Cours Gtk+

http://www.gtk-fr.org

Version du 27 février 2006

Table des matières

Présentation de Gtk+

1. Qu'est-ce que GTK+?

La bibliothèque GTK+ permet de créer des interfaces graphiques (GUI) très facilement.

A l'origine, GTK+ a été développé pour donner des bases solides au logiciel de traitement d'images GIMP (GNU Image Manipulation Program). Aujourd'hui, le domaine d'application de cette bibliothèque ne se limite pas à GIMP : elle est utilisée dans d'autres projets. Le développement phare est l'environnement GNOME (GNU Network Object Model Environment).

L'utilisation de GTK+ pour la création de GUI est très intéressante :

- GTK+ est sous licence GNU LGPL. Cela fait de GTK+ une bibliothèque libre, permettant ainsi de l'utiliser voire de la modifier sans aucune contrainte financière. Si vous souhaitez en savoir plus, le plus simple est de visiter le site du projet GNU ;
- GTK+ existe sur plusieurs plates-formes : Linux et Bsd, Windows, BeOs;
- GTK+ est utilisable avec plusieurs langages de programmation. Même si les créateurs de GTK+ la développe en C, sa structure orientée objets et sa licence permettent à d'autres d'adapter GTK+ à leur langage préféré. Il est possible de programmer des GUI GTK+ en C, C++, Ada, Perl, Python, PHP et bien d'autres.

Les créateurs de GTK+ sont :

- Peter Mattis ;
- Spencer Kimball ;
- Josh MacDonald.

Actuellement, GTK+ est maintenu par :

- Owen Taylor ;
- Tim Janik.

Le site officiel de GTK+ est http://www.gtk.org.

2. Objectif du cours.

L'objectif de ce cours est de vous offrir un support en français pour la création de vos applications GTK+ en langage C. Ce cours développera en détail la majorité des fonctions de GTK+ tout en fournissant des exemples concrets. De ce fait, ce cours sera une sorte de tutorial couplé à un manuel de référence complet.

3. A qui s'adresse ce cours?

Ce cours est destiné plus particulièrement à trois types de programmeurs :

- les novices en programmation GUI ;
- les personnes connaissant d'autres GUI (API Win32, wxWindow) ;

- les personnes connaissant GTK+ 1.2.

Pour profiter pleinement de ce cours, vous devez avoir une connaissance du langage C.

4. Comment y contribuer ?

Tout le monde peut contribuer à ce cours grâce à l'interface Wiki que propose GtkFr pour la diffusion du cours. Il suffit pour cela de créer un compte afin de pouvoir éditer les différents chapitres du cours.

Installer GTK+

1. Installation de GTK+.

Avant toute chose, GTK+ s'appuie sur plusieurs bibliothèques. Il faut absolument les installer avant de pouvoir utiliser GTK+. Voici les différentes méthodes d'installation suivant l'outil avec lequel vous travaillez :

- Sous Linux, votre distribution inclut une version de GTK+. Seules les distributions les plus récentes proposent la version 2 de GTK+. La première étape consiste donc à vérifier quelle version est installée sur votre système à l'aide de la ligne de commande suivante : **pkg-config --modversion gtk+-2.0**. Si vous possédez déjà la version 2 de GTK+, vérifiez si une version plus récente n'existe pas sur le site officiel de GTK+. Si tel est le cas, vous avez deux solutions, soit il vous faut télécharger les fichiers sources de GTK+ sur le ftp officiel : ftp://ftp.gtk.org/pub/gtk/ puis utiliser les commandes d'installation classique (./configue make et install), soit utiliser le système de gestion de package de votre distribution.
- Sous Windows cela diffère un peu :
  - Avec DevCpp, vous pouvez soit utiliser le package Win32 disponible sur (section Téléchargement), soit utiliser un package auto-installer disponible à divers endroit sur Internet (section Liens) ,
- Avec Visual C++, le plus simple est d'utiliser le package Win32 disponible sur ce site ,
- Dans tous les cas vous pouvez vous faire votre propre pack en récupérant les fichiers nécessaires sur le site Tor Lillqvist en charge du portage de GTK+ sur les plates-formes Win32. Les fichiers à télécharger sont : libiconv, libintl, dirent, zlib, libpng, libjpeg, libtiff, freetype2, glib, atk, pango et gtk.

2. Configuration de votre compilateur.

Cette étape est obligatoire à la création d'un exécutable GTK+. Elle consiste surtout à configurer l'éditeur de liens et intervient à des moments différents selon votre outil de travail :

- avec gcc, il faut rajouter l'option `pkg-config --cflags --libs gtk+-2.0` à votre ligne de compilation. Exemple : gcc `pkg-config --libs --cflags gtk+-2.0` monprog.c -o monprog;
- avec Dev-Cpp, c'est la création du projet qui est importante. Dans la boîte de dialogue "New Project", sélectionner l'onglet GUI puis GTK+ et C Project ;
- avec Visual C++, il faut dans un premier temps, créer un nouveau projet Console. L'étape suivante consiste à modifier les propriétés du projet en ajoutant les librairies suivantes en entrée de l'éditeur de lien : gtk-win32-2.0.lib, gobject-2.0.lib et glib-2.0.lib.
- Vous pouvez aussi ajouter les options suivantes pour détecter lors de la compilation toutes les fonctions devenues obsolètes depuis la mise à jour des librairies GTK. Intéressant pour optimiser votre code :
  - DGTK_DISABLE_DEPRECATED=1
- DGDK_DISABLE_DEPRECATED=1
- DGDK_PIXBUF_DISABLE_DEPRECATED=1

- DG_DISABLE_DEPRECATED=1

Vous êtes maintenant prêt à créer votre première application GTK+.

