exemple de dialogue

```
QVBoxLayout *outerLayout = new QVBoxLayout(this);

QHBoxLayout *topLayout = new QHBoxLayout();
topLayout->addWidget(new QLabel("Printer:"));
topLayout->addWidget(c=new QComboBox());
outerLayout->addLayout(topLayout);
```

```
QHBoxLayout *groupLayout = new QHBoxLayout();
...
outerLayout->addLayout(groupLayout);
outerLayout->addSpacerItem(new QSpacerItem(...));
QHBoxLayout *buttonLayout = new QHBoxLayout();
buttonLayout->addSpacerItem(new QSpacerItem(...));
buttonLayout->addWidget(new QPushButton("Print"));
buttonLayout->addWidget(new QPushButton("Cancel"));
outerLayout->addLayout(buttonLayout);
```

digia 10

**Qt** Exemple de dialogue

```

QVBoxLayout *outerLayout = new QVBoxLayout(this);

QHBoxLayout *topLayout = new QHBoxLayout();
topLayout->addWidget(new QLabel("Printer:"));
topLayout->addWidget(c=new QComboBox());
outerLayout->addLayout(topLayout);

QHBoxLayout *groupLayout = new QHBoxLayout();
    ...
outerLayout->addLayout(groupLayout);

outerLayout->addSpacerItem(new QSpacerItem(...));

QHBoxLayout *buttonLayout = new QHBoxLayout();
buttonLayout->addSpacerItem(new QSpacerItem(...));
buttonLayout->addWidget(new QPushButton("Print"));
buttonLayout->addWidget(new QPushButton("Cancel"));
outerLayout->addLayout(buttonLayout);

```

The screenshot shows the Qt Designer interface with the code for creating a print dialog. The layout consists of three main sections: a top section with a printer dropdown, a middle section with page orientation and color options, and a bottom section with Print and Cancel buttons.

**digia** 11

**Qt** Exemple de dialogue

```

QVBoxLayout *outerLayout = new QVBoxLayout(this);

QHBoxLayout *topLayout = new QHBoxLayout();
topLayout->addWidget(new QLabel("Printer:"));
topLayout->addWidget(c=new QComboBox());
outerLayout->addLayout(topLayout);

QHBoxLayout *groupLayout = new QHBoxLayout();
    ...
outerLayout->addLayout(groupLayout);

outerLayout->addSpacerItem(new QSpacerItem(...));

QHBoxLayout *buttonLayout = new QHBoxLayout();
buttonLayout->addSpacerItem(new QSpacerItem(...));
buttonLayout->addWidget(new QPushButton("Print"));
buttonLayout->addWidget(new QPushButton("Cancel"));
outerLayout->addLayout(buttonLayout);

```

The screenshot shows the Qt Designer interface with the same code for creating a print dialog. The layout is identical to the one in the previous slide, featuring a top section with a printer dropdown, a middle section with page orientation and color options, and a bottom section with Print and Cancel buttons.

**digia** 12

**Qt** Exemple de dialogue

```

QVBoxLayout *outerLayout = new QVBoxLayout(this);

QHBoxLayout *topLayout = new QHBoxLayout();
topLayout->addWidget(new QLabel("Printer:"));
topLayout->addWidget(c=new QComboBox());
outerLayout->addLayout(topLayout);

QHBoxLayout *groupLayout = new QHBoxLayout();
    
```

Printer: Office Printer

Page Orientation      Color Options

Landscape      Black and White  
Portrait      Color

```

outerLayout->addLayout(groupLayout);

outerLayout->addSpacerItem(new QSpacerItem(...));

QHBoxLayout *buttonLayout = new QHBoxLayout();
buttonLayout->addSpacerItem(new QSpacerItem(...));
buttonLayout->addWidget(new QPushButton("Print"));
buttonLayout->addWidget(new QPushButton("Cancel"));
outerLayout->addLayout(buttonLayout);
    
```

Print      Cancel

**digia** 13

**Qt** Exemple de dialogue

```

QVBoxLayout *outerLayout = new QVBoxLayout(this);

QHBoxLayout *topLayout = new QHBoxLayout();
topLayout->addWidget(new QLabel("Printer:"));
topLayout->addWidget(c=new QComboBox());
outerLayout->addLayout(topLayout);

QHBoxLayout *groupLayout = new QHBoxLayout();
    
```

Printer: Office Printer

Page Orientation      Color Options

Landscape      Black and White  
Portrait      Color

```

outerLayout->addLayout(groupLayout);

outerLayout->addSpacerItem(new QSpacerItem(...));

QHBoxLayout *buttonLayout = new QHBoxLayout();
buttonLayout->addSpacerItem(new QSpacerItem(...));
buttonLayout->addWidget(new QPushButton("Print"));
buttonLayout->addWidget(new QPushButton("Cancel"));
outerLayout->addLayout(buttonLayout);
    
```

Print      Cancel

**digia** 14



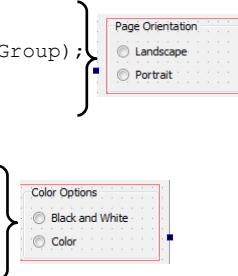
## Exemple de dialogue

- La boîte horizontale contient
  - Deux boîtes groupes qui contiennent chacune
    - Une boîte verticale qui contient
      - Des boutons radios

```
QHBoxLayout *groupLayout = new QHBoxLayout();
```

```
QGroupBox *orientationGroup = new QGroupBox();
QVBoxLayout *orientationLayout = new QVBoxLayout(orientationGroup);
orientationLayout->addWidget(new QRadioButton("Landscape"));
orientationLayout->addWidget(new QRadioButton("Portrait"));
groupLayout->addWidget(orientationGroup);
```

```
QGroupBox *colorGroup = new QGroupBox();
QVBoxLayout *colorLayout = new QVBoxLayout(colorGroup);
colorLayout->addWidget(new QRadioButton("Black and White"));
colorLayout->addWidget(new QRadioButton("Color"));
groupLayout->addWidget(colorGroup);
```



digia

15



## Exemple de dialogue

- Même structure avec l'EDI de QtCreator : QtDesigner

Object	Class
Form	QWidget
horizontalLayout	QHBoxLayout
label	QLabel
printerBox	QComboBox
horizontalLayout_2	QHBoxLayout
cancelButton	QPushButton
horizontalSpacer	Spacer
printButton	QPushButton
horizontalLayout_3	QHBoxLayout
groupBox	QGroupBox
landscapeButton	QRadioButton
portraitButton	QRadioButton
groupBox_2	QGroupBox
bwButton	QRadioButton
colorButton	QRadioButton
verticalSpacer	Spacer

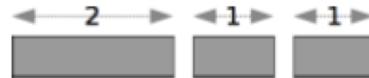
digia

16



## Quelques propriétés

- Stretch ⇒ définit un facteur de taille relatif



```
QBoxLayout::addWidget(widget, stretch)
QBoxLayout::addStretch(stretch)
QGridLayout::setRowStretch(row, stretch)
QGridLayout::setColumnStretch(col, stretch)
```

digia

17



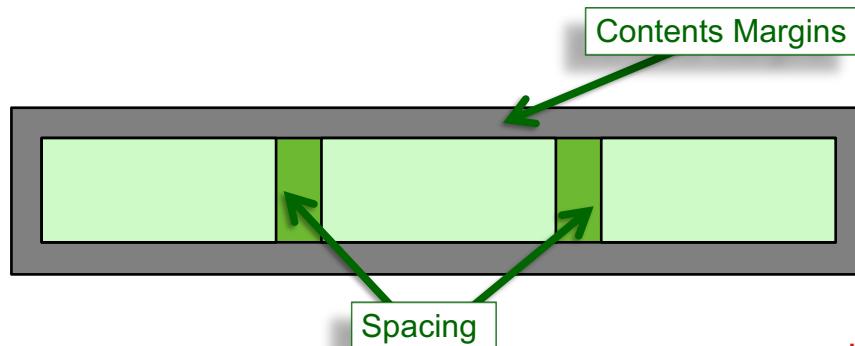
## Quelques propriétés

- Contents Margins ⇒ espace autour des widgets gérées

```
QLayout::setContentsMargins(l, t, r, b)
```

- Spacing ⇒ espace réservé entre les widgets

```
QBoxLayout::addSpacing(size)
```



digia

18

# Qt Styles multiplateformes

- Widgets ⇒ look natif en utilisant un style

The image shows three separate windows of the Qt style switcher dialog. Each dialog has tabs for "Styles" and "Margins". The "Styles" tab is active. In each window, there are four radio buttons: "Heading", "Paragraph", "List", and "Footnote". In the first window (Plastique), "Heading" is selected. In the second window (Windows), "Paragraph" is selected. In the third window (Mac OS X), "Paragraph" is also selected. All three windows show the same basic interface with tabs and radio buttons.

**digia 19**

# Qt Stratégie multiplateforme

- Multiplateforme ⇒ pas qu'une question de style
  - Agencement des formulaires

The image shows four separate windows of the Qt file properties dialog. Each dialog displays the following information: Name: HPIM1273.jpeg, Tags: car garage house, Description: Size: 2048x1536, DPI: 72, Bitdepth: 24, and Access: Read and Write. The windows are labeled at the bottom: "Plastique", "Windows", "ClearLooks", and "Mac OS X". The labels "Plastique" and "ClearLooks" are in green, while "Windows" and "Mac OS X" are in black.

