# PoP C++ Série 7 niveau 0

## Usage de GTKmm :Fenêtre de dialogue / (basic) Timer

Adapté du manuel de référence en-ligne de GTKmm 3

**Exercice 1.**(niveau 0) : **fenêtre de dialogue pour choisir un nom de fichier**

Cet exercice simplifie l'exemple FileChooserDialog fourni par le manuel GTKmm 3. Par contre on a ajouté un Gtk ::Label dont on change le texte selon l'interaction avec la fenêtre de dialogue. Le programme principal définit un widget ExampleWindow dérivé de la classe Window

```
#include "examplewindow.h"
#include <gtkmm/application.h>

int main(int argc, char *argv[])
{
  auto app = Gtk::Application::create(argc, argv, "org.gtkmm.example");

  ExampleWindow window;

  //Shows the window and returns when it is closed.
  return app->run(window);
}
```

L'interface du module **exampleWindow** définit un ButtonBox, un Button avec son signal handler, et un Label :

```
#ifndef GTKMM_EXAMPLEWINDOW_H
#define GTKMM_EXAMPLEWINDOW_H

#include <gtkmm.h>

class ExampleWindow : public Gtk::Window
{
public:
  ExampleWindow();
  virtual ~ExampleWindow();

protected:
  //Signal handlers:
  void on_button_file_clicked();

  //Child widgets:
  Gtk::ButtonBox m_ButtonBox;
  Gtk::Button    m_Button_File;
  Gtk::Label     m_Label_Info;
};

#endif //GTKMM_EXAMPLEWINDOW_H
```

La première partie de l'implémentation contient le constructeur. On dispose le Button et le Label verticalement dans un ButtonBox (équivalent à Box vu dans la série6 niveau 0). On connecte le Button à son signal handler.

```
#include "examplewindow.h"
#include <iostream>


ExampleWindow::ExampleWindow()
: m_ButtonBox(Gtk::ORIENTATION_VERTICAL),
  m_Button_File("Choose File"),
  m_Label_Info("Init")
{
  set_title("Gtk::FileSelection example");

  add(m_ButtonBox);

  m_ButtonBox.pack_start(m_Button_File);
  m_Button_File.signal_clicked().connect(sigc::mem_fun(*this,
              &ExampleWindow::on_button_file_clicked) );

  m_ButtonBox.pack_start(m_Label_Info);

  show_all_children();
}

ExampleWindow::~ExampleWindow()
{
}
```

L'aspect intéressant est le signal handler du Button. Cette méthode commence par déclarer une instance **dialog** de **Gtk ::FileChooserDialog** qui servira de support pour gérer le choix d'un nom de fichier dans une fenêtre temporaire. Cet objet de dialogue dispose d'une méthode **run()** qui va capter l'activité de l'interface, c'est-à-dire qu'on ne pourra rien faire d'autre tant que ce dialogue sera en cours.

Le caractère « temporaire » de l'instance **dialog** est spécifié par l'appel de la méthode **set_transient_for**() qui suit. Cela garantit le placement de la fenêtre de dialogue au-dessus de la fenêtre de l'application.

On doit explicitement demander quels boutons de dialogue on veut : ici on demande un button CANCEL et un bouton OPEN qui confirme le choix.

Les constantes **`Gtk::RESPONSE_CANCEL`** et **`Gtk::RESPONSE_OK,`** fournies en second paramètres, sont testées plus loin dans le code.

Dans cet exemple nous avons encadré l'appel de la méthode **run**() par une modification du Label à l'aide de la méthode **set_text**(). Il suffit de déplacer la fenêtre de dialogue pour constater que le Label a bien changé avant l'appel de **run**().

```
void ExampleWindow::on_button_file_clicked()
{
  Gtk::FileChooserDialog dialog("Please choose a file",
          Gtk::FILE_CHOOSER_ACTION_OPEN);
  dialog.set_transient_for(*this);

  //Add response buttons the the dialog:
  dialog.add_button("_Cancel", Gtk::RESPONSE_CANCEL);
  dialog.add_button("_Open", Gtk::RESPONSE_OK);

  m_Label_Info.set_text("choosing a file");

  //Show the dialog and wait for a user response:
  int result = dialog.run();

  m_Label_Info.set_text("Done choosing a file");

  //Handle the response:
  switch(result)
  {
    case(Gtk::RESPONSE_OK):
    {
      std::cout << "Open clicked." << std::endl;

      //Notice that this is a std::string, not a Glib::ustring.
      std::string filename = dialog.get_filename();
      std::cout << "File selected: " <<  filename << std::endl;
      break;
    }
    case(Gtk::RESPONSE_CANCEL):
    {
      std::cout << "Cancel clicked." << std::endl;
      break;
    }
    default:
    {
      std::cout << "Unexpected button clicked." << std::endl;
      break;
    }
  }
}
```

Il est très important de récupérer dans la variable **result** la valeur entière renvoyée par la méthode **run**() car c'est elle qui pilote la suite du programme avec un switch. On y retrouve les deux constantes utilisées au moment de la création des buttons CANCEL et OPEN.

