

# GTK+

## Initialiser GTK

```
gtk_init(&argc,&argv);
```

## GTK+ Windows

```
GtkWidget *window=gtk_window_new(GTK_WINDOW_TOPLEVEL)
```

crée la fenêtre principale.

GTK\_WINDOW\_POPUP est une autre possibilité (pour créer tooltips et menus).

```
void gtk_window_set_title(GtkWindow *window, const gchar *title)
```

pour donner le titre à une fenêtre.

```
void gtk_widget_show(GtkWidget *widget) et void gtk_widget_hide(GtkWidget *widget)
```

pour afficher/cacher le widget.

La boucle principale lancée avec `gtk_main()` L'appel `gtk_main_quit()` termine l'application.

## Compilation

Sous linux:

```
gcc -Wall -g helloworld.c helloworld `pkg-config --cflags gtk+-2.0` \  
`pkg-config --libs gtk+-2.0`
```

## Conteneur

GtkWindow est une sous-classe de GtkContainer.

```
void gtk_container_add(GtkContainer *container, GtkWidget *child)
```

pour ajouter un widget dans un container.

```
void gtk_container_set_border_width(GtkContainer *container,  
guint border_width)
```

pour définir la largeur de bord.

## 3 Signaux et fonction callback

Les signaux sont émis par les widgets (contrairement aux événements qui sont émis par le serveur X, voir les événements dans la section suivante). Un événement peut déclencher un signal (voir les événements dans la section suivante).

Donc chaque widget possède sa liste de signaux.

Le seul signal que nous allons utiliser dans les exemples c'est `clicked` associé à `GtkButton` et envoyé quand l'utilisateur appuie sur le bouton.

Connecter un signal à une fonction callback:

```
,
gulong g_signal_connect(gpointer object,
                        const gchar *signal_name,
                        GCallback callback_function,
                        gpointer data);
```

Le premier paramètre: le widget qui envoie le signal, faire le cast comme `G_OBJECT(window)`.

`signal_name` le nom du signal.

`G_CALLBACK(callback_function)` la fonction callback qui sera appelée si le signal est émis.

`data` permet de passer des données à une fonction callback. Le format de la fonction callback dépend du signal.

### Fonctions callback

```
static void
callback_function(GtkWidget *widget,
                 /* d'autres paramètres éventuellement , */
                 gpointer data)
```

### Émission et suppression des signaux

```
void g_signal_emit_by_name(gpointer instance,
                          const gchar *signal_name,
                          ...);
```

Les paramètres à la fin de la liste de paramètres à passer au signal et la location pour mettre la valeur de retour.

```
void g_signal_stop_emission_by_name(gpointer instance,
                                    const gchar *signal_name);
```

pour arrêter l'émission d'un signal.

## Événements

Les événements sont émis par le serveur X-Window.

Quelques événements que nous allons utiliser:

- `delete-event` – window manager (l'utilisateur) demande la destruction de la fenêtre principale.
- `button-press-event` – si l'utilisateur a appuyé sur un bouton de la souris,
- `button-release-event` – le bouton de la souris est relâché.
- `button-notify-event` – le curseur de la souris bouge,

- `key-press-event` – on a appuyé sur une touche de clavier,
- `expose-event` – une partie ou tout widget apparaît sur l'écran et il faut le redessiner. L'évènement est émis si le widget a été caché par un autre objet. La fonction callback associée à cet évènement joue le même rôle que la méthode `paintComponent` en java, c'est là où dessine le widget.

On connecte les évènements de la même façon que les signaux (`g_signal_connect`). Par contre la fonction callback aura la forme

```
static gboolean
callback_function(GtkWidget *widget,
                  GdkEvent *event,
                  gpointer data);
```

c'est-à-dire elle retourne un booléen. Si TRUE retourné alors GTK+ assume que le traitement de l'évènement est terminé.

## 5 Liste

Chaque `GObject` possède une liste qui associe les chaînes de caractères à des pointeurs. Ce mécanisme peut être utilisé pour passer des données à une fonction callback.

```
void g_object_set_data(GObject *object,
                      const gchar *key,
                      gpointer data)
```

ajoute (`key,data`) sur la liste de l'objet `object`.

`g_object_get_data()` retourne le pointeur associé à `key`.

