[Apprendre GTK]

Autheur : SPL3EN

Le but de ce tutorial sera de vous apprendre à coder, de façon simple, une application fenêtrée en utilisant GTK.

/!\ Attention :
Gardez à l'esprit que la manière dont nous allons procéder n'est pas forcément la plus « propre », mais seulement la manière la plus simple.
Une fois ces notions acquises, il vous sera aisé d'organiser votre code comme il vous semble.

Introduction

Tout d'abord, qu'est-ce que GTK ? Je ne vais pas faire un long discours, pour cela je vous invite à regarder la page Wikipedia concernant ce sujet.

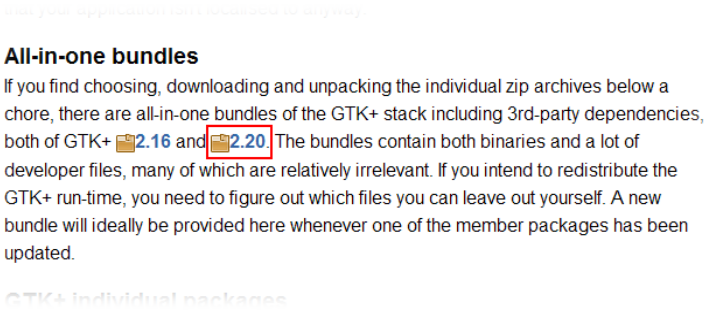
En résumé :
=> Une librairie multi-plateformes utile pour créer des GUI
=> Un ensemble de widgets codé en C, mais gardant un esprit orienté objet (existe aussi dans beaucoup d'autres langages : Python, C++, Php, Ruby, Java, Perl, ...)
=> **GNOME, Wireshark, GIMP, Inkscape, VMware, Pidgin** entre autres, ont été développés avec GTK. Sa renommée n'est donc plus à faire.

Ce tutorial demande quelques prérequis afin de pouvoir le suivre aisément :
- Vous devez connaître le langage C, au moins un peu. (même si chaque fonction seront décrites)
- Vous devez installer le **Framework GTK** ; Lisez la partie I (Installation) si vous souhaitez être guidé pour cette installation. (j'utilise l'IDE Code::Blocks ici)

Sommaire

I) Installation

Tout d'abord, vous pouvez télécharger la dernière version stable à ce lien : http://www.gtk.org/download.html
Pour Windows, je vous conseille d'installer le « *Bundle all-in-one* » ; Et de choisir la dernière version. (2.20 sur le screen, mais la version sera probablement différente lorsque vous lirez ce PDF)

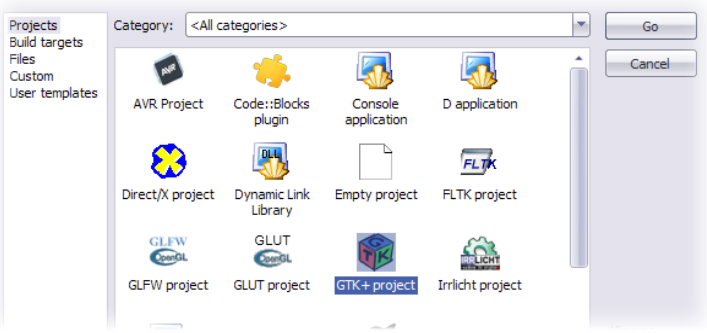


All-in-one bundles

If you find choosing, downloading and unpacking the individual zip archives below a chore, there are all-in-one bundles of the GTK+ stack including 3rd-party dependencies, both of GTK+ 2.16 and 2.20. The bundles contain both binaries and a lot of developer files, many of which are relatively irrelevant. If you intend to redistribute the GTK+ run-time, you need to figure out which files you can leave out yourself. A new bundle will ideally be provided here whenever one of the member packages has been updated.

Mettez l'archive dans le dossier de votre compilateur ; Par exemple, si vous avez MinGW avec Code::Blocks, mettez tout dans C:/Program Files/CodeBlocks/MinGW/

Après cela, vous pouvez créer un nouveau projet, puis choisir « GTK project ». Code::Blocks se chargera de faire les linkages à votre place de cette manière.



Essayez de compiler le code automatiquement généré par Code::Blocks … s'il n'y a aucune erreur, votre installation est fonctionnelle !