3. Créer une application GTK+.

La première chose à faire est d'ajouter le fichier en-tête à votre code source :

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

Ensuite, dans votre fonction main, il faut initialiser GTK+ avec cette fonction :

```
void gtk_init(int *argc, char ***argv);
```

Les deux paramètres à passer à cette fonction correspondent aux paramètres reçus par la fonction main. Cette fonction récupère les valeurs passées par la ligne de commande et configure GTK+. Les différentes valeurs reconnues par GTK+ sont :

- --gtk-module ;
- --g-fatal-warnings ;
- --gtk-debug ;
- --gtk-no-debug ;
- --gdk-debug ;
- --gdk-no-debug ;
- --display ;
- --sync ;
- --name ;
- --class.

Voici le code source complet de notre première application. Elle ne fait strictement rien, mais il s'agit bien d'une application GTK+.

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

int main(int argc, char **argv)
{
    /* Initialisation de GTK+ */
    gtk_init(&argc, &argv);

    return EXIT_SUCCESS;
}
```

Les bases de la programmation Gtk+

1. La notion d'objet

Avant de commencer l'étude de la création d'interface graphique grâce à Gtk+, il faut savoir que les objets graphiques de Gtk+ sont appelés des widgets (Window Gadget). Un widget est en fait une structure définissant les propriétés d'un objet associé à une large panoplie de fonctions permettant de manipuler ces objets.

Ici, le terme "objet" est à prendre au sens littéral, mais aussi au sens Programmation Orientée Objet (POO). En effet, bien que GTK+ soit écrit en C, il introduit la notion d'héritage et les widgets de Gtk+ suivent une hiérarchie bien définie. Ainsi tous les objets graphiques héritent des propriétés et des fonctions d'un widget de base qui s'appelle GtkWidget.

Ainsi le widget permettant d'afficher une fenêtre (GtkWindow) a bien sûr ses propres fonctions, mais grâce à l'héritage il bénéficie aussi des fonctions des autres widgets dont il dérive.

2. La gestion des évènements

Afin de faire réagir une application aux actions de l'utilisateur, Gtk+les traite comme des évènements dans une boucle événementielle.

Lorsque l'utilisateur interagit avec l'application, qu'il clique sur un bouton, qu'il ferme une fenêtre, le widget concerné émet un signal spécifique. Chaque widget possède un signal pour chaque action possible.

Le signal est intercepté par une boucle événementielle. Celle-ci met en correspondance le signal envoyé et l'action spécifique prévue pour le signal. Si la correspondance est prévue, la fonction associée sera exécutée. Ces fonctions s'appellent fonction "callback".

Programmer avec les Widget revient à créer ces fonctions "callback" pour chacun des évènements susceptible d'avoir lieu pendant l'exécution du programme et à associer ces fonctions avec les signaux.

La première étape consiste donc à créer une fonction "callback". Dans la majorité des cas, ce sera sous cette forme :

```
void nom_de_la_fonction(GtkWidget *widget, gpointer data)
```

Le paramètre *widget* est le widget qui a émis le signal, et *data* est une donnée supplémentaire utile à l'application.

Pour connecter un signal à notre fonction "callback", Gtk+ utilise une fonction de la bibliothèque GLib :

```
gulong g_signal_connect(gpointer *object, const gchar *name, GCallback func, gpointer
func_data );
```

Le paramètre *object*, correspond au Widget qui émet le signal. Cependant, la variable demandée par g_signal_connect étant de type gpointer* (correspond à void* du C standard) et le widget de type GtkWidget*, il faut convertir ce dernier pour ne pas

provoquer d'erreur lors de la compilation. Pour cela Gtk+ (ou dans ce cas GLib) fournit une macro de conversion (G_OBJECT) qui sera à utiliser à chaque utilisation de cette fonction.

Le paramètre *name*, est le signal qui doit être intercepté par la boucle événementielle. Dans ce cours, certains signaux seront utilisés dans les exemples, mais la rubrique "En savoir plus" donnera une liste complète des signaux qui peuvent être émis par un widget.

Le paramètre *func*, définit la fonction callback à associer au signal. Cette fois encore, il faudra utiliser une macro de conversion qui est G_CALLBACK(func).

Le paramètre *func_data*, permet de spécifier une donnée quelconque à laquelle la fonction callback peut avoir accès pour effectuer son travail correctement.

Une fois que les signaux sont connectés, il faut lancer la boucle événementielle en appelant cette fonction :

```
void gtk_main(void);
```

Cela aura pour effet de faire entrer Gtk+ dans une boucle infinie qui ne pourra être stoppée que par l'appel de la fonction de fin de boucle qui est :

```
void gtk_main_quit(void);
```

Ces fonctions correspondent au minimum nécessaire afin de pouvoir créer une application Gtk+ correcte. D'autres fonctions permettent une utilisation avancée des signaux et de la boucle événementielle et sont traitées dans le chapitre GSignal du cours.

Les fenêtres

1. Présentation

Le widget GtkWindow est l'élément de base d'une interface graphique, car c'est ce widget qui va vous permettre de créer la fenêtre principale ainsi que toutes les autres fenêtres de votre programme.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin -> GtkWindow

2. Utilisation de base

2.1 Création de la fenêtre

Dans un premier temps, il faut déclarer un pointeur sur notre objet fenêtre. Bien que nous voulions créer un objet GtkWindow, il faut déclarer un objet GtkWidget.

