Architecture d'un programme interactif graphique

Partie 2: Programmation par événement

Objectifs:

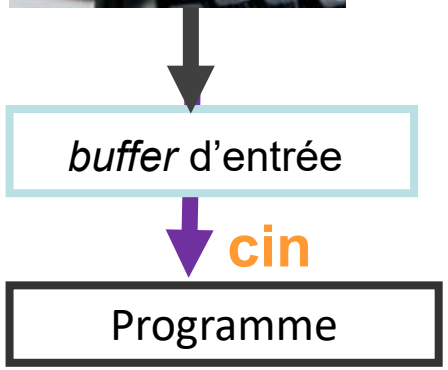
- maîtriser le concept de programmation par événement

Plan:

- Principe de la lecture non-bloquante
- Programmation par événements
- Exemple1 myEvent : interaction par variable d'état
- Exemple2 BasicTimer
- Exemple3 lecture et ecriture de fichier

Principe de la lecture non-bloquante => programmation par événements

Entrée conversationnelle
= lecture bloquante



Tant qu'on n'a pas validé ce qui est tapé au clavier avec **Enter**, le *buffer* d'entrée est vide :

→ il n'y a rien à «lire» pour le programme

→ Le programme attend…

Programmation par événements
= lecture non-bloquante

L'essentiel du temps d'exécution est passé pendant l'exécution de la méthode **run** sur l'Application **app** de GTKmm.

```
…
int main(…)
{
  auto app = Gtk::Application …
  …
  return app->... ;
}
```

La gestion de l'interaction est non-bloquante car la méthode **run** gère une boucle infinie de traitement des ***événements***.

L'***événement*** est l'atome de l'interaction.

C'est un **changement d'état** d'un élément de l'interface GTKmm, y compris de la fenêtre graphique, du clavier et des boutons de la souris,

Dans la terminologie de **GTKmm** à chaque ***événement*** correspond la production d'un ***signal*** spécifique

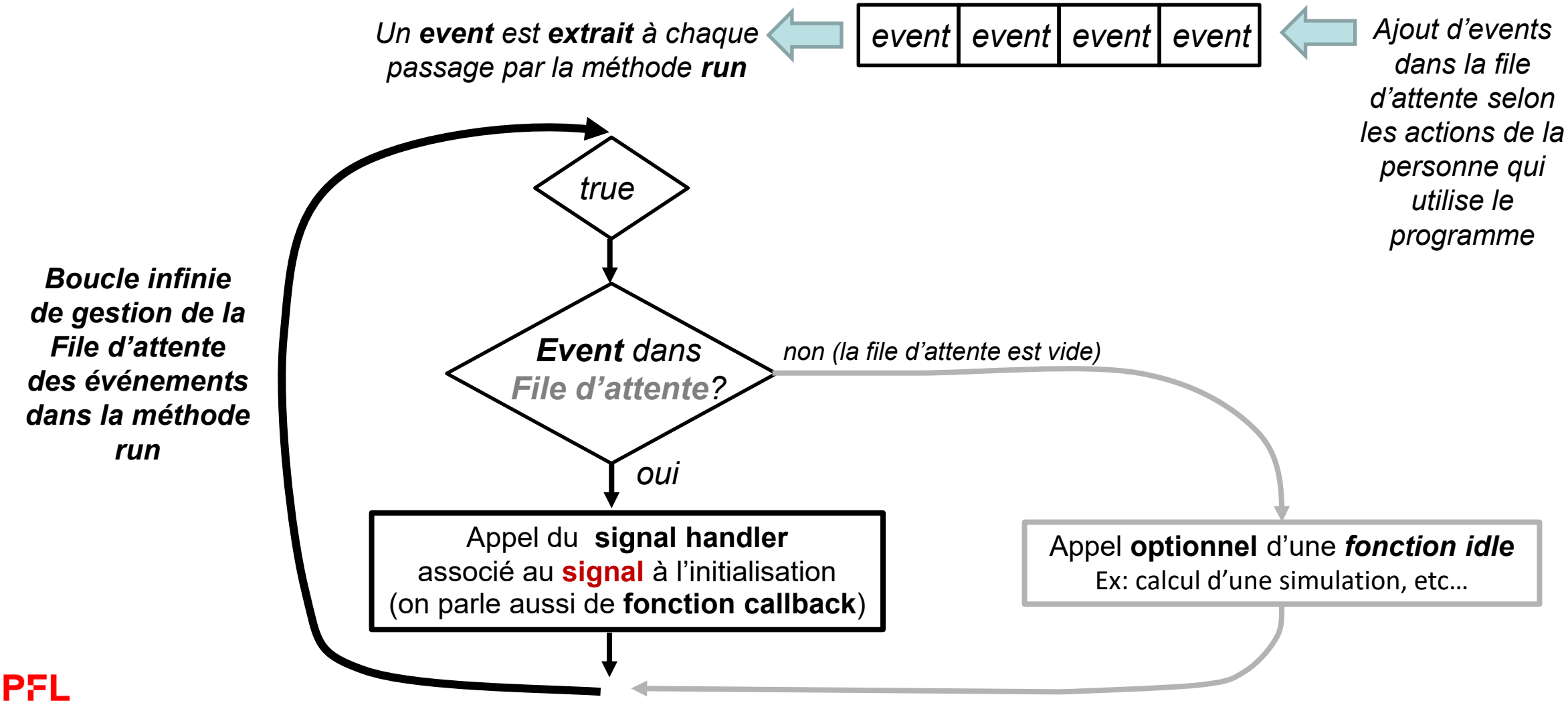
Ex: appuyer sur un `Button` produit `signal_clicked`