II) Créer une fenêtre

Codons un petit peu maintenant !

En premier lieu, nous allons nous intéresser à la manière dont on affiche une fenêtre.
> En GTK, chaque « objet » (buttons, fenêtres, entry, labels) est appelé « **GtkWidget** »
Pour cela, nous allons déclarer nos **GtkWidget** comme cela :

```
//Un pointeur sur GtkWidget portant le nom window, qui sera notre fenêtre :
GtkWidget *window;
```

Maintenant que c'est déclaré, voyons comment on l'initialise :

```
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

Et c'est tout !

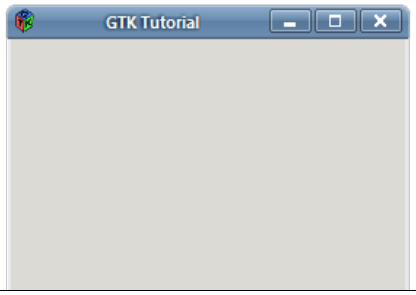
> Oui, bah j'ai essayé de mettre ça et compiler, ça ne marche pas ...

Evidemment : Nous n'avons pas encore initialisé GTK !
Appelons donc GTK, et demandons lui de garder cette fenêtre ouverte avec gtk_main().

On procède de cette façon :

```
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>

int main (int argc, char **argv)
{
    //On déclare notre fenêtre window
    GtkWidget *window;

    // On demande à GTK de s'initialiser : il faut toujours faire passer argc et argv en argument
    gtk_init ( &argc , &argv );

    // On initialise notre window
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    // On demande à GTK d'afficher notre window et tout ce qu'elle contient (rien pour l'instant)
    gtk_widget_show_all(window);

    // On demande à GTK de ne pas quitter et de laisser notre fenêtre ouverte
    gtk_main();

    return 0;
}
```

On compile ça,
et regardez ce qui apparaît...



... Magnifique !
Voici notre première fenêtre GTK :).

III) Ajouter quelques Widgets

Bon, maintenant, rajoutons à cette fenêtre quelques widgets afin de la rendre *un petit peu* plus intéressante :)

> J'ai entendu dire que ma fenêtre était considérée par GTK comme un "GtkContainer". Ca veut dire quoi ?

Un **GtkContainer** est, en GTK, un widget pouvant contenir **un seul** autre widget.
La fonction permettant d'ajouter un widget dans le container est :

```
gtk_container_add(GTK_CONTAINER(window), widget);
```

En premier argument, le GtkContainer, et en deuxième argument, le GtkWidget à ajouter.

> Oula... Et c'est quoi GTK_CONTAINER() ?

C'est une macro qui permet de caster un pointeur GtkWidget en un pointeur GtkContainer.
En effet, quand on déclare notre fenêtre, on utilise un pointeur sur GtkWidget.
Or, la fonction `gtk_container_add` prend pour premier argument un pointeur sur GtkContainer ...
Le compilateur émet donc un warning lors de la compilation, même si le programme marchera bien.
Pour pallier à ce warning, on utilise donc cette macro !
Il en existe généralement une pour chaque type de widget : GTK_WIDGET, GTK_WINDOW, GTK_LABEL, ...

La hiérarchie des GtkContainer en terme d'héritage est la suivante :