**20**

## Qt Stratégie multiplateforme

- Multiplateforme ⇒ pas qu'une question de style
  - Ordre des boutons

**digia 21**

## Qt Stratégie multiplateforme

- Multiplateforme ⇒ pas qu'une question de style
  - Dialogues standards

**QFileDialog**      **QColorDialog**

**digia 22**

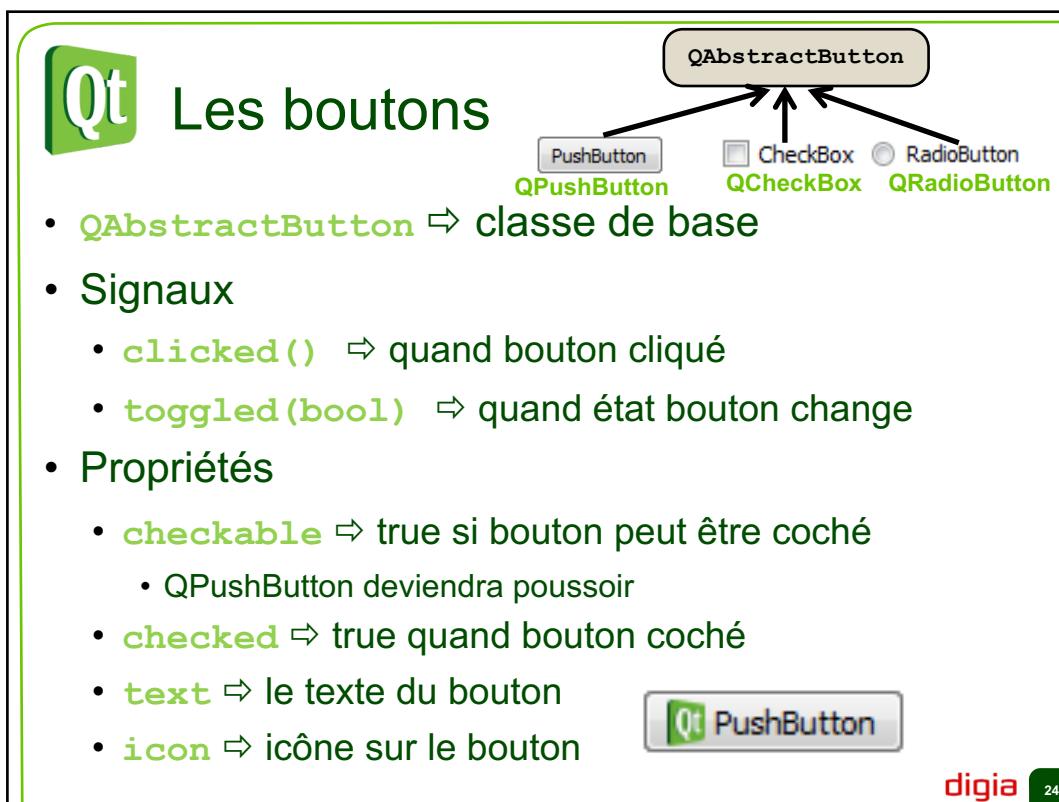


**Qt Les widgets courantes**

- Pour toutes les situations courantes
- Le concepteur montre une vue d'ensemble

Widgets regroupés par catégorie

digia 23



**Qt Les boutons**

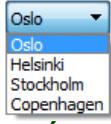
- **QAbstractButton** ⇒ classe de base
- Signaux
  - `clicked()` ⇒ quand bouton cliqué
  - `toggled(bool)` ⇒ quand état bouton change
- Propriétés
  - `checkable` ⇒ true si bouton peut être coché
    - QPushButton deviendra poussoir
  - `checked` ⇒ true quand bouton coché
  - `text` ⇒ le texte du bouton
  - `icon` ⇒ icône sur le bouton

digia 24

## Qt Les listes



**QListWidget**

- **QListWidget** ⇒ pour montrer une liste d'éléments
- Ajout d'éléments
  - `addItem(QString)` ⇒ ajout en fin de liste
  - `insertItem(int row, QString)` ⇒ ajout à une position
- Sélection
  - `selectedItems` ⇒ donne une liste **QListWidgetItem**s
  - `QListWidgetItem::text` donne le texte
- Signaux
  - `itemSelectionChanged` ⇒ quand sélection a changé
- **QComboBox**

- **QComboBox** ⇒ liste dans un format compacte

**digia** 25

## Qt Les conteneurs



**QGroupBox**



**QFrame**



**QTabWidget**

- Pour structurer l'interface
- Considérés comme passifs (pas toujours vrai !)
- Concepteur
  - Placer widgets dans conteneur
  - Ajouter ensuite agencement au conteneur
- Code
  - Créer un agencement pour le conteneur
  - Ajouter ensuite widgets à l'agencement

```
QGroupBox *box = new QGroupBox();
QVBoxLayout *layout = new QVBoxLayout(box);
layout->addWidget(...);
```

**digia** 26

**Qt** Les entrées



- `QLineEdit` ⇒ pour saisir une ligne de texte
- Signaux
  - `textChanged(QString)` ⇒ quand texte modifié
  - `editingFinished()` ⇒ quand widget est quittée
  - `returnPressed()` ⇒ quand touche entrée saisie
- Propriétés
  - `text` ⇒ le texte du widget
  - `maxLength` ⇒ limite la longueur de la saisie
  - `readOnly` ⇒ true pour empêcher la saisie
    - Copie du texte possible (# libellé)

digia 27

**Qt** Les entrées

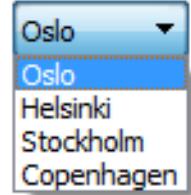
- `QTextEdit` OU `QPlainTextEdit` ⇒ plusieurs lignes
- Signaux
  - `textChanged()` ⇒ quand texte modifié
- Propriétés
  - `plainText` ⇒ texte non formaté
  - `html` ⇒ texte au format HTML
  - `readOnly` ⇒ true pour empêcher la saisie



digia 28

## Qt Les entrées

QComboBox

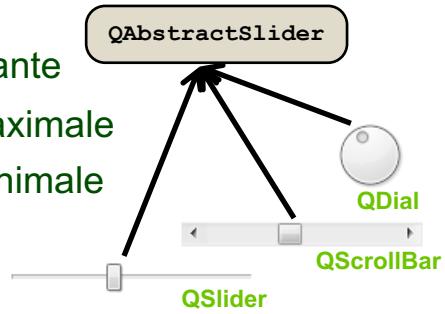


- `QComboBox` peut être éditable
  - Propriété `editable`
- Signaux
  - `editTextChanged(QString)` ⇒ quand texte édité
- Propriétés
  - `currentText` ⇒ texte courant dans la boîte à option

**digia** 29

## Qt Les entrées

QAbstractSlider



- Large choix de widgets pour éditer des entiers
- Il y en a d'autres pour doubles, durées et dates
- Signaux
  - `valueChanged(int)` ⇒ valeur mise à jour
- Propriétés
  - `value` ⇒ valeur courante
  - `maximum` ⇒ valeur maximale
  - `minimum` ⇒ valeur minimale

**digia** 30



## Rétroaction visuelle

- `QLabel` ⇒ affiche un texte ou une image
  - Propriétés
    - `text` ⇒ text du libellé
    - `pixmap` ⇒ image à afficher
- `QLCDNumber` ⇒ affiche un entier
  - Propriétés
    - `intValue` ⇒ valeur à montrer (à définir avec `display(int)`)



HelloWorld  
QLabel

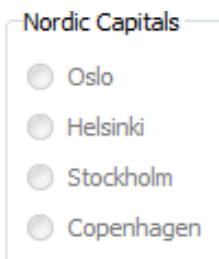


digia 31



## Propriétés communes

- Héritées de classe de base `QWidget`
- `enabled` ⇒ autorise ou non l'interaction
- `visible` ⇒ montre ou non
  - Alternative : méthodes `show` et `hide`
- Ces propriétés affectent les widgets enfants
  - Exemple : désactiver un conteneur



PushButton



PushButton

digia 32

# Qt Politiques de taille

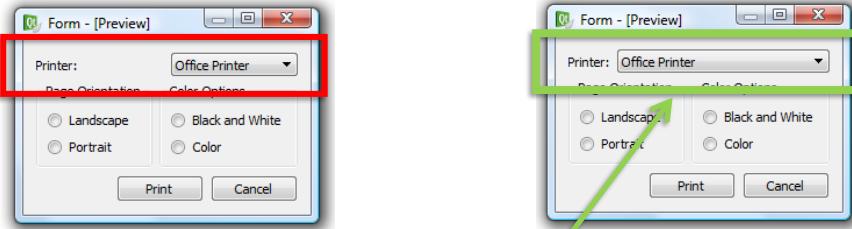
- Agencement = processus widgets ⇔ agenceurs
- Agenceurs apportent une structure
  - Boîtes horizontales et verticales
  - Grille
- Widgets fournissent
  - Politiques de taille pour chaque direction ⇒ QSizePolicy
  - Tailles minimales et maximales




**digia** 33

# Qt Politiques de taille

- L'exemple n'était pas complet !



```
printerList->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed)
```



Comportement horizontal



Comportement vertical

**digia** 34



## Politiques de taille

- Widget  $\Rightarrow$  un indice de taille + 1 politique/direction
  - **Fixed**  $\Rightarrow$  indice = taille widget
  - **Minimum**  $\Rightarrow$  indice = + petite taille possible
  - **Maximum**  $\Rightarrow$  indice = + grande taille possible
  - **Preferred**  $\Rightarrow$  indice = taille préférée, mais pas requise
  - **Expanding**  $\Rightarrow$  ~ preferred, mais veut grandir
  - **MinimumExpanding**  $\Rightarrow$  ~ minimum, mais veut grandir
  - **Ignored**  $\Rightarrow$  indice de taille ignoré
    - Widget prend toute la place disponible

digia

35



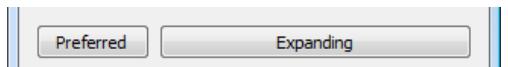
## Politiques de taille

- Widget  $\Rightarrow$  un indice de taille + 1 politique/direction
  - **Fixed**  $\Rightarrow$  fixée à celle de l'indice
  - **Minimum**  $\Rightarrow$  peut s'agrandir
  - **Maximum**  $\Rightarrow$  peut se réduire
  - **Preferred**  $\Rightarrow$  peut s'agrandir et se rétrécir
  - **Expanding**  $\Rightarrow$  peut s'agrandir, se rétrécir, veut grandir
  - **MinimumExpanding**  $\Rightarrow$  peut s'agrandir, veut grandir
  - **Ignored**  $\Rightarrow$  indice ignoré, peut s'agrandir et se rétrécir

digia

36

**Qt Quelle combinaison?**

- 2 preferred
- 1 preferred, 1 expanding
- 2 expanding
- Pas assez de widget pour remplir l'espace (fixed)


