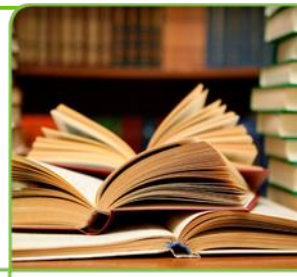
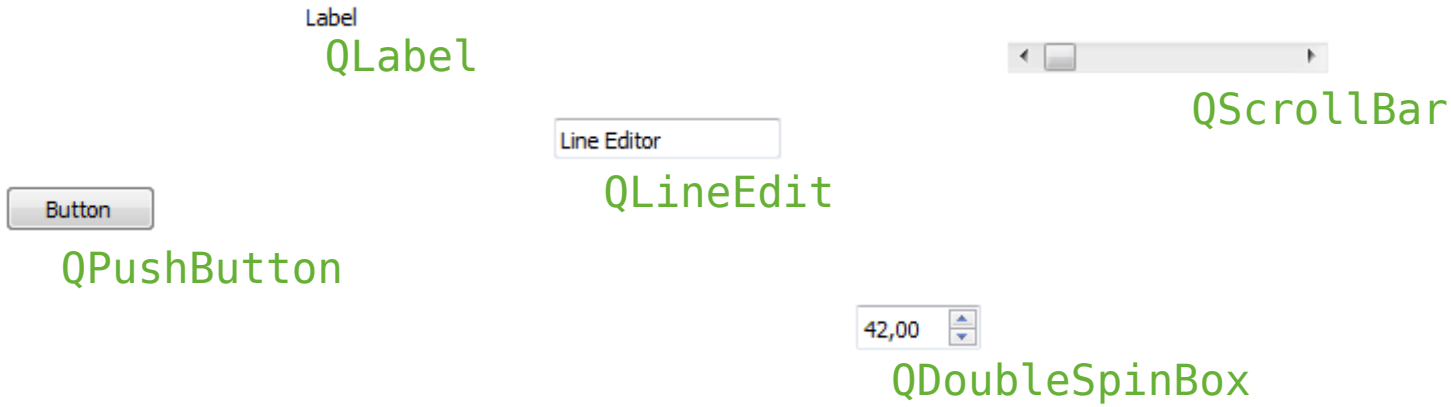
Widgets et Layouts

digia



Composants d'IHM

- Les widgets



- 46 widgets avec QtDesigner
- 59+ composants héritants de `QWidget`

Caractéristiques d'un Widget

- Il occupe une zone rectangulaire à l'écran
- Il reçoit des événements des périphériques d'entrée
- Il émet des signaux lors de changements notables

- Il fait partie d'une structure hiérarchique
- Il peut contenir d'autres widgets

Les différentes fenêtres

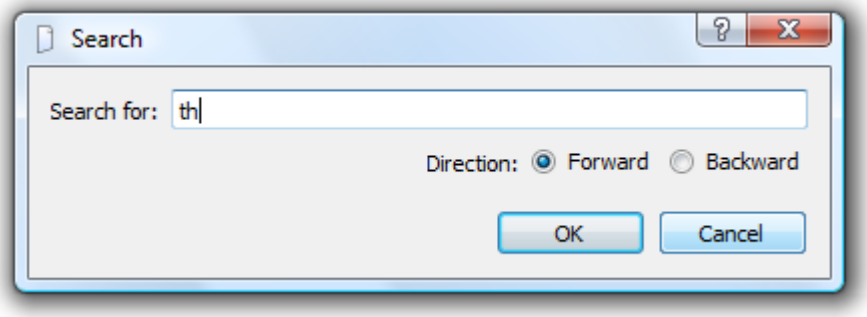
- Des Widgets sans widget parent deviennent automatiquement des fenêtres
  - `QWidget` – Une fenêtre ordinaire
  - `QDialog` – Une fenêtre de dialogue, généralement utilisé pour afficher des boutons OK, Cancel, etc
  - `QMainWindow` – Une fenêtre application avec des menus, des toolbars, statusbar, etc
- `QDialog` et `QMainWindow` héritent de `QWidget`

QWidget comme fenêtre

- Utilisez `setWindowModality` pour rendre la fenêtre “modal”
  - `NonModal` – toutes les fenêtres peuvent être utilisées en même temps.
  - `WindowModal` – La fenêtre parent est bloquée
  - `ApplicationModal` – Toutes les autres fenêtres sont bloquées.