```
GtkWidget *pWindow;
```

Par la suite, quel que soit l'objet à créer, il faudra toujours déclarer un GtkWidget.

Une fois l'objet pWindow déclaré, il faut l'initialiser. Pour cela, une fonction est à notre disposition :

```
GtkWidget* gtk_window_new(GtkWindowType type);
```

Le paramètre *type* définit le type de la fenêtre en cours de création, et peut prendre une des deux valeurs suivantes :

- GTK_WINDOW_TOPLEVEL : pour créer une fenêtre complète avec une zone réservée dans la barre des tâches ;
- GTK_WINDOW_POPUP : pour créer une fenêtre sans aucune décoration (barre de titre, bordure, ...).

La valeur de retour est le pointeur sur notre nouvel objet fenêtre. Cette fonction est donc à utiliser comme ceci :

```
pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

La création de la fenêtre est maintenant terminée.

2.2 Titre de la fenêtre

Pour définir le titre d'une fenêtre une fonction spécifique est à notre disposition. Cette dernière est très simple d'utilisation :

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

Le premier paramètre *window* correspond au widget que nous venons de créer. Mais

comme *pWindow* est de type GtkWidget il faut utiliser la macro de conversion GTK_WINDOW().

Le deuxième paramètre *title* est bien entendu le titre que l'on veut donner à la fenêtre (gchar correspond à char en C).

Au contraire, pour connaître le titre d'une fenêtre, la fonction est :

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

La variable qui recevra la valeur de retour de cette fonction devra être de type const gchar*.

2.3 Taille de la fenêtre

La fonction pour définir la taille par défaut de la fenêtre est :

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

Le paramètre *width* est la largeur de la fenêtre tandis que le paramètre *height* est sa hauteur.
Et très logiquement, la fonction pour connaître la taille par défaut de la fenêtre est :

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

2.4 Position de la fenêtre

La première fonction étudiée permet de définir la position de la fenêtre avant son affichage. Son prototype est le suivant :

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

Le deuxième paramètre *position* est la position que l'on veut donner à la fenêtre. Les valeurs acceptées sont :

- 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.

La deuxième fonction est utilisable à tout moment du programme et permet de donner la position exacte de la fenêtre :

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

Le premier paramètre est toujours la fenêtre dont on veut modifier la position. Les deux autres paramètres sont la nouvelle position de la fenêtre. Le paramètre *x* est la position suivant l'axe X (axe horizontal) et le paramètre *y*, la position suivant l'axe Y (axe vertical).

À l'inverse, pour connaître la position de la fenêtre, il faut utiliser cette fonction :

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

2.5 Affichage de la fenêtre

Le widget GtkWindow, comme tous les autres widgets, n'a pas de fonctions particulières concernant son affichage. Pour cela il faut utiliser une fonction du widget de base GtkWidget qui est :

```
void gtk_widget_show(GtkWidget *widget);
```

Cependant cette fonction ne s'occupera d'afficher que la fenêtre et pas les widgets qui ont été ajoutés à l'intérieur de celle-ci. Pour simplifier les choses, Gtk+ nous offre cette fonction qui affichera tout d'un seul coup :

```
void gtk_widget_show_all(GtkWidget *widget);
```

Il en est de même en ce qui concerne la destruction d'un widget, qui se fait avec cette fonction :

```
void gtk_widget_destroy(GtkWidget *widget);
```

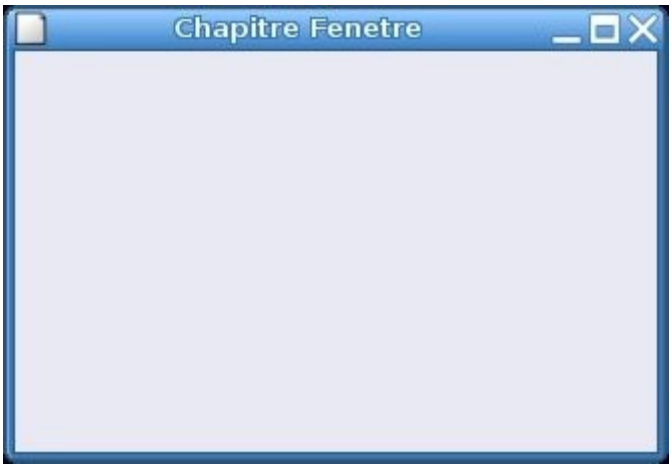
2.6 Programme exemple

Nous allons créer une fenêtre de taille 320x200, dont le titre sera "Chapitre Fenetre" et qui sera centrée à l'écran.

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

void OnDestroy(GtkWidget *pWidget, gpointer pData);

int main(int argc,char **argv)
{
    /* Declaration du widget */
    GtkWidget *pWindow;

    gtk_init(&argc,&argv);

    /* Creation de la fenetre */
    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    /* Definition de la position */
    gtk_window_set_position(GTK_WINDOW(pWindow), GTK_WIN_POS_CENTER);
    /* Definition de la taille de la fenetre */
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    /* Titre de la fenetre */
    gtk_window_set_title(GTK_WINDOW(pWindow), "Chapitre Fenetre");

    /* Connexion du signal "destroy" */
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(OnDestroy), NULL);
    /* Affichage de la fenetre */
    gtk_widget_show(pWindow);

    /* Demarrage de la boucle evenementielle */
    gtk_main();
```

```
    return EXIT_SUCCESS;
}

void OnDestroy(GtkWidget *pWidget, gpointer pData)
{
    /* Arret de la boucle evenementielle */
    gtk_main_quit();
}
```

Résultat



Les Labels

1. Présentation

Le widget GtkLabel va nous permettre d'afficher du texte à l'intérieur de nos fenêtres.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkMisc? -> GtkLabel

2. Utilisation de base

2.1 Création d'un label

Après avoir déclarer un objet pLabel de type GtkWidget*, il faut utiliser la fonction :

