Chapitre 8

Initiation au développement d'interfaces graphiques avec Qt

Arborescence de composants

Signaux et slots

Placement des composants

Interfaces graphiques et C++

Le langage C++ et la bibliothèque standard ne fournissent pas d'outils pour développer des interfaces graphiques.

Plusieurs bibliothèques peuvent être utilisées *wxWidgets*, *GTK*, *CEGUI*, etc.

Qt `https://www.qt.io` est une bibliothèque C++ permettant de développer des applications disposant d'une interface graphique (et bien plus).

- Desktop, Mobile, IOT
- Portable, Apparence native, Efficace
- *Bindings* dans différents langages
- Très utilisé (Dreamworks, Lucasfilm, Philips, Samsung, Blizzard, AMD, Valve, etc). KDE, VLC, Photoshop (Album/Elements), Telegram, Virtualbox, etc.

Arborescence de composants

Arborescence de composants

Signaux et slots

Placement des composants

Principe de Qt

- Une fenêtre d'interface graphique est composée d'une arborescence de widgets.
- Une action de l'utilisateur est représentée par un signal reçu par l'application.
- Un signal peut être connecté à un slot qui exécutera une action en réponse au signal.
- Signaux et slots ne sont pas accessibles directement depuis le compilateur C++ : Qt fournit un compilateur de meta-objets (`moc`) qui génère du code C++ gérant slots et signaux.

`QObject` : Arborescence d'objets

- `QObject` est la racine d'héritage de la plupart des classes fournies par Qt.
- `QObject` gère une arborescence d'objets `parent()`, `children()`. La destruction d'un objet détruit sa descendance.
  - → un objet qui a un parent doit être alloué dynamiquement, et jamais détruit.
  - → un objet qui n'a pas de parent doit être détruit après utilisation... Pas de raison d'utiliser l'allocation dynamique.

`QWidget` : Arborescence de composants

- `QWidget` (sous-classe de `QObject`) est la racine des composants d'interface graphique.
- Qt fournit un grand nombre de widgets… on peut définir les nôtres.
- Un `QWidget` sans parent est affiché dans une fenêtre. Un `QWidget` ayant un parent est affiché dans son parent.

`QApplication` : Application

Le `main` ne fait « rien »… si ce n'est…

- instancier `QApplication` (avec les arguments reçus en ligne de commande);
- créer le composant de la fenêtre principale de l'application (et sa descendance);
- afficher ce composant (méthode `show`);
- appeller la méthode `run` de `QApplication` qui se charge de gérer toutes les interactions avec l'utilisateur.

Quand la méthode `QApplication::run` se termine, l'application se termine.

Exemple d'application Qt minimale

Application minimale : `main` (qt/premier/main.cc)

```
#include <QtWidgets>

int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  QWidget window;
  window.resize(320, 240);
  window.setWindowTitle("Ma première fenêtre");
  window.show();
  return app.exec();
}
```

Exemple d'application Qt minimale

Construction de l'exemple

(qt/premier/CMakeLists.txt)

```
1  cmake_minimum_required(VERSION 3.1.0)
2  project(qtpremier CXX)
3
4  set(CMAKE_CXX_STANDARD 14)
5  set(CMAKE_CXX_STANDARD_REQUIRED on)
6  set(CMAKE_CXX_EXTENSIONS off)
7
8  find_package(Qt5Widgets REQUIRED)
9  set(CMAKE_INCLUDE_CURRENT_DIR ON)
10 set(CMAKE_AUTOMOC ON)
11 set(CMAKE_AUTOUIC ON)
12
13 if(CMAKE_CXX_COMPILER_ID MATCHES "GNU")
14   add_compile_options(-Wall -Wpedantic)
15 endif()
16
17 add_executable(qtpremier main.cc)
18 target_link_libraries(qtpremier Qt5::Widgets)
```

Construction d'une application Qt

Ce qui est nouveau par rapport à un `CMakeLists.txt` C++14

- Lignes 8-11. Recherche de la bibliothèque Qt5Widgets (8) ; Utilisation simplifiée du moc (9-10) et de l'uic (11) (non utilisés dans cet exemple).
- Ligne 18. L'exécutable est lié à la bibliothèque Qt5Widgets.