QDialog comme fenêtre

- Par exemple pour une fenêtre de recherche

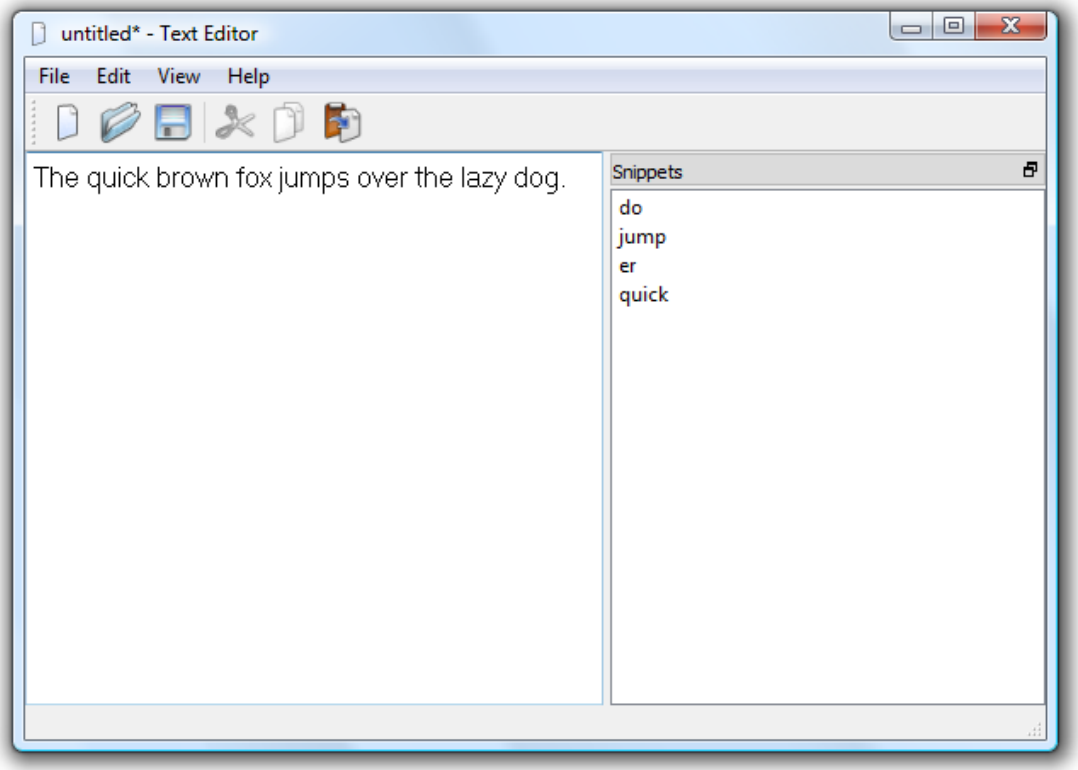


- Hérite de `QDialog`

QMainWindow comme fenêtre

`QMainWindow` peut contenir

- Menus
- Toolbar
- Statusbar
- Docks
- Central widget



Layouts

- Les différents layouts

QVBoxLayout

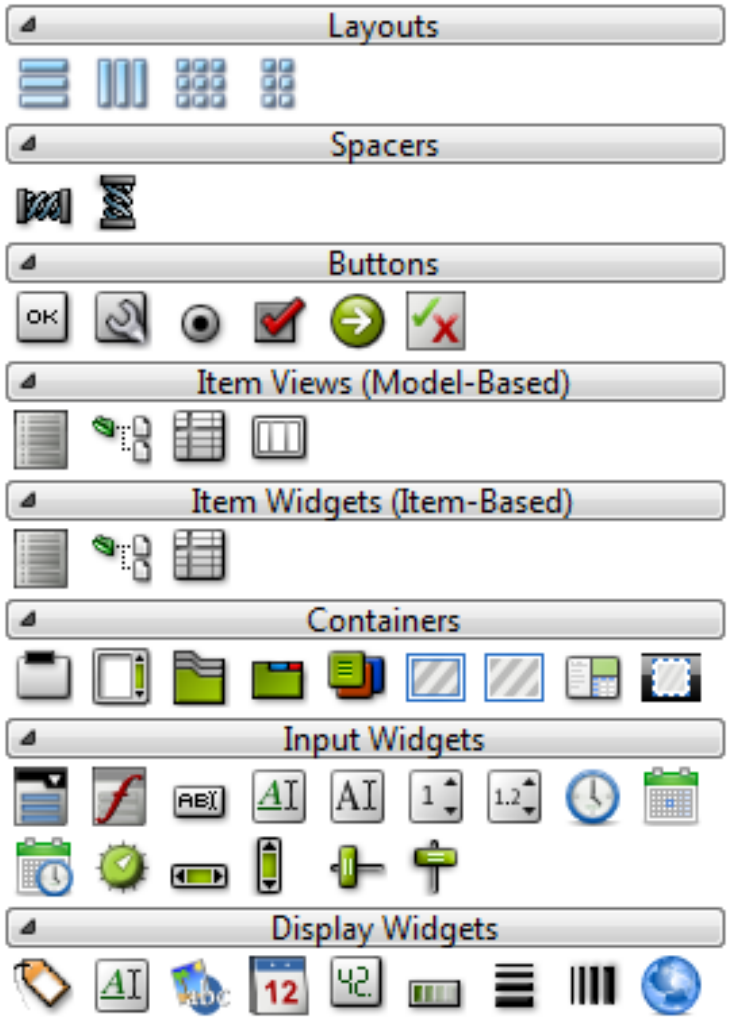
QHBoxLayout

QGridLayout

- Layouts et widgets “s'arrange” pour les tailles et les positions
- Des spacer (ressorts) peuvent être utiliser pour combler les vides.

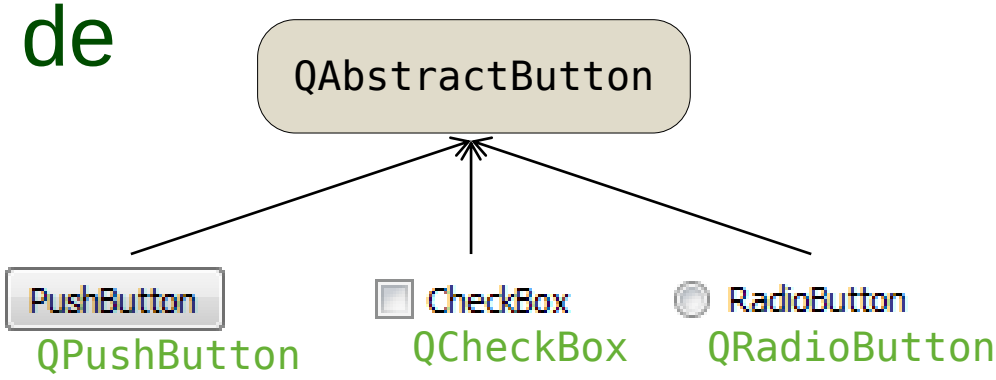
Les Widgets les plus communs

- La plupart des widgets sont disponibles avec QtDesigner



Les boutons

- Tous les boutons héritent de `QAbstractButton` .



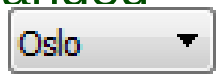
- Signaux
  - `clicked()`
  - `toggled(bool)` – emit lorsque l'état coché du bouton à changé.

- Propriétés
  - `checkable` – true if the button can be checked. Makes a push button toggle.
  - `checked` – true when the button is checked.
  - text – the text of the button.
  - icon – an icon on the button (can be displayed together with text).

les widgets “déroulants”

QListWidget

- `QListWidget` utilisé pour afficher une liste d'items
- Ajouter des items
  - `addItem(QString)` – appends an item to the end of the list
  - `insertItem(int row, QString)` – inserts an item at the specified row
- Selection
  - `selectedItems` – returns a list of `QListWidgetItem`s used `QListWidgetItem::text` to determine the text
- Signaux
  - `itemSelectionChanged` – emitted when the selection is changed



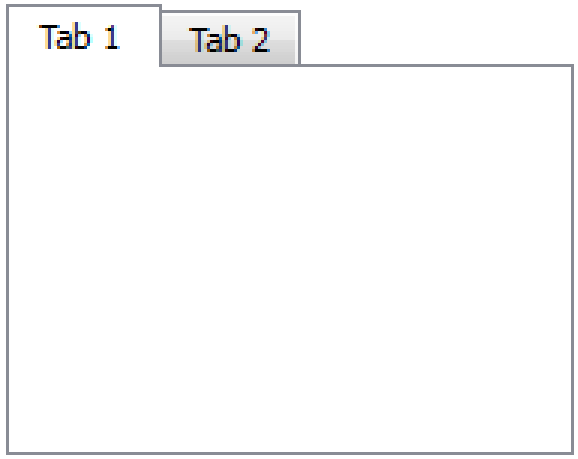
QComboBox

- `QComboBox` est utilisé pour faire des listes déroulantes.

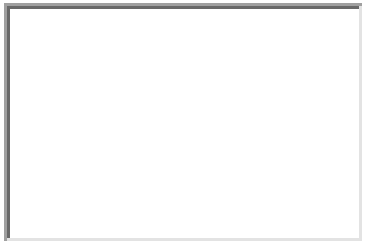
Widgets containers

- Utilisé pour structurer l'interface
- Composant “passifs”

QGroupBox



QTabWidget



QFrame

Widgets d'entrée

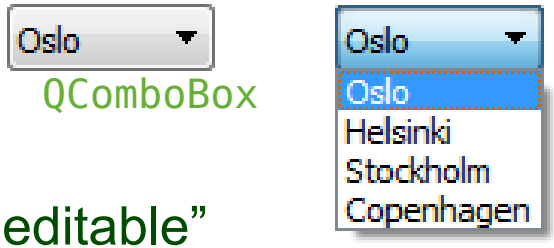
- `QLineEdit` pour une ligne de saisie de texte
- Signaux:
  - `textChanged(QString)` - emitted when the text is altered
  - `editingFinished()` - emitted when the widget is left
  - `returnPressed()` - emitted when return is pressed
- Propriétés
  - `text` – the text of the widget
  - `maxLength` – limits the length of the input
  - `readOnly` – can be set to true to prevent editing (still allows copying)