```
GtkWidget* gtk_label_new(const char *str);
```

Le paramètre *str* n'est autre que le texte qui sera affiché par le GtkLabel.

2.2 Modification du label

Il peut arriver qu'au cours d'un programme, l'on veuille modifier le texte qui est affiché. La fonction à utiliser est la suivante :

```
void gtk_label_set_label(GtkLabel *label, const gchar *str);
```

Le premier paramètre *label* correspond au widget dont on veux modifier le texte, et le paramètre *str* est le nouveau texte qui sera affiché.

2.3 Récupération du label

Pour récupérer le texte qui est affiché par un label, la fonction est :

```
G_CONST_RETURN gchar* gtk_label_get_label(GtkLabel *label);
```

Attention la valeur de retour est une variable constante, il ne faut donc pas essayer de la modifier.

2.4 Alignement du texte

Maintenant, nous allons voir comment aligner notre texte, lorsque celui-ci comporte plusieurs lignes. Comme d'habitude, GTK+ nous fourni une fonction très simple d'utilisation :

```
void gtk_label_set_justify (GtkLabel *label, GtkJustification? jtype);
```

Le paramètre jtype correspond à l'alignement du texte et peut prendre une de ces valeurs :

- GTK_JUSTIFY_LEFT pour aligner le texte à gauche (par défaut) ;
- GTK_JUSTIFY_RIGHT pour aligner le texte à droite ;
- GTK_JUSTIFY_CENTER pour centrer le texte ;
- GTK_JUSTIFY_FILL pour justifier le texte.

Au contraire, pour obtenir l'alignement du texte, la fonction est :

```
GtkJustification? gtk_label_get_justify (GtkLabel *label);
```

2.5 Programme exemple

Nous allons, comme programme exemple, utiliser le classique "Hello Word".

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

int main(int argc,char **argv)
{
    GtkWidget* pWindow;
    GtkWidget* pLabel;

    gtk_init(&argc,&argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "Les labels");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);

    /* Creation du label */
    pLabel=gtk_label_new("Hello World!");

    /* On ajoute le label a l'interieur de la fenetre */
    gtk_container_add(GTK_CONTAINER(pWindow), pLabel);

    /* Affichage de la fenetre et de tout ce qu'il contient */
    gtk_widget_show_all(pWindow);

    /* Connexion du signal
    /* On appelle directement la fonction de sortie de boucle */
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    gtk_main();

    return EXIT_SUCCESS;
}
```

Résultats

3. Les caractères accentués

Vous avez peut-être déjà essayé d'afficher du texte contenant des caractères tel que "é, è, à, ...", et lors de l'exécution, votre texte ne s'affiche complètement. Nous allons maintenant voir le pourquoi du comment.

3.1 Application test

Pour vous montrer tout cela, nous allons reprendre l'exemple précédent en remplaçant " Hello Word " par " Texte à afficher ", et cela ne marche pas! En effet vous obtenez cela :

Et en plus sur la console, vous avez le message d'erreur suivant :

```
** (Label2.exe) : WARNING **: Invalid UTF8 string passed to pango_layout_set_text()
```

3.2 Comment ça marche ?

Pour afficher du texte, Gtk utilise la librairie Pango qui s'occupe du rendu et de l'affichage du texte. Le but de Pango est de permettre l'internationalisation des applications, et pour cela Pango utilise l'encodage UTF8. Ce charset est codée sur 16 bits, ce qui offre la possibilité d'afficher plus 65000 caractères, permettant ainsi d'afficher des caractères accentués et bien plus (ex : pour le grec, le chinois, ...).

Puisque Pango sait faire tout cela, pourquoi le test n'a-t-il pas fonctionné? Et bien, tout simplement parce que votre système d'exploitation n'utilise pas ce charset. Lorsque vous souhaitez afficher "Texte à afficher", Pango va avant tout vérifier l'encodage d'un caractère et si l'un de ces caractères n'est pas correct, Pango va arrêter son processus de rendu et envoyer un message d'erreur.

Le moyen le plus simple pour voir les effets d'une erreur d'encodage est de changer celui de votre navigateur. Modifiez les options de votre navigateur afin d'utiliser l'encodage UNICODE. Cette page web utilisant le charset iso-8859-1, les caractères accentués deviendront des carrés ou autre chose.

3.3 Solution

Une seule chose s'impose à nous : il faut convertir notre chaîne de caractère en UTF8. Pour cela, il faut utiliser une fonction de Glib qui est :

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

La valeur de retour est le pointeur sur la chaîne de caractère fraîchement convertie. Si une erreur est survenue lors de la conversion, g_locale_to_utf8 renvoie NULL. Il faudra tout de même libérer le mémoire allouée par cette fonction lorsque la variable ne sera plus utile.

Le premier paramètre *opsysstring* est la chaîne de caractère à convertir et le second paramètre len correspond à la longueur de la chaîne. Ce paramètre sera en général égal à -1 (on laisse la bibliothèque calculer la longueur toute seule).
Les trois derniers paramètres sont utiles en cas d'erreur lors de la conversion. Tout d'abord *bytes_read* est le nombre d'octet qui ont été lus dans le texte à convertir, et *bytes_writen* le nombre d'octet qui ont été écris dans la nouvelle chaîne de caractères. Le dernier paramètre *error*, donne plus de précision en cas d'erreur. Voici la liste des messages d'erreur possibles :

- G_CONVERT_ERROR_NO_CONVERSION ;
- G_CONVERT_ERROR_ILLEGAL_SEQUENCE ;
- G_CONVERT_ERROR_FAILED ;
- G_CONVERT_ERROR_PARTIAL_INPUT ;
- G_CONVERT_ERROR_BAD_URI ;
- G_CONVERT_ERROR_NOT_ABSOLUTE_PATH.