**digia** 37

**Qt Plus sur les tailles...**

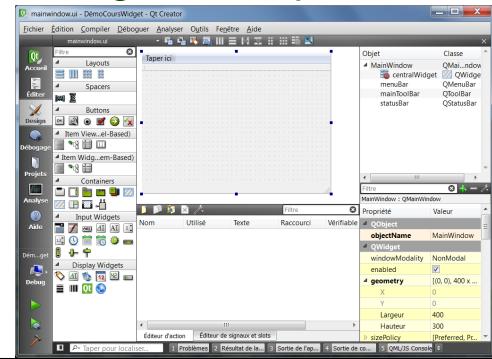


- Contrôle de tailles avec propriétés
  - taille maximum
  - taille minimum
- `maximumSize` ⇒ plus grande largeur possible
- `minimumSize` ⇒ plus petite largeur

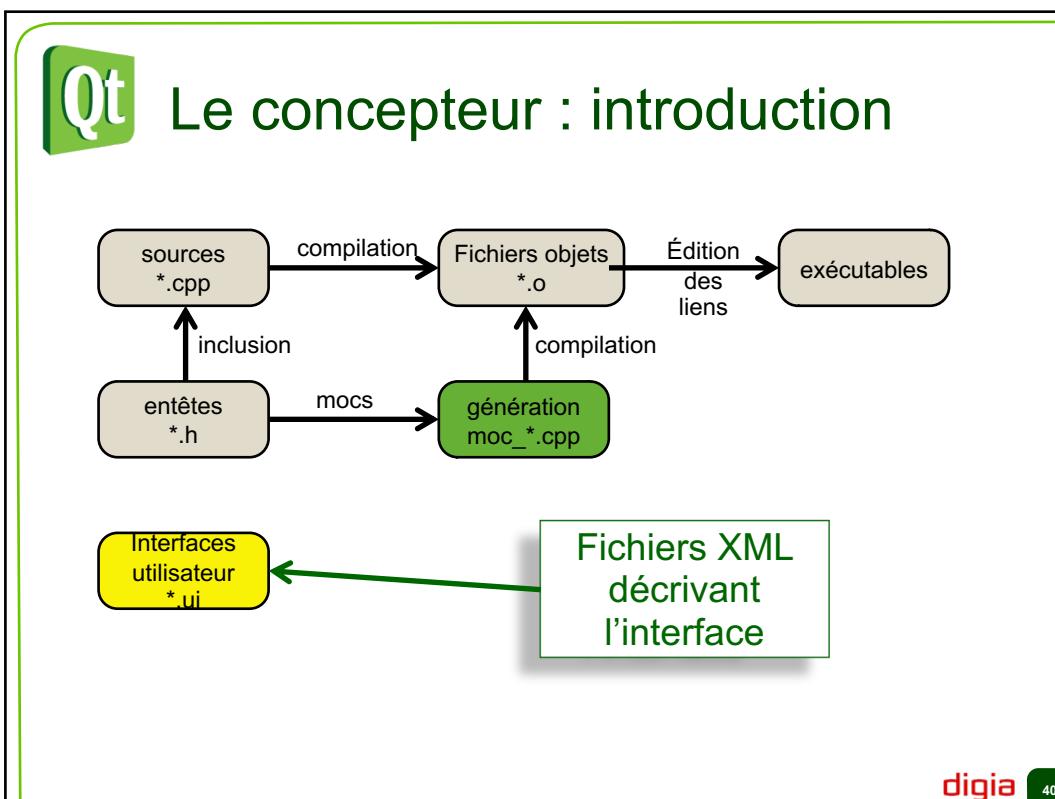
```
ui->pushButton->setMinimumSize(100, 150);
ui->pushButton->setMaximumHeight(250);
```

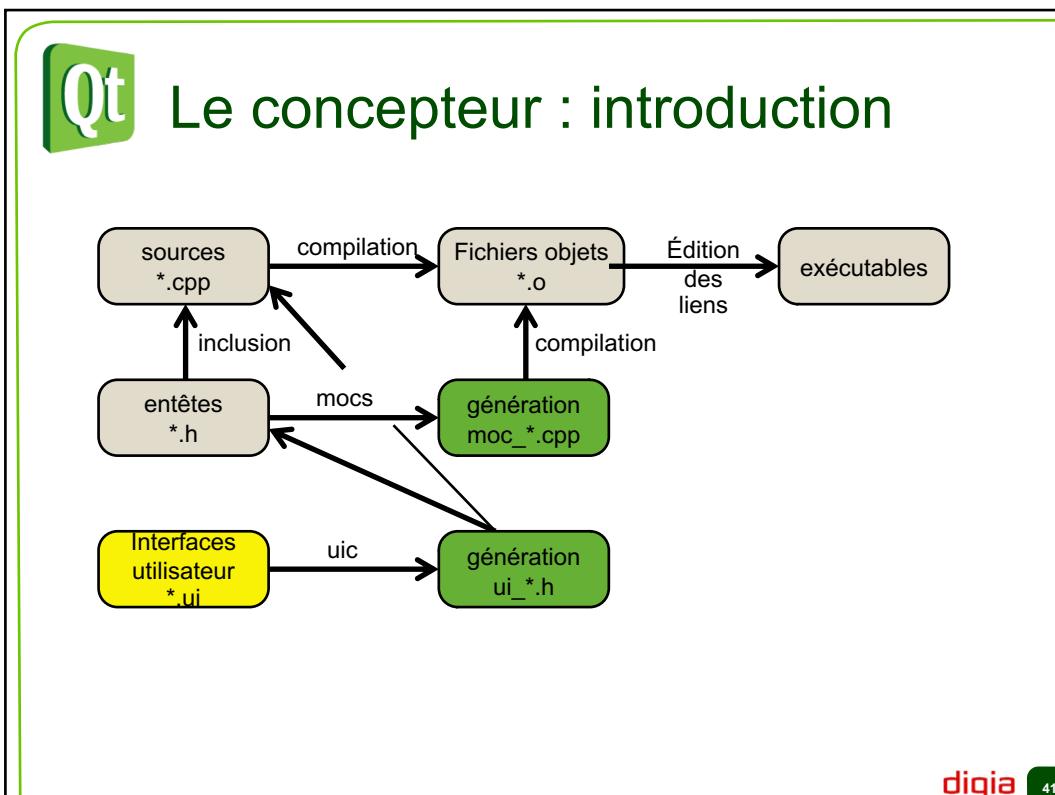
**digia** 38

## Qt Le concepteur

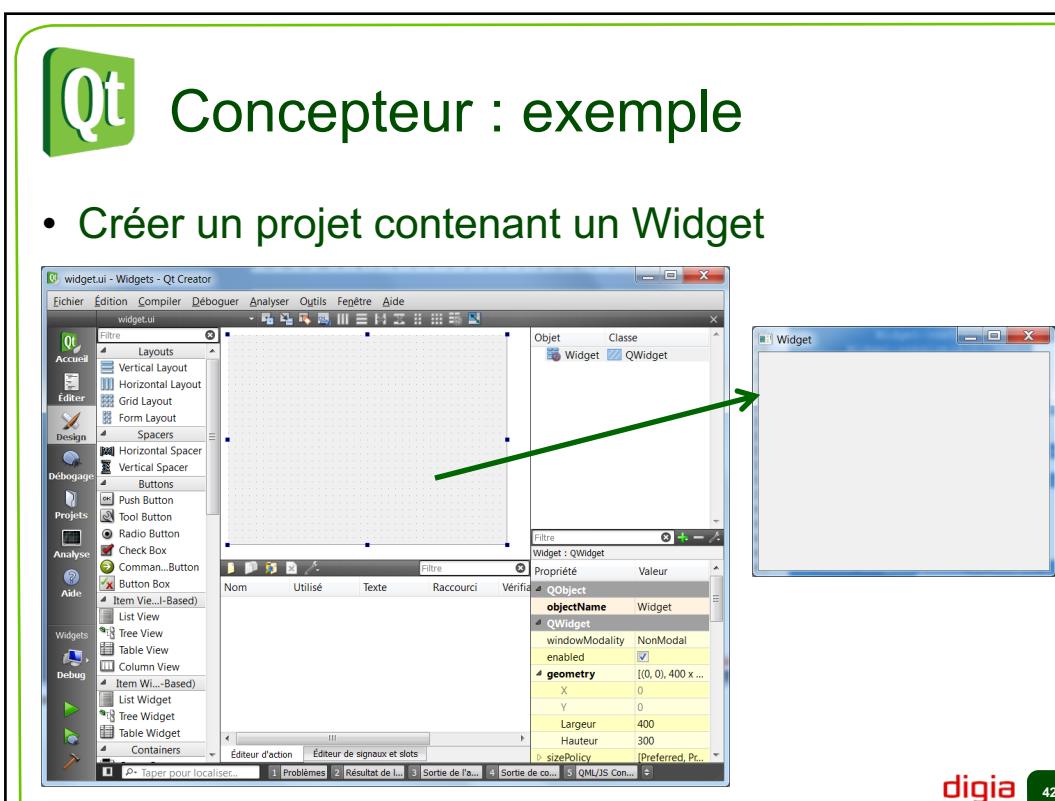


- Historiquement outil séparé
- Maintenant ⇒ partie de QtCreator
- Un éditeur visuel de formulaires
- Manipulation directe avec drag-and-drop
- Gère les agencements
- Réalise les connexions





digia 41



digia 42

**Qt Exemple : code généré**

- Avant compilation

```
main.cpp
widget.cpp
widget.h
widget.ui
Widgets.pro
Widgets.pro.user
```

- Après compilation
- Dans répertoire build

```
ui_widget.h
moc_widget.cpp
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>Widget</class>
<widget class="QWidget" name="Widget">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>400</width>
<height>300</height>
</rect>
</property>
<property name="windowTitle">
<string>Widget</string>
</property>
</widget>
<layoutdefault spacing="6"
margin="11"/>
<resources/>
<connections/>
</ui>
```