`QLineEdit`

Widgets d'entrée

- `QTextEdit` ou `QPlainTextEdit` pour une saisie multiligne
- Signaux
  - `textChanged()` - emitted when the text is altered
- Propriétés
  - `plainText` – unformatted text
  - `html` – HTML formatted text
  - `readOnly` – can be set to prevent editing

`QTextEdit`

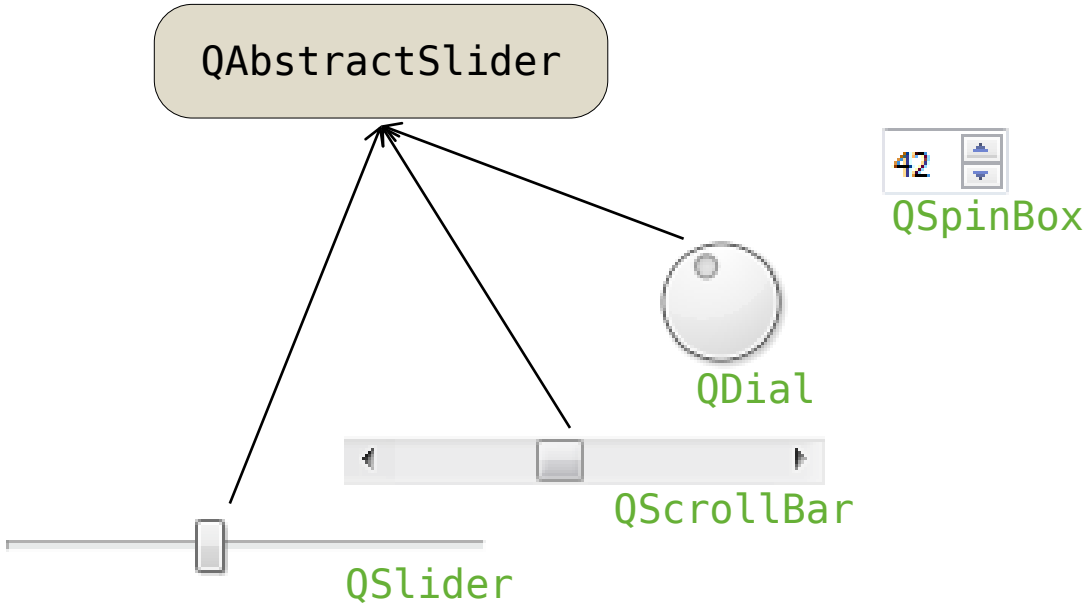


`QComboBox`

- `QComboBox` peut être rendu éditable grâce à la propriété “editable”
- Signaux
  - `editTextChanged(QString)` – emitted while the text is being edited
- Propriétés
  - `currentText` – the current text of the combo box

Widgets d'entrée

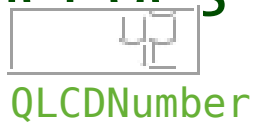
- Il existe plusieurs widgets permettant de modifier des valeurs numériques
- Signaux:
  - `valueChanged(int)` - emitted when the value is updated
- Propriétés
  - `value` – the current value
  - `maximum` – the maximum value
  - `minimum` – the minimum value



Widgets d'affichage

- `QLabel` affiche du texte ou une image
- Propriétés
  - `text` – a text for the label
  - `pixmap` – a picture to show

- `QLCDNumber` peut être utilisé pour afficher des valeurs numériques

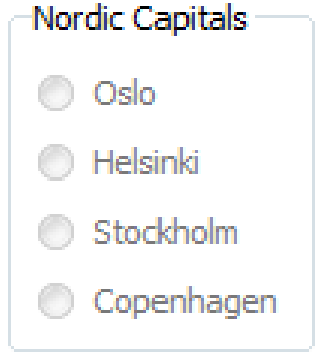


- Propriétés
  - `intValue` – the value shown (set using `display(int)`)

Propriétés

- Tous les widgets ont des propriétés héritées de `QWidget`

- `enabled` – active/désactive les interactions utilisateurs

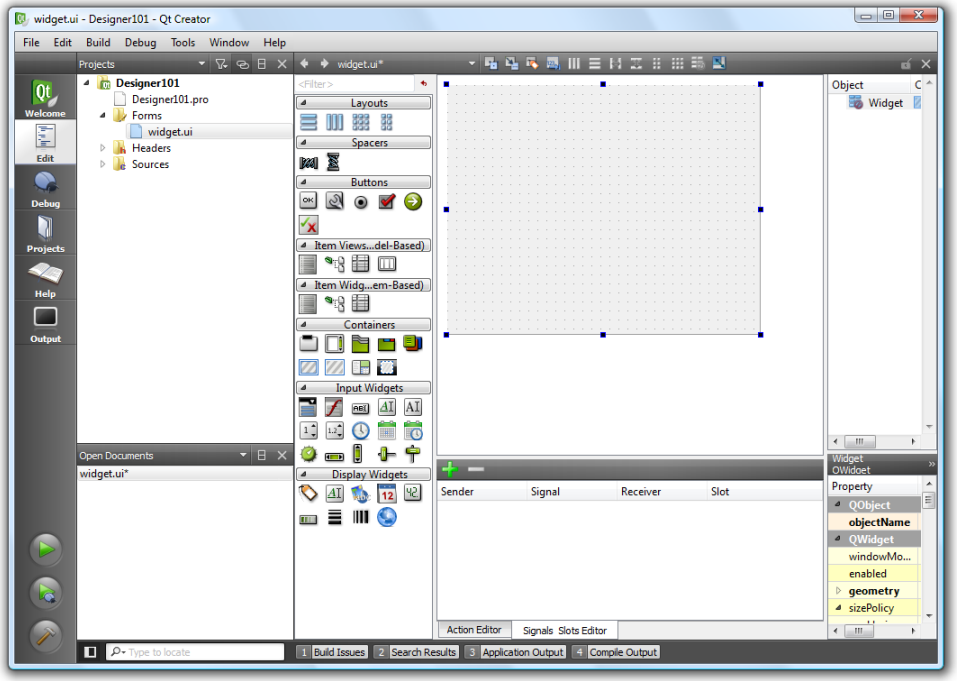


- `visible` – visible ou pas.

- Ses propriétés affectent les widgets présents dans un container.