Si la classe dérivée de `Window` a initialisé **un signal handler**, celui-ci est appelé automatiquement

Principe de la lecture non-bloquante => programmation par événements (2)

Pseudocode de la boucle infinie de gestion de la file d'attente des événements dans la méthode run(…)

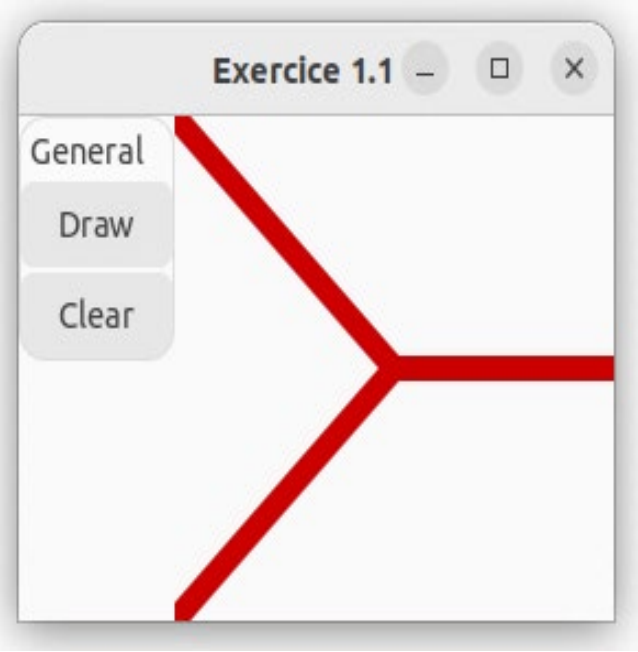
Chaque ***événement*** **=** ***event*** **=** ***signal*** est mémorisé dans une **File d'attente d'événements** selon son instant de création

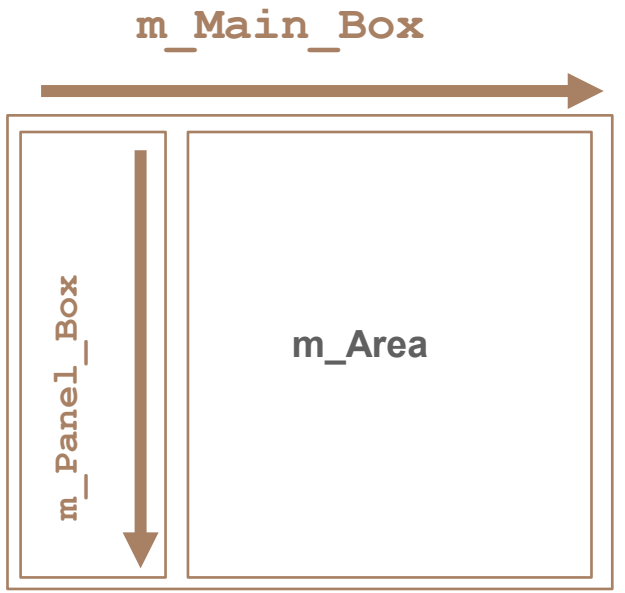


myevent.h

Exemple1 MyEvent(1)

```
10  class MyEvent : public Gtk::Window
11  {
12  public:
13      MyEvent();
14
15  private:
16      // GUI layout
17      Gtk::Box m_Main_Box;
18      Gtk::Box m_Panel_Box;
19      Gtk::Box m_Buttons_Box;
20      Gtk::Frame m_Panel_Frame;
21      Gtk::Button m_Button_Draw;
22      Gtk::Button m_Button_Clear;
23      Gtk::DrawingArea m_Area;
24
25      bool draw ; // current drawing state
26
27      //Button Signal handlers:
28      void on_button_clicked_clear();
29      void on_button_clicked_draw();
30
31      // DrawingArea signal handler:
32      void on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
33                   int width, int height);
34  };
```