**digia** 43

**Qt Exemple : code généré**



Déclaration de l'application

```
#include "widget.h"
#include <QApplication>
```

Déclaration du widget

```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
```

Affichage du widget

```
    Widget w;
```

Démarrage de l'application

```
    w.show();
    return a.exec();
}
```

Fichier main.cpp

**digia** 44

**Qt** Exemple : code généré




```
#include <QWidget>
namespace Ui { } // Déclaration forward car inclusion de ui_widget.h
class Widget; // dans ui::Widget

class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent=0);
    ~Widget();

private:
    Ui::Widget *ui; // ui donnera accès à tous les widgets
};

Fichier widget.h
```

digia 45

**Qt** Exemple : code généré



Fichier widget.cpp

Fichier pour créer l'interface  
⇒ généré à la compilation

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent)
: QWidget(parent),
ui(new Ui::Widget) // Crée tous les widgets comme enfant de this
{
    ui->setupUi(this); // Initialise ui
}

Widget::~Widget()
{
    delete ui; // Détruit ui
}
```

Crée tous les widgets comme enfant de this

Initialise ui

Détruit ui

digia 46



## Exemple : code généré

```

class Ui_Widget
{public:
    void setupUi(QWidget *Widget)
    {
        if (Widget->objectName().isEmpty())
            Widget->setObjectName(QStringLiteral("Widget"));
        Widget->resize(400, 300);

        retranslateUi(Widget);

        QMetaObject::connectSlotsByName(Widget);
    } // setupUi

    void retranslateUi(QWidget *Widget)
    {
        Widget->setWindowTitle(QApplication::translate("Widget", "Widget", 0));
    } // retranslateUi
};

namespace Ui {
    class Widget: public Ui_Widget {};
} // namespace Ui

```

Fichier  
ui\_widget.h

digia

47



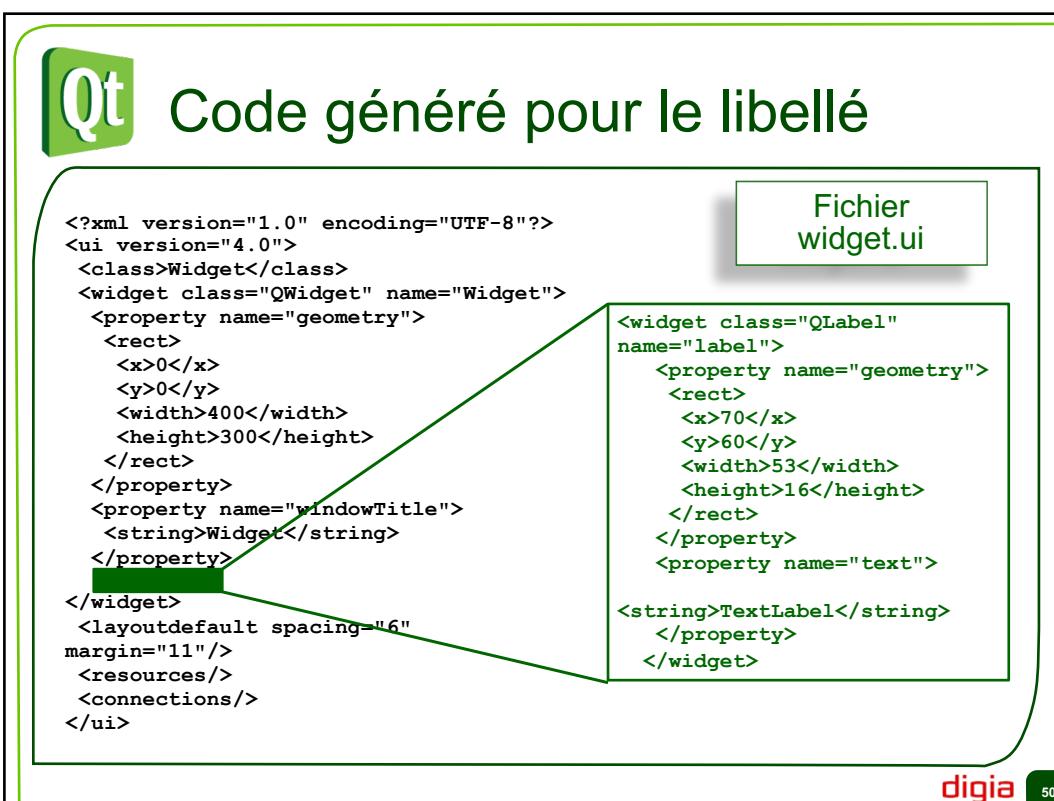
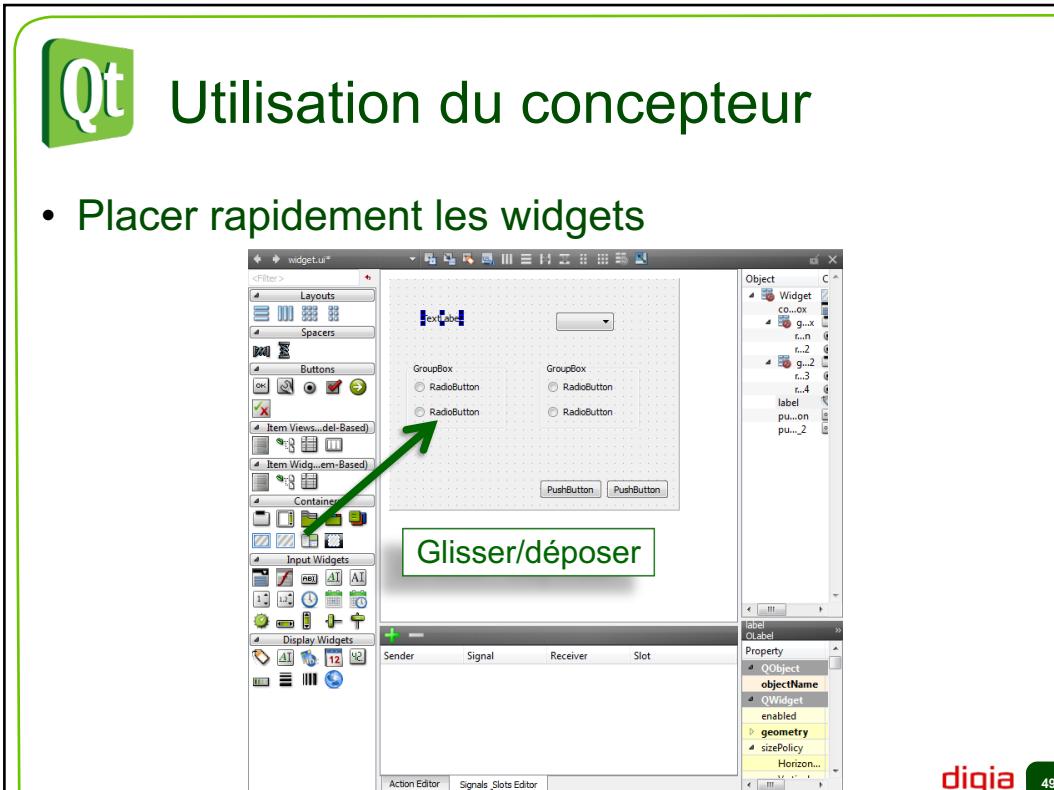
## Utilisation du concepteur



- 5 tâches à réaliser
  - Placer rapidement les widgets
  - Appliquer les agencements de l'intérieur vers l'extérieur
    - Ajouter des espaces si nécessaires
  - Editer et mettre à jour les propriétés
  - Réaliser les connexions
  - Coder l'interface à partir de ce qui est généré
- Le concepteur nécessite de la pratique !

digia

48



## Qt Code généré pour le libellé

```

class Ui_Widget
{public:
    QLabel *label;

    void setupUi(QWidget *Widget)
    {
        if (Widget->objectName().isEmpty())
            Widget->setObjectName(QStringLiteral("Widget"));
        Widget->resize(400, 300);
        label = new QLabel(Widget);
        label->setObjectName(QStringLiteral("label"));
        label->setGeometry(QRect(70, 60, 53, 16));

        retranslateUi(Widget);

        QMetaObject::connectSlotsByName(Widget);
    } // setupUi

    void retranslateUi(QWidget *Widget)
    {
        Widget->setWindowTitle(QApplication::translate("Widget", "Widget", 0));
        label->setText(QApplication::translate("Widget", "TextLabel", 0));
    } // retranslateUi
};

