

# Architecture d'un programme interactif graphique

## event clavier, timer, Lecture/Sauvegarde de fichier

### Objectifs:

- Usage des événements du clavier
  - exemple pour la gestion de l'activation d'une simulation
- Mise à jour synchronisée à un timer
- Lecture / Sauvegarde d'un fichier

### Remarques:

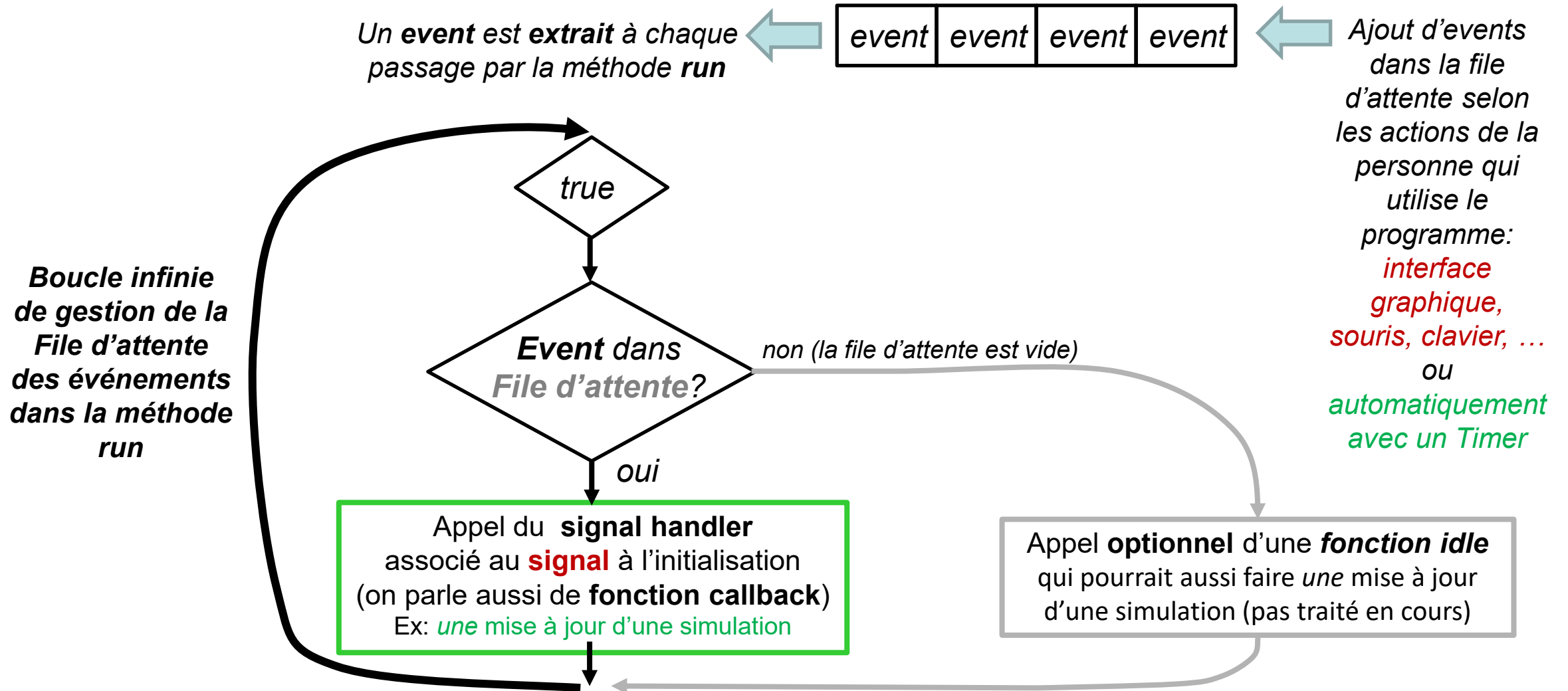
Revoir le cours de la semaine précédente sur la gestion d'événements liés à des boutons

Le code présenté dans ce cours est fourni et détaillé dans la série6 niveau 0.