Attention cette fonction alloue de la mémoire. Lorsque vous n'avez plus besoin de la valeur de retour de cette fonction, il faut libérer la mémoire à l'aide de la fonction **g_free**.

3.4 Programme exemple

Pour notre exemple, nous n'allons pas utiliser les trois derniers paramètres. En cas d'erreur, il n'y aura donc aucun moyen d'avoir plus de précision sur l'erreur survenue.

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

int main(int argc,char **argv)
{
```

```
    GtkWidget* pWindow;
    GtkWidget* pLabel;
    gchar* sUtf8;

    gtk_init(&argc, &argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "Les labels II");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);

    /* Creation du label avec g_locale_to_utf8 */
    sUtf8 = g_locale_to_utf8("Texte à afficher", -1, NULL, NULL, NULL);
    pLabel=gtk_label_new(sUtf8);
    g_free(sUtf8);

    /* On ajoute le label a l'interieur de la fenetre */
    gtk_container_add(GTK_CONTAINER(pWindow), pLabel);

    /* Affichage de la fenetre et de tout ce qu'il contient */
    gtk_widget_show_all(pWindow);

    /* Connexion du signal
    /* On appelle directement la fonction de sortie de boucle */
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    gtk_main();

    return EXIT_SUCCESS;
}
```

Résultats

4. Formatage du texte

Nous allons maintenant voir comment modifier l'apparence de notre texte (police, couleur, ...).
Pour notre application test, nous allons afficher une ligne en Courier gras taille 10, une ligne en Times New Roman italique bleu taille 12, et une ligne en Verdana souligné taille 16. Et tout cela centré (bien sûr).

4.1 Définition du format

Une fois encore, nous allons utiliser les propriétés de Pango (même si nous n'allons pas utiliser les fonctions de Pango directement).

Pour définir le format du texte, il suffit d'insérer des balises à l'intérieur même de notre texte. Pango s'occupe ensuite de rechercher les balises puis de formater le texte à notre convenance.

4.2 Les balises rapides

Les balises rapides servent à mettre le texte en gras, italique ou autre de manière très simple. Voici l'intégralité de ces balises :

- : Met le texte en gras
- <big> : Augmente légèrement la taille du texte
- <i> : Met le texte en italique
- <s> : Barre le texte
- <sub> : Met le texte en indice
- <sup> : Met le texte en exposant
- <small> : Diminue légèrement la taille du texte
- <tt> : Met le texte en télétype
- <u> : Souligne le texte

Pour utiliser ces balises, il suffit d'encadrer le texte à modifier des balises de formatage. Par exemple, pour mettre le texte en gras, il faudra entrer : "Normal vs Gras".
Mais cela ne suffit pas, il faut aussi dire que le texte utilise les balises Pango, ce qui est possible avec la fonction suivante :

```
void gtk_label_set_use_markup(GtkLabel *label, gboolean setting);
```

Il faut mettre le paramètre *setting* à TRUE, et le texte sera alors formaté en conséquence.
Un autre moyen de spécifier l'utilisation des balises, est d'utiliser cette fonction :

```
void gtk_label_set_markup(GtkLabel *label, const gchar *str);
```

Cette fonction dit à Pango qu'il faut utiliser les balises, mais elle modifie aussi le label en affichant la chaîne de caractères *str* (deuxième paramètre).

4.3 La balise

Cette fois-ci, nous allons étudier la balise en détail. Avec celle-ci, nous allons pouvoir changer la police de caractères, la couleur du texte et bien d'autres choses. Cette balise s'utilise comme les précédentes si ce n'est qu'elle accepte des paramètres. Le tableau ci-dessous présente tous les paramètres et les effets sur le texte.

- ***font_family*** : Pour définir la police à utiliser. Si la valeur donnée à ce paramètre est incorrecte, la police par défaut (Sans Serif) sera utilisée. Exemple : <span font_family=\"Courier New\">Courier New.
- ***face*** : Identique à font_family. Exemple : <span face=\"Times New

Roman\">Times New Roman.

- ***size*** : Pour définir la taille du texte (par défaut 10). Exemple : <span size=\"12\">Taille 12
- ***style*** : Pour définir le style du texte. Trois valeurs possibles : normal, oblique, italic. Exemple : <span style=\"oblique\">Oblique
- ***font_desc*** : Permet de combiner les paramètres précédents en un seul. Exemple : <span font_desc=\"Courier New italic 12\">Courier New italic 12.
- ***weight*** : Permet de définir l'épaisseur des lettres. Les valeurs peuvent être *ultralight, light, normal, bold, utrabold, heavy* ou encore une valeur numérique. (Vous remarquerez qu'il n'y a que peu ou pas de différence). Exemple : <span weight=\"bold\">Bold
- ***variant*** : Pour définir si l'on veut du texte normal (normal) ou en petite majuscule (smallcaps). Exemple : <span variant=\"smallcaps\"> Smallcaps . Afin de pourvoir l'utiliser, il faut avoir la police : nom smallcaps
- ***stretch*** : Permet d'étirer le texte. La valeur de ce paramètre peut être : *ultracondensed, extracondensed, condensed, semicondensed, normal, semiexpanded, expanded, extraexpanded, ultraexpanded*. Afin de pourvoir l'utiliser, il faut avoir la police : nom condensed (ou autre).
- ***foreground*** : Pour définir la couleur du texte. Il faut donner la valeur hexadécimale de la palette RVB. Exemple : <span foreground=\"#FF0000\"> Couleur Texte
- ***background*** : Pour définir la couleur du fond. Il faut donner la valeur hexadécimale de la palette RVB. Exemple : <span background=\"#FF0000\"> Couleur Texte
- ***underline*** : Pour souligner le texte. Exemple : <span underline=\"double\">Double.
- ***rise*** : Pour déplacer le texte verticalement.
- ***strikethrough*** : Pour barrer le texte. Exemple : <span strikethrough=\"true\">Striketrough = "true"
- ***lang*** : Pour indiquer la langue du texte.

Tous ces paramètres peuvent être mis à l'intérieur d'une seule et même balise .

4.4 Programme exemple

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

int main(int argc,char **argv)
{
    GtkWidget* pWindow;
    GtkWidget* pLabel;

    gtk_init(&argc,&argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow),"Les labels III");
    gtk_window_set_default_size(GTK_WINDOW(pWindow),320,200);

    /* Creation du label avec g_locale_to_utf8 */
    pLabel=gtk_label_new(NULL);
```

```
    /* On utilise les balises */
    gtk_label_set_markup(GTK_LABEL(pLabel), "<span face=\"Courier
New\"><b>Courier New 10 Gras</b></span>\n"
        "<span font_desc=\"Times New Roman italic 12\"
foreground=\"#0000FF\">Times New Roman 12 Italique</span>\n");

    /* On centre le texte */
    gtk_label_set_justify(GTK_LABEL(pLabel), GTK_JUSTIFY_CENTER);

     /* On ajoute le label a l'interieur de la fenetre */
     gtk_container_add(GTK_CONTAINER(pWindow),pLabel);

     /* Affichage de la fenetre et de tout ce qu'il contient */
    gtk_widget_show_all(pWindow);

    /* Connexion du signal */
    /* On appelle directement la fonction de sortie de boucle */
    g_signal_connect(G_OBJECT(pWindow),"destroy",G_CALLBACK(gtk_main_quit),0);

     gtk_main();

    return EXIT_SUCCESS;
}
```