Quelques composants

- `QLabel` Texte fixe.
- `QPushButton` Bouton.
- `QCheckBox` Case à cocher.
- `QLineEdit` Champ d'édition.
- `QComboBox` Liste déroulante.
- `QRadioButton` Bouton radio.

Et bien d'autres
`http://doc.qt.io/qt-5/widget-classes.html`

Arborescence de composants

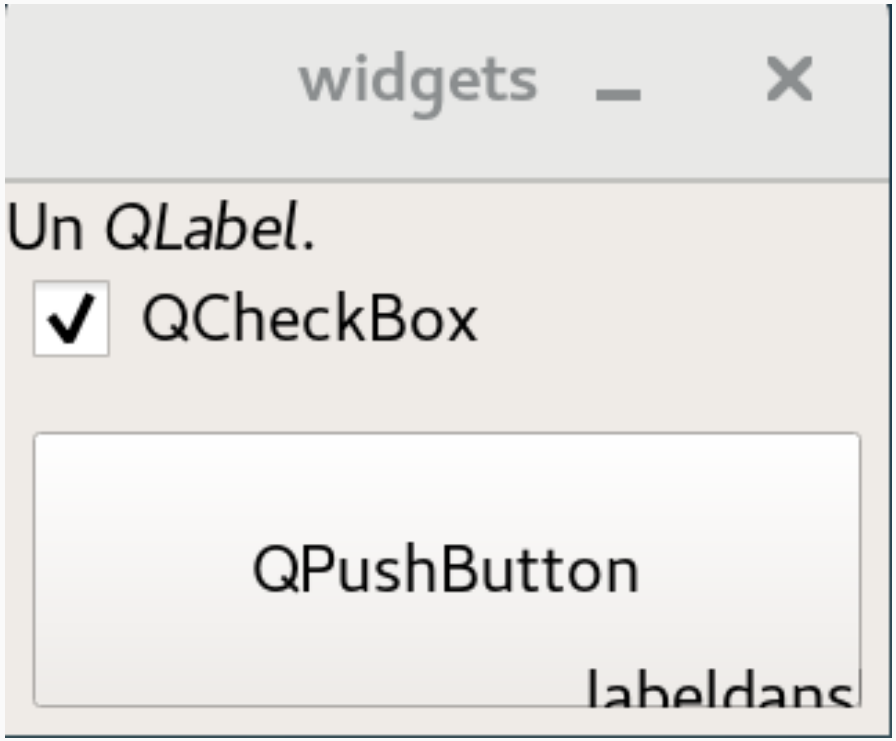
Quelques composants

(qt/widgets/main.cc)

```
#include <QtWidgets>

int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  QWidget window; window.resize(320, 200);
  QLabel* monlabel(new QLabel("Un <i>QLabel</i>.", &window));
  monlabel->move(0,0);
  QCheckBox * macheckbox(new QCheckBox("QCheckBox", &window));
  macheckbox->move(10,30);
  QPushButton* monbutton(new QPushButton("QPushButton", &window));
  monbutton->setGeometry(10,90,300,100);
  QLabel* autrelabel(new QLabel("labeldansbouton", monbutton));
  autrelabel->move(200,80);
  window.show();
  return app.exec();
}
```

Arborescence de composants



Arborescence de composants

- Un composant ayant un parent ne doit pas être détruit.
- Les coordonnées passées à `move(x,y)` sont celles du point supérieur gauche du composant et sont relatives au composant parent.
- Un composant de tout type peut avoir comme parent un composant de tout type.
- Un composant ne peut pas sortir de son parent.
- `QLabel` comprend un sous-ensemble d'HTML.