digia 51

```

Fichier ui\_widget.h

## Qt Utilisation du concepteur

- Appliquer les agencements
  - Commencer par les boutons radios

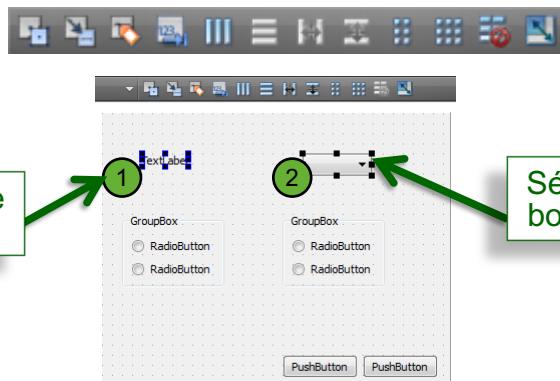
Sélectionner chaque boîte groupe

Appliquer un agencement vertical

digia 52

## Qt Utilisation du concepteur

- Appliquer les agencements
  - Régler l'agencement pour la première ligne



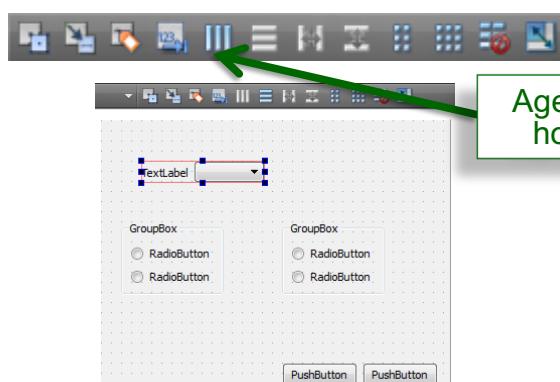
Sélectionner le libellé

Sélectionner la boîte à options

digia 53

## Qt Utilisation du concepteur

- Appliquer les agencements
  - Régler l'agencement pour la première ligne
    - Y mettre une boîte horizontale



Agencement horizontal

digia 54

## Qt Utilisation du concepteur

- Appliquer les agencements

The screenshot shows the Qt Designer interface with a grid layout. A green box labeled "Agencement horizontal pour la 2<sup>e</sup> ligne" points to a horizontal layout containing a TextLabel and two GroupBox widgets. A green box labeled "Ajout d'un espace" points to a vertical space between two push buttons. A green box labeled "Agencement horizontal pour la 3<sup>e</sup> ligne" points to another horizontal layout containing two push buttons.

digia 55

## Qt Utilisation du concepteur

- Appliquer les agencements

The screenshot shows the Qt Designer interface with a vertical layout. A green box labeled "Agencement vertical" points to the vertical layout icon in the toolbar. A green box labeled "Ajouter un espace vertical" points to a vertical space between two push buttons. A green box labeled "Sélectionner le formulaire" points to the "Select Form" icon in the toolbar. A green box labeled "Prévisualiser l'interface ⇒ Alt+Shift+R" points to a preview window titled "Widget - [prévisualisation]" showing the interface.

digia 56

## Qt Utilisation du concepteur

- Définir les connexions entre widgets

- Passer en mode édition signal/slot
- Glisser du widget source au widget destination
- Choisir le signal clicked() et le slot close()
- La nouvelle connexion apparaît dans l'éditeur

The screenshot shows the Qt Designer interface. At the top, there's a toolbar with various icons. Below it is a palette containing 'TextLabel', 'GroupBox', 'RadioButton', and 'PushButton'. A 'PushButton' is selected. In the center, there's a main window with a 'Configure connection' dialog open. This dialog has two dropdown menus: 'Emetteur' (pushButton) and 'Récepteur' (Widget). Underneath, it lists 'Signal' (clicked()) and 'Slot' (close()). A green callout points to the 'Signal' dropdown with step 4. On the right, there's a 'digia' logo and the number 57.

## Qt Utilisation du concepteur

- Définir les connections entre widget et code

- Mode édition de widgets
- Clic bouton droit sur widget source puis choisir « Aller au slot... »
- Choisir le signal auquel on doit répondre
- Définir l'action du slot généré

The screenshot shows the Qt Designer interface. At the top, there's a toolbar with various icons. Below it is a palette containing 'PushButton'. A 'PushButton' is selected. In the center, there's a main window with a 'Go to slot' dialog open. This dialog lists signals like 'clicked()', 'clicked(bool)', and 'pressed()'. A green callout points to this list with step 3. To the right, there's a context menu with items like 'Contrainte de taille', 'Layout Alignment', 'Promouvoir en...', and 'Aller au slot...'. A green callout points to the 'Aller au slot...' item with step 2. At the bottom, there's a code editor window showing C++ code. A green callout points to the code with step 4. The code is as follows:

```

15
16 void Widget::on_pushButton_2_clicked()
17 {
18
19 }

```

On the right, there's a 'digia' logo and the number 58.



## Utilisation du concepteur

- Coder l'interface à partir de ce qui est généré
  - Accès aux widgets via donnée membre `ui`

```
class Widget : public QWidget {
    ...
private:
    Ui::Widget *ui;
};
```

```
void Widget::memberFunction()
{
    ui->pushButton->setText(...);
}
```

**digia** 59



## Fenêtres de premier niveau



- Widgets sans parent = fenêtre
- `QWidget` ⇒ simple fenêtre en général non modale
- `QDialog` ⇒ boîte de dialogue
  - Attend en général un résultat (`Ok`, `Cancel`, etc)
- `QMainWindow` ⇒ fenêtre d'application avec
  - Menus
  - Barre d'outils
  - Barre d'état, etc
- `QDialog` et  `QMainWindow` hérite de  `QWidget`

**digia** 60



## QWidget comme fenêtre



- Toute widget peut être une fenêtre
- Widgets sans parent ⇒ fenêtre automatique
- Widgets avec parent ⇒ flag `Qt::Window`
  - À définir pour constructeur `QWidget`
- `setWindowModality` ⇒ pour rendre modal
  - `NonModal` ⇒ toutes les fenêtres utilisables ensemble
  - `WindowModal` ⇒ la fenêtre parent est bloquée
  - `ApplicationModal` ⇒ les autres fenêtres sont bloquées

**digia** 61



## Fenêtre : propriétés

- Titre de la fenêtre ⇒ `setWindowTitle`
- Constructeur de `QWidget`

```
QWidget::QWidget(QWidget *parent, Qt::WindowFlags f=0)
```

- `f` : définir le style
  - `Qt::Window` ⇒ créer une fenêtre
  - `Qt::CustomizeWindowHint` ⇒ personnaliser ses contrôles
    - `Qt::WindowMinimizeButtonHint` ⇒ bouton pour réduire la fenêtre
    - `Qt::WindowMaximizeButtonHint` ⇒ bouton pour agrandir la fenêtre
    - `Qt::WindowCloseButtonHint` ⇒ bouton pour fermer la fenêtre
    - etc

**digia** 62

## Qt QMainWindows

- **QMainWindow** ⇒ fenêtre principale
  1. “bureau” de l’application
  2. Menus
  3. Barre d’outils
  4. Barre d’état
  5. Fenêtre ancrée
  6. Widget centrale

**digia** 63

## Qt Boîte de dialogue : exemple

- Exemple : dialogue de recherche

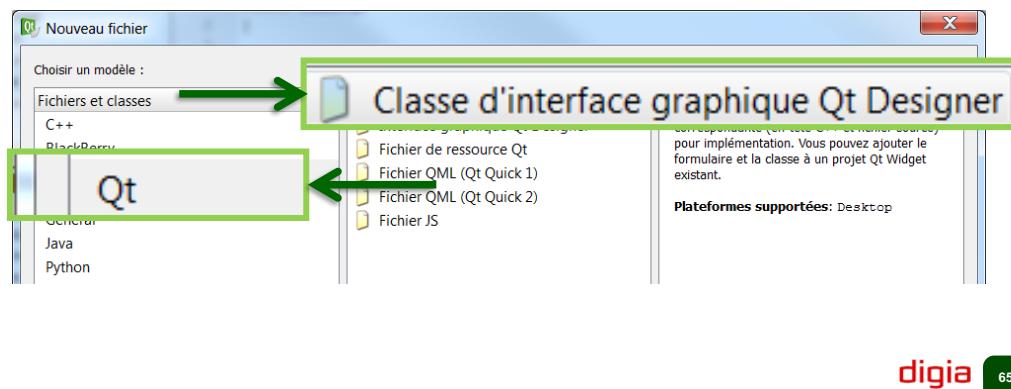
- Hérite de **QDialog**
- Interface créée avec concepteur ou code
  - **QLineEdit** et **QRadioButton** sont les “sorties”
  - Boutons pour accepter ou rejeter

**digia** 64



## Boîte de dialogue : exemple

- Crédation via concepteur
  - Ajout dans une application de type `QMainWindow`
  - Menu du projet ⇒ Ajouter|nouveau...



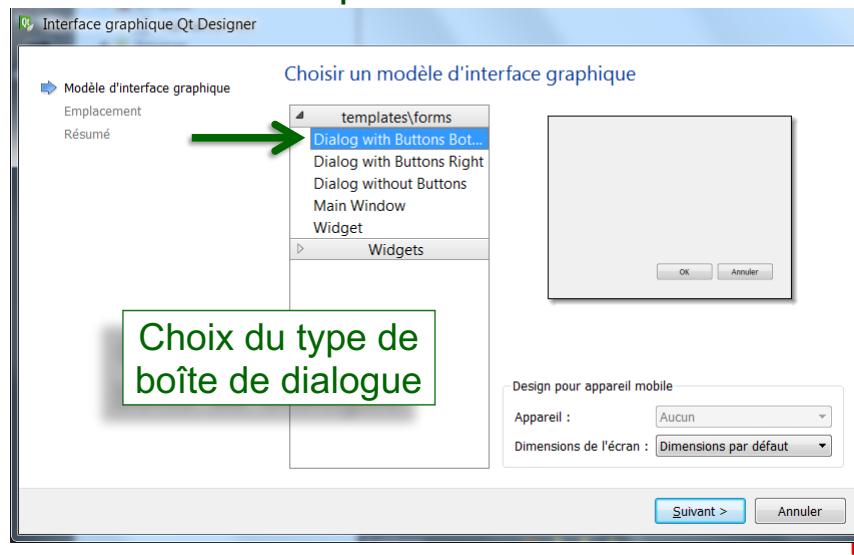
digia

65



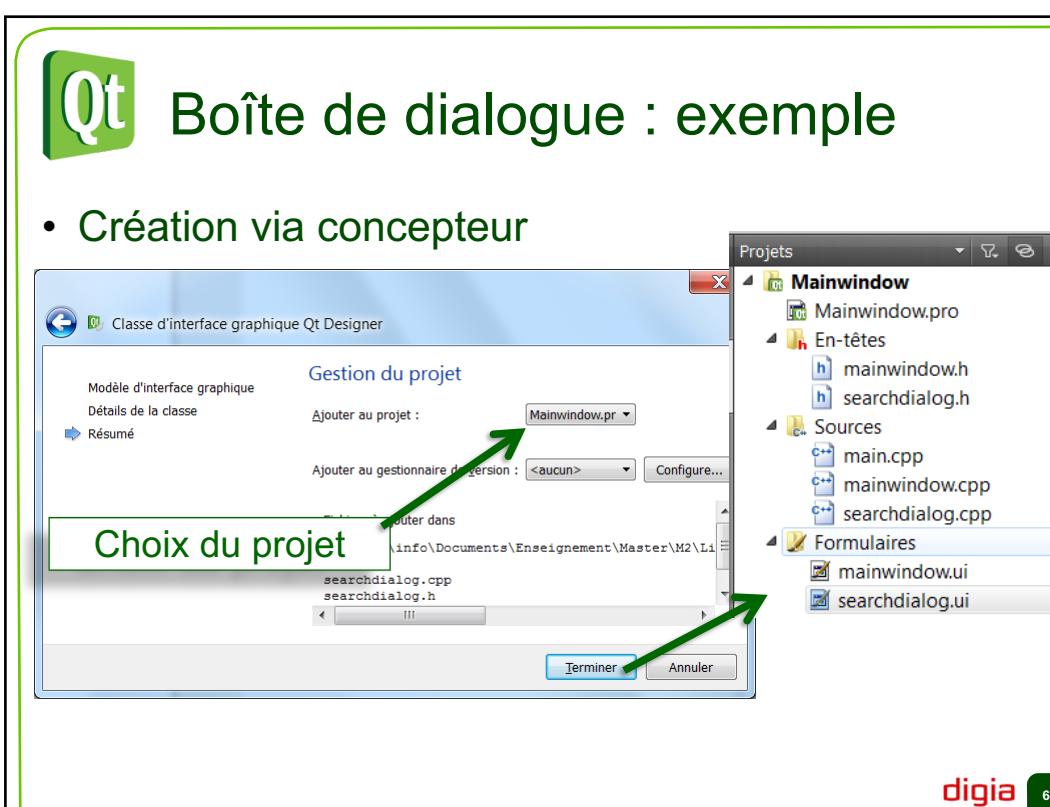
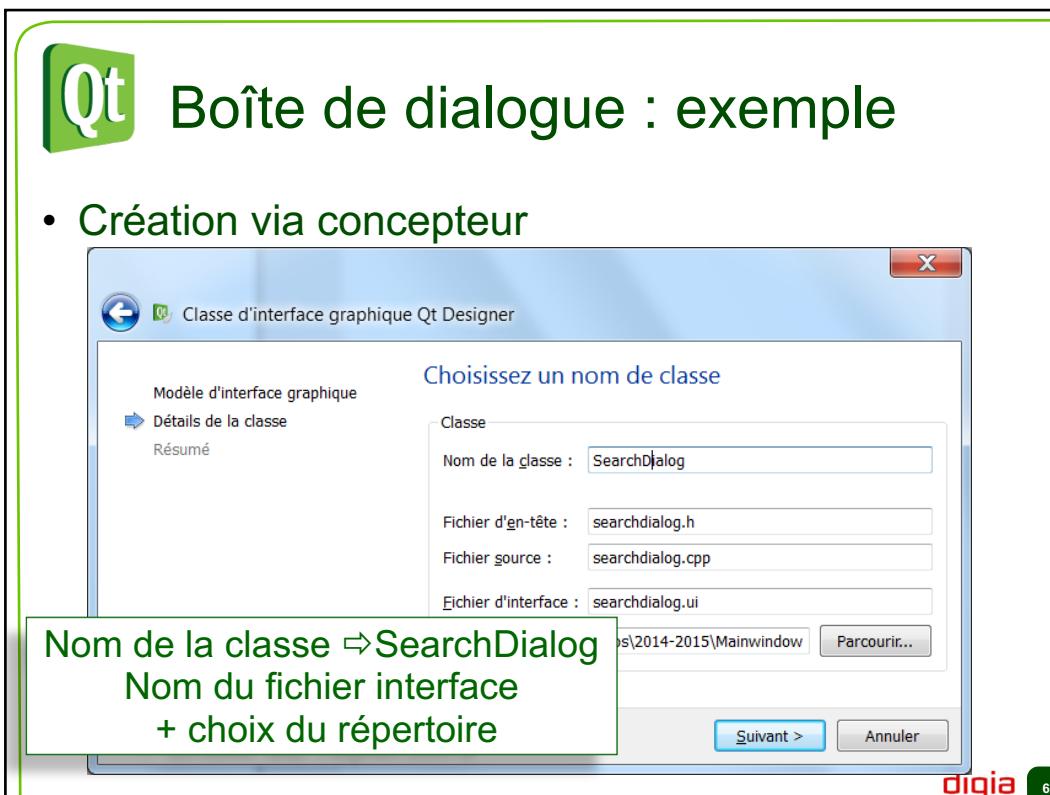
## Boîte de dialogue : exemple

- Crédation via concepteur



digia

66





## Boîte de dialogue : exemple

- Ajout d'une boîte SearchDialog

