Cours d'Interfaces Graphiques n°2

Edouard THIEL

Faculté des Sciences
Université d'Aix-Marseille (AMU)

Janvier 2016

Les transparents de ce cours sont téléchargeables ici :
http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ens/igra/

Lien court : http://j.mp/optigra

Plan du cours n°2

1. Scripts de compilation
2. Le type `GObject`
3. Conteneurs
4. Boutons et cases à cocher
5. Menus

1 - Scripts de compilation

Script pour compiler un programme GTK+ sans ou avec modules :

```
$ ./compgtk3.sh ex01.c
gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) -c ex01.c
gcc ex01.o -o ex01 $(pkg-config --libs gtk+-3.0)
Success, can run ./ex01

$ ./compgtk3.sh myapp.c gui.c info.c
gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) -c myapp.c
gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) -c gui.c
gcc -Wall -std=c99 $(pkg-config --cflags gtk+-3.0) -c info.c
gcc myapp.o gui.o info.o -o myapp $(pkg-config --libs gtk+-3.0)
Success, can run ./myapp
```

Un script de compilation avec modules : compgtk3.sh

```
#! /bin/bash

if (($#<1)); then
  echo "Usage: $0 prog1.c [module.c ...]" >&2 ; exit 1
fi

cf="-Wall -std=c99" ; gv="gtk+-3.0" ; ob=()
for i ; do
  n=${i%.c}
  test "$i" = "$n.c" || { echo "Not a C file." >&2 ; exit 1 ;}
  ob+=("$n.o")
  echo "gcc $cf \$(pkg-config --cflags $gv) -c $i"
  gcc $cf $(pkg-config --cflags $gv) -c "$i" || exit 1
done

echo "gcc "${ob[@]}" -o "${1%.c}" \$(pkg-config --libs $gv)"
gcc "${ob[@]}" -o "${1%.c}"  $(pkg-config --libs $gv) || exit 1
echo "Success, can run ./${1%.c}"
```

Scripts complémentaires (sans modules)

Pour recompiler ce qui a changé : `allcomp.sh`

```
#! /bin/bash

for f in *.c ; do
    test -s "$f" || continue  # fichier vide
    test "$f" -nt "${f%.c}.o" || continue
    ./compgtk3.sh "$f"
done
```

Pour supprimer les `.o` et exécutables : `allclean.sh`

```
#! /bin/bash

for f in *.c ; do
    test -f "${f%.c}.o" && rm -f "${f%.c}.o"
    test -f "${f%.c}" && rm -f "${f%.c}"
    test -f "$f~" && rm -f "$f~"
done
```

2 - Le type `GObject`

Le type d'objet de base est `GObject`, défini dans la GLib.

- Utilisé dans toutes les librairies : GTK+, Pango, Cairo, etc
- Utilisé pour les widgets, les pixbufs, les surfaces, etc
- Méthodes pour construire et détruire d'autres objets
- Méthodes pour accéder aux attributs (`get`, `set`)
- Méthodes pour gérer les signaux
- Macros de construction

Hiérarchie des widgets

```
GObject
   |-- GtkWidget
   |    |-- GtkContainer
   |    |    |-- GtkBin                    container avec 1 fils
   |    |    |    |-- GtkWindow
   |    |    |    |    |-- GtkDialog
   |    |    |    |    |    |-- GtkAboutDialog
   |    |    |    |    |    |-- GtkFileChooserDialog
   |    |    |    |    |    ...
   |    |    |    |    ...
   |    |    |    |-- GtkButton
   |    |    |         |-- GtkToggleButton
   |    |    |         |    |-- GtkCheckButton
   |    |    |         |    |    |-- GtkRadioButton
   |    |    |         |    |-- GtkMenuButton
   |    |    |         ...
   |    |    |-- GtkBox                      avec plusieurs fils
```