QtDesigner

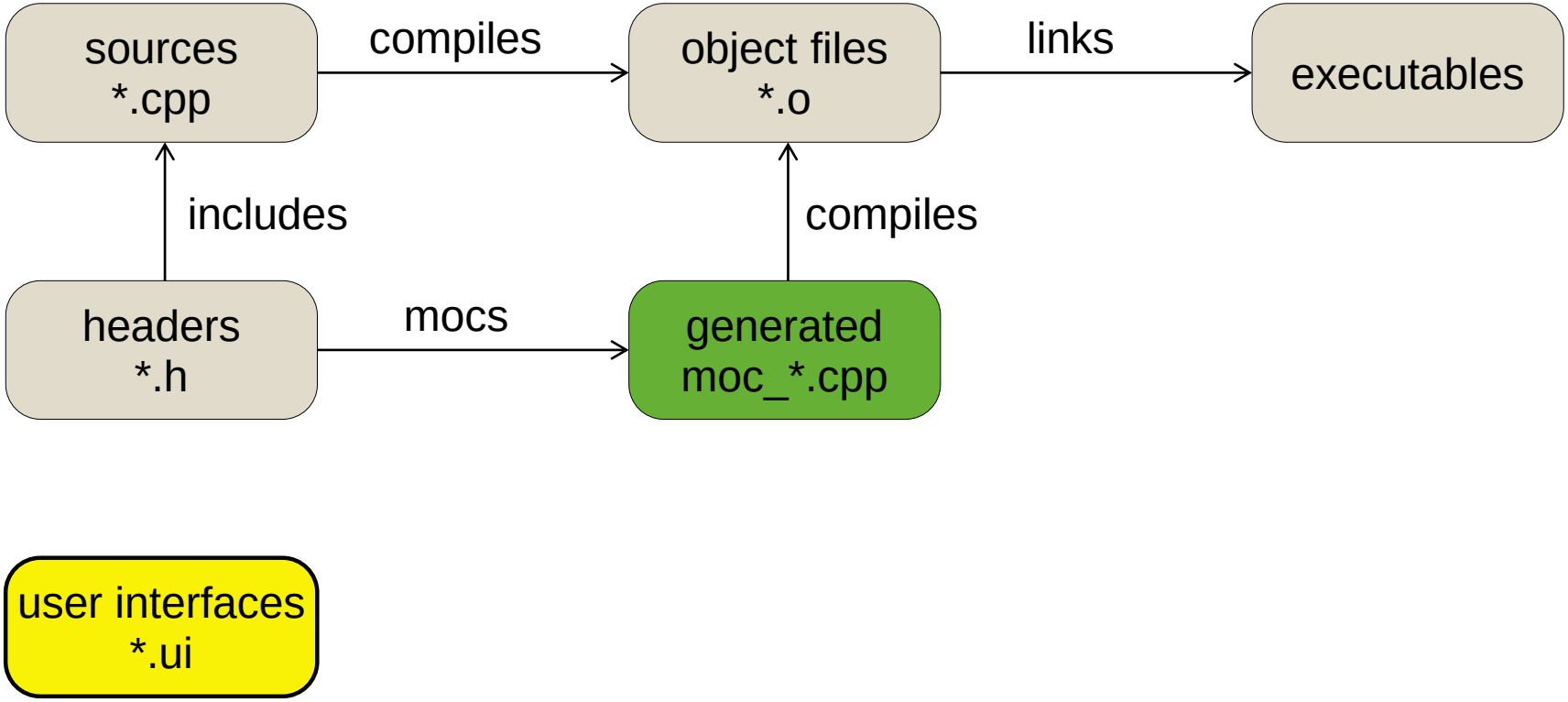
- Historiquement QtDesigner était un outil indépendant de l'outil de développement, mais il est à présent intégré à QtCreator



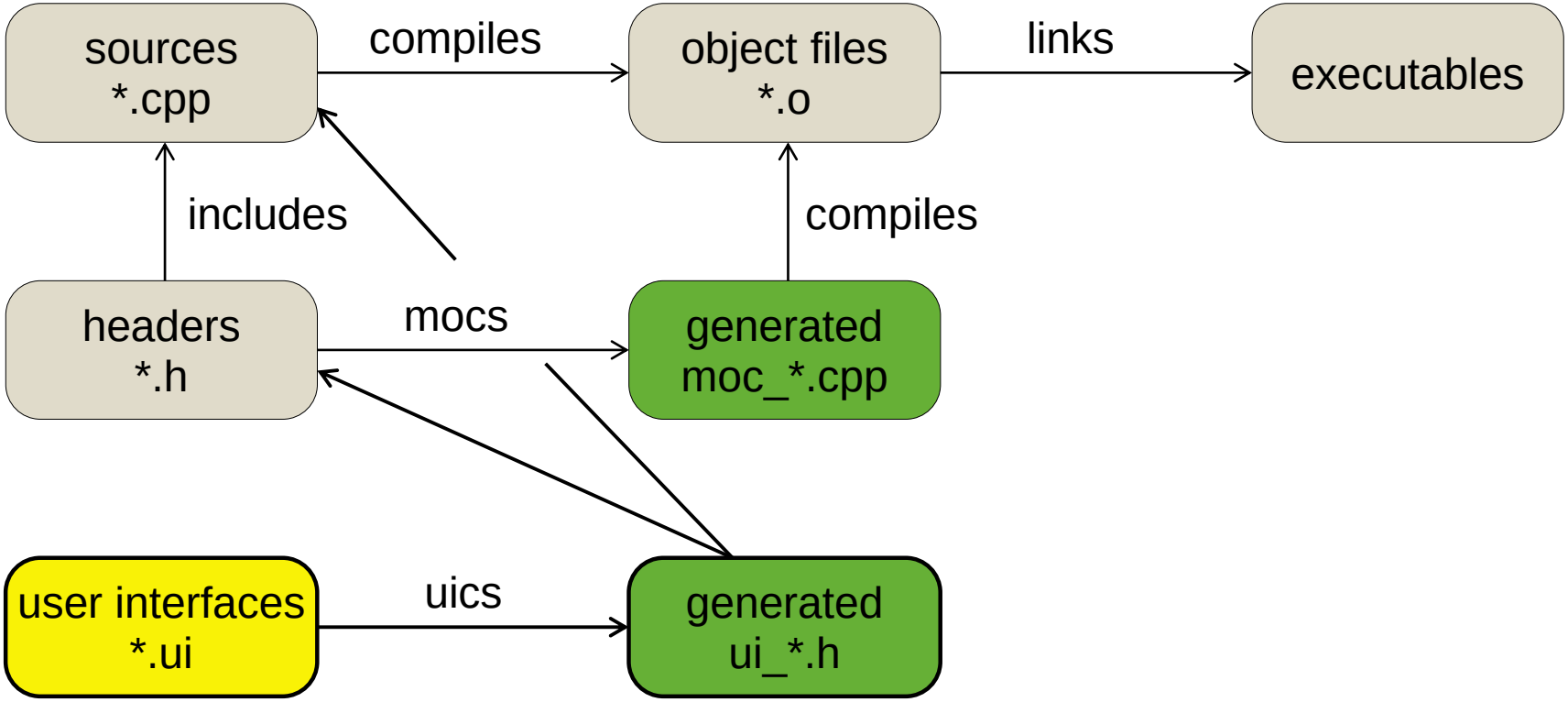
- C'est un éditeur visuel

- Application de layouts
- Création de connexions
- Glisser/déposer d'éléments

QtDesigner


sources
*.cpp
compiles
object files
*.o
links
executables
includes
compiles
headers
*.h
mocs
generated
moc_*.cpp
user interfaces
*.ui

QtDesigner


sources
*.cpp
compiles
object files
*.o
links
executables
includes
compiles
headers
*.h
mocs
generated
moc_*.cpp
user interfaces
*.ui
uics
generated
ui_*.h

Code généré (.h)

Forward declaration of the `Ui::Widget` class

A `Ui::Widget` pointer, `ui`, refers to all widgets

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>

namespace Ui {
    class Widget;
}

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

private:
    Ui::Widget *ui;
};