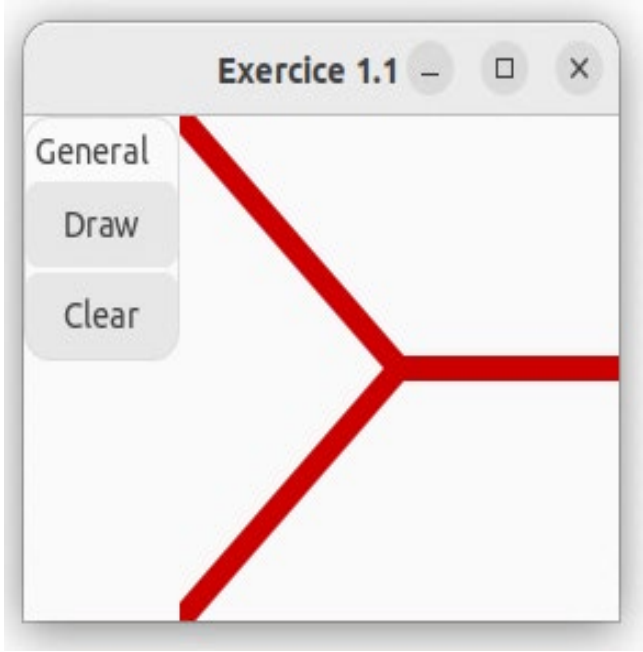


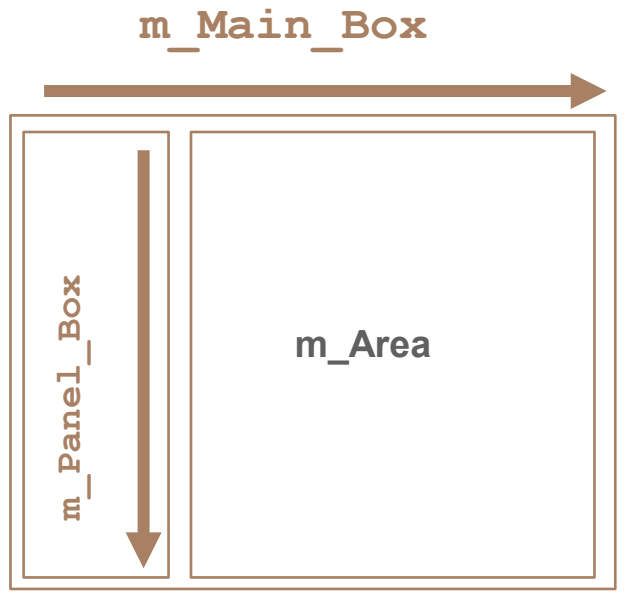
EPFL

myevent.cc

MyEvent(2)

```
8   MyEvent::MyEvent():
9       m_Main_Box(Gtk::Orientation::HORIZONTAL, 0),
10      m_Panel_Box(Gtk::Orientation::VERTICAL, 2),
11      m_Buttons_Box(Gtk::Orientation::VERTICAL, 2),
12      m_Panel_Frame("General"),
13      m_Button_Draw("Draw"),
14      m_Button_Clear("Clear"),
15      draw(true)
16  {
17      // init layout
18      set_title("Exercice 1.1");
19      set_child(m_Main_Box);
20
21      m_Main_Box.append(m_Panel_Box);
22      m_Main_Box.append(m_Area);
23
24      m_Panel_Box.append(m_Panel_Frame);
25      m_Panel_Frame.set_child(m_Buttons_Box);
26      m_Buttons_Box.append(m_Button_Draw);
27      m_Buttons_Box.append(m_Button_Clear);
28
29      // init buttons signal handlers
30      m_Button_Clear.signal_clicked().connect(
31          sigc::mem_fun(*this, &MyEvent::on_button_clicked_clear));
32
33      m_Button_Draw.signal_clicked().connect(
34          sigc::mem_fun(*this, &MyEvent::on_button_clicked_draw));
35
36      // init drawing sub-window parameters
37      m_Area.set_content_width(area_side);
38      m_Area.set_content_height(area_side);
39      m_Area.set_expand();
40      m_Area.set_draw_func(sigc::mem_fun(*this, &MyEvent::on_draw));
41
42  }
```