```
class SearchDialog : public QDialog
{
    Q_OBJECT
public:
    explicit SearchDialog(const QString &initialText,
                          bool isBackward, QWidget *parent = 0);

    bool isBackward() const;
    const QString &searchText() const;

private:
    Ui::SearchDialog *ui;
};
```

Initialiser le dialogue dans le constructeur

Fonctions de consultation pour un accès propre aux données

**digia**

69



## Boîte de dialogue : exemple

```
SearchDialog::SearchDialog(const QString &initialText,
                           bool isBackward, QWidget *parent) :
    QDialog(parent), ui(new Ui::SearchDialog)
{
    ui->setupUi(this);

    ui->searchText->setText(initialText);
    if(isBackward)
        ui->directionBackward->setChecked(true);
    else
        ui->directionForward->setChecked(true);
}

bool SearchDialog::isBackward() const
{
    return ui->directionBackward->isChecked();
}

const QString &SearchDialog::searchText() const
{
    return ui->searchText->text();
}
```

Initialiser le dialogue

Fonction de consultation

**digia**

70



## Boîte de dialogue : exemple

- Utilisation simple de la boîte de dialogue
  - `QDialog::exec()` ⇒ affichage en mode modal
  - retourne `QDialog::Accepted` OU `QDialog::Rejected`

```
void MainWindow ::myFunction()
{
    SearchDialog dlg("", false, this); ←
    if(dlg.exec() == QDialog::Accepted)
    {
        QString text = dlg.searchText();
        bool backwards = dlg.isBackward();
        ...
    }
}
```

Initialiser le dialogue dans le constructeur

digia 71



## Boîtes de dialogue standard

- `QFileDialog` ⇒ sélection fichiers / répertoires

```
QString fileName =
    QFileDialog::getOpenFileName(this, tr("Open File"));
if(!fileName.isNull()) {
    // do something useful
}
```

- `QMessageBox` ⇒ informer/interroger

```
QMessageBox::StandardButton ret =
    QMessageBox::question(parent, title, text);
if(ret == QMessageBox::Ok) {
    // do something useful
}
```

digia 72



## Boîtes de dialogue standard

- QProgressDialog ⇒ feedback sur opération lente

```
QProgressDialog dialog("Copy", "Abort", 0, count, this);
dialog.setWindowModality(Qt::WindowModal);
for (int i = 0; i < count; i++) {
    dialog.setValue(i);
    if (dialog.wasCanceled()) { break; }
    //... copy one file
}
dialog.setValue(count); // ensure set to maximum
```

digia 73



## Boîtes de dialogue standard

- QErrorMessage ⇒ comme QMessageBox

- Avec une case à cocher

```
m_error = new QErrorMessage(this);
m_error->showMessage(message, type);
```



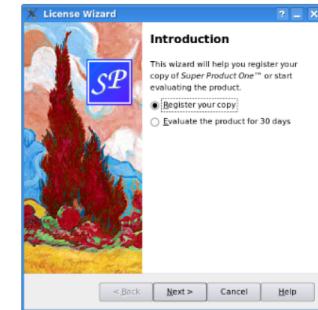
- QInputDialog ⇒ saisie de texte
- QColorDialog ⇒ définition de couleur
- QFontDialog ⇒ définition de police

digia 74



## Boîtes de dialogue avancé

- QWizard ⇒ succession de pages
  - Linéaire ou non
  - Titre, sous-titre
  - Logo, bannière, filigrane, arrière plan
  - Accès à une page par identifiant
  - Une page = QWizardPage



digia 75



## Boîtes de dialogue avancé

- QWizard ⇒ succession de pages

```
QWizardPage *createIntroPage() {
    QWizardPage *page = new QWizardPage;
    page->setTitle("Introduction");
// create widgets and layout them

    return page;
}

QWizardPage *createRegistrationPage() { ... }

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QWizard wizard;
    wizard.setWindowTitle("License Wizard");
    wizard.addPage(createIntroPage());
    wizard.addPage(createRegistrationPage());
    wizard.show();
    return app.exec();
}
```



digia 76

## Qt QAction : introduction

- Plusieurs composants pour une même action

• QAction pour représenter toutes ces interfaces

- Gère les conseils des barres d'état, aide, etc.

digia 77

## Qt QAction : introduction

- QAction encapsule les paramètres
  - Des menus, barres d'outils et raccourcis claviers
- Les propriétés courantes
  - `text` ⇒ texte utilisé partout
  - `icon` ⇒ l'icône à utiliser partout
  - `shortcut` ⇒ raccourci
  - `checkable` ⇒ vrai si action cochable
  - `checked` ⇒ état courant de la coche
  - `toolTip` ⇒ conseil affiché (survol+attente)
  - `statusTip` ⇒ conseil barre d'état (survol,sans attente)

digia 78

## Qt QAction : définition

- Par programme

```

QAction *action = new QAction(parent);
action->setText("text");
action->setIcon(QIcon(":/icons/icon.png"));
action->setShortcut(QKeySequence("Ctrl+G"));
action->setData(myDataQVariant);
    
```

Créer une nouvelle action

Définir les propriétés pour le texte, l'icône et le raccourci clavier

Associer des données via objet QVariant

**digia** 79

## Qt QAction : définition

- En utilisant le concepteur

The screenshot shows the Qt Designer application window. At the top, there is a toolbar with various icons. Below the toolbar, a table-like interface lists items with columns for Nom, Utilisé, Texte, Raccourci, Vérifiable, and Info-bulle. A red circle highlights the first icon in the Utilisé column. In the foreground, a modal dialog box titled "Nouvelle action" is open. This dialog contains fields for Texte, Nom de l'objet, ToolTip, Icon theme, Icône (with a dropdown menu showing "Arrêt normal"), Checkable (with a checkbox), and Shortcut. At the bottom of the dialog are OK and Annuler buttons. A red arrow points from the "Editeur d'action" tab in the toolbar to the "Nouvelle action" dialog.

**digia** 80



## QAction : liens avec composants

- Par programme ⇒ `addAction`