#endif // WIDGET_H
```

Basically a standard `QWidg`

Code généré (.cpp)

n to the given parent (`this`).

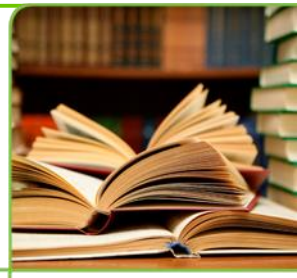
```
#include "widget.h"
#include "ui_widget.h"

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

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

Instanciates the `Ui::Widget` class as `ui`

Deletes the `ui` object

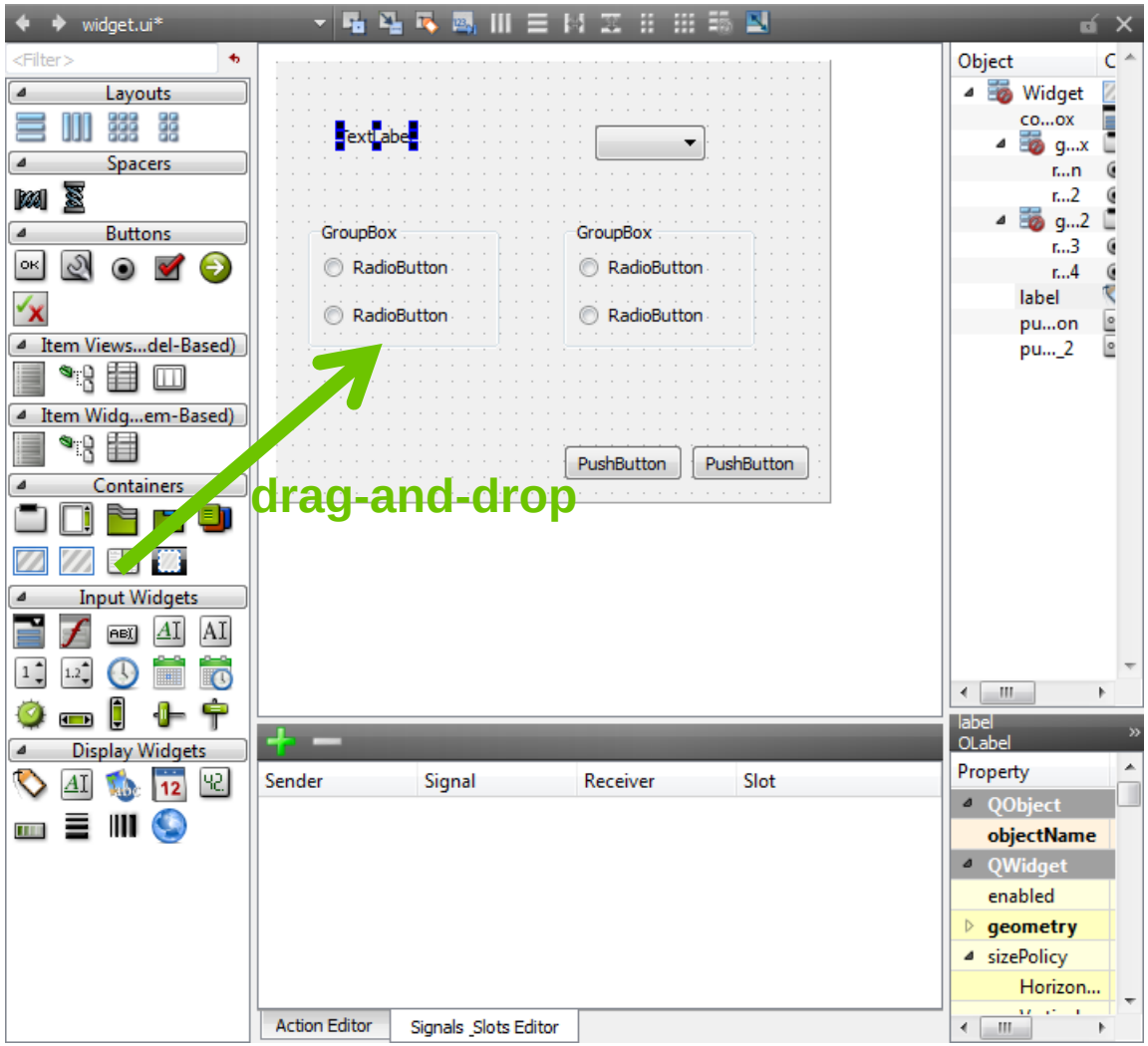


Utilisation de QtDesigner

- Basiquement
  1. Placez les widgets sur l'interface et nommez les.
  2. Arrangez les widgets avec des layout et des ressorts
  3. Créer les connexions entre les signaux et les slots pour chaque widget en ayant besoin.
  4. Coder le contenu des slots