Signaux et slots

Arborescence de composants

Signaux et slots

Placement des composants

Signaux et slots - Principe

- Dans une interface graphique (mais pas uniquement), l'application doit réagir à des événements externes : actions de l'utilisateur.
- Un événement sur un composant est représenté dans Qt par l'émission d'un signal. Exemple : clic sur un bouton.
- Un slot est une méthode d'un `QObject` effectuant un traitement. Exemple : fermer la fenêtre.
- Un slot peut être connecté à un signal (d'un autre objet ou du même). Quand le signal est émis, le slot est déclenché. Exemple : un clic sur un bouton (signal) provoque la fermeture d'une fenêtre (slot).

Signaux - Principe

Mécanisme de communication entre objets avec couplage faible :
Quand un signal est émis, l'émetteur ne sait pas « qui » traitera ce signal.

- Un signal est émis quand l'état de l'émetteur change, ce qui peut intéresser d'autres objets.
- Un signal a une signature correspondant aux paramètres qu'il porte.

Slots - Principe

- Un slot est une méthode. Un slot a une signature. Un slot peut être appelé comme une méthode.
- Un signal d'une signature donnée ne peut être connecté qu'à des slots qui ont une signature compatible. (vérification à la compilation).
- Un slot peut être connecté à plusieurs signaux, un signal peut être connecté à plusieurs slots.

Signaux - Fonctionnement

Une classe qui émet des signaux doit :

- Être une sous classe de `QObject`.
- Utiliser la macro `Q_OBJECT` au début de sa déclaration.
- Déclarer les signaux émis dans une section `signals:`
  Un signal a la syntaxe d'une méthode (le type de retour est souvent `void`).
- Utiliser l'instruction `emit` pour émettre un signal.

Compiler simplement cette classe par un compilateur C++ n'est pas suffisant.

Signaux - Le *Meta-Object Compiler*

Le moc gère des extensions au langage C++ propres à Qt, dont les signaux et les slots.

- Quand un fichier source contient une déclaration classe utilisant la macro `Q_OBJECT`, le moc génère un fichier définissant le code de méthodes devant être compilées et liées à l'exécutable (entre autres pour la gestion des signaux et slots).
- Cela est fait automatiquement par CMake à condition que les lignes suivantes soient présentes dans le `CMakeLists.txt`.

CMakeLists et moc (qt/slotssignals/CMakeLists.txt)

```
set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_AUTOMOC ON)
```

Signaux - Exemple

Classe émettrice de signaux (qt/slotssignals/setofstring.hh)

```
#pragma once
#include <QObject>
#include <vector>
#include <string>
class setofstring: public QObject {
  Q_OBJECT
  public:
    void ajoutervaleur(std::string const &);
    void supprimervaleur(std::string const &);
    bool contientvaleur(std::string const &) const;
  signals:
    void valeurajoutee(std::string);
    void valeursupprimee(std::string);
    void taillemodifiee(std::size_t);
  private:
    std::vector<std::string> _contenu;
};
```

Signaux - Exemple

Classe émettrice de signaux (qt/slotssignals/setofstring.cc)

```
#include "setofstring.hh"
#include <algorithm>

void setofstring::ajoutervaleur(std::string const & v) {
  if (!contientvaleur(v)) {
    _contenu.push_back(v);
    emit valeurajoutee(v);
    emit taillemodifiee(_contenu.size());
  }
}
```

Signaux - Exemple

Classe émettrice de signaux (qt/slotssignals/setofstring.cc)

```
void setofstring::supprimervaleur(std::string const & v) {
  auto i(std::find(_contenu.begin(), _contenu.end(), v));
  if (i != _contenu.end()) {
    _contenu.erase(i);
    emit valeursupprimee(v);
    emit taillemodifiee(_contenu.size());
  }
}

bool setofstring::contientvaleur(std::string const & v) const {
  return std::find(_contenu.begin(), _contenu.end(), v) != _contenu.
      end();
}
```

Slots

Un slot est une méthode qui répond à l'émission d'un signal.

Dans quelle classe déclarer une telle méthode ?

Ça dépend...

Habituellement on définit une classe pour représenter une fenêtre d'interface graphique (contrairement à l'exemple `widgets` donné précédemment).

Cette classe peut être utilisée pour définir des méthodes slots.