## 6 Conteneur widget

### 6.1 Boîte verticale et zone

Constructeur:

```
GtkWidget *gtk_vbox_new(gboolean homogeneous,
                        gint spacing)
```

crée `GtkVBox`. Si `homogeneous` est TRUE tous les enfants de même taille. `spacing` espace entre les enfants voisins.

Pour ajouter les enfants (avec les propriétés par défaut utilisez

```
void gtk_box_pack_start_defaults(GtkBox *box, GtkWidget *widget)
```

Si les propriétés par défaut ne convient pas alors on utilise:

```
void gtk_box_pack_start(GtkBox *box,
                        GtkWidget *child,
                        gboolean expand,
                        gboolean fill,
                        guint padding)
```

`expand` – si les cellules prennent la place libre, `fill` – si le widget enfant occupe toute la cellule.

## 2.1.2. Les macros

Les types de données Glib: `gboolean`, `gchar`, `guchar`, `gdouble`, `gfloat`, `gint`, `guint`, `gint8`, `guint8`, `gint16`, `guint16`, `gint32`, `gint64`, `glong`, `gulong`, `gpointer`, `gshort`, `gsize`, `gssize`.  
`gsize` unsigned int utilisé pour stocker une taille, `gssize` même chose signé.

Les macros :

`ABS(s)` valeur absolue,  
`G_DIR_SEPARATOR`, `G_DIR_SEPARATOR_S` séparateur de répertoires (slash backslash) selon le système, le premier retourne un caractère, le deuxième une chaîne de caractères,  
`TRUE`, `FALSE`  
`G_PI` - pi (précision 49 chiffres)  
`G_E` - e (précision 49 chiffres)

## 2.2. Utilisation de la mémoire

Utilisation de `malloc()` déconseillée.

Pour allouer les blocs de mémoire de même taille:

```
#include <glib.h>
gpointer g_slice_alloc(gsize block_size);
gpointer g_slice_alloc0(gsize block_size);
```

La deuxième fonction initialise la mémoire à 0. Pas besoin de vérification d'erreur, si l'appel échoue alors l'application est terminée automatiquement.

Pour libérer la mémoire

```
void g_slice_free1(gsize block_size, gpointer mem_block);
```

Impossible de changer la taille de la mémoire allouée avec `g_slice_alloc`.

Pour allouer un objet:

```
#define g_slice_new(type)
#define g_slice_new0(type)
```

retournent le pointeur vers type (pas besoin de faire cast).

Les fonctions suivantes sont juste “wrappers” des fonctions C standard:

```
gpointer g_malloc(gsize n_bytes);
gpointer g_malloc0(gsize n_bytes);
gpointer g_realloc(gpointer mem, gsize n_bytes);
gpointer g_try_malloc(gsize n_bytes);
gpointer g_try_malloc0(gsize n_bytes);
gpointer g_try_realloc(gpointer mem, gsize n_bytes);

void g_free(gpointer mem);
```

Seul avantage par rapport aux fonctions C – pas de vérification d'erreur sauf pour `try` où il faut vérifier si la valeur retournée est non NULL.

## 2.3. Fonctions utiles

```
g_get_current_dir(), g_get_home_dir()
```

## Timers

```
GTimer* g_timer_new(void);
void    g_timer_start(GTimer *timer);
void    g_timer_stop(GTimer *timer);
void    g_timer_continue(GTimer *timer);
gdouble g_timer_elapsed(GTimer *timer, gulong *microseconds);
void    g_timer_reset(GTimer *timer);
void    g_timer_destroy(GTimer *timer);
```

`g_timer_continue()` redémarre le timer arrêté avec `g_timer_stop()`

`g_timer_start()` peut être utilisée à chaque moment (`g_timer_reset` est inutile).

### 3.3 Timeout

C'est une fonction appelée à des intervalles réguliers jusqu'à ce qu'elle retourne `FALSE`.

```
guint g_timeout_add(guint interval,
                   GSourceFunc function,
                   gpointer data);
```

fonction c'est la fonction qui sera appelée, `data` – le paramètre que cette fonction recevra, `interval` en millisecondes entre deux appels.

```
guint g_timeout_add_full(gint priority,
                        guint interval,
                        GSourceFunc function,
                        gpointer data,
                        GDestroyNotify notify);
```

même chose mais on pourra spécifier la priorité: `G_PRIORITY_HIGH`, `G_PRIORITY_DEFAULT`, `G_PRIORITY_LOW` et d'autres. `notify` est la fonction appelé qu'on le timeout retourne `FALSE` (pour nettoyage).