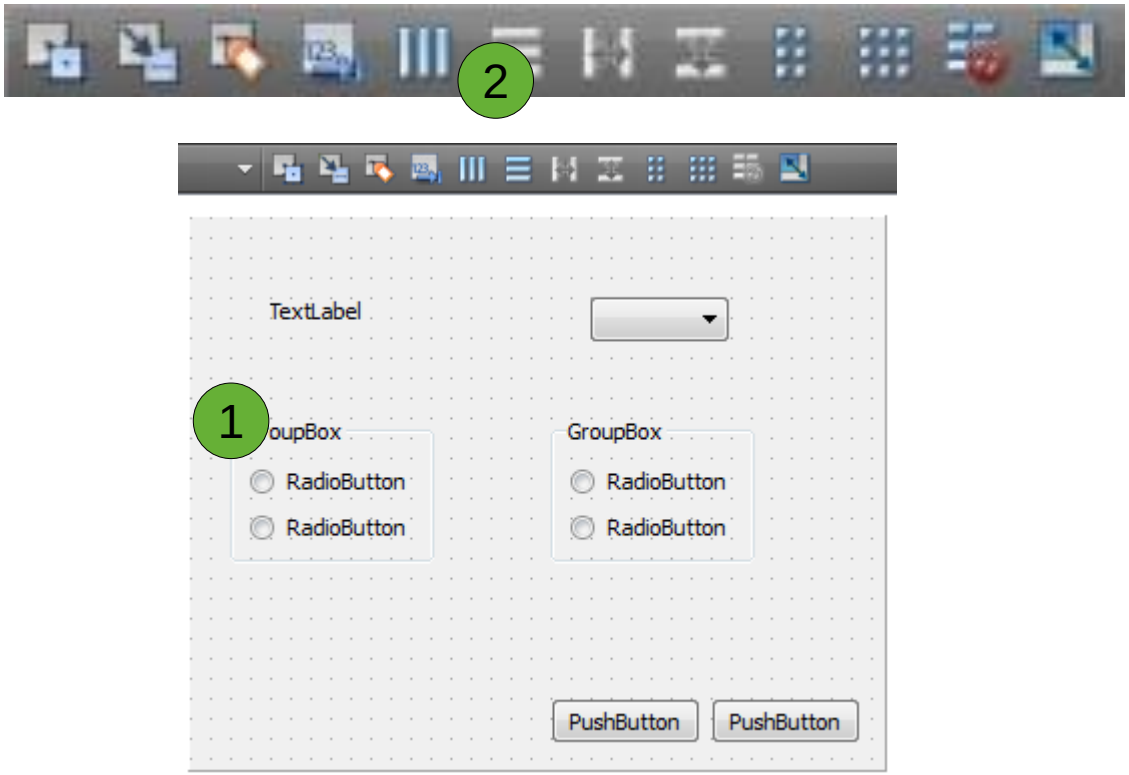
Utilisation de QtDesigner

Placer les widgets



Utilisation de QtDesigner

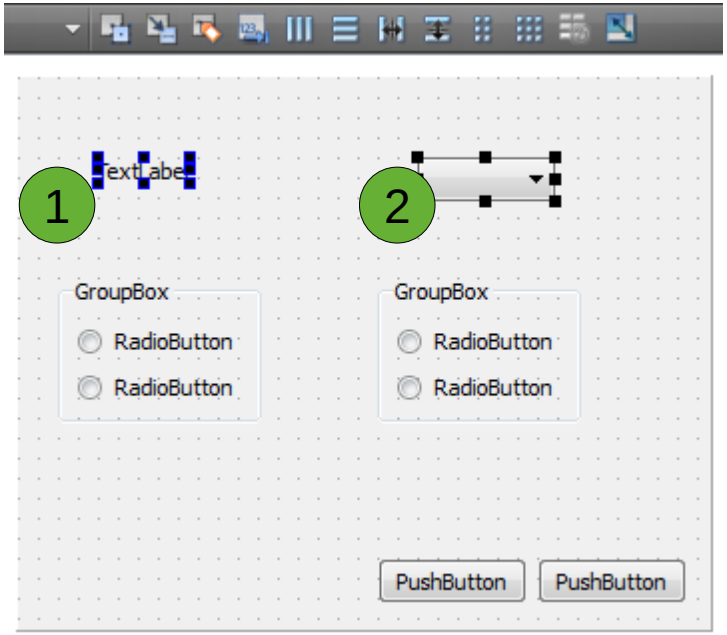
Utilisez des layouts et des ressorts



1. pour chaque group box, **2**. utilisez un vertical box layout

Utilisation de QtDesigner

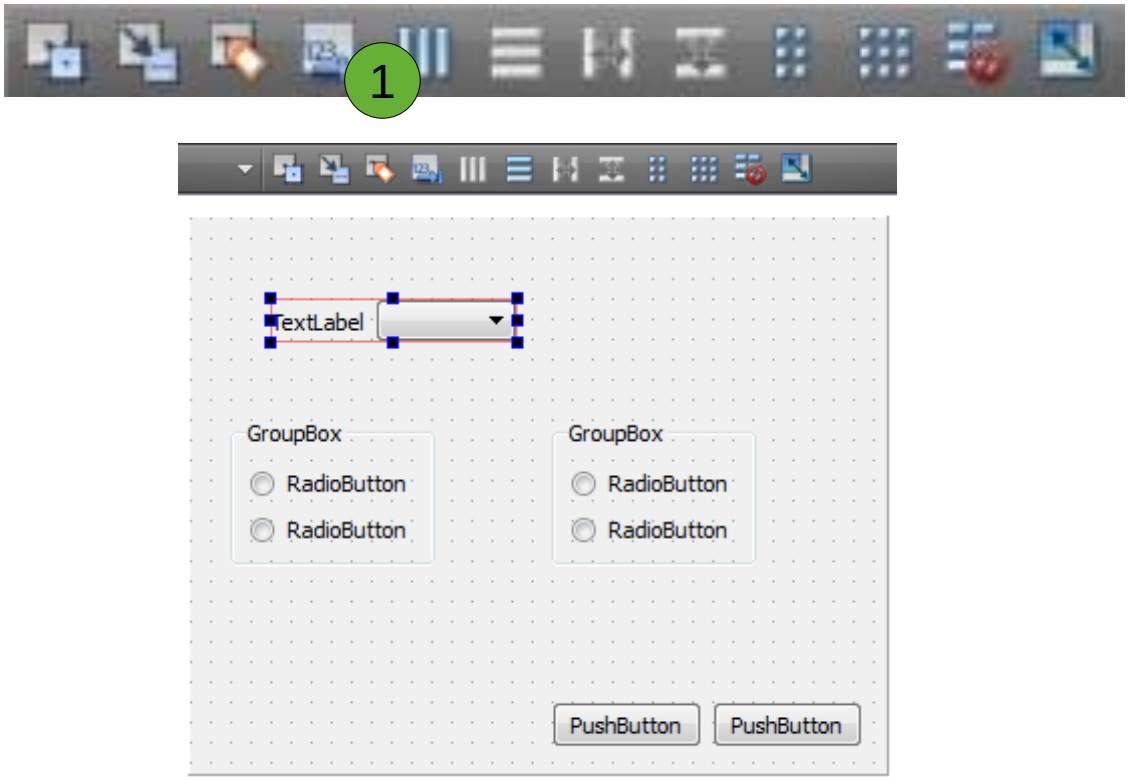
Utilisez des layouts et des ressorts



1. Sélectionnez le label (click), **2**. Sélectionnez aussi le combobox (Ctrl+click)

Utilisation de QtDesigner

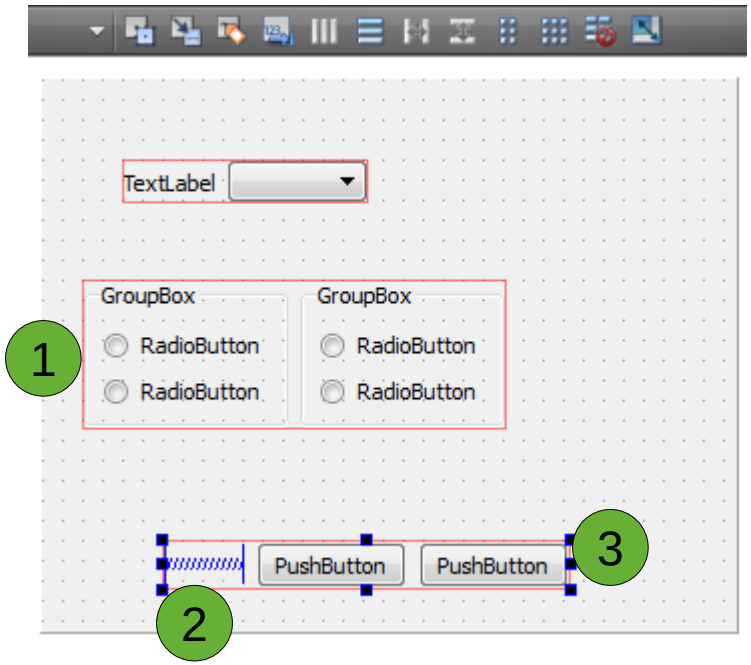
Utilisez des layouts et des ressorts



1. Utilisez un horizontal box layout