Slots

Classe fenêtre (qt/slotssignals/fenetrev1.hh)

```
#include <QtWidgets>

class fenetre: public QWidget {
  Q_OBJECT
  public:
    fenetre();
  public:
    QLineEdit * _saisie;
    QPushButton * _ajout;
    QLabel * _taille;
    QPushButton * _quitter;
};
```

Classe fenêtre

(qt/slotssignals/fenetrev1.cc)

```
#include "fenetrev1.hh"

fenetre::fenetre( )
  :QWidget( ) {
  resize(320,200);
  _saisie = new QLineEdit("", this);
  _ajout = new QPushButton("Ajouter", this);
  _taille = new QLabel("taille", this);
  _quitter = new QPushButton("Quitter", this);
  _saisie->setGeometry(10,10,300,40);
  _ajout->setGeometry(10,50,300,50);
  _taille->move(10,100);
  _quitter->setGeometry(10,140,300,50);
}
```

Connexion slot/signal

Connexion

La méthode (de classe) `QObject::connect` permet de connecter un signal d'un objet à un slot d'un objet.
Paramètres : Pointeur sur l'objet émetteur, Signal émis, Pointeur sur l'objet receveur, Slot déclenché.

- Un slot, comme un signal, est identifié par un pointeur sur son code.
- Les classes de Qt fournissent des slots qui peuvent être appelés comme méthodes classiques ou connectées à un signal.

Connexion slot/signal

Par exemple, `QApplication` dispose d'une méthode `quit()` qui termine l'application.

Connexion à un slot existant (qt/slotssignals/fenetrev1.cc)

```
int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  fenetre f;
  QObject::connect(f._quitter, &QPushButton::clicked, &app, &
      QApplication::quit);
  f.show();
  return app.exec();
}
```

Déclaration de slots

Dans une section `public slots:` d'une classe (sous-classe de `QObject`, etc.)

Déclaration de slots

(qt/slotssignals/fenetrev2.hh)

```
#include <QtWidgets>
#include "setofstring.hh"
class fenetre: public QWidget {
  Q_OBJECT
  public:
    fenetre(setofstring& sos);
  public slots:
    void onclicajout( );
    void ontaillemodifiee(std::size_t);
    void onajoutervaleur(std::string const &);
  private:
    setofstring& _sos;
    QLineEdit * _saisie;
    QPushButton * _ajout;
```

Connexions

Connexions

(qt/slotssignals/fenetrev2.cc)

```
fenetre::fenetre(setofstring& sos)
  :QWidget(), _sos(sos) {
  resize(320,200);
  _saisie = new QLineEdit("", this);
  _ajout = new QPushButton("Ajouter", this);
  _taille = new QLabel("taille", this);
  _quitter = new QPushButton("Quitter", this);
  _saisie->setGeometry(10,10,300,40);
  _ajout->setGeometry(10,50,300,50);
  _taille->move(10,100);
  _quitter->setGeometry(10,140,300,50);
  connect(_ajout, &QPushButton::clicked, this, &fenetre::onclicajout
      );
  connect(&_sos, &setofstring::taillemodifiee, this, &fenetre::
      ontaillemodifiee);
  connect(&_sos, &setofstring::valeurajoutee, this, &fenetre::
      onajoutervaleur);
}
```

Définition de slots

Définition de slots

(qt/slotssignals/fenetrev2.cc)

```
void fenetre::onclicajout() {
  _sos.ajoutervaleur(_saisie->text().toStdString());
}

void fenetre::ontaillemodifiee(std::size_t ns) {
  _taille->setText(QString::number(ns));
}

void fenetre::onajoutervaleur(std::string const & n) {
  std::cout << "Ajout de " << n << "\n";
}
```

Un clic sur le bouton d'ajout émet le signal `QPushButton::clicked` ce qui provoque l'exécution du slot `fenetre::onclicajout`. Cette méthode appelle `setofstring::ajoutervaleur` qui peut émettre les signaux `setofstring::valeurajoutee` et `taillemodifiee` ...