Rôle de la variable d'état **draw** et appel indirect de **on_draw()**

```
44  void MyEvent::on_button_clicked_clear()
45  {
46      std::cout << "Drawing cancelled" << std::endl;
47      draw = false;
48      m_Area.queue_draw();
49  }
50
51  void MyEvent::on_button_clicked_draw()
52  {
53      std::cout << "Drawing activated" << std::endl;
54      draw = true;
55      m_Area.queue_draw();
56  }
```

```
58  void MyEvent::on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
59                        int width, int height)
60  {
61      if(draw)
62      {
63          // coordinates for the center of the window
64          int xc, yc;
65          xc = width / 2;
66          yc = height / 2;
67
68          cr->set_line_width(10.0);
69
70          // draw red lines out from the center of the window
71          cr->set_source_rgb(0.8, 0.0, 0.0);
72          cr->move_to(0, 0);
73          cr->line_to(xc, yc);
74          cr->line_to(0, height);
75          cr->move_to(xc, yc);
76          cr->line_to(width, yc);
77          cr->stroke();
78      }
79      else
80      {
81          std::cout << "Nothing to draw !" << std::endl;
82      }
83  }
```

clic! → Clear ⇨ Appel du signal handler : **on_button_clicked_clear()** ⇨ 1) **draw** passe à **false** 2) **queue_draw()** ⟹

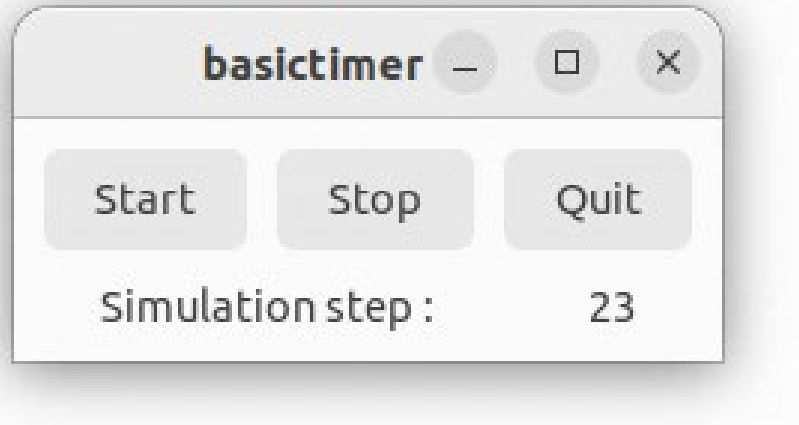
clic! → Draw ⇨ Appel du signal handler : **on_button_clicked_draw()** ⇨ 1) **draw** passe à ***true*** 2) **queue_draw()** ⟹

La méthode **queue_draw()** produit un **event** qui va causer l'appel de **on_draw()**

Exemple2: mise à jour synchronisée à un timer

But: pouvoir demander l'activation/l'arret de la production d'événements à intervalles réguliers

```
8   class BasicTimer : public Gtk::Window
9   {
10  public:
11      BasicTimer();
12
13  private:
14      Gtk::Box m_box, top_box,   bottom_box;
15      Gtk::Button     start_timer, stop_timer, quit_button;
16      Gtk::Label      text_label, data_label;
17
18      // state variable to handle a single timer
19      bool timer_added;
20      // state variable for the timer disconnect request
21      bool disconnect;
22      // duration initialized in constructor
23      const int timeout_value;
24
25      // button signal handlers
26      void on_button_add_timer();
27      void on_button_delete_timer();
28      void on_button_quit();
29
30      // Timer signal handler
31      bool on_timeout();
32  };
```



2 boutons gérent la demande d'activation et d'arrêt d'un timer à l'aide de 2 attributs d'état **timer_added** et **disconnect**

l'attribut **timeout_value** mémorise la durée séparant 2 events du timer (exprimée en ms)

le signal handler du timer qui est activé est :