Poupées gigognes

```
typedef struct {
  GTypeInstance  g_type_instance;
  volatile guint ref_count;
  ...
} GObject;

typedef struct {
  GtkObject object;    // EN PREMIER
  guint8 state;
  ...
  GtkWidget *parent;
} GtkWidget;

typedef struct {
  GtkWidget widget;    // EN PREMIER
  GtkWidget *focus_child;
  ...
} GtkContainer;
```

```
GObject
    |-- GtkWidget
    |   |-- GtkContainer
```

Macros

```
typedef struct {
  GTypeInstance  g_type_instance;        // type opaque
  ...
} GObject;
```

`G_IS_OBJECT(obj)`
`GTK_IS_WIDGET(obj)`
`GTK_IS_CONTAINER(obj)`
} vérifie `obj != NULL` et `obj->g_type_instance`

`G_OBJECT(obj)`	caste en `(GObject*)`	+ vérifications
`GTK_WIDGET(obj)`	caste en `(GtkWidget*)`	+ vérifications
`GTK_CONTAINER(obj)`	caste en `(GtkContainer*)`	+ vérifications

...

Vérifications par les fonctions GTK

```
void gtk_container_add (GtkContainer *container,
                        GtkWidget    *widget)
{
  g_return_if_fail (GTK_IS_CONTAINER (container));
  g_return_if_fail (GTK_IS_WIDGET (widget));

  ...
}
```

`g_return_if_fail(expr)` : revient si `expr` est fausse + log erreur critique

Gestion de la mémoire

Tous les objets sont alloués dynamiquement.

Certains objets référencent d'autres objets :

un widget `image` utilise un `pixbuf`

Problème : quand libérer un objet ?

$\rightarrow$ Compteur de références

Compteur de référence

```
typedef struct {
  ...
  volatile guint ref_count;
  ...
} GObject;
```

```
GObject *obj;
```

`obj = g_object_new()`	compteur initialisé à 1.
`g_object_ref(obj)`	compteur incrémenté
`g_object_unref(obj)`	compteur décrémenté ; 0 $\rightarrow$ libération
`g_object_clear(&obj)`	idem + `obj := NULL`

3 - Conteneurs

Contient un ou plusieurs fils.

Donne leur géométrie et le focus clavier.

```
GObject
   |-- GtkWidget
       |-- GtkContainer
           |-- GtkBin                    container avec 1 fils
           |-- GtkBox                      avec plusieurs fils
           |-- GtkGrid
           |-- GtkToolbar
           |-- GtkMenuShell
           |-- ...
```

```
typedef struct {
  GtkWidget widget;
  GtkWidget *focus_child;
  ...
} GtkContainer;
```

Attachement simple

```
void gtk_container_add (GtkContainer *container,
                        GtkWidget *widget);
```

Utilisé pour attacher

- un seul fils dans `GtkWindow`, `GtkFrame`, `GtkButton`,...
- plusieurs fils avec réglages par défaut (éviter)

```
void gtk_container_remove (GtkContainer *container,
                           GtkWidget *widget);
```

Détache `widget` + unref

GtkBin

```
typedef struct {
  GtkContainer container;
  GtkWidget *child;
} GtkBin;
```

Utilisé en interne pour dériver des widgets avec un seul fils : `GtkWindow`, `GtkFrame`, `GtkButton`,...

`gtk_container_add` appelle la méthode *privée* `gtk_bin_add`

GtkBox

```
typedef struct {
  GtkContainer    container;
  GList           *children;
  GtkOrientation  orientation;
  gint16          spacing;
  ...
} GtkBox;
```

```
GtkWidget *gtk_box_new (GtkOrientation orientation,
                        gint spacing);
```

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

`expand`	peut se dilater
`fill`	prend toute la place qu'on lui donne
`padding`	espace supplémentaire autour du widget

GtkGrid

```
GtkWidget *gtk_grid_new (void);

void gtk_grid_attach (GtkGrid *grid,
                      GtkWidget *child,
                      gint left,
                      gint top,
                      gint width,     // nb colonnes
                      gint height);   // nb lignes
```