Utilisation de QtDesigner

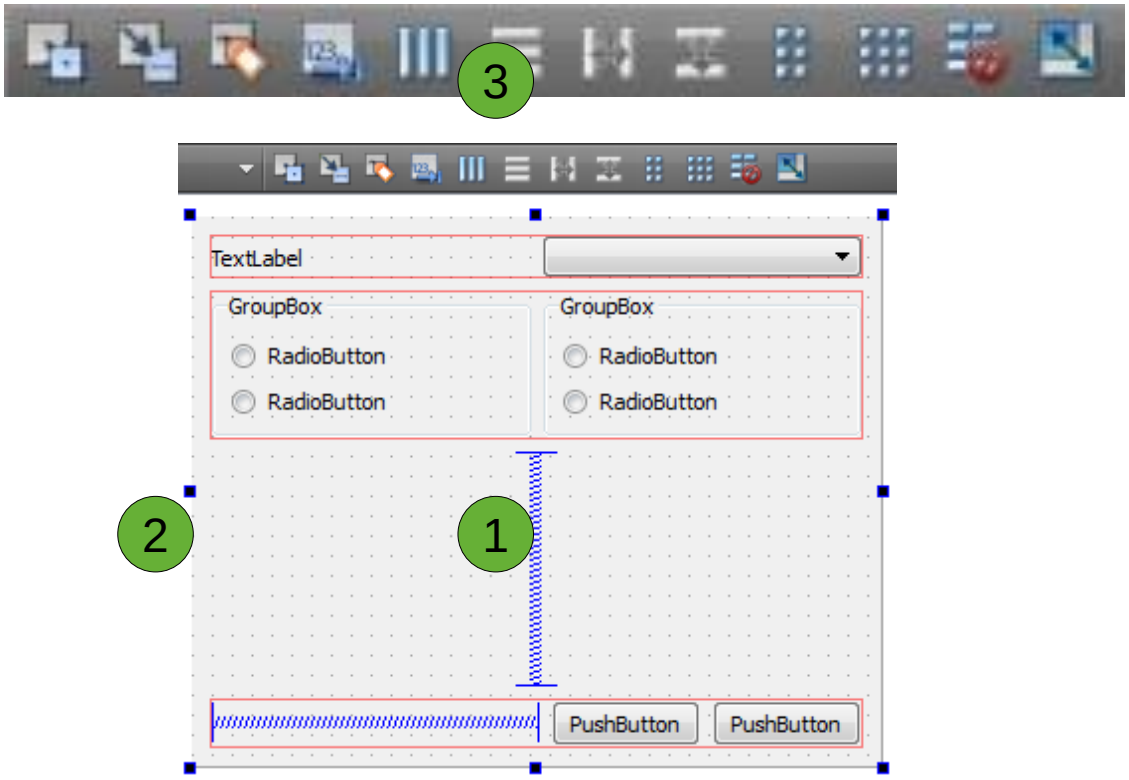
Utilisez des layouts et des ressorts



1. Utilisez un horizontal box layout sur les deux group box,
2. Ajouter un horizontal spacer devant les boutons,
3. Utilisez un horizontal box layout sur les boutons et le spacer

Utilisation de QtDesigner

Utilisez des layouts et des ressorts

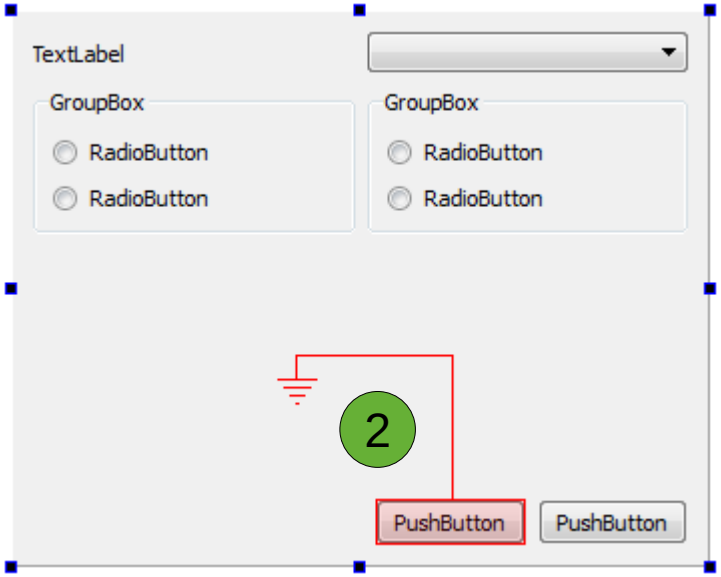


1. Ajoutez vertical spacer, **2**. sélectionnez la fenêtre, **3**. Utilisez un vertical box layout

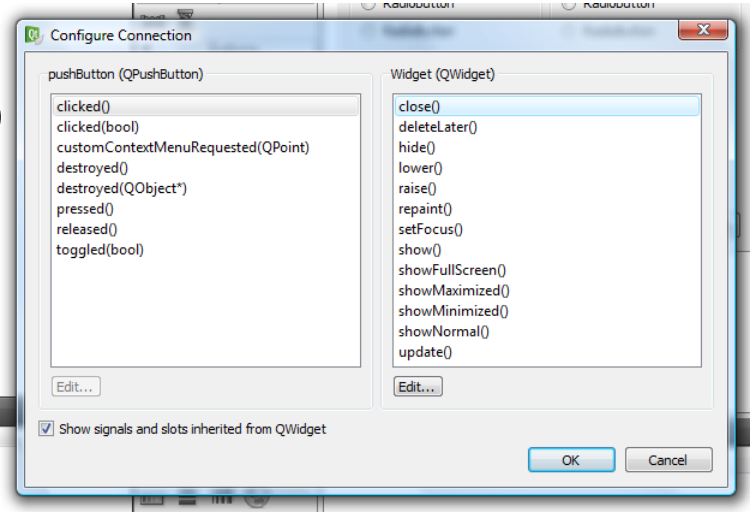
Utilisation de QtDesigner

Créer les connexions (entre widgets)