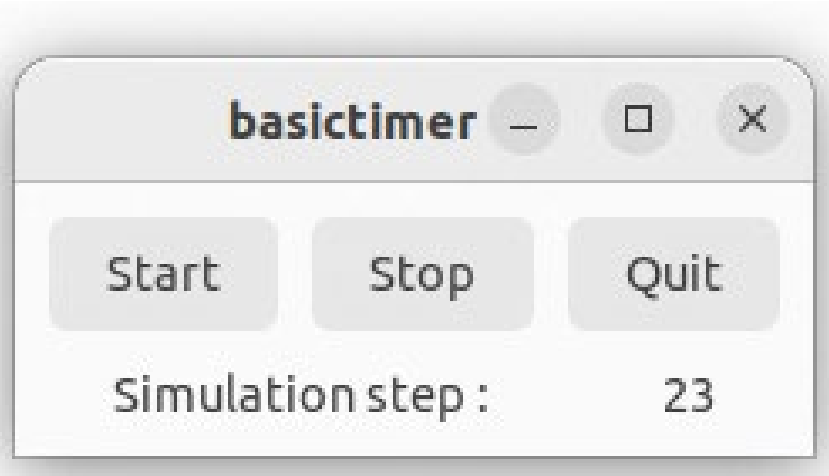
```
bool on_timeout();
```

Exemple2: BasicTimer(2)

```
4   BasicTimer::BasicTimer() :
5       m_box(Gtk::Orientation::VERTICAL, 10),
6       top_box(Gtk::Orientation::HORIZONTAL, 10),
7       bottom_box(Gtk::Orientation::HORIZONTAL, 10),
8       start_timer("_Start", true),
9       stop_timer("_Stop", true),
10      quit_button("_Quit", true),
11      text_label("Simulation step : "),
12      data_label("0"),
13      timer_added(false),// to handle a single timer
14      disconnect(false), // to handle a single timer
15      timeout_value(500) // 500 ms = 0.5 seconds
16  {
17      m_box.set_margin(10);
18      set_child(m_box);
19
20      m_box.append(top_box);
21      m_box.append(bottom_box);
22
23      top_box.append(start_timer);
24      top_box.append(stop_timer);
25      top_box.append(quit_button);
26
27      start_timer.set_expand();
28      stop_timer.set_expand();
29      quit_button.set_expand();
30
31      bottom_box.append(text_label);
32      bottom_box.append(data_label);
33
34      text_label.set_expand();
35      data_label.set_expand();
36
37      // Connect the three buttons:
38      quit_button.signal_clicked().connect(sigc::mem_fun(*this,
39                &BasicTimer::on_button_quit));
40      start_timer.signal_clicked().connect(sigc::mem_fun(*this,
41                &BasicTimer::on_button_add_timer));
42      stop_timer.signal_clicked().connect(sigc::mem_fun(*this,
43                &BasicTimer::on_button_delete_timer));
44  }
```

le constructeur **BasicTimer()** initialise

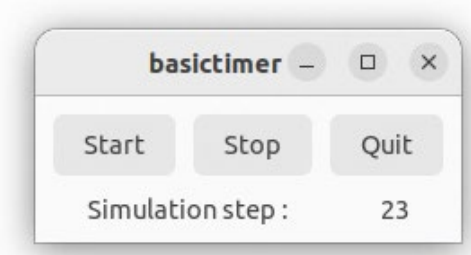
les attributs d'état à ***false*** et la durée à 500 ms



Exemple2: BasicTimer(3)

```
51  void BasicTimer::on_button_add_timer()
52  {
53      if(not timer_added)
54      {
55          // Creation of a new object prevents long lines and shows us a little
56          // how slots work.  We have 0 parameters and bool as a return value
57          // after calling sigc::bind.
58          sigc::slot<bool()> my_slot = sigc::bind(sigc::mem_fun(*this,
59                                                  &BasicTimer::on_timeout));
60
61          // This is where we connect the slot to the Glib::signal_timeout()
62          auto conn = Glib::signal_timeout().connect(my_slot,timeout_value);
63
64          timer_added = true;
65
66          std::cout << "Timer added" << std::endl;
67      }
68      else
69      {
70          std::cout << "The timer already exists : nothing more is created"
71                    << std::endl;
72      }
73  }
74
75  void BasicTimer::on_button_delete_timer()
76  {
77      if(not timer_added)
78      {
79          std::cout << "Sorry, there is no active timer at the moment." << std::endl;
80      }
81      else
82      {
83          std::cout << "manually disconnecting the timer " << std::endl;
84          disconnect  = true;
85          timer_added = false;
86      }
87  }
```

le signal handler du bouton **Start** crée un timer seulement si **timer_added** est ***false*** et fait alors passer cet attribut dans l'état ***true***