Résultats

Les Boutons (Partie 1)

1. Présentation

Nous allons cette fois nous intéresser à un élément essentiel d'une interface graphique : le bouton. En effet celui-ci permet à l'utilisateur d'effectuer une action grâce à un simple click de souris. GTK+ nous permet de les utiliser grâce au widget GtkButton :

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkButton

2. Utilisation de base

2.1 Création d'un bouton

Pour créer un bouton, GTK+ permet l'utilisation de quatre fonctions différentes :

```
GtkWidget* gtk_button_new(void);
GtkWidget* gtk_button_new_with_label(const gchar *label);
GtkWidget* gtk_button_new_with_mnemonic(const gchar *label);
GtkWidget* gtk_button_new_from_stock(const gchar *stock_id);
```

La première fonction permet de créer un bouton vide. Cela permet de personnaliser complètement le bouton car GtkButton dérive de GtkContainer. On peut donc inclure n'importe quel type de widget dans le bouton (GtkLabel, GtkImage, ...).

La deuxième fonction s'occupe en plus d'insérer un label à l'intérieur du bouton. Le paramètre *label* correspond au texte à afficher. Comme pour le widget GtkLabel, si un caractère accentué est utilisé, il faudra appeler la fonction *g_locale_to_utf8* pour avoir un affichage correct du texte.

La troisième fonction ajoute à cela une nouvelle fonctionnalité. Elle permet, en plus d'afficher un *label*, de faire réagir le bouton à l'aide d'un raccourci clavier. La touche servant de raccourci est spécifiée dans le paramètre label. Il suffit de mettre "_" devant la lettre souhaitée pour que la combinaison Atl+Touche active le bouton.

La quatrième fonction permet de créer un bouton avec un label, un raccourci clavier et une image. Cependant, pour faire cela, GTK+ utilise les GtkStockItem? qui est une structure contenant les informations sur le label et l'image à afficher. GTK+ comporte déjà beaucoup de GtkStockItem? prédéfinis (en tout cas les plus courant). Le paramètre *stock_id* est donc l'identifiant du GtkStockItem? à utiliser.

Une fois le bouton créé, il ne reste plus qu'à l'inclure dans une fenêtre et de connecter le signal "clicked", pour qu'un clic de souris sur le bouton corresponde à une action précise.

2.2 Modifier le texte d'un bouton

A tout moment pendant l'exécution du programme, nous pouvons connaître ou modifier le texte d'un bouton à l'aide d'une de ces deux fonctions :

G_CONST_RETURN gchar gtk_button_get_label (GtkButton *button);*
*void gtk_button_set_label (GtkButton *button, const gchar *label);*

La première fonction renvoie le texte du bouton *button* (premier paramètre), tandis que la deuxième modifie le texte du bouton par la valeur du deuxième paramètre *label*.

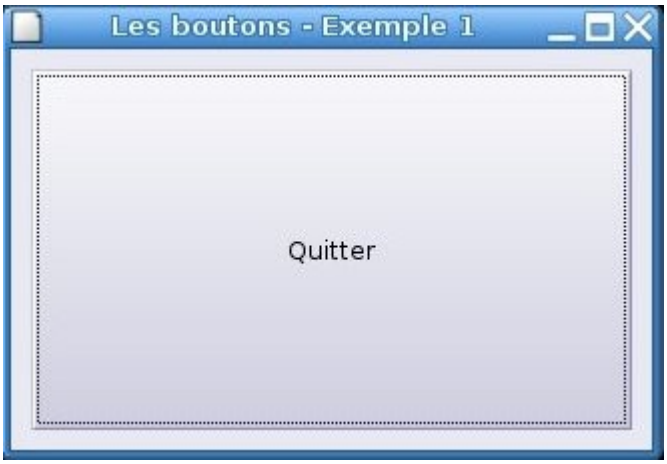
2.3 Programme exemple

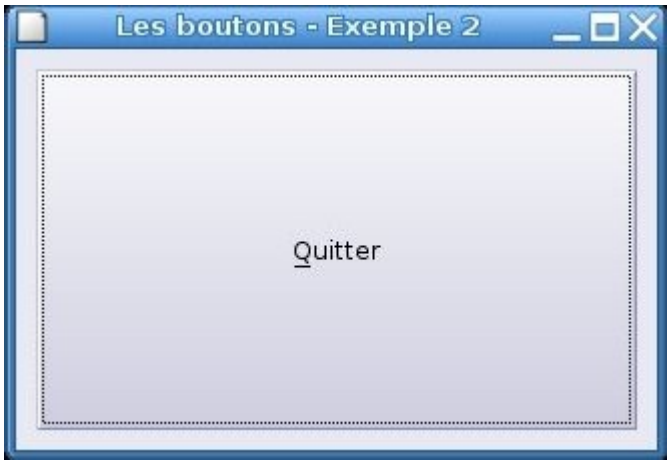
Pour ce chapitre, notre objectif sera de créer une application contenant un bouton qui permettra de quitter l'application.

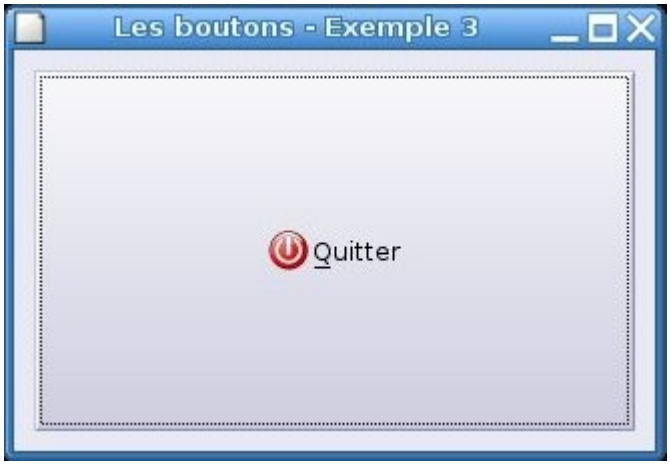
```
#include <stdlib.h>
#include <gtk/gtk.h>

#define EXEMPLE_1 0
#define EXEMPLE_2 1
#define EXEMPLE_3 2

void add_btn(GtkWidget *pWindow, gint iExemple);

int main(int argc, char **argv)
{
    GtkWidget* pWindow;

    gtk_init(&argc, &argv);

    /* Creation de la fenetre */
    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320 ,200);
    gtk_container_set_border_width(GTK_CONTAINER(pWindow), 10);

    /* Connexion du signal "destroy" de la fenetre */
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    /* Insertion du bouton */
    add_btn(pWindow, EXEMPLE_1);

    /* Affichage de la fenetre */
    gtk_widget_show_all(pWindow);

    /* Demarrage de la boucle evenementielle */
    gtk_main();

    return EXIT_SUCCESS;
}

/*
/* void add_btn(GtkWidget *pWindow, gint iExemple)
/*
/* Fonction en charge d'inserer le bouton dans la fenetre
/*
/* Parametre :
/* - pWindow : fenetre parente
/* - iExemple : mode de creation
/* EXEMPLE_1 pour un bouton label
/* EXEMPLE_2 pour un bouton EXEMPLE_1 + raccourci
/* EXEMPLE_3 pour un bouton EXEMPLE_2 + image
*/
```

```
void add_btn(GtkWidget *pWindow, gint iExemple)
{
    GtkWidget *pQuitBtn;

    switch(iExemple)
    {
    default:
    case EXEMPLE_1:
        /* Bouton avec un label */
        pQuitBtn = gtk_button_new_with_label("Quitter");
        gtk_window_set_title(GTK_WINDOW(pWindow), "Les boutons - Exemple 1");
        break;
    case EXEMPLE_2:
        /* Bouton avec un label et un raccourci */
        pQuitBtn = gtk_button_new_with_mnemonic("_Quitter");
        gtk_window_set_title(GTK_WINDOW(pWindow), "Les boutons - Exemple 2");
        break;
    case EXEMPLE_3:
        /* Bouton avec un label, un raccourci et une image */
        pQuitBtn = gtk_button_new_from_stock(GTK_STOCK_QUIT);
        gtk_window_set_title(GTK_WINDOW(pWindow), "Les boutons - Exemple 3");
        break;
    }

    /* Connexion du signal "clicked" du bouton */
    g_signal_connect(G_OBJECT(pQuitBtn), "clicked", G_CALLBACK(gtk_main_quit),
NULL);

    /* Insertion du bouton dans la fenetre */
    gtk_container_add(GTK_CONTAINER(pWindow), pQuitBtn);
}
```

Résultats




Les boutons - Exemple 2
Quitter


Les boutons - Exemple 3
Quitter

Les Box

1. Présentation

Vous avez sûrement dû essayer de mettre plusieurs widgets dans une fenêtre mais sans succès. Cela est dû au fait qu'un widget de type GtkContainer ne peut contenir qu'un seul widget. La solution à ce problème est l'utilisation des widgets de type GtkBox qui permettent d'inclure plusieurs widgets à l'intérieur.

Il existe deux catégories de GtkBox :

- les GtkHBox qui permettent de disposer les widgets horizontalement ;
- les GtkVBox pour les disposer verticalement.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBox -> GtkHBox

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBox -> GtkVBox

2. Utilisation de base

2.1 Création

Comme toujours, la création de ces widgets est très simple. Les fonctions suivantes permettent de créer respectivement une GtkHBox et une GtkVBox :

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

Le paramètre *homogeneous* définit si tous les widgets contenus dans la GtkBox utilisent un espace équivalent. C'est à dire que si ce paramètre est à TRUE, la zone d'affichage de la GtkBox sera divisée en x zone(s) de taille égale (x étant le nombre de widgets contenus).

Le paramètre *spacing* permet de définir l'espacement entre chacun des widgets contenus.

2.2 Ajout de widget

Avant de pouvoir ajouter un widget dans une GtkHBox ou GtkVBox, il faut savoir que pour une GtkBox il existe la notion de premier et de dernier widget. Nous allons donc pouvoir ajouter des widgets soit par le début soit par la fin avec une de ces deux fonctions :

```
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);
```

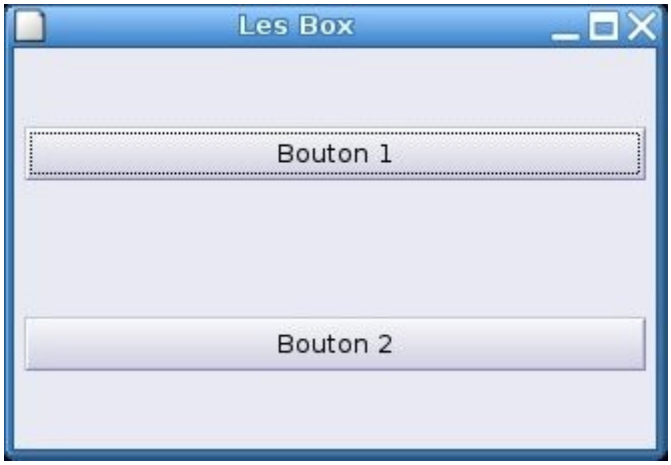
La première fonction ajoute les widgets de haut en bas et la deuxième de bas en haut. Pour les paramètres, ils sont les mêmes pour les deux fonctions. Le premier *box* est en

fait le widget conteneur dans lequel nous allons insérer le widget *child* qui vient en deuxième paramètre.

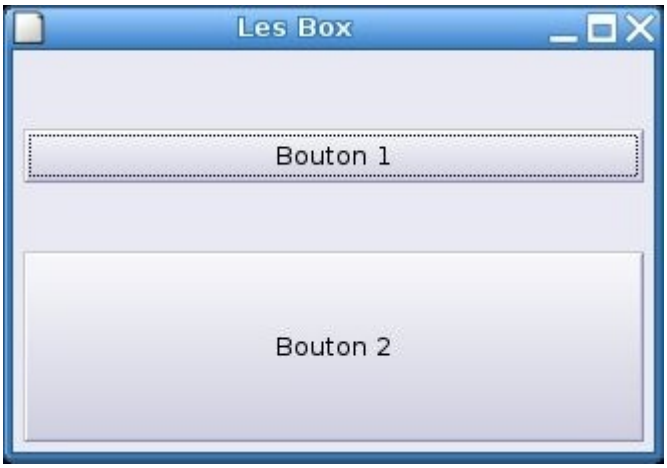
Le troisième paramètre *expand* nécessite un petite explication. Nous ne le voyons pas ici, mais lors de la création d'une GtkHBox ou GtkVBox, il faut définir une propriété appelée *homogeneous*. Cette propriété détermine si les widgets qui sont à l'intérieur de la GtkBox vont se partager l'intégralité de l'espace fourni, ou alors réserver uniquement la place qui leur est nécessaire.

Petit exemple, nous avons un espace de 3 pixel dans lequel nous voulons insérer deux widgets de 1 pixel. Si le paramètre *homogeneous* est à TRUE, chaque widget réservera un espace de 1.5 pixel. Au contraire si il est à FALSE, chaque widget réservera un espace de 1 pixel, laissant donc un espace libre de 1 pixel à la fin. Revenons donc au paramètre *expand*. Si la GtkBox est homogène, la valeur de ce paramètre n'a aucune importance. Par contre si la GtkBox n'est pas homogène, le fait de mettre *expand* à TRUE à un widget va lui permettre de partager l'espace restant dans la GtkBox avec les autres widgets qui auront été insérés avec *expand* à TRUE. Reprenons notre exemple. La GtkBox n'est pas homogène, donc normalement chaque widget va réserver 1 pixel de l'espace. Si nous mettons le premier widget avec *expand* à TRUE et le deuxième avec *expand* à FALSE, le deuxième widget réservera un espace de 1 pixel, le premier aussi mais il doit se partager l'espace libre (1 pixel) avec les autres widgets qui ont aussi la propriété *expand* à TRUE. Comme il est tout seul avec cette propriété, il va donc réserver 2 pixel.

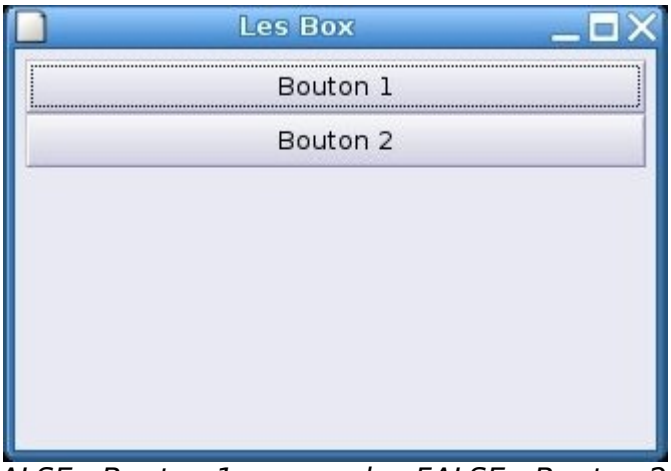
Le paramètre *fill* permet de définir si le widget enfant occupe toute la zone qui lui est réservée.
Et enfin, le paramètre *padding* permet d'ajouter de l'espace autour du widget. Voici quelques exemples avec différentes valeurs pour ces paramètres :