```
GObject
  +----GInitiallyUnowned
         +----GtkObject
                +----GtkWidget
                       +----GtkContainer
```

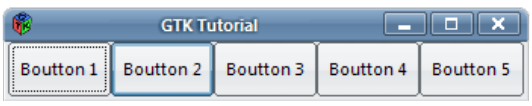
Ici, tout ce qui nous intéresse est de savoir qu'un GtkContainer est un GtkWidget.

Le problème qui se pose donc est le suivant :
Nous voulons mettre plusieurs widget dans notre fenêtre, et pas uniquement qu'un seul !
Pour résoudre ce souci, on va mettre dans la fenêtre un autre type de Widget, qui permet de mettre pas seulement qu'un seul, mais une multitude de widgets :
On appelle ce type de widget les **GtkBox**.

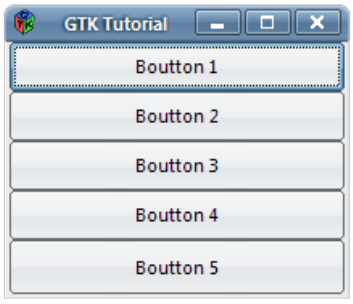
```
GObject
  +----GInitiallyUnowned
         +----GtkObject
                +----GtkWidget
                       +----GtkContainer
                              +----GtkBox
                                     +----GtkVBox
                                     +----GtkHBox
```

On voit bien ici que la GtkBox hérite du GtkContainer, ce qui semble tout à fait logique :)

On remarque l'existance deux différents types de GtkBox : GtkHBox et GtkVBox.
La différence entre ces deux là est la manière dont les widgets seront disposés lorsqu'ils seront ajoutés :
> La **GtkHBox** va ordonner ses widgets comme ceci :



> Alors que la **GtkVBox** comme cela :



> Comment as-tu fait pour mettre quelque chose dans ta box ?! Vite, donne moi le code !
Ah, ne soit pas impatient :)
Tout d'abord, nous allons voir la signification de quelques termes :

- L' homogenious, d'après la documentation :
"The homogeneous argument controls whether each object in the box has the same size (i.e., the same width in an hbox, or the same height in a vbox).
Bref, en d'autres termes, c'est ce qui permet de répartir les widgets à l'intérieur de la box de façon à ce qu'il aie tous la même taille ;
Par exemple sur le screenshot au dessus, on remarque que tous les boutons ont la même taille, car j'ai utilisé l'homogeneous.

- Le padding :
C'est tout simplement l'espacement entre chaque widget de la box, en pixels.
Exemple, si elle est égale à 10, il y aura un espacement de 10 pixels entre chaque widgets dans la box.

Maintenant qu'on sait cela, on peut voir la tête des fonctions d'initialisation des GtkBox :

```
GtkWidget *hbox = gtk_hbox_new( TRUE, 0 );
GtkWidget *vbox = gtk_vbox_new( TRUE, 0 );
```

Le TRUE correspond à l'homogenious, ici on l'active en le mettant à « vrai », et le 0 … au padding, comme vous l'avez certainement deviné :)

On crée donc ici une GtkVBox et une GtkHBox, toutes les deux homogènes et dont les widgets n'auront pas d'espacement entre eux.

> Ok, et donc, comment on met des widgets dedans ? Je veux savoir :(

On va procéder tout simplement comme cela :

```
gtk_box_pack_start_defaults(GTK_BOX(box), widget);
```

- En premier argument, la box où l'on veut rajouter quelque chose, (notez la macro GTK_BOX)
- En deuxième argument, le GtkWidget à rajouter...

Simple non ? :)

GTK se chargera de rajouter les uns à la suite des autres les widgets dans la Box, en **fonction de l'ordre** dans lequel vous les aurez ajoutés grâce à cette fonction.

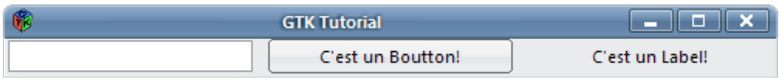
IV) Quelques différents types de Widgets

Bien, maintenant que l'on sait créer une fenêtre, créer des box, ajouter des widgets dans les box...
Quelle sorte de widgets allons nous utiliser ?

Voici une petite liste des widgets les plus communs :
Pour cela, nous allons étudier un bout de code, suivi de son explication, ça sera certainement plus explicite.

```c
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>

int main (int argc, char **argv)
{
    // On déclare nos Widgets
    GtkWidget *window;
    GtkWidget *hbox;
    GtkWidget *button;
    GtkWidget *label;
    GtkWidget *entry;

    // On initialise GTK
    gtk_init( &argc, &argv );

    // On initialise les Widgets
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    hbox = gtk_hbox_new( TRUE, 10 );
    button = gtk_button_new_with_label("C'est un Boutton !");
    label = gtk_label_new("C'est un Label !");
    entry = gtk_entry_new();

    // Tout d'abord, on met la GtkHBox dans la GtkWindow :
    gtk_container_add(GTK_CONTAINER(window), hbox );

    // Puis on met les Widgets dans la GtkHBox:
    gtk_box_pack_start (GTK_BOX(hbox), entry, TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX(hbox), label, TRUE, TRUE, 0);

    // On demande enfin à GTK de montrer notre Window et ce qu'elle contient :
    gtk_widget_show_all(window);

    // Puis on entre dans la boucle GTK qui garde la fenêtre ouverte
    gtk_main();

    return 0;
}
```

On le compile ...



Et voilà le travail :) !
Voyons ici plus en détail les Widgets utilisés :

- GtkLabel : C'est un Widget qui permet de mettre du texte non éditable dedans.

On peut y effectuer quelques fonctions amusantes dessus, mais sa principale utilisation se limitera à mettre du texte dedans, c'est tout ce qu'on lui demande après tout.

- GtkButton : C'est un Widget qui, comme son nom l'indique, porte la forme d'un bouton.

Il possède comme principale vocation celle d'être cliqué (si, si, en vrai), et nous verrons à la suite du tutorial comment faire. Le lien ci-dessus vous mènera à la documentation de ce Widget, qui référence absolument toutes les fonctions liées aux GtkButton. C'est parfois dur de s'y retrouver au début, mais une fois habitué, vous verrez que cette documentation sera votre meilleure amie :)
Remarquez que j'utilise la fonction `gtk_button_new_with_label` qui permet d'ajouter directement un GtkLabel à l'intérieur du bouton afin d'y ajouter du texte dedans.

- GtkEntry : C'est le Widget qui vous servira généralement à entrer du texte.

En d'autres termes, c'est l'équivalent d'une <input type="text"/> en HTML si ça peut simplifier !
Les fonctions les plus intéressantes et les plus basiques à utiliser avec ce widget sont :

```c
const gchar*        gtk_entry_get_text                  (GtkEntry *entry);
void                gtk_entry_set_text                  (GtkEntry *entry, const gchar *text);
```

Ces fonctions sont assez explicites :
La première fonction permet de récupérer le texte d'une GtkEntry, et la deuxième permet de mettre un texte dans la GtkEntry.

Pas d'inquiétude vis à vis des gchar : utilisez les de la même manière que les char habituels.
Glib (faisant partie du framework GTK+) offre en effet une multitudes de fonctions sur les chaînes de caractères, à la manière de celles de stdio.h ou string.h.
Vous les trouverez ici. Certaines sont intéressantes, donc cela vaut le détour :).
De la même manière, on retrouve des gint, gdouble, gfloat etc, avec leurs fonctions associés.
Bref, libre à vous d'utiliser des gint ou int, les gint n'étant que des simples *defines* d'int au final.

V) Fonctions Callback

C'est bien beau, mais rien ne se passe quand on clique sur le bouton … On va mettre un peu de vie dedans !
Dans le jargon GTK, on dit que l'on va **connecter** un **signal** sur le bouton, grâce à cette fonction :

```
g_signal_connect(button, "button-press-event", (GCallback)on_click_button, NULL);
```

button : Le Widget sujet à l'évenement
"button-press-event" : Le type de l'évenement ; Ici, quand on clic sur le bouton
on_click_button : La fonction qui sera appelée lorsque l'évenement se produit : on l'appelle fonction Callback.
NULL : Un pointeur sur une variable, si l'on souhaite faire passer quelque chose en argument dans la fonction.
Attention ! `g_signal_connect` n'autorise qu'un seul pointeur en dernier argument.

Maintenant, voyons comment se forme la fonction "on_click_button" :

```
gboolean on_click_button (GtkWidget *button, GdkEventButton *event, gpointer data)
/*
        La fonction Callback pour l'évenement « clicked » se présente toujours ainsi :
        - 1er argument, le GtkWidget sujet de l'event, ici notre bouton
        - 2èm argument, un pointeur sur l'évenement ; cette variable nous sera utile quand on
voudra traiter tel ou tel évenement de manière spécifique
        - 3èm argument, le pointeur sur ce qu'on a passé en paramètre dans le g_signal_connect.
        gpointer est la même chose qu'un (void *), le pointeur générique.
*/
{
    printf ("Hello World!");

    // On redonne la main à GTK
    return FALSE;
}
```

Vos fonctions Callback devront **toujours** avoir cette forme là.
Un petit point concernant la valeur de retour :

Ici, j'ai retourné FALSE. Pourquoi ?
Lors de l'appel d'une fonction Callback, il se peut que plusieurs Widget réagissent **au même signal**, ou bien qu'un même Widget soit connecté plusieurs fois pour le même signal.

En retournant FALSE, on demande à GTK de poursuivre l'évenement pour les autres Widgets.

Dans le cas présent pour la fonction on_click_button, si je retourne TRUE, vous verrez que l'animation du bouton « pressé » ne se fait pas ;
C'est normal : vous demandez avec cette valeur de retour à GTK de stopper toute interaction avec l'évenement, donc votre clic n'est plus pris en compte à la fin de votre fonction Callback si la valeur de retour est TRUE.
Faîtes le test, cela sera sûrement plus clair ainsi !

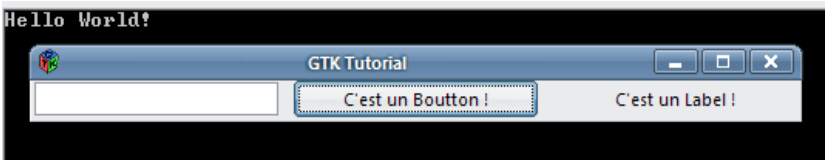
Pour prends un exemple plus simple, imaginons deux Widgets qui capturent ce que vous tapez au clavier.
Si l'un de ces Widget retourne TRUE à la fin de la fonction Callback appelée, l'autre Widget ne capturera pas le signal et sa fonction Callback ne s'effectuera pas !

Cette notion de valeur de retour n'est pas évidente au départ, mais le deviendra certainement avec un peu de temps...

Retour à nos moutons, testons maintenant ce code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>

gboolean on_click_button (GtkWidget *button, GdkEventButton *event, gpointer data)
{
    printf ("Hello World!");

    // On redonne la main à GTK
    return FALSE;
}

int main (int argc, char **argv)
{
    // On déclare nos Widgets
    GtkWidget *window;
    GtkWidget *hbox;
    GtkWidget *button;
    GtkWidget *label;
    GtkWidget *entry;

    // On initialise GTK
    gtk_init( &argc, &argv );

    // On initialise les Widgets
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    hbox = gtk_hbox_new( TRUE, 10 );
    button = gtk_button_new_with_label("C'est un Boutton !");
    label = gtk_label_new("C'est un Label !");
    entry = gtk_entry_new();

    // Tout d'abord, on met la GtkHBox dans la GtkWindow :
    gtk_container_add(GTK_CONTAINER(window), hbox );

    // Puis on met les Widgets dans la GtkHBox:
    gtk_box_pack_start (GTK_BOX(hbox), entry, TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX(hbox), label, TRUE, TRUE, 0);

    // On connecte le bouton à l'évenement « clicked »
    g_signal_connect(button, "button-press-event", (GCallback)on_click_button, NULL);

    // On demande enfin à GTK de montrer notre Window et ce qu'elle contient :
    gtk_widget_show_all(window);

    // Puis on entre dans la boucle GTK qui garde la fenêtre ouverte
    gtk_main();

    return 0;
}
```

Cela donne en console après appui sur le bouton :



Ca semble bien marcher :)

> Bon, bah maintenant j'aimerais bien récupérer ce qu'il y a dans la GtkEntry et le mettre dans le GtkLabel, c'est possible ?

Oui, tout à fait, mais on va se confronter au problème précédemment soulevé :
Rappelez vous de la fonction g_signal_connect : On ne peut faire passer qu'**UN** seul pointeur à la fonction Callback.
Or, ici nous avons besoin à la fois du pointeur sur la GtkEntry pour récupérer le texte, ainsi que celui sur le GtkLabel pour y mettre le texte …

Comment faire ?
Il existe plusieurs solutions, mais celle la plus simple est la suivante :
Nous allons créer une structure contenant tous nos pointeurs sur GtkWidget qui nous serons utiles.