# Rappel : La programmation par événements

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



# Usage des événements du clavier

Pop\_s6\_MyEvent\_avec\_clavier\_GTKmm4

`myevent.h`: ajouter le signal handler des événements du clavier dans la liste des signal handlers de la classe `MyEvent`:

```
bool on_window_key_pressed(guint keyval, guint keycode, Gdk::ModifierType state);
```

`myevent.cc`: on n'utilise que le premier paramètre `keyval` que nous devons convertir en code UNICODE pour faire un switch

```
bool MyEvent::on_window_key_pressed(guint keyval, guint, Gdk::ModifierType state)
{
    switch(gdk_keyval_to_unicode(keyval))    // les autres cas sont visibles dans myevent.cc
    {
        case 'C':
            std::cout << " action sur le lable du bouton Clear" << std::endl;
            m_Button_Clear.set_label("CLEAR");
            return true;

        }
        //the event has not been handled
        return false;
    }
}
```

Le slide suivant illustre comment 2 events du clavier peuvent permettre de modifier des attributs “d’état” de l’interface de la même manière que les boutons.

# Usage des événements du clavier (2)

## Fonctionnement général de la programmation par événements:

- les attributs “d’état” sont modifiés par des signals handlers en réponse à des événements (ex: **empty**)
- leur valeur courante est utilisée dans d’autres signal handlers pour adapter le comportement de l’application (on\_draw, timer...)

Si le Modèle est une simulation qu’il faut exécuter un grand nombre de fois pour voir sa progression, alors une des responsabilités du module «Controler» de l’interface graphique (gui) est de décider *quand demander* ces mises à jour au Modèle

Deux attributs booléens de l’interface gui suffisent :

- **started** initialisé à **false** pour lancer/stopper la simulation «en continu» avec un timer
- **step** initialisé à **false** pour demander *une seule mise à jour* lorsque la simulation est stoppée

Deux events du clavier suffisent pour gérer les changements d’état (illustration avec ceux demandés pour le projet):

- 's' fait basculer la valeur de **started** dans l’état opposé: **started = !started ;**
- '1' fait passer step à **true** si **started** est **false** : **step = !started ;**

valeur initiale et modifications en réponse à aux 2 events du clavier décrit ci-dessus :

event		's'	's'	'1'	'1'	's'	'1'
<b>started</b>	<b>false</b>	<b>true</b>	<b>false</b>	<b>false</b>	<b>false</b>	<b>true</b>	<b>true</b>
<b>step</b>	<b>false</b>	<b>false</b>	<b>false</b>	<b>true</b>	<b>true</b>	<b>false</b>	<b>false</b>

# Mise à jour synchronisée à un timer

Pop\_s6\_GTKBasicTimer

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

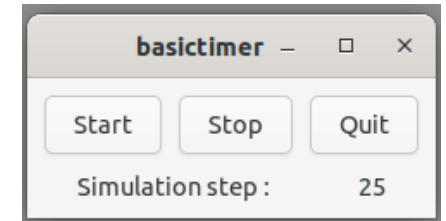
**basictimer.h:**

- 2 boutons gèrent 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() ;`
- cet exemple illustre aussi comment utiliser les **Gtk::Label**

pour faire afficher de l'information éventuellement variable

**basictimer.cc:**

- le constructeur **BasicTimer()** initialise les attributs d'état à **false** et la durée à 500 ms
- 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**
- le signal handler **on\_timeout()** du timer supprime le timer avec **return false** si l'attribut **disconnect** est **true** sinon il réalise sa tâche d'affichage et (important) **renvoie true**.



```
KBasicTimer$ ./basictimer
libEGL warning: DRI2: failed to authenticate
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
```

```

void BasicTimer::on_button_add_timer()
{
    if(not timer_added)
    {
        sigc::slot<bool()> my_slot = sigc::bind(sigc::mem_fun(*this,
                                                    &BasicTimer::on_timeout));
        auto conn = Glib::signal_timeout().connect(my_slot, 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

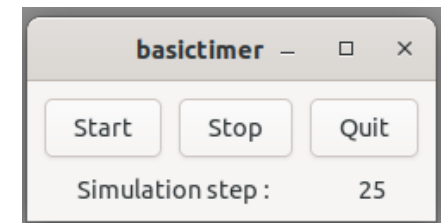
        return false; destruction du timer
    }
    // display the simulation clock
    data_label.set_text(std::to_string(val));

    std::cout << "This is simulation update number : " << val << std::endl;
    ++val;

    return true; conservation du timer
}

```

création du timer



```

KBasicTimer$ ./basictimer
libEGL warning: DRI2: failed to authenticate
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