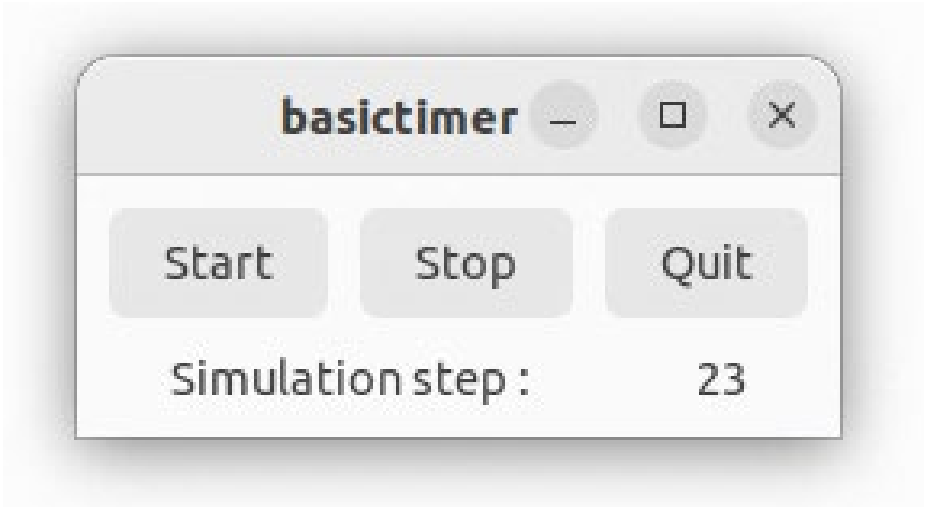
le signal handler du bouton **Stop** met à jour les attributs d'état seulement si **timer_added** est ***true*** ; dans ce cas **timer_added** passe à ***false*** et l'attribut **disconnect** passe à ***true***

Exemple2: BasicTimer(4)

```
89  bool BasicTimer::on_timeout()
90  {
91      static unsigned int val(1);
92
93      if(disconnect)
94      {
95          disconnect = false; // reset for next time a Timer is created
96
97          return false; // End of Timer
98      }
99
100     data_label.set_text(std::to_string(val));  // display the clock
101
102     std::cout << "This is simulation update number : " << val << std::endl;
103
104     ++val;
105     return true;
106 }
```

destruction du timer

conservation du timer



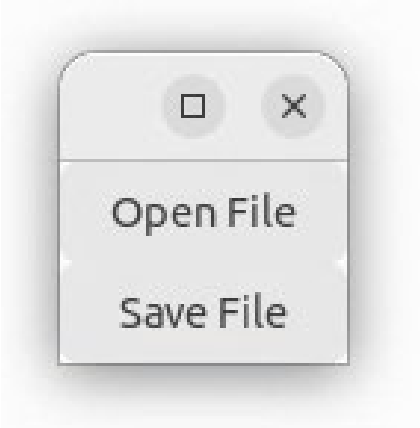

```
KBasicTimer$ ./basictimer
libEGL warning: DRI2: failed to authentica
te
Timer added
This is simulation update number : 1
This is simulation update number : 2
This is simulation update number : 3
This is simulation update number : 4
This is simulation update number : 5
This is simulation update number : 6
This is simulation update number : 7
This is simulation update number : 8
This is simulation update number : 9
manually disconnecting the timer
Timer added
This is simulation update number : 10
This is simulation update number : 11
The timer already exists : nothing more is
 created
This is simulation update number : 12
This is simulation update number : 13
This is simulation update number : 14
manually disconnecting the timer
```

Exemple3: lecture / écriture d'un fichier

But: gestion d'une fenêtre pop-up qui permet de **choisir un fichier** à ouvrir ou de donner un nom de fichier pour lancer une sauvegarde

examplewindow.h:

```
 9  public:
10    ExampleWindow();
11
12  private:
13    //Child widgets:
14    Gtk::Box m_Box;
15    Gtk::Button m_Button_Open;
16    Gtk::Button m_Button_Save;
17
18    std::string filename;
19
20    //Signal handlers:
21    void on_button_open_clicked();
22    void on_button_save_clicked();
23    void on_file_dialog_response(int response_id, Gtk::FileChooserDialog* dialog);
24  };
```



2 boutons, chacun avec son signal handler, resp.