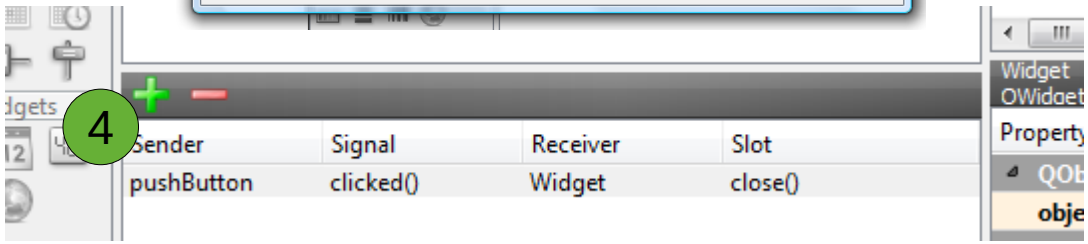
1



3



4

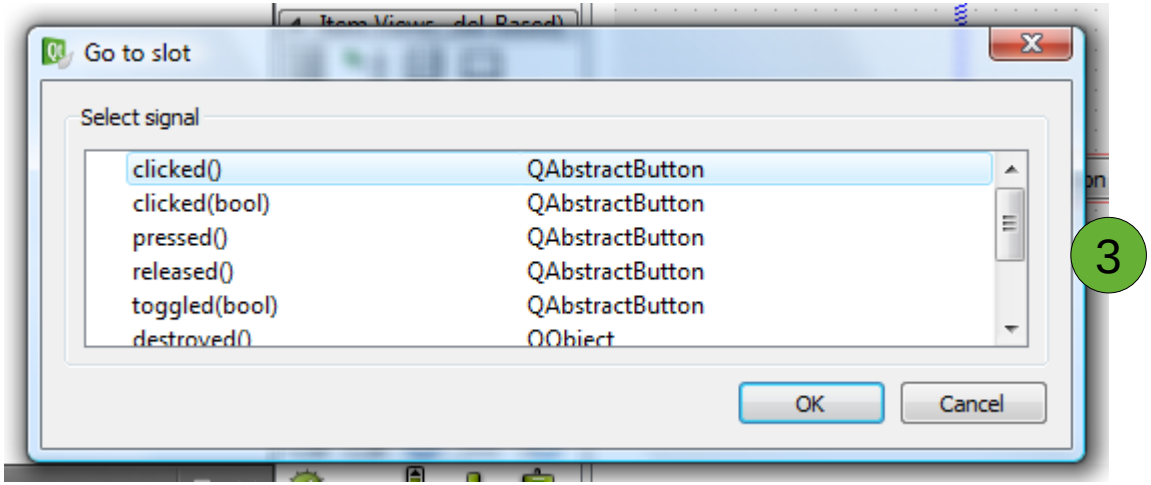


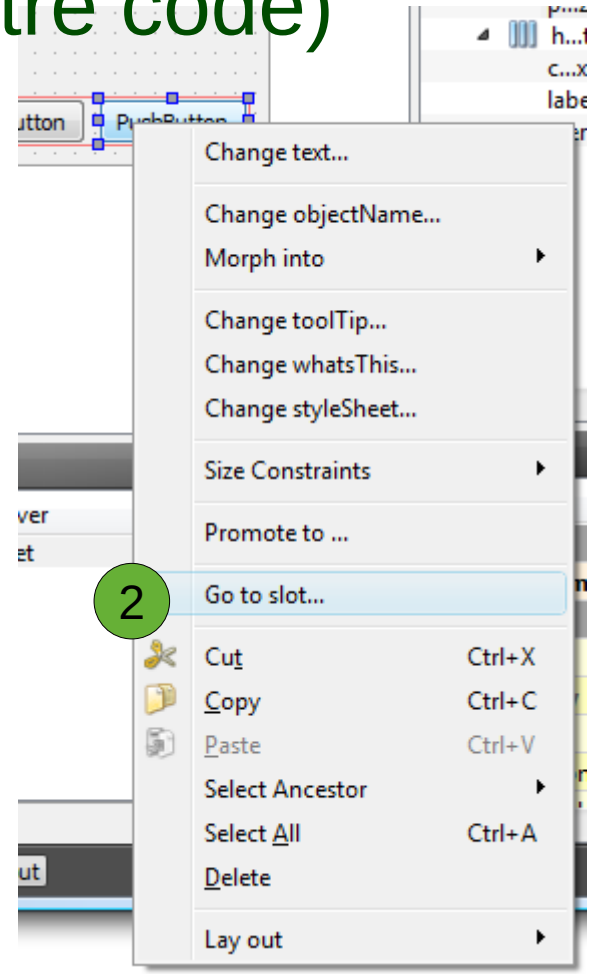
1. Choisissez le mode édition signals/slot, **2**. faire un glisser d'un widget vers la fenêtre, **3**. choisissez un signal et un slot, **4**. regardez le résultat dans la fenêtre de connexion

Utilisation de QtDesigner

Créer les connexions (vers votre code)

1





1. Revenez en mode édition de widget,
2. click droit sur un widget sélectionnez *Go to slot...*
3. choisissez le signal qui sera géré par votre code

Utilisation de QtDesigner

Utilisation depuis votre code

L'accès à tous les éléments de votre ihm sont accessibles via l'objet ui

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

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

Conventions de nommage

Prenez pour habitude de nommez vos éléments de la façon suivante:

TypeDeLElementDescription

Exemple:

pushButtonQuitter

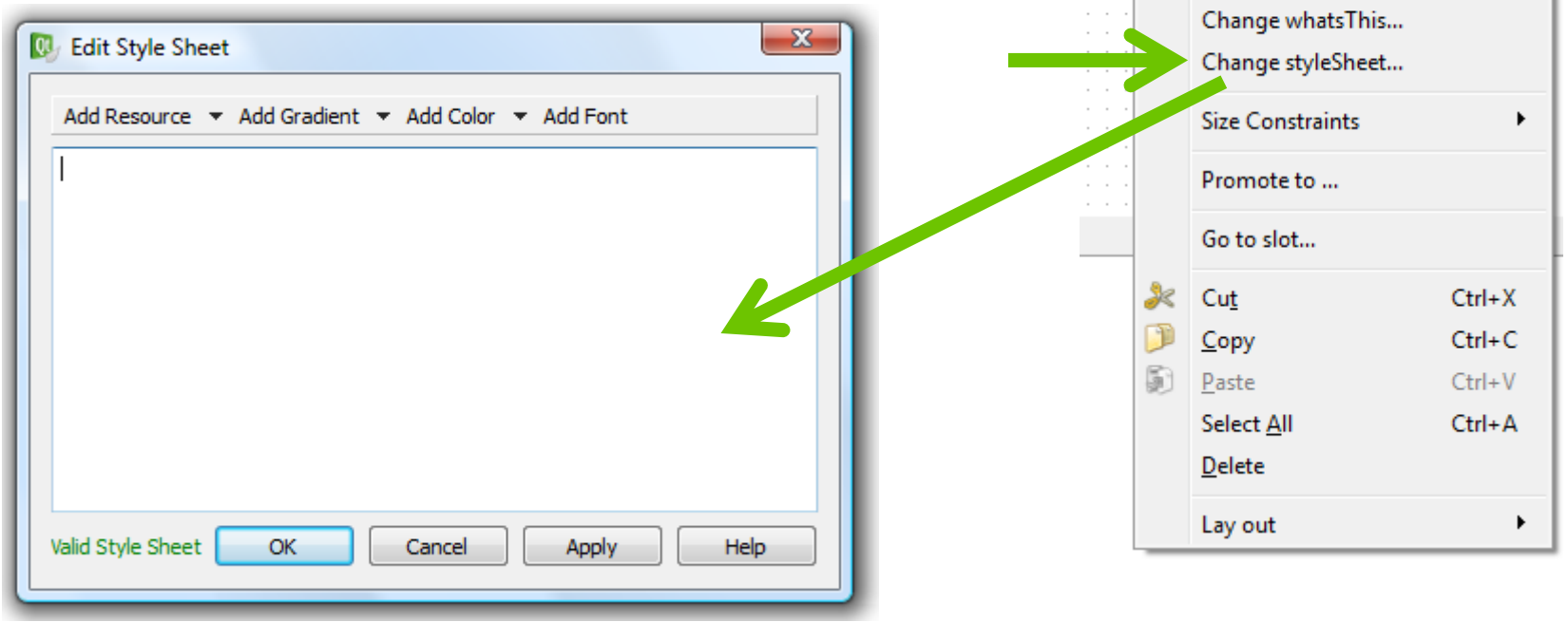
Style sheets

- Toutes les classes `QWidget` possèdent la propriété `styleSheet`
- Style sheets est une variante du CSS
- Il est donc possible d'appliquer des “styles” à chaque widget

- Le boooooooo bouton que voici:

Style sheets

- QtDesigner reste l'interface la plus simple pour appliquer un style à un widget



Stylesheet

- Il est possible d'appliquer un style à l'ensemble de l'application avec `QApplication::setStyleSheet`

Select a class

Select an object by name

Use images

Build these in Designer's editor

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

xtEdit, QListView {
    background-color: white;
    background-image: url(draft.png);
    background-attachment: scroll;
}

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;
}
```

A vous de jouer

- Développez une interface ayant l'aspect et les fonctionnalités suivantes: