C. Drocourt – 2007
UPJV - IUT Amiens
CNAM Picardie

Les widgets

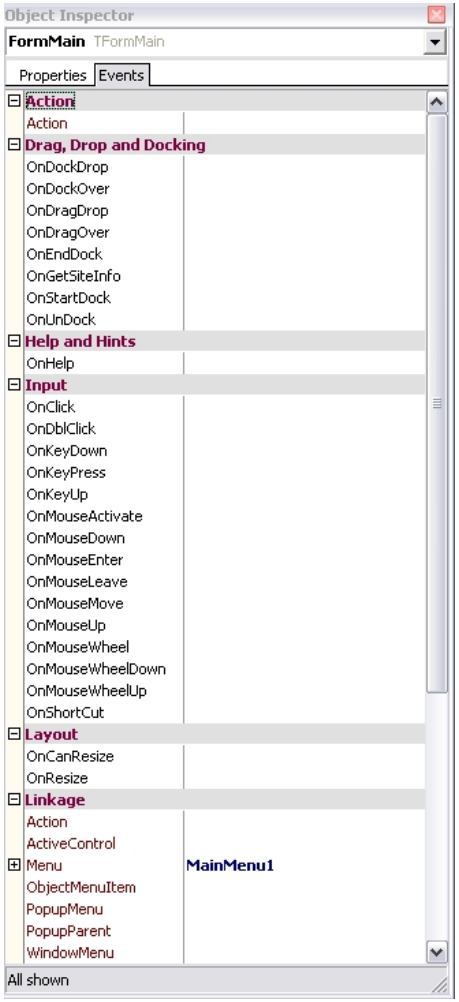
Nous avons vu dans les cours précédents la complexité de programmer en utilisant uniquement la Xlib, en effet il nous manque des objets essentiels comme les boutons, les menus, ... que l'on appelle les **widgets**. Voici une définition du site wikipedia :

> *Widget est un mot anglais qui est apparu aux États-Unis dans les années 1920. De manière générale, il est utilisé pour désigner un objet banal et quelconque, en français on peut le traduire par « machin » ou gadget. L'origine du mot est d'ailleurs probablement une déformation de ce dernier. Selon l'Office québécois de la langue française (www.granddictionnaire.com), on peut traduire widget par « métachose » ou bien « machin » ou encore « gadget logiciel ».*
> *En informatique, le mot widget recouvre deux notions distinctes en relation avec les interfaces graphiques (en ce sens, certains pensent que widget est un mot-valise formé des mots window (fenêtre) et gadget, ce qui signifierait donc « gadget de fenêtre »),il peut désigner :*
> - *Un élément de base d'une interface graphique (liste déroulante, bouton, etc.) que l'on peut désigner également par le terme calqué de l'anglais contrôle ;*
> - *Un petit outil qui permet d'obtenir des informations (météo, actualité, dictionnaire, carte routière, pense-bête (en anglais post-it), traducteur etc.)*

Plutôt que d'avoir à redéfinir ces objets soi même (ce qui reste possible évidemment), on utilisera une librairie qui proposera ce type d'objets avec évidemment un style différent pour chacune d'entres elles. Il est alors évident que l'ensemble de ces librairies reposent toutes sur la Xlib standard (en ce qui concerne la programmation Xwindow).

Dans ce cours nous allons nous intéresser plus particulièrement à l'une d'entre elle : GTK, qui a été créée initialement pour le projet GIMP, en effet GTK signifie Gimp ToolKit. Il existe évidemment d'autres toolkit comme Xt/Motif ou encore Qt (utilisé par KDE).

Programmation événementielle



Nous avons vu dans la programmation Xlib que les fenêtres d'un programme pouvaient recevoir des événements particuliers comme le déplacement de la souris, l'appui d'une touche, ... Le programmeur se doit ensuite de faire le tri, et de déterminer quels sont les actions à entreprendre en fonction de l'évènement, du contexte, ... Ce type de programmation est inadapté à la programmation par widgets, en effet, lorsque le programme reçoit un évènement, il faudrait déterminer par exemple ou se trouve le pointeur de souris pour savoir de quel objet il s'agit par exemple, d'où la nécessité d'une autre approche : La programmation évènementielle.

Dans la programmation événementielle ce sont les actions de l'utilisateur qui déclenchent des évènements associés à un objet particulier, par exemple, le fait de cliquer avec le curseur de la souris sur le bouton « A » va déclencher l'évènement « click » associé à l'objet « Bouton A », et uniquement celui ci. Le programmeur va donc définir pour chacun des objets qu'il utilise, les évènements gérés pour cet objet, et donc les actions associées à la gestion de cet évènement pour cet objet.

Premier programme

```
/* prog1.c */
#include <gtk/gtk.h>
int main (int argc, char *argv[])
{
    GtkWidget *window;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_widget_show (window);

    gtk_main ();

    return 0;
}
```

La compilation de ce programme va s'effectuer par la commande suivante :

```
gcc prog1.c -o prog1 `gtk-config --cflags --libs`
```

Ceci utilise GTK version 1, si vous voulez utiliser GTK2, il faudra taper :

```
gcc prog1.c -o prog1 `pkg-config --cflags --libs gtk+-2.0`
```

Exercice 1 : Testez ce programme avec les deux librairies.

Deuxième programme

```
/* prog2.c */
#include <gtk/gtk.h>
int main (int argc, char *argv[])
{
    GtkWidget *window;
    GtkWidget *button;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    button = gtk_button_new_with_label ("Bonjour");

    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button);
    gtk_widget_show (window);

    gtk_main ();

    return 0;
}
```

Exercice 2 : Testez ce programme avec les deux librairies.

Remarque : Plutôt que de réaliser un appel à la fonction `gtk_widget_show()` pour chaque objet, on peut le faire en une seule fois : `gtk_widget_show_all (window);`

Les fenêtres

- Nous avons vu dans les précédents programmes comment créer une fenêtre principale avec la fonction **`gtk_window_new ()`** qui prend en argument le type de fenêtre que l'on souhaite créer, le paramètre peut être GTK_WINDOW_TOPLEVEL, GTK_WINDOW_DIALOG ou GTK_WINDOW_POPUP.

- Pour donner un titre à notre fenêtre on utilisera la fonction :

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

  On peut récupérer le titre avec la fonction inverse :

```
G_CONST_RETURN gchar* gtk_window_get_title (GtkWindow *window);
```

- On pourra donner une taille spécifique à notre fenêtre en utilisant la fonction :

```
void gtk_window_set_default_size (GtkWindow *window, gint largeur, gint hauteur);
```

  Et récupérer cette taille avec la fonction :

```
void gtk_window_get_default_size (GtkWindow *window, gint *width, gint *height);
```

- Pour positionner une fenêtre avant son affichage on utilise :

```
void gtk_window_set_position(GtkWindow* window, GtkWindowPosition position);
```

  Avec l'argument « position » :
  - GTK_WIN_POS_NONE: la fenêtre aura une position aléatoire lors de son affichage ;
  - GTK_WIN_POS_CENTER : la fenêtre sera centrée à l'écran ;
  - GTK_WIN_POS_MOUSE : le coin supérieur droit de la fenêtre correspondra à la position de la souris au moment de l'affichage ;
  - GTK_WIN_POS_CENTER_ALWAYS : la fenêtre sera centrée et ne pourra être déplacée ;
  - GTK_WIN_POS_CENTER_ON_PARENT : la fenêtre sera centrée par rapport à la fenêtre parente.

  Pour déplacer cette fenêtre pendant le programme :

```
void gtk_window_move(GtkWindow *window, gint x, gint y);
```

  Pour récupérer cette position :

```
void gtk_window_get_position(GtkWindow *window, gint *root_x, gint *root_y);
```

Exercice 3 : Faites un programme *prog3.c* qui utilisera les fonctions précédentes (taille, position, titre), vous aurez besoin de la fonction GTK_WINDOW(<widget fenêtre>) pour éviter les messages d'avertissements.

Placement des objets

Nous avons vu précédemment que pour insérer un objet dans la fenêtre on utilisait la fonction *gtk_container_add()*, qui fonctionne pour un objet mais pas plusieurs !!! Pour cela nous allons devoir utiliser d'autres conteneurs ou boites (ou box) qui sont aussi des widgets. On peut par exemple créer des boites horizontales ou verticales avec :

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

où homogeneous précise si tous les widgets utilisent un espace identique et *spacing* précise l'espacement. Ensuite nous pouvons placer nos objets (boutons, ...) dans ces boites avec (la première insère de haut en bas et la deuxième fait l'inverse) :

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

avec *box* le conteneur (si celui ci est du type GtkWidget, on utilisera la fonction GTK_BOX(objet) pour réaliser un cast et éviter un message d'avertissement), *child* l'objet, *expand* l'expansion globale, *fill* l'expansion de la cellule et *padding* l'espace autour de l'objet.

Exercice 4 : Faites un programme *prog4.c* qui utilisera une boite horizontale contenant trois boutons, vous testerez également les différentes valeurs de *homogeneous*, *spacing*, *expand*, *fill* et *padding*.

On peut également utiliser des séparateurs avec :

```
GtkWidget* gtk_hseparator_new (void);
GtkWidget* gtk_vseparator_new (void);
```

Exercice 5 : Faites un programme *prog5.c* qui utilisera une boite horizontale contenant trois boites verticales, contenant chacune deux boutons et un séparateur.

La gestion des évènements

Pour associer la gestion d'un évènement particulier à un objet on utilise la fonction suivante :

```
gtk_signal_connect (GtkObject *object, const gchar *name, GtkSignalFunc *func, gpointer data);
```

Avec :

- *object* : l'objet considéré, ATTENTION, il faudra faire un cast avec **`GTK_OBJECT (<Objet Widget>)`** pour éviter les messages d'avertissements,
- *name* : le nom du signal à intercepter comme : `event`, `destroy`, `clicked`, `button_press_event`, `button_release_event`, `motion_notify_event`, `delete_event`, `destroy_event`, `expose_event`, `key_press_event`, `key_release_event`, `enter_notify_event`, `leave_notify_event`, `configure_event`, `focus_in_event`, `focus_out_event`, `map_event`, `unmap_event`, `property_notify_event`, `selection_clear_event`, `selection_request_event`, `selection_notify_event`, `proximity_in_event`, `proximity_out_event`, `drag_begin_event`, `drag_request_event`, `drag_end_event`, `drop_enter_event`, `drop_leave_event`, `drop_data_available_event`, `other_event`.
- *func* : la fonction qui sera appelée, ATTENTION : il faudra aussi faire un cast avec **`GTK_SIGNAL_FUNC (<fonction>)`**, de plus la fonction doit avoir le prototype suivant :

  ```
  void callback_func(GtkWidget *widget, gpointer *callback_data);
  ```

  Où *widget* est la widget concerné par l'évènement, et *callback_data* le paramètre passé à la fonction.
- *data* : un paramètre éventuel à passer à la fonction.

Remarques :

- On quitte proprement un programme GTK en appelant la fonction `gtk_main_quit ();`,
- Lorsque l'on ferme une fenêtre, le système génère d'abord l'évènement `delete_event`, puis si la fonction de callback existe et retourne 0, l'évènement `destroy` est généré.

Exercice 6 : Faites un programme *prog6.c* basé sur le précédent, qui quittera proprement, qui possèdera un bouton « Quit », un bouton permettant d'afficher un message sur la console, et un bouton qui déplacera la fenêtre.

Le widget label

Ce widget permet de placer du texte statique, c'est à dire non modifiable par l'utilisateur, sur l'interface (le label reste modifiable par le programme évidemment). Voici les principales fonctions utilisées avec les label :

- `GtkWidget* gtk_label_new(const char *str);`
- `void gtk_label_set_label(GtkLabel *label, const gchar *str);`
- `G_CONST_RETURN gchar* gtk_label_get_label(GtkLabel *label);`
- `void gtk_label_set_justify (GtkLabel *label, GtkJustification? Jtype);`
- `GtkJustification gtk_label_get_justify (GtkLabel *label);`

qui permettent respectivement de créer un label, de positionner un nouveau texte, de récupérer le texte, de positionner l'alignement (GTK_JUSTIFY_LEFT, GTK_JUSTIFY_RIGHT, GTK_JUSTIFY_CENTER ou GTK_JUSTIFY_FILL) et de récupérer l'alignement.

Afin de pouvoir manipuler des chaînes de caractères contenant des caractères accentués, il faudra la convertir auparavant en codage utf8, pour cela il faudra utiliser la fonction :

```
gchar* g_locale_to_utf8(const gchar *opsysstring,gsize len,gsize *bytes_read,gsize *bytes_written,GError **error);
```

qui convertit la chaîne *opsysstring*, et retourne la nouvelle chaîne, les paramètres suivants sont la taille (on mettra -1 pour laisser le système calculer lui même), le nombre d'octets réellement lus en cas d'erreur (NULL si pas besoin), le nombre d'octets réellement ecrits en cas d'erreur (NULL si pas besoin), et le type réel de l'erreur (NULL si pas besoin).

Attention : La chaîne de caractère renvoyée par la fonction est allouée dynamiquement et doit être libérée lorsque celle ci n'est plus utilisée par un appel à la fonction *g_free()*.

Exercice 7 : Faites un programme *prog7.c* qui utilisera plusieurs labels, vous testerez l'encodage utf8 et les caractères accentués.

Le widget GtkEntry

Afin de pouvoir utiliser une zone d'édition courte, pour la saisie d'un formulaire par exemple, on utilisera le widget GtkEntry, avec les fonctions disponibles suivantes :

- `GtkWidget* gtk_entry_new(void);`
- `G_CONST_RETURN gchar* gtk_entry_get_text(GtkEntry *entry);`
- `void gtk_entry_set_text(GtkEntry *entry, const gchar *text);`
- `void gtk_entry_set_max_length(GtkEntry *entry, gint max);`

qui permettent respectivement de créer une zone de saisie, de récupérer le texte de la zone, de positionner le texte voulu dans la zone de saisie et enfin de spécifier la taille maximale de la chaîne de caractères entrée par l'utilisateur. De plus, lorsque l'utilisateur appuie sur la touche entrée et que le focus se trouve sur une zone de saisie, un évènement « *activate* » est généré.

Lorsque vous devez demander à l'utilisateur la saisie d'un mot de passe, vous pouvez cacher les caractères entrés en utlisant la fonction :

```
void gtk_entry_set_visibility(GtkEntry *entry, gboolean visible);
```

Exercice 8 : Faites un programme *prog8.c* qui contiendra un widget de saisie, un bouton et un label. Lorsque l'utilisateur appuiera sur la touche entrée OU qu'il cliquera sur le bouton, le contenu de la zone de saisie sera recopié dans le label.

Le widget GtkImage

Pour l'affichage d'objets images, on utilisera la widget GtkImage, avec les fonctions de création suivantes :

- `GtkWidget* gtk_image_new (void);`
- `GtkWidget* gtk_image_new_from_file (const gchar *filename);`
- `GtkWidget* gtk_image_new_from_stock (const gchar *stock_id, GtkIconSize size);`

qui permettent de créer un objet vide, de créer un objet à partir d'un fichier image de type jpeg, png ou tiff, et enfin de créer un objet image à partir d'objets internes prédéfinis.

Pour modifier un objet de ce type on utilisera :

- `void gtk_image_set_from_file (GtkImage *image, const gchar *filename);`
- `void gtk_image_set_from_stock (GtkImage *image, const gchar *stock_id, GtkIconSize size);`

Exercice 9 : Faites un programme *prog9.c* qui contiendra un widget de type GtkImage et un bouton, lorsque l'utilisateur cliquera sur le bouton, une autre image s'affichera dans le widget.

Le widget GtkDrawingArea

Pour dessiner sur une zone particulière de la fenêtre, il faut utiliser le widget GtkDrawingArea. En fait, celui ci intègre un objet GDK (GIMP Drawing Kit), dans lequel nous allons pouvoir dessiner. Nous commençons par l'utilisation des fonctions suivantes :

- `GtkWidget* gtk_drawing_area_new (void);`
- `void gtk_drawing_area_size (GtkDrawingArea *darea, gint width,gint height);`

qui permettent respectivement de créer l'objet en question et de définir sa taille en pixels. La différence avec les autres widget est que les système ne gère pas les problèmes de ré affichage, c'est à dire que si l'objet passe de l'état caché à l'état visible, un évènement est généré, et c'est au programme de s'occuper du rafraîchissement de l'objet. Pour cela, il faut donc traiter le signal *expose_event*.

Pour dessiner dans notre zone, nous utiliserons les primitives suivantes :

- `void gdk_draw_point (GdkDrawable *drawable, GdkGC *gc, gint x, gint y);`
- `void gdk_draw_points(GdkDrawable *drawable, GdkGC *gc, GdkPoint *points, gint npoints);`
- `void gdk_draw_line(GdkDrawable *drawable, GdkGC *gc, gint x1_, gint y1_, gint x2_, gint y2_);`
- `void gdk_draw_lines(GdkDrawable *drawable, GdkGC *gc, GdkPoint *points, gint npoints);`
- `void gdk_draw_segments(GdkDrawable *drawable, GdkGC *gc, GdkSegment *segs, gint nsegs);`
- `void gdk_draw_rectangle(GdkDrawable *drawable, GdkGC *gc, gboolean filled, gint x, gint y, gint width, gint height);`
- `void gdk_draw_arc(GdkDrawable *drawable, GdkGC *gc, gboolean filled, gint x, gint y, gint width, gint height, gint angle1, gint angle2);`
- `void gdk_draw_polygon(GdkDrawable *drawable, GdkGC *gc, gboolean filled, GdkPoint *points, gint npoints);`
- `void gdk_draw_pixmap(GdkDrawable *drawable, GdkGC *gc, GdkDrawable *src, gint xsrc, gint ysrc, gint xdest, gint ydest, gint width, gint height);`
- `void gdk_draw_drawable(GdkDrawable *drawable, GdkGC *gc, GdkDrawable *src, gint xsrc, gint ysrc, gint xdest, gint ydest, gint width, gint height);`
- `void gdk_draw_image(GdkDrawable *drawable, GdkGC *gc, GdkImage *image, gint xsrc, gint ysrc, gint xdest, gint ydest, gint width, gint height);`
- `GdkImage* gdk_drawable_get_image(GdkDrawable *drawable, gint x, gint y, gint width, gint height);`
- `GdkImage* gdk_drawable_copy_to_image(GdkDrawable *drawable, GdkImage *image, gint src_x, gint src_y, gint dest_x, gint dest_y, gint width, gint height);`
- `void gdk_draw_string(GdkDrawable *drawable, GdkFont *font, GdkGC *gc, gint x, gint y, const gchar *string);`
- `void gdk_draw_text (GdkDrawable *drawable, GdkFont *font, GdkGC *gc, gint x, gint y, const gchar *text, gint text_length);`

Voici les explications pour les champs les plus essentiels :

- *drawable* : La zone dans laquelle il faudra dessiner. Cela pourra être directement le sous élément window de l'objet GtkDrawingArea (Exemple : monobjet->window), ou une autre surface de dessin (pixmap, ...).
- *GdkGC* : Le contexte graphique. On pourra utiliser par exemple les valeurs prédéfinies white_gc ou black_gc de l'élément style de l'objet GtkDrawngArea (Exemple : monobjet->style->black_gc).
- *x, y, src_x, src_y, ...* : Les positions de départs, d'arrivées, ... des éléments de dessin considérés.

Exercice 10 : Faites un programme *prog10.c* qui contiendra un widget de type GtkDrawingArea et essayez d'utiliser quelque unes des primitives de dessin vues précédemment. Vous devrez traiter l'évènement « *expose_event* ».

Le problème de cette méthode est qu'il nous faut dessiner à chaque fois dans notre objet GtkDrawingArea. Cela peut être coûteux si le dessin est complexe, et ceci nous oblige à connaître l'ensemble des primitives de dessin qui ont été utilisées. En fait, dans la pratique, on utilisera plutôt un pixmap, qui est un objet mémoire dans lequel nous allons pouvoir dessiner, que nous recopierons à chaque fois que cela sera nécessaire. Pour créer un objet de ce type nous allons utiliser :

```
GdkPixmap * pixmap=NULL;
GdkPixmap* gdk_pixmap_new(GdkDrawable *drawable, gint width, gint height, gint depth);
```

qui nous permet de créer un pixmap compatible avec la zone de dessin passé en argument (drawable), nous pouvons mettre « -1 » comme paramètre de profondeur. A l'inverse, si nous mettons NULL comme zone de dessin, il nous faut absolument préciser la profondeur « depth ». Nous pouvons maintenant utiliser ce pixmap exactement de la même manière que précédemment et le recopier à l'écran si nécessaire avec la primitive *gdk_draw_drawable()*. Nous pouvons récupérer le GC par défaut du dessin avec : *objet_dessin->style->fg_gc[GTK_WIDGET_STATE(objet_desin)]*.

Exercice 11 : Faites un programme *prog11.c* qui utilisera un pixmap pour dessiner dans la zone GtkDrawingArea