Voir aussi : `gtk_grid_attach_next_to()`

4 - Boutons et cases à cocher

```
GObject
   |-- GtkWidget
       |-- GtkContainer
           |-- GtkBin
               |-- GtkButton
```

```
typedef struct {
    GtkBin bin;
    gchar *label_text;
    GtkWidget *image;
    guint button_down;
    ...
} GtkButton;

GtkWidget *gtk_button_new_with_label (const gchar *label);

GtkWidget *gtk_button_new (void);
void gtk_button_set_image (GtkButton *button, GtkWidget *image);
```

Signal `clicked` pour un `GtkButton`

```
GtkWidget *button1;

button1 = gtk_button_new_with_label ("Press me");

g_signal_connect (button1, "clicked",
                  G_CALLBACK(on_button1_clicked), data);

void on_button1_clicked (GtkWidget *widget, gpointer data)
{
    printf("Bouton1 pressé\n");
}
```

GtkToggleButton : bouton qui reste enfoncé

```
GObject
    |-- GtkWidget
        |-- GtkContainer
            |-- GtkBin
                |-- GtkButton
                    |-- GtkToggleButton
```

```
typedef struct {
    GtkButton button;
    guint active;
    ...
} GtkToggleButton;
```

```
GtkWidget *gtk_toggle_button_new_with_label (const gchar *label);
gboolean gtk_toggle_button_get_active (GtkToggleButton *toggle);
void gtk_toggle_button_set_active (GtkToggleButton *toggle,
        gboolean is_active); // + signal "toggled"
```

Signal toggled pour un GtkToggleButton

```
GtkWidget *toggle1;

toggle1 = gtk_toggle_button_new_with_label ("Toggle me");

g_signal_connect (toggle1, "toggled",
                  G_CALLBACK(on_toggle1_toggled), data);

void on_toggle1_toggled (GtkWidget *widget, gpointer data)
{
    int actif = gtk_toggle_button_get_active (
        GTK_TOGGLE_BUTTON(widget));
    printf("Le toggle1 a changé d'état, il est maintenant %s\n",
        actif ? "enfoncé" : "relaché");
}
```

GtkCheckButton

```
GObject
   |-- GtkWidget
       |-- GtkContainer
           |-- GtkBin
               |-- GtkButton
                   |-- GtkToggleButton
                       |-- GtkCheckButton
```

```
GtkWidget *gtk_check_button_new_with_label (
    const gchar *label);
```

- Même comportement et même signal `toggled`
- Décoration différente : case cochée à gauche

GtkRadioButton

```
GObject
   |-- GtkWidget
       |-- GtkContainer
           |-- GtkBin
               |-- GtkButton
                   |-- GtkToggleButton
                       |-- GtkCheckButton
                           |-- GtkRadioButton
```

Boutons dans un groupe ; un seul actif à la fois

```
GtkWidget *gtk_radio_button_new_with_label_from_widget (
    GtkRadioButton *radio_group_member,  // NULL pour le 1er
    const gchar *label);
```

Signal `toggled` pour les boutons qui changent d'état
Utiliser `gtk_toggle_button_get_active()`

Exemple

```
char *noms[3] = {"Poireau", "Radis", "Tomate"};
GtkWidget *radios[3];

for (int i = 0; i < 3; i++) {
  radios[i] = gtk_radio_button_new_with_label_from_widget (
      i == 0 ? NULL : GTK_RADIO_BUTTON(radios[0]),
      noms[i]);

  g_signal_connect (radios[i], "toggled",
                    G_CALLBACK(on_radio_toggled), NULL);

  gtk_box_pack_start (GTK_BOX (...),
                      radios[i], FALSE, FALSE, 0);
}
```