Ainsi, généralement, la structure doit avoir à peu près cette forme là :

```
typedef struct s_Window
{
    GtkWidget *widget;
    GtkWidget *box;
    GtkWidget *label;
    GtkWidget *entry;
    GtkWidget *button;
} t_Window;
```

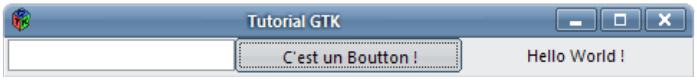
On va donc stocker dans notre variable de type t_Window ce qui nous intéresse.
Dans un code en C, ça donnerait :

```
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>

typedef struct s_Window
{
    GtkWidget *widget;
    GtkWidget *box;
    GtkWidget *label;
    GtkWidget *entry;
    GtkWidget *button;
} t_Window;

int main (int argc, char **argv)
{
    // On initialise notre t_Window
    t_Window *my_window = (t_Window*) malloc (sizeof (t_Window));
    if (my_window == NULL)
        exit(EXIT_FAILURE);

    // On initialise GTK
    gtk_init( &argc, &argv );

    // On initialise les Widgets de la variable my_window
    my_window->widget = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    my_window->box = gtk_hbox_new( TRUE, 0 );

    // … etc
```

La fonction Callback de ce que nous voulons faire sera la suivante :

```
gboolean on_click_button (GtkWidget *button, GdkEventButton *event, gpointer data)
{
    t_Window *my_w = (t_Window*) data;
    const gchar *buffer;

    // On récupère le texte contenu dans la GtkEntry
    buffer = gtk_entry_get_text(GTK_ENTRY(my_w->entry));

    // Puis on met ce texte dans le label
    gtk_label_set_text(GTK_LABEL(my_w->label), buffer);

    // Et pour finir, on vide la GtkEntry
    gtk_entry_set_text(GTK_ENTRY(my_w->entry), "");

    // On redonne la main à GTK
    return FALSE;
}
```

Maintenant, je compile, je met « Hello World ! » à l'intérieur de la GtkEntry puis j'appuie sur le bouton ...



Génial, ça marche ! :)

Pour ceux qui seraient un peu perdu, voici le code en entier :

```c
#include <stdio.h>
#include <stdlib.h>
#include <gtk/gtk.h>

typedef struct s_Window
{
    GtkWidget *widget;
    GtkWidget *box;
    GtkWidget *label;
    GtkWidget *entry;
    GtkWidget *button;
} t_Window;

gboolean on_click_button (GtkWidget *button, GdkEventButton *event, gpointer data)
{
    t_Window *my_w = (t_Window*) data;
    const gchar *buffer;

    // On récupère le texte contenu dans la GtkEntry
    buffer = gtk_entry_get_text(GTK_ENTRY(my_w->entry));

    // Puis on met ce texte dans le label
    gtk_label_set_text(GTK_LABEL(my_w->label), buffer);

    // Et pour finir, on vide la GtkEntry
    gtk_entry_set_text(GTK_ENTRY(my_w->entry), "");

    // On redonne la main à GTK
    return FALSE;
}

int main (int argc, char **argv)
{
    t_Window *my_window = (t_Window*) malloc (sizeof (t_Window));
    if (my_window == NULL)
        exit(EXIT_FAILURE);

    // On initialise GTK
    gtk_init( &argc, &argv );

    // On initialise les Widgets
    my_window->widget = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    my_window->box = gtk_hbox_new( TRUE, 0 );
    my_window->button = gtk_button_new_with_label("C'est un Boutton !");
    my_window->label = gtk_label_new("C'est un Label !");
    my_window->entry = gtk_entry_new();

    // Tout d'abord, on met la GtkHBox dans la GtkWindow :
    gtk_container_add(GTK_CONTAINER(my_window->widget), my_window->box );

    // Puis on met les Widgets dans la GtkHBox:
    gtk_box_pack_start (GTK_BOX(my_window->box), my_window->entry, TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX(my_window->box), my_window->button, TRUE, TRUE, 0);
    gtk_box_pack_start (GTK_BOX(my_window->box), my_window->label, TRUE, TRUE, 0);

    // On connecte le bouton à l'évenement « clicked »
    g_signal_connect(my_window->button, "button-press-event", (GCallback)on_click_button, my_wind
ow);

    // On demande enfin à GTK de montrer notre Window et ce qu'elle contient :
    gtk_widget_show_all(my_window->widget);

    // Puis on entre dans la boucle GTK qui garde la fenêtre ouverte
    gtk_main();

    return 0;
}
```

Voilà, ce tutorial est maintenant terminé !

Maintenant que vous savez connecter un signal et utiliser les fonctions des différents Widgets, il ne reste plus qu'à potasser la Documentation, accompagné des Tutoriaux mis à dispositions sur le site !
N'hésitez pas aussi à lire l'Annexe qui suit, qui vous renseignera sur les fonctions à savoir absolument sur les Widgets !

Bon courage, et n'hésitez pas à me poser vos questions par mail si vous en avez besoin.

VI) Annexe : ce qui est utile

-> GtkWidget

Documentation

Détruit un GtkWidget :
```
void gtk_widget_destroy (GtkWidget *widget);
→ Si ce widget est un GtkContainer, cette fonction s'occupera de détruire tous les GtkWidgets contenu dans ce GtkContainer... Pratique pour détruire une fenêtre proprement :)
```

Fais apparaître un GtkWidget à l'écran :
```
void gtk_widget_show (GtkWidget *widget);
```

Fais apparaître un GtkWidget ainsi que ses enfants (dans le cas d'un GtkContainer par exemple) à l'écran :
```
void gtk_widget_show_all (GtkWidget *widget);
```

Définit la taille minimale d'un GtkWidget :
```
void gtk_widget_set_size_request (GtkWidget *widget, gint width,  gint height);
→ Attention, l'utilisateur ne pourra pas resizer la fenêtre dans une taille en dessous de celle fixée par cette fonction. Utile pour des GtkWindow non resizable.
```

Récupère le parent d'un GtkWidget (s'il est contenu dans un GtkContainer) :
```
GtkWidget *gtk_widget_get_parent (GtkWidget *widget);
```

Récupère la fenêtre principale d'un GtkWidget
```
GdkWindow *gtk_widget_get_root_window (GtkWidget *widget);
→ Utile pour récupérer la GtkWindow dans une fonction Callback
```

-> GtkWindow

Documentation

Initialise une GtkWindow :
```
GtkWidget *gtk_window_new (GtkWindowType type);
```

Changer le titre de la fenêtre :
```
void gtk_window_set_title (GtkWindow *window, const gchar *title);
```

Initialiser la taille de la fenêtre :
```
void gtk_window_set_default_size (GtkWindow *window, gint width, gint height);
```

Définir si la fenêtre est redimensionnable ou non :
```
void gtk_window_set_resizable (GtkWindow *window, gboolean resizable);
```

Changer la position de la fenêtre :
```
void gtk_window_set_position (GtkWindow *window, GtkWindowPosition position);
```

Définir si la fenêtre possède des décorations :
```
void gtk_window_set_decorated (GtkWindow *window, gboolean setting);
```

Récupérer la taille de la fenêtre :
```
void gtk_window_get_size (GtkWindow *window, gint *width, gint *height);
```

Redimensionner la fenêtre :
```
void gtk_window_resize (GtkWindow *window, gint width, gint height);
```

Changer l'icône de la fenêtre :
```
void gtk_window_set_icon (GtkWindow *window, GdkPixbuf *icon);
```

Changer l'opacité de la fenêtre :
```
void gtk_window_set_opacity (GtkWindow *window, gdouble opacity);
```

-> GtkEntry

Documentation

Initialise une GtkEntry vide :
```
GtkWidget* gtk_entry_new (void);
```

Initialise une GtkEntry avec du texte :
```
GtkWidget* gtk_entry_new_with_buffer (GtkEntryBuffer *buffer);
```

Récupère le texte de la GtkEntry :
```
const gchar* gtk_entry_get_text (GtkEntry *entry);
```

Définit le texte de la GtkEntry :
```
void gtk_entry_set_text (GtkEntry *entry, const gchar *text);
```

Définit si le texte de la GtkEntry doit être visible (utile pour les mots de passe) :
```
void gtk_entry_set_visibility (GtkEntry *entry, gboolean visible);
```

Définit si la GtkEntry doit être éditable :
```
void gtk_entry_set_editable (GtkEntry *entry, gboolean editable);
```

Définit la taille maximum du texte de la GtkEntry :
```
void gtk_entry_set_max_length (GtkEntry *entry, gint max);
```

Définit une icône dans la GtkEntry :
```
void gtk_entry_set_icon_from_pixbuf (GtkEntry *entry, GtkEntryIconPosition icon_pos, GdkPixbuf
*pixbuf);
```