La constante liée à OPEN est **`Gtk::RESPONSE_OK.`** Attention il n'y a pas eu d'ouverture de fichier à ce stade. Cette réponse veut simplement dire qu'on peut récupérer le nom de fichier choisi. La string renvoyée par **get_filename()** contient le chemin absolu depuis la racine de l'arborescence des fichiers comme on peut le constater quand on l'affiche dans le terminal.

Activité : utiliser le filename pour ouvrir un fichier et l'afficher dans le terminal.

## **Complément** : Comment passer de **OPEN** à **SAVE** ?

Le même objet **Gtk ::FileChooserDialog** sert pour les deux opérations ; il faut seulement le configurer avec un symbole différent selon l'utilisation.

Pour OUVRIR : on indique le symbole **`Gtk::FILE_CHOOSER_ACTION_OPEN`** dans la déclaration du **Gtk ::FileChooserDialog** :

```
Gtk::FileChooserDialog dialog("Please choose a file",
        Gtk::FILE_CHOOSER_ACTION_OPEN);
```

Ensuite, on ajoute le Button de dialogue qui signale l'action **_Open** dans la fenêtre de dialogue :

```
dialog.add_button("_Open", Gtk::RESPONSE_OK);
```

Pour SAUVEGARDER : on indique le symbole **`Gtk::FILE_CHOOSER_ACTION_SAVE`** dans la déclaration du **Gtk ::FileChooserDialog** :

```
Gtk::FileChooserDialog dialog("Please choose a file",
        Gtk::FILE_CHOOSER_ACTION_SAVE);
```

Ensuite, on ajoute le Button de dialogue qui signale l'action **_Save** (au lieu de **_Open**) dans la fenêtre de dialogue :

```
dialog.add_button("_Save", Gtk::RESPONSE_OK);
```

---

**Exercice 2.**(niveau 0) : **contrôle du temps avec un Timer**

Cet exercice simplifie autant que possible l'exemple Timerexample fourni par le manuel GTKmm 3 dans le sens où on gère un seul Timer au lieu d'un nombre quelconque. Le programme principal définit un widget SimpleExampleWindow dérivé de la classe Window

```
#include "basictimer.h"
#include <gtkmm/application.h>

int main (int argc, char *argv[])
{
  auto app = Gtk::Application::create(argc, argv, "org.gtkmm.example");

  BasicTimer example;
  return app->run(example);
}
```

Le but est de faire exécuter une action particulière à intervalle régulier, dans cet exemple toutes les 0,5 s. L'exemple illustre également comment arrêter puis relancer le Timer.

```
#ifndef GTKMM_EXAMPLE_TIMEREXAMPLE_H
#define GTKMM_EXAMPLE_TIMEREXAMPLE_H

#include <gtkmm.h>
#include <iostream>

class BasicTimer : public Gtk::Window
{
public:
  BasicTimer();

protected:
  // button signal handlers
  void on_button_add_timer();
  void on_button_delete_timer();
  void on_button_quit();

  // This is the standard prototype of the Timer callback function
  bool on_timeout();

  // Member data:
  Gtk::Box m_Box;
  Gtk::Button m_ButtonAddTimer, m_ButtonDeleteTimer, m_ButtonQuit;

  // Keep track of the timer status (created or not)
  bool timer_added;

  // to store a timer disconnect request
  bool disconnect;

  // This constant is initialized in the constructor's member initializer:
  const int timeout_value;
};

#endif // GTKMM_EXAMPLE_TIMEREXAMPLE_H
```

**Interface du module :**

Les actions de lancer puis d'arrêter le Timer sont demandées à l'aide des Buttons **m_ButtonAddTimer** et **m_ButtonDeleteTimer**. Chacun dispose de son signal handler qui va respectivement contrôler ces actions de lancer ou arrêter le Timer.

La Box **m_Box** sert à disposer les Buttons horizontalement.

Le booléen **timer_added** sert à mémoriser si un Timer a déjà été créé pour éviter des actions incorrectes (ex : créer plusieurs Timers ou détruire un Timer qui n'existe pas).

Le booléen **disconnect** va servir de relai pour une demande d'arrêt du Timer.

La *constante* entière **timeout_value** est un nombre de millisecondes qui sera initialisée à la déclaration de l'instance de SimpleTimerExample.