`on_button_open_clicked()` **et**

`on_button_save_clicked()`

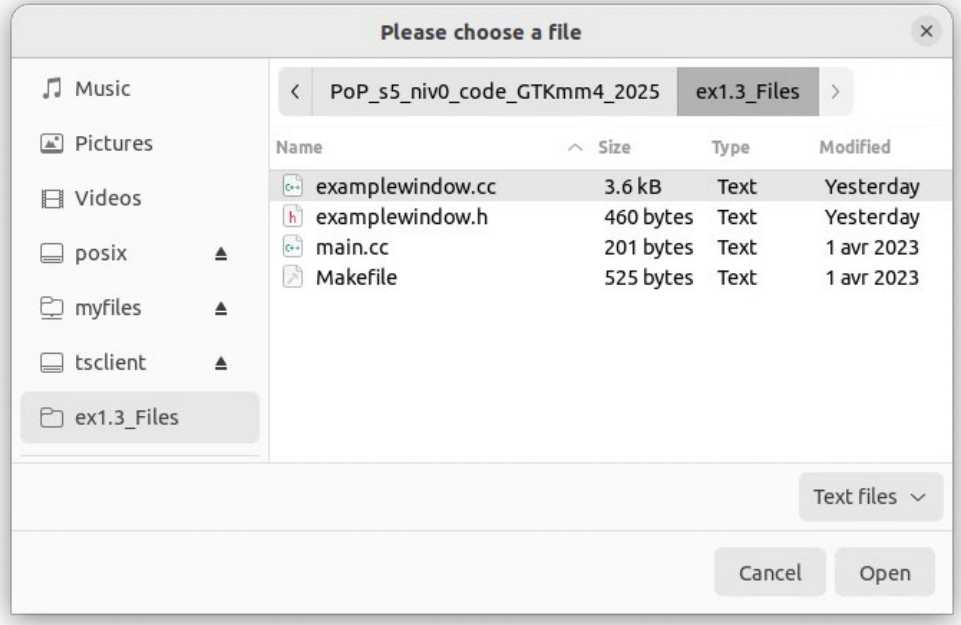
\+ le signal handler de la fenêtre pop-up de dialogue qui servira pour les 2 actions Open et Save:

`void on_file_dialog_response`

```cpp
void ExampleWindow::on_button_open_clicked()
{
    auto dialog = new Gtk::FileChooserDialog("Please choose a file",
          Gtk::FileChooser::Action::OPEN);
    dialog->set_transient_for(*this);
    dialog->set_modal(true);
    dialog->signal_response().connect(sigc::bind(
    sigc::mem_fun(*this, &ExampleWindow::on_file_dialog_response), dialog));

    //Add response buttons to the dialog:
    dialog->add_button("_Cancel", Gtk::ResponseType::CANCEL);
    dialog->add_button("_Open", Gtk::ResponseType::OK);

    //Add filters, so that only certain file types can be selected:

    auto filter_text = Gtk::FileFilter::create();
    filter_text->set_name("Text files");
    filter_text->add_mime_type("text/plain");
    dialog->add_filter(filter_text);

    auto filter_cpp = Gtk::FileFilter::create();
    filter_cpp->set_name("C/C++ files");
    filter_cpp->add_mime_type("text/x-c");
    filter_cpp->add_mime_type("text/x-c++");
    filter_cpp->add_mime_type("text/x-c-header");
    dialog->add_filter(filter_cpp);

    auto filter_any = Gtk::FileFilter::create();
    filter_any->set_name("Any files");
    filter_any->add_pattern("*");
    dialog->add_filter(filter_any);

    //Show the dialog and wait for a user response:
    dialog->show();
}
```

Le signal handler alloue dynamiquement une instance de **Gtk::FileChooserDialog** à laquelle on connecte le signal handler **on_file_dialog_response()**

C'est dans ce second signal handler qu'on va récupérer le nom du fichier choisi



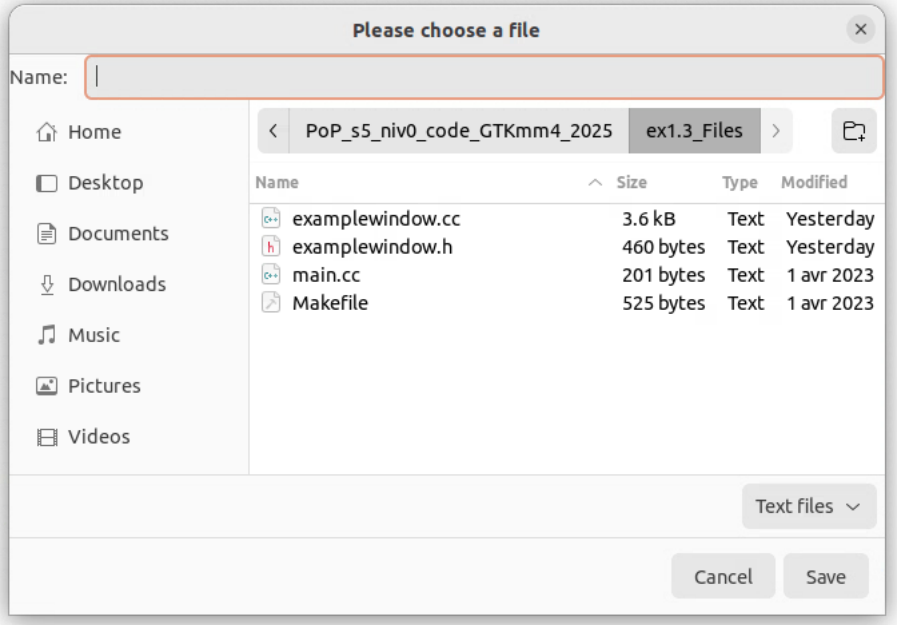
Exemple3: fichier(2)