```

# Lecture / sauvegarde d'un fichier

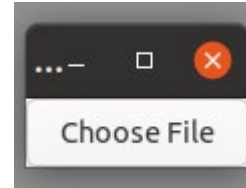
Pop\_s6\_GTKFile\_Chooser

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

**examplewindow.h:**

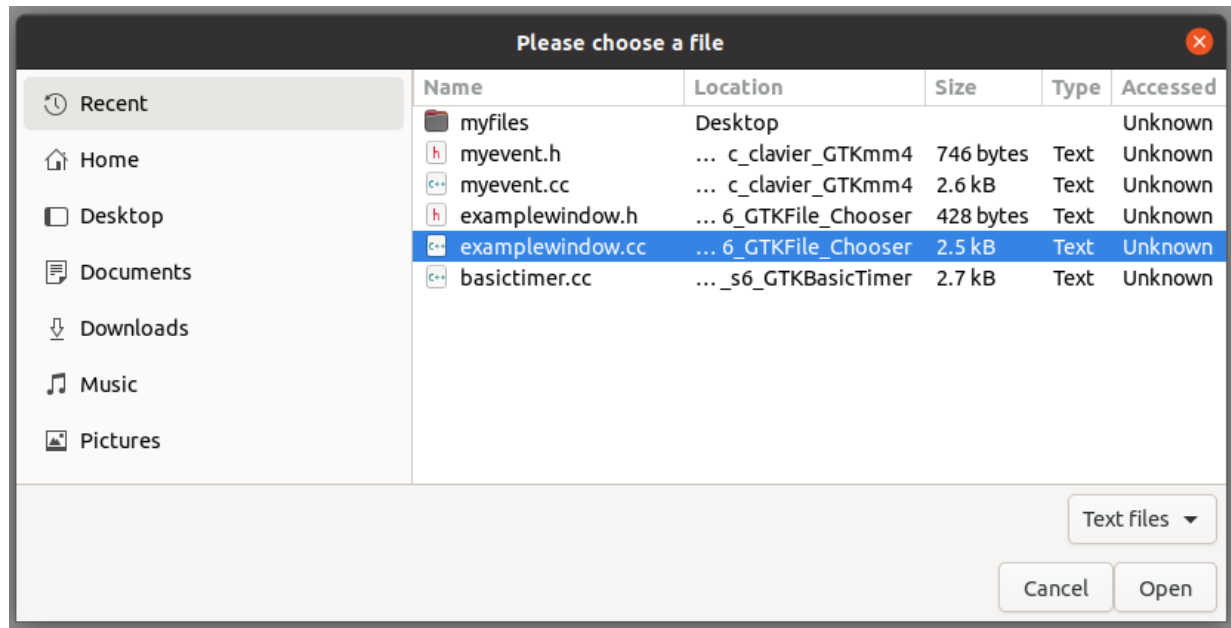
- un bouton avec son signal handler `on_button_file_clicked()`
- le signal handler de la fenêtre pop-up de dialogue:

```
void on_file_dialog_response(int response_id, Gtk::FileChooserDialog* dialog)
```

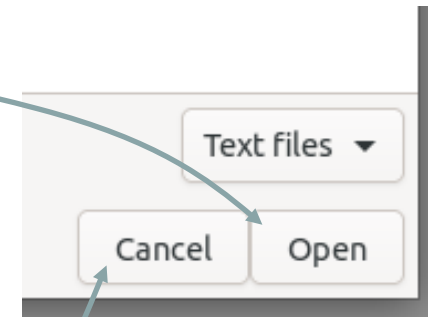


**examplewindow.cc:**

- le constructeur `ExampleWindow()` initialise seulement le signal handler du bouton
- Ce signal handler `on_button_file_clicked()` 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



```
void ExampleWindow::on_file_dialog_response(int response_id,  
                                           Gtk::FileChooserDialog* dialog)  
{  
    //Handle the response:  
    switch (response_id)  
    {  
        case Gtk::ResponseType::OK:  
        {  
            std::cout << "Open clicked." << std::endl;  
  
            //Notice that this is a std::string, not a Glib::ustring.  
  
            auto filename = dialog->get_file()->get_path();  
  
            std::cout << "File selected: " << filename << std::endl;  
            break;  
        }  
        case Gtk::ResponseType::CANCEL:  
        {  
            std::cout << "Cancel clicked." << std::endl;  
            break;  
        }  
        default:  
        {  
            std::cout << "Unexpected button clicked." << std::endl;  
            break;  
        }  
    }  
    delete dialog;  
}
```



la série6 niveau0 page9  
décrit comment adapter le  
code fourni pour réaliser la  
sauvegarde