```
myMenu->addAction(action);
myToolBar->addAction(action);
```

- Avec le concepteur ⇒ “drag and drop” actions
  - Sur barre d’outils
  - Sur menu

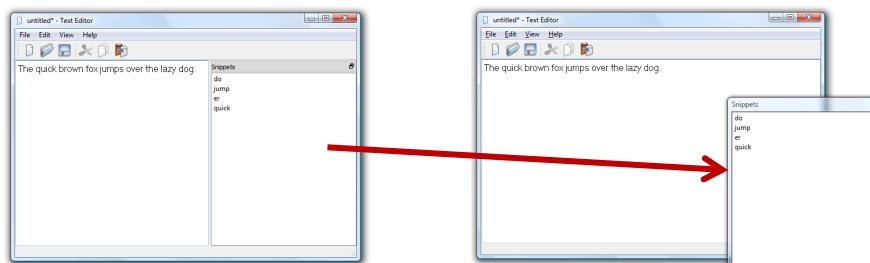
**digia** 81



## Widgets Dock



- Widgets détachables
  - placées sur les bords d’une `QMainWindow`
  - définies dans `QDockWidget`
  - `QMainWindow::addDockWidget`
    - Ajouter les widgets détachables à la fenêtre



**digia** 82

## Qt Widgets Dock : par programme

```

void MainWindow::createDock()
{
    QDockWidget *dock = new QDockWidget("Dock", this);
    dock->setFeatures(QDockWidget::DockWidgetMovable | QDockWidget::DockWidgetFloatable);

    dock->setAllowedAreas(Qt::LeftDockWidgetArea | Qt::RightDockWidgetArea);
    dock->setWidget(actualWidget);
    ...
    addDockWidget(Qt::RightDockWidgetArea, dock);
}

```

Un nouveau dock avec un titre

Peut-être déplacé et flottant, mais pas fermé

La vrai widget est ce qui interagit avec l'utilisateur

Peut être ancré sur les côtés droit et gauche

Ajouter dock à la fenêtre



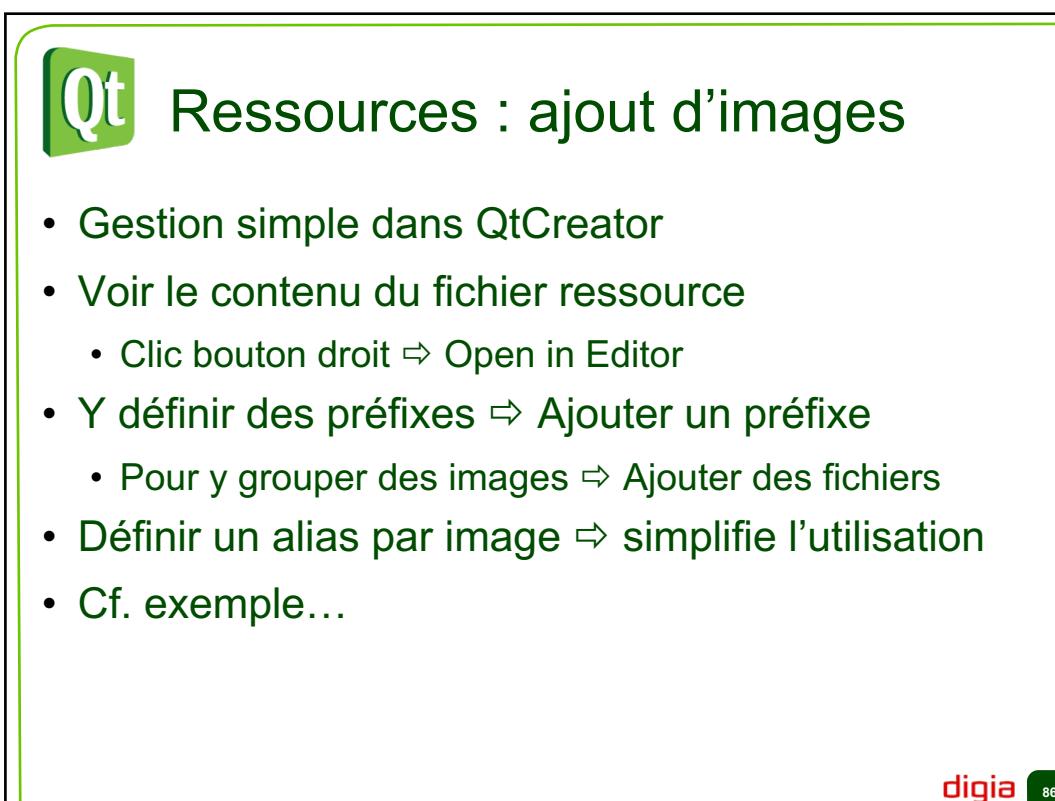
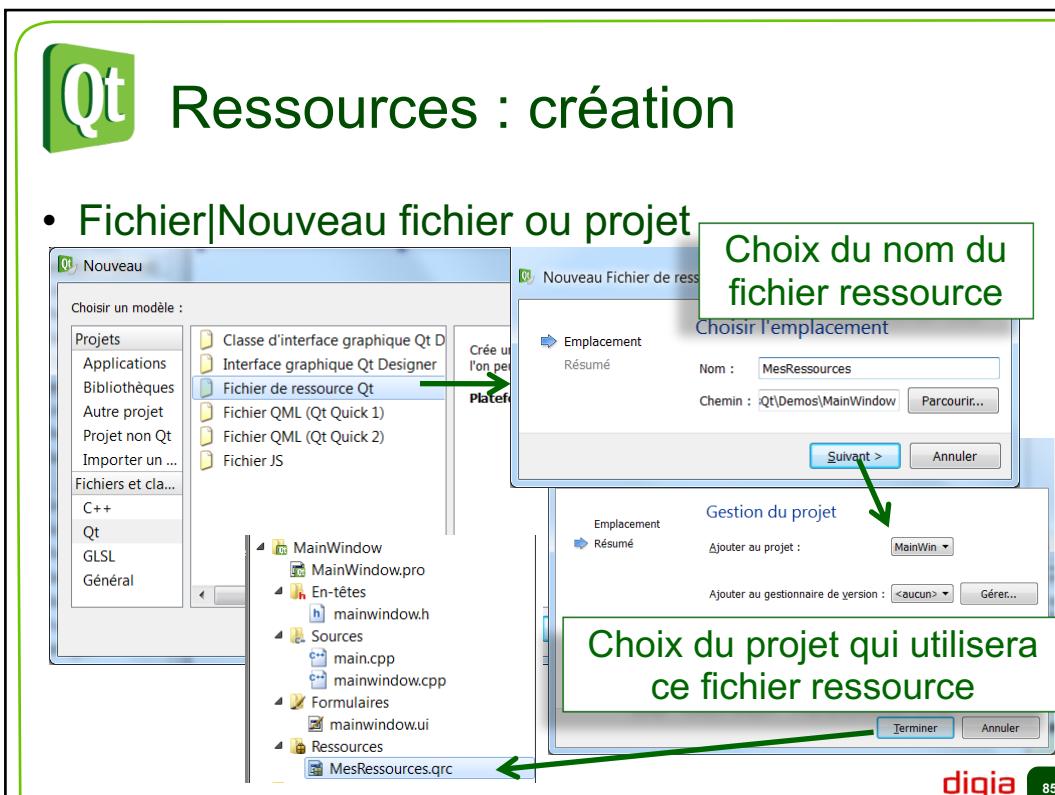
83

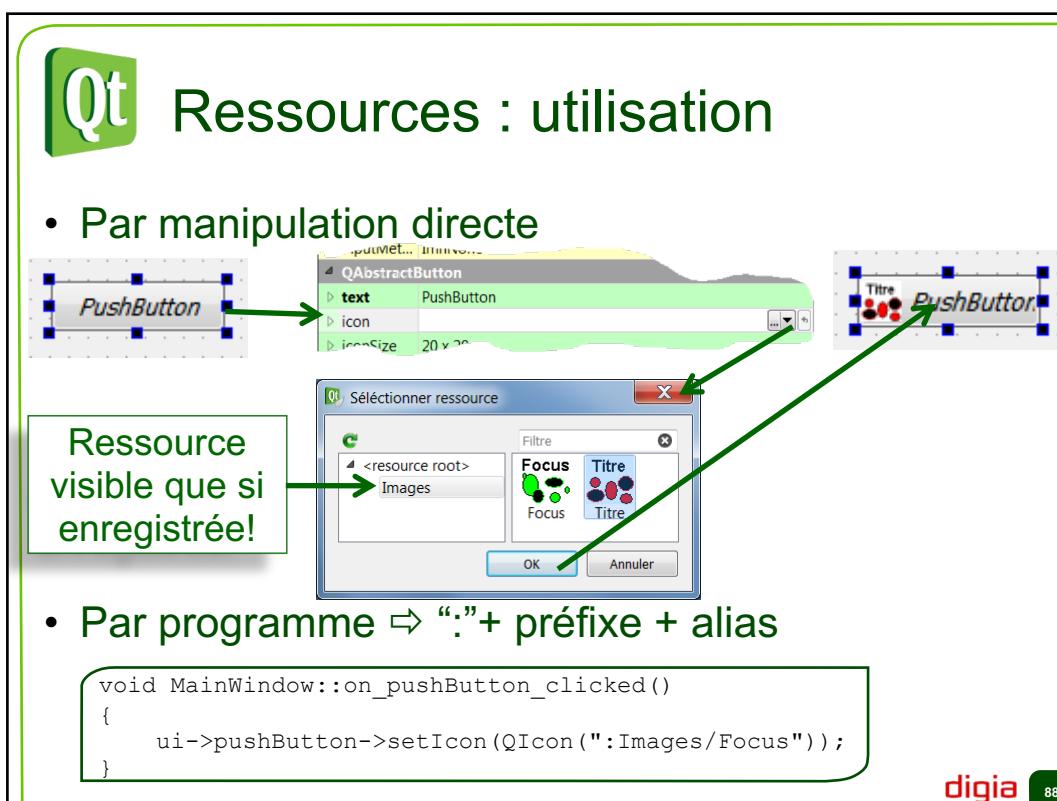
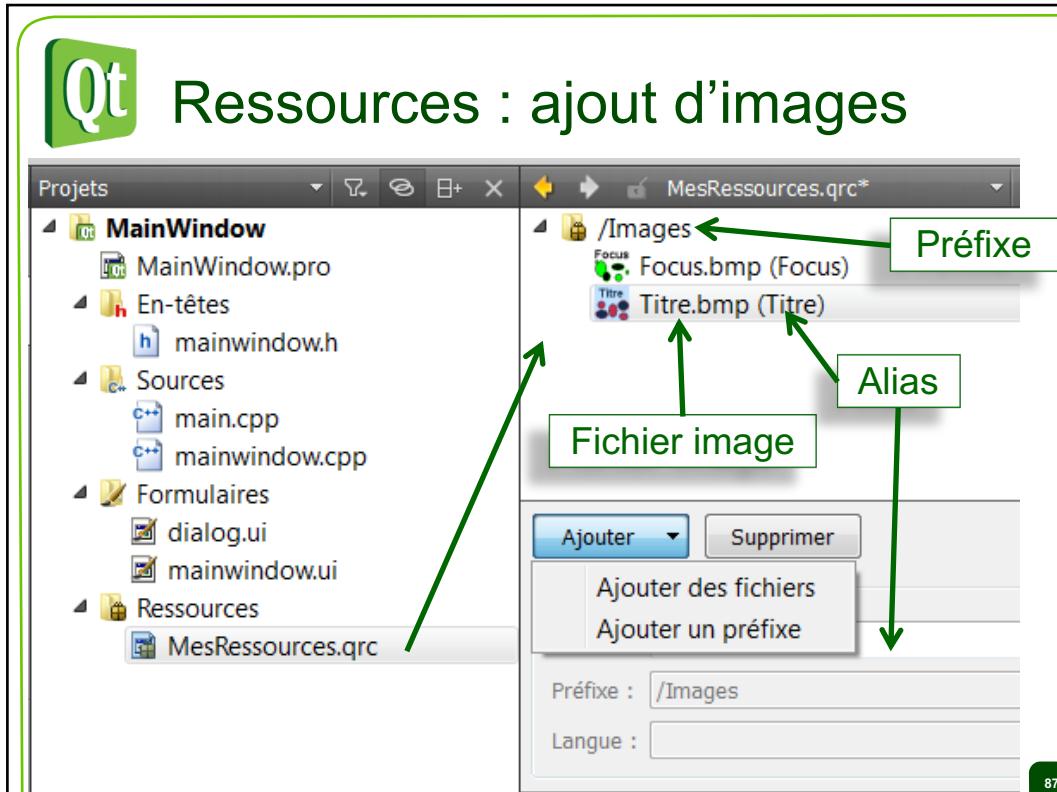
## Qt Gérer des ressources

- Icônes dans fichier ressource ⇒ dans exécutable
  - Évite de gérer plusieurs fichiers
  - Pas besoin de définir le chemin pour les icônes
  - Tout est intégré à la construction
  - ...
- Pas que des icônes dans ressources

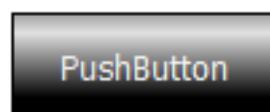



84





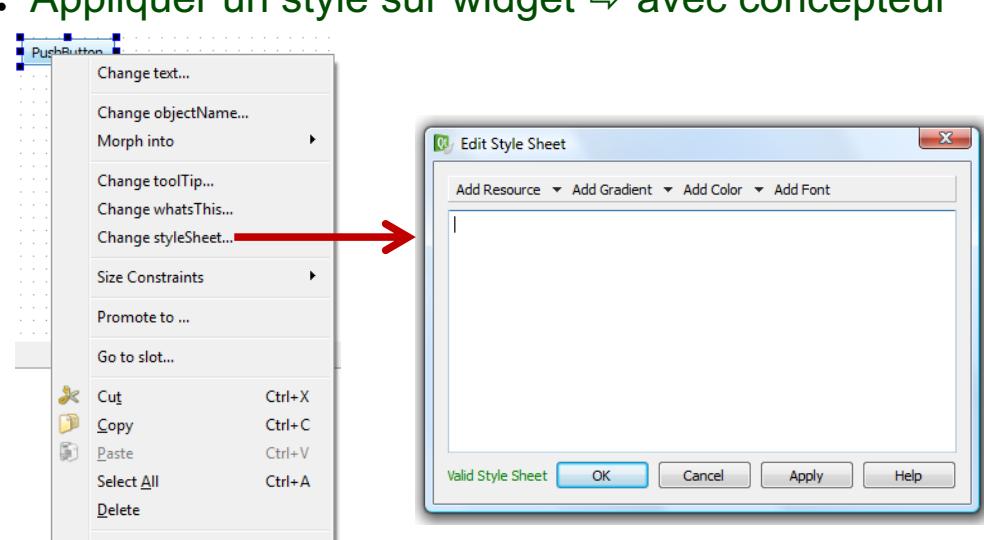
## Qt Feuille de style

- Propriété `stylesheet` pour chaque `QWidget`
  - Inspirée des CSS
- Utilisation
  - Pour mettre en évidence
  - Pour de petites variations
- Pour une révision complète de toute l'interface
 

**digia** 89

## Qt Feuille de style

- Appliquer un style sur widget ⇒ avec concepteur



**digia** 90



## Feuille de style

- Appliquer un style à toute l'application

- `QApplication::setStyleSheet`

Sélection de la classe