Association de données

On peut associer des données à un GObject à l'aide de clés :

```
radio = gtk_radio_button_new_with_label_from_widget (...);
g_signal_connect (radio, "toggled", gtk_settings_get_default
                  G_CALLBACK(on_radio_toggled), data);
g_object_set_data (G_OBJECT(radio), "gout", "sucré");
g_object_set_data (G_OBJECT(radio), "numero", GINT_TO_POINTER(2));

void on_radio_toggled (GtkWidget *widget, gpointer data)
{
  int actif = gtk_toggle_button_get_active (
      GTK_TOGGLE_BUTTON(widget));
  const char *nom = gtk_button_get_label(GTK_BUTTON(widget));
  const char *gout = g_object_get_data (G_OBJECT(widget), "gout");
  gint numero = GPOINTER_TO_INT(
      g_object_get_data (G_OBJECT(widget), "numero"));
}
```

Exemple bis

```
char *noms[3] = {"Poireau", "Radis", "Tomate"};
GtkWidget *radios[3];

for (int i = 0; i < 3; i++) {
  radios[i] = gtk_radio_button_new_with_label_from_widget (
      i == 0 ? NULL : GTK_RADIO_BUTTON(radios[0]),
      noms[i]);

  g_object_set_data (G_OBJECT(radios[i]), "numero",
      GINT_TO_POINTER(i));

  g_signal_connect (radios[i], "toggled",
                    G_CALLBACK(on_radio_toggled), NULL);

  gtk_box_pack_start (GTK_BOX (...),
                      radios[i], FALSE, FALSE, 0);
}
```

5 - Menus

```
GObject
   |-- GtkWidget
       |-- GtkContainer
           |-- GtkMenuShell
               |-- GtkMenuBar
               |-- GtkMenu
```

Conteneurs de `GtkMenuItem`

```
GtkWidget *gtk_menu_bar_new (void);
GtkWidget *gtk_menu_new (void);
```

Menu-items

```
GObject
   |-- GtkWidget
       |-- GtkContainer
           |-- GtkBin
               |-- GtkMenuItem
                   |-- GtkCheckMenuItem
                   |   |-- GtkRadioMenuItem
                   |-- GtkImageMenuItem
                   |-- GtkSeparatorMenuItem
```

Éléments de menus

```
GtkWidget *gtk_menu_item_new_with_label (const gchar *label);
```

Attacher un menu-item dans un menu

```
void gtk_menu_shell_append (GtkMenuShell *menu_shell,
                            GtkWidget *child);
```

Exemple :

```
GtkWidget *menu_bar = gtk_menu_bar_new ();

GtkWidget *item1 = gtk_menu_item_new_with_label ("File");

gtk_menu_shell_append (GTK_MENU_SHELL(menu_bar), item1);
```

Attacher un sous-menu à un menu-item

```
void gtk_menu_item_set_submenu (GtkMenuItem *menu_item,
                                GtkWidget *submenu);
```

Exemple :

```
GtkWidget *item1 = gtk_menu_item_new_with_label ("File");

GtkWidget *submenu1 = gtk_menu_new ();

gtk_menu_item_set_submenu (GTK_MENU_ITEM(item1), submenu1);
```

Signal activate pour un menu-item

Pour les menu-item terminaux (sans sous-menu)

```
GtkWidget *item1 = gtk_menu_item_new_with_label ("File");

g_signal_connect (item1, "activate",
                  G_CALLBACK(on_item1_activate), data);

void on_item1_activate (GtkMenuItem *menuitem, gpointer data)
{
  const char *nom = gtk_menu_item_get_label(menuitem);
  printf ("Le menu-item %s a été activé\n", nom);
}
```

Très utile : `g_object_data_set()`, `g_object_data_get()`

Bouton avec menu popup

```
GObject
   |-- GtkWidget
       |-- GtkContainer
           |-- GtkBin
               |-- GtkButton
                   |-- GtkToggleButton
                       |-- GtkMenuButton
```

```
GtkWidget *gtk_menu_button_new (void);

void gtk_menu_button_set_popup (
    GtkMenuButton *menu_button, GtkWidget *menu);
```

Menubar déporté

Sous Unity ou MacOS, par défaut le menubar est déporté en haut de l'écran.

Pour réintégrer le menubar dans la fenêtre, rajouter dans `on_app_activate()` :

```
g_object_set (gtk_settings_get_default(),
  "gtk-shell-shows-menubar", FALSE, NULL);
```