exemple: FileChooser pour Open

```cpp
void ExampleWindow::on_button_save_clicked()
{
    auto dialog = new Gtk::FileChooserDialog("Please choose a file",
          Gtk::FileChooser::Action::SAVE);
    dialog->set_transient_for(*this);
    dialog->set_modal(true);
    dialog->signal_response().connect(sigc::bind(
    sigc::mem_fun(*this, &ExampleWindow::on_file_dialog_response), dialog));

    //Add response buttons to the dialog:
    dialog->add_button("_Cancel", Gtk::ResponseType::CANCEL);
    dialog->add_button("_Save", Gtk::ResponseType::OK);

    //Add filters, so that only certain file types can be selected:

    auto filter_text = Gtk::FileFilter::create();
    filter_text->set_name("Text files");
    filter_text->add_mime_type("text/plain");
    dialog->add_filter(filter_text);

    auto filter_cpp = Gtk::FileFilter::create();
    filter_cpp->set_name("C/C++ files");
    filter_cpp->add_mime_type("text/x-c");
    filter_cpp->add_mime_type("text/x-c++");
    filter_cpp->add_mime_type("text/x-c-header");
    dialog->add_filter(filter_cpp);

    auto filter_any = Gtk::FileFilter::create();
    filter_any->set_name("Any files");
    filter_any->add_pattern("*");
    dialog->add_filter(filter_any);

    //Show the dialog and wait for a user response:
    dialog->show();
}
```

Le signal handler alloue dynamiquement une instance de **Gtk::FileChooserDialog** à laquelle on connecte le signal handler **on_file_dialog_response()**

C'est dans ce second signal handler qu'on va récupérer le nom du fichier choisi

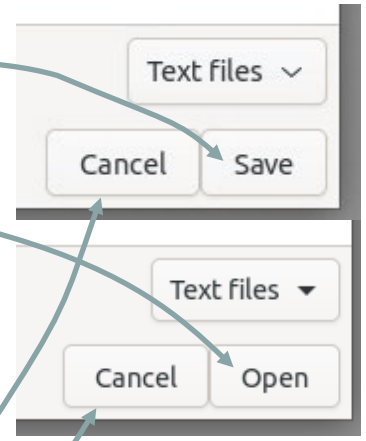


Exemple3: fichier(3)

exemple: FileChooser pour Save

Exemple3: fichier(4)

```
102 void ExampleWindow::on_file_dialog_response(int response_id,
103                                             Gtk::FileChooserDialog* dialog)
104 {
105     //Handle the response:
106     switch (response_id)
107     {
108         case Gtk::ResponseType::OK:
109         {
110             std::cout << "Open or Save clicked." << std::endl;
111
112             //Notice that this is a std::string, not a Glib::ustring.
113             filename = dialog->get_file()->get_path();
114             std::cout << "File selected: " <<  filename << std::endl;
115             break;
116         }
117         case Gtk::ResponseType::CANCEL:
118         {
119             std::cout << "Cancel clicked." << std::endl;
120             break;
121         }
122         default:
123         {
124             std::cout << "Unexpected button clicked." << std::endl;
125             break;
126         }
127     }
128     delete dialog;
129 }
```



Résumé

- **Le code d'une application graphique interactive est structuré par une boucle infinie de traitement de la file d'attente des événements dont GTKmm a le contrôle.**
- **La conception d'une interface avec GTKmm et plus généralement avec la programmation par événements implique de réfléchir différemment à la *manière de transmettre l'information* entre fonctions ou methodes.**
- **Le passage de paramètres habituel n'est pas toujours possible car les signal handlers (fonctions callback) doivent respecter des prototypes prédéfinis.**
- **Certains attributs sont des *variables d'état* qui jouent le role de relai / boite à lettre pour indiquer des changements d'état de l'application.**
- **on appelle `queue_draw()` pour forcer (indirectement) une demande de dessin**
- **L'outils de timer permet de produire des événements à intervalle régulier**
- **l'outil de File Chooser permet d'indiquer un nom de fichier à lire ou à créer**