**Implémentation du module :**

Le constructeur précise les label des Buttons dans la liste d'initialisation (Start, Stop et Quit). On y trouve aussi la valeur de la période du Timer, ici **500ms**. Enfin the Timer n'existe pas au lancement du programmet ; le booléen **timer_added** est initialisé à **false** ainsi que le booléen **disconnect**.

Le constructeur dispose les Buttons dans la Box horizontalement (conformément au paramètre de la liste d'initialisation) et les connectent à leur signal handler.

```
#include "basictimer.h"

BasicTimer::BasicTimer() :
  m_Box(Gtk::ORIENTATION_HORIZONTAL, 10),
  m_ButtonAddTimer("_Start", true),
  m_ButtonDeleteTimer("_Stop", true),
  m_ButtonQuit("_Quit", true),
  timer_added(false),
  disconnect(false),
  timeout_value(500) // 500 ms = 0.5 seconds
{
  set_border_width(10);

  add(m_Box);
  m_Box.pack_start(m_ButtonAddTimer);
  m_Box.pack_start(m_ButtonDeleteTimer);
  m_Box.pack_start(m_ButtonQuit);

  // Connect the three buttons:
  m_ButtonQuit.signal_clicked().connect(sigc::mem_fun(*this,
              &BasicTimer::on_button_quit));
  m_ButtonAddTimer.signal_clicked().connect(sigc::mem_fun(*this,
              &BasicTimer::on_button_add_timer));
  m_ButtonDeleteTimer.signal_clicked().connect(sigc::mem_fun(*this,
              &BasicTimer::on_button_delete_timer));

  show_all_children();
}

void BasicTimer::on_button_add_timer()
{
  if(not timer_added)
  {
        Glib::signal_timeout().connect( sigc::mem_fun(*this,
              &BasicTimer::on_timeout), timeout_value );

        timer_added = true;

        std::cout << "Timer added" << std::endl;
  }
  else
  {
        std::cout << "The timer already exists: nothing more is created"
                  << std::endl;
  }
}

bool BasicTimer::on_timeout()
{
  static unsigned int val(1);

  if(disconnect)
  {
        disconnect = false; // reset for next time a Timer is created
        return false; // End of Timer
  }

  std::cout << "This is timer value : " << val << std::endl;
  ++val; // tic the clock

  return true; // keep the Timer working
}
```

```
void BasicTimer::on_button_delete_timer()
{
  if(not timer_added)
  {
    std::cout << "Sorry, there is no active timer at the moment."
              << std::endl;
  }
  else
  {
    std::cout << "manually disconnecting the timer " << std::endl;
    disconnect  = true;
    timer_added = false;
  }
}

void BasicTimer::on_button_quit()
{
  hide();
}
```

La nouveauté de cet exemple provient des signal handlers des Buttons Add et Delete :

- Pour **m_ButtonAddTimer** le signal handler **on_button_add_timer()** s'assure que le booléen est à false avant de créer le Timer. Celui-ci est associé à une fonction callback **on_timeout()** qui sera automatiquement appelée chaque fois que la période de 500ms est écoulée. Cela fait on change l'état du booléen **timer_added** à vrai pour ne pas créer d'autres Timer par erreur.
- Pour **m_ButtonDeleteTimer** le signal handler **on_button_delete_timer()** s'assure qu'il y a bien un Timer en testant **timer_added**. Une demande de désactivation est mémorisée en faisant passer le booléen **disconnect** à true. Le booléen **timer_added** doit bien sûr repasser à faux pour permettre de re-créer le Timer ultérieurement avec l'autre Button.
- La fonction callback **on_timeout** gère le Timer en fonction des demandes exprimées par les Buttons.
    - Si une demande de fin d'existence du Timer a été enregistrée en faisant passer le booléen **disconnect** à **true**, alors la fonction renvoie **false** ce qui a pour effet de supprimer le Timer. Auparavant il faut seulement remettre le booléen **disconnect** à **false** au cas où un Timer est créée par une demande du Button addTimer.
    - Si par contre il n'y a pas de demande de fin du Timer, on se contente d'afficher la valeur d'une variable **static** locale initialisée à 1 et qui est incrémentées à chaque appel. Remarquer que la fonction renvoie **true**, ce qui exprime qu'on veut que le Timer continue son activité.

Activité : changer la valeur de la période et constater la qualité de la synchronisation avec le temps du monde réel (wall-clock time). *On se satisfera de cette qualité pour le projet*. Par exemple, commencez en indiquant une période de 1000ms puis regardez votre montre au moment où vous appuyez sur Start et lorsque le programme affiche 10. Continuez en diminuant la période : (100ms, affiche 100), (10ms, affiche 1000),(1ms, affiche 10000).

Que constatez-vous ?