Placement des composants

Arborescence de composants

Signaux et slots

Placement des composants

Placement des composants

Positionner et dimensionner les composants de façon absolue a deux défauts :

- Fastidieux.
- Ne gère pas l'agrandissement des fenêtres.

Qt propose d'utiliser des gestionnaires de placements, permettant de ranger (et redimensionner) automatiquement les composants.
`QLayout` est la classe mère des gestionnaires de placements de Qt.

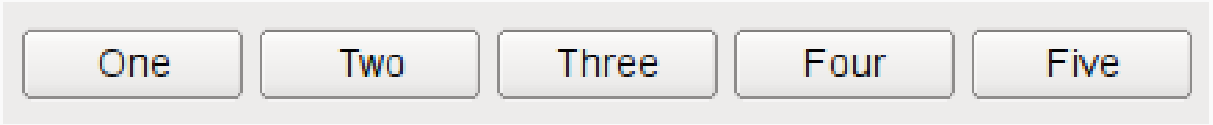
Principe d'un gestionnaire de placement

- Instancier une sous-classe de `QLayout` correspondant au type de « rangement » désiré.
- Rattacher ce gestionnaire au composant contenant les composants à ranger.
- Rattacher les composants à ranger au gestionnaire.

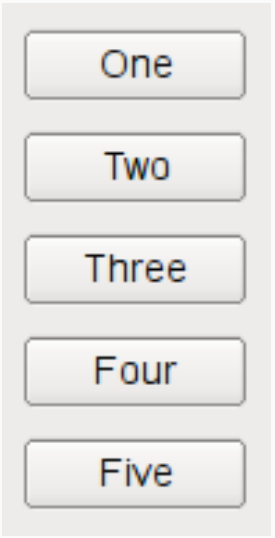
Principaux gestionnaires de placement

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

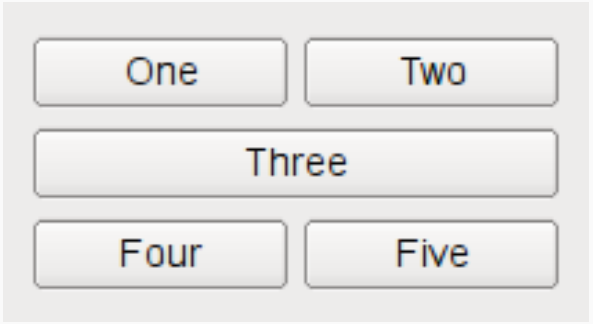
- QHBoxLayout



- QVBoxLayout



- QGridLayout



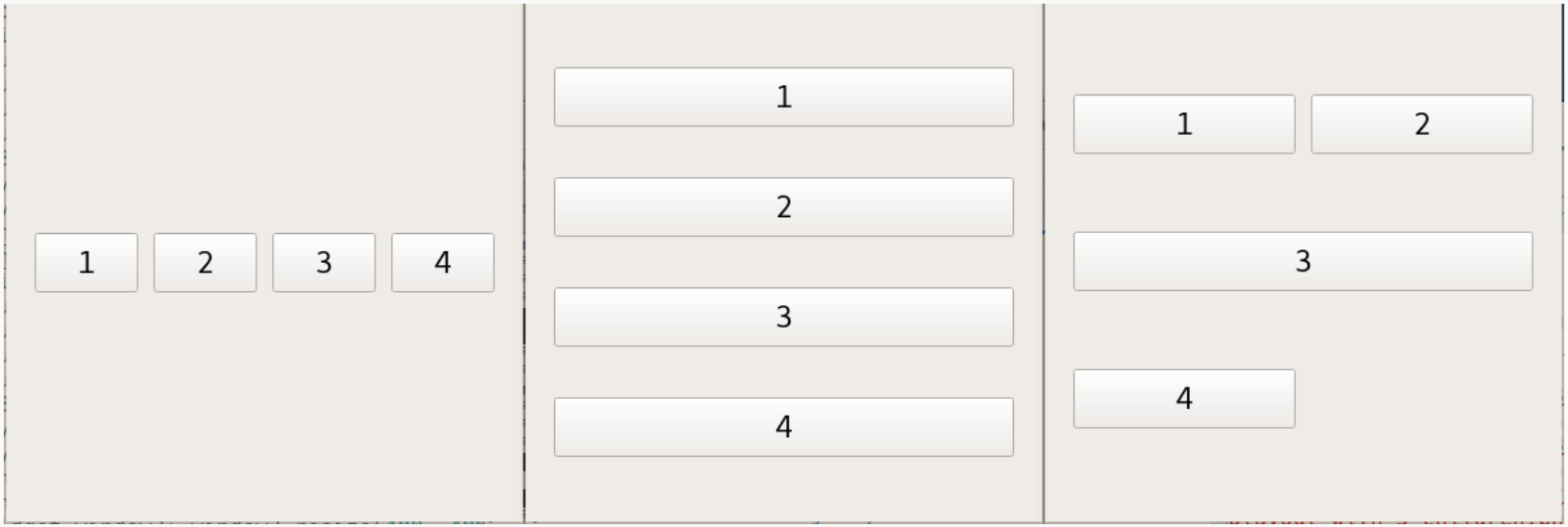
- …

QHBoxLayout (qt/layout/main.cc)

```
QWidget window1; window1.resize(400, 400);
QPushButton* b11=new QPushButton("1");
QPushButton* b12=new QPushButton("2");
QPushButton* b13=new QPushButton("3");
QPushButton* b14=new QPushButton("4");
QHBoxLayout * layout1 = new QHBoxLayout;