```
QLineEdit { background-color: yellow }
QLineEdit#nameEdit { background-color: yellow }
```

Sélection de l'objet par son nom

```
QTextEdit, QListWidget {
    background-color: white;
    background-image: url(draft.png);
    background-attachment: scroll;
}
```

Utilise une image

```
QGroupBox {
    background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                      stop: 0 #E0E0E0, stop: 1 #FFFFFF);
    border: 2px solid gray;
    border-radius: 5px;
    margin-top: 1ex;
}
```

Construit dans l'éditeur de style du concepteur

digia

91



## Validateurs

- Rôles

- Valider saisie de texte  $\Rightarrow$  `validate()`

- 3 valeurs renvoyées

- `Acceptable`  $\Rightarrow$  saisie correcte

- `Intermediate`  $\Rightarrow$  saisie presque juste

- `Invalid`  $\Rightarrow$  saisie incorrecte

- Corriger les erreurs de saisie  $\Rightarrow$  `fixup()`

À surcharger si crée nouveau validateur

- Widgets qui les utilisent classiquement

- `QLineEdit`

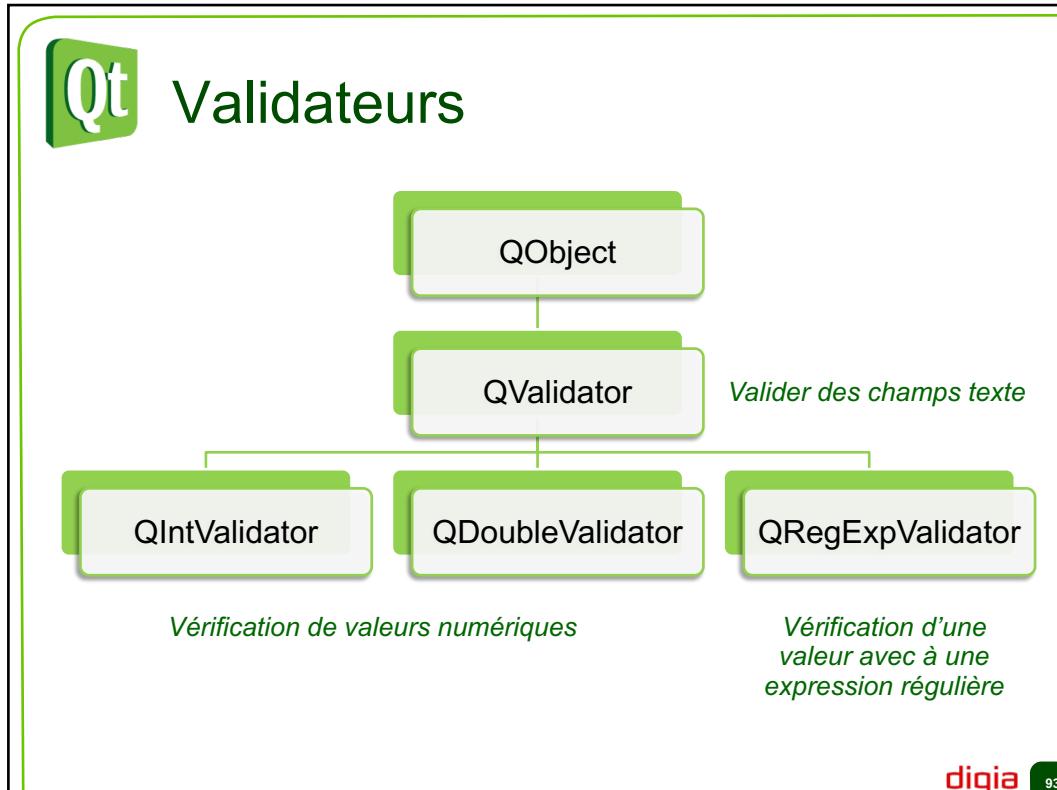
Lien  $\Rightarrow$  méthode `setValidator`

- `QSpinBox`

- `QComboBox`

digia

92



digia 93

## Qt Validator : exemple

- Avec expression régulière
  - Widget : zone de saisie
  - Validité : que des lettres minuscules

```

lineEdit = new QLineEdit(centralwidget);
QRegExp regExp("[a-z]+");
QRegExpValidator *validator = new QRegExpValidator( regExp,0 );
lineEdit->setValidator(validator);

```

**digia** 94