-> GtkButton

Documentation

Initialise un GtkButton vide :
```
GtkWidget *gtk_button_new (void);
```

Initialise un GtkButton avec un texte :
```
GtkWidget *gtk_button_new_with_label (const gchar *label);
```

Initialise un GtkButton avec une image :
```
GtkWidget *gtk_button_new_from_stock (const gchar *stock_id);
```
→ voir http://library.gnome.org/devel/gtk/stable/gtk-Stock-Items.html

-> GtkBox

Documentation

Initialise une GtkVBox :
```
GtkWidget *gtk_vbox_new (gboolean homogeneous, gint spacing);
```

Initialise une GtkHBox :
```
GtkWidget *gtk_hbox_new (gboolean homogeneous, gint spacing);
```

Introduit un GtkWidget dans la GtkBox :
```
void gtk_box_pack_start_defaults (GtkBox *box, GtkWidget *widget);
```

Introduit un GtkWidget dans la GtkBox avec plus de configurations :
```
void gtk_box_pack_start (GtkBox *box, GtkWidget *child, gboolean expand, gboolean fill, guint
padding);
```

-> GtkLabel

Documentation

Initialise un GtkLabel :
```
GtkWidget *gtk_label_new (const gchar *str);
```

Change le contenu du GtkLabel :
```
void gtk_label_set_text (GtkLabel *label, const gchar *str);
```

Définit des styles de polices du GtkLabel (voir la doc) :
```
void gtk_label_set_markup (GtkLabel *label, const gchar *str);
```

Définit le positionnement du texte dans le GtkLabel :
```
void gtk_label_set_justify (GtkLabel *label, GtkJustification jtype);
```

Récupère le contenu du GtkLabel :
```
const gchar *gtk_label_get_text (GtkLabel *label);
```

Définit si le GtkLabel est selectionnable :
```
void gtk_label_set_selectable (GtkLabel *label, gboolean setting);
```

Définit un angle de rotation du GtkLabel :
```
void gtk_label_set_angle (GtkLabel *label, gdouble angle);
```

-> GtkTable

Documentation

Initialise une GtkTable :
```
GtkWidget *gtk_table_new (guint rows, guint columns, gboolean homogeneous);
```

Attache un Widget sur la GtkTable aux coordonnées passées en paramètre :
```
void gtk_table_attach_defaults (GtkTable *table, GtkWidget *widget, guint left_attach, guint
right_attach, guint top_attach, guint bottom_attach);
```

Change la taille d'une GtkTable :
```
void gtk_table_resize (GtkTable *table, guint rows, guint columns);
```

-> GtkImage

Documentation

Initialise un GtkImage depuis un fichier (le format est reconnu automatiquement) :
```
GtkWidget *gtk_image_new_from_file  (const gchar *filename);
```

Initialise un GtkImage depuis un GdkPixbuf :
```
GtkWidget *tk_image_new_from_pixbuf (GdkPixbuf *pixbuf);
```

Initialise un GtkImage depuis un GdkPixmap :
```
GtkWidget *gtk_image_new_from_pixmap (GdkPixmap *pixmap, GdkBitmap *mask);
```

-> Les Events

Documentation

Il existe sous GTK une multitude d'évenements que vous pouvez connecter à vos Widgets, qui permettent d'enclencher une fonction en réaction à tel ou tel evénement produit.
Voici une liste de ceux les plus courants : (non exhaustive, voir la documentation)

Quand on appuie sur le bouton de la souris sur le Widget :
button_press_event

Quand on relâche le bouton de la souris :
button_release_event

Quand on effectue un scroll sur le Widget :
scroll_event

Quand on déplace la souris sur le Widget :
motion_notify_event

Quand on cherche à détruire le Widget (par exemple via la croix en haut à droite, ou bien gtk_widget_destroy) :
destroy_event

Quand on appuie sur une touche du clavier :
key_press_event

Quand on entre avec la souris dans la Widget :
enter_notify_event

Quand on sort la souris du Widget :
leave_notify_event

Note importante concernant l'event destroy :
Dans la plupart des cas, vous souhaiterez quitter correctement l'application en effectuant :

```
g_signal_connect ( window, «destroy_event», gtk_main_quit, NULL );
```

Ainsi, en appelant la fonction `gtk_main_quit`, vous quitter la boucle gtk_main, et votre programme se termine.