layout1->addWidget(b11); layout1->addWidget(b12);
layout1->addWidget(b13); layout1->addWidget(b14);
window1.setLayout(layout1);
window1.show();
```

L'ordre des appels à `addWidget` est important. Les appels au constructeur de `QPushButton` qui ne précisent pas le composant parent : en étant ajouté au `QLayout`, les composants prennent comme parent le composant rattaché au `QLayout`.

QHBoxLayout



QVBoxLayout (qt/layout/main.cc)

```
QWidget window2; window2.resize(400, 400);
QPushButton* b21=new QPushButton("1");
QPushButton* b22=new QPushButton("2");
QPushButton* b23=new QPushButton("3");
QPushButton* b24=new QPushButton("4");
QVBoxLayout * layout2 = new QVBoxLayout;
layout2->addWidget(b21); layout2->addWidget(b22);
layout2->addWidget(b23); layout2->addWidget(b24);
window2.setLayout(layout2);
window2.show();
```

QGridLayout (qt/layout/main.cc)

```
QWidget window3; window3.resize(400, 400);
QPushButton* b31=new QPushButton("1");
QPushButton* b32=new QPushButton("2");
QPushButton* b33=new QPushButton("3");
QPushButton* b34=new QPushButton("4");
QGridLayout * layout3 = new QGridLayout;
layout3->addWidget(b31,0,0); layout3->addWidget(b32,0,1);
layout3->addWidget(b33,1,0,1,2); layout3->addWidget(b34,2,0);
window3.setLayout(layout3);
window3.show( );
```

QGridLayout

`QGridLayout::addWidget` prend 5 paramètres

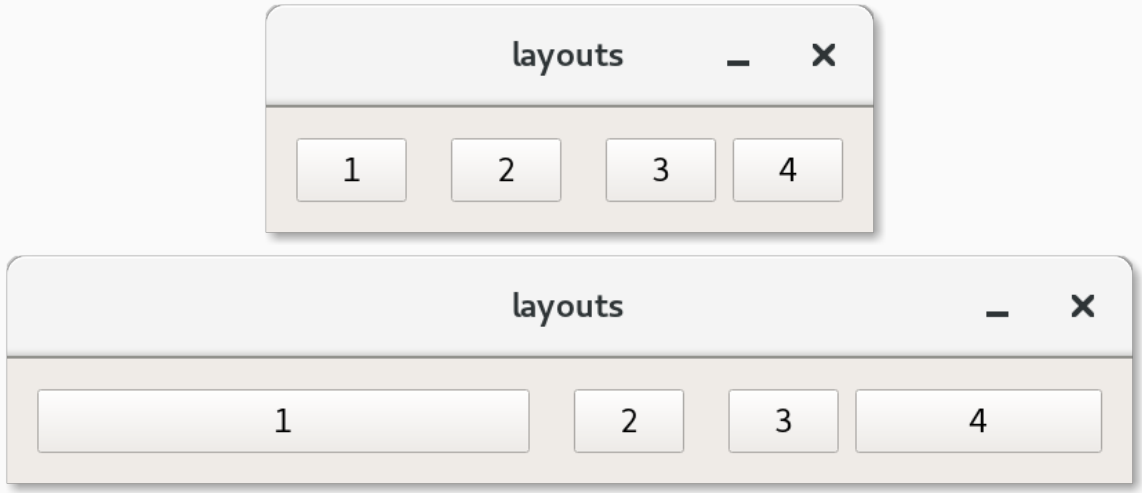
- Le composant à ajouter
- Les coordonnées de la case (ligne, colonne, en commençant à 0)
- Le nombre de cases (verticalement, horizontalement) (par défaut 1,1)

Le nombre de colonnes/lignes est déterminé automatiquement.

`QGridLayout` - Étirement

Une colonne/ligne peut être « vide » pour gérer l'espacement entre cases.

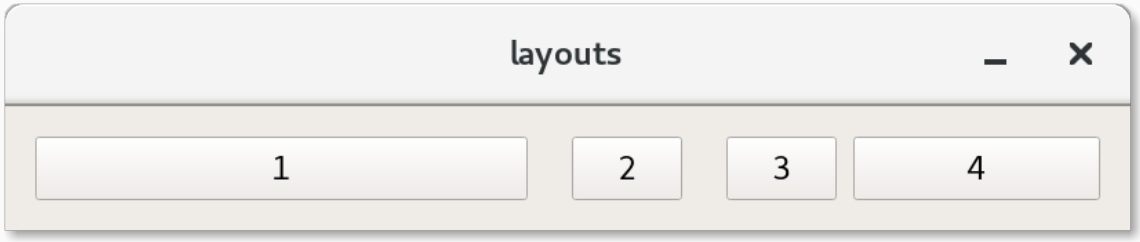
`QGridLayout::setColumnMinimumWidth` (`setRowMinimumHeight`) permettent de fixer la taille minimale d'une colonne/ligne.
`QGridLayout::setColumnStretch` (`setRowStretch`) permet de choisir le facteur d'étirement de la colonne/ligne.
Une colonne/ligne ayant un facteur d'étirement de 0 a une taille fixée.
Sauf si toutes les colonnes/lignes ont un facteur d'étirement de 0.



QGridLayout (qt/layout/main.cc)

```
QGridLayout * layout4 = new QGridLayout;
layout4->addWidget(b41,0,0);
layout4->setColumnStretch(0,4);
layout4->setColumnMinimumWidth(1, 20);
layout4->addWidget(b42,0,2);
layout4->setColumnMinimumWidth(3, 20);
layout4->addWidget(b43,0,4);
layout4->addWidget(b44,0,5);
layout4->setColumnStretch(5,2);
window4.setLayout(layout4);
```

QGridLayout - Étirement



- Colonne 0 : Bouton 1, Étirement 4
- Colonne 1 : Rien, Largeur minimale 20 (constante)
- Colonne 2 : Bouton 2, Largeur constante
- Colonne 3 : Rien, Largeur minimale 20 (constante)
- Colonne 4 : Bouton 3, Largeur constante
- Colonne 5 : Bouton 4, Étirement 2

`QGridLayout` - Séparateurs

- Entre les colonnes 4 et 5, espacement standard.
- Changer l'espacement entre widgets à l'intérieur du layout par `QGridLayout::setHorizontalSpacing(int)` (`setVerticalSpacing(int)`)
- Changer les marges (espace autour des widgets du layout) par `QLayout::setContentsMargins(int left, int top, int right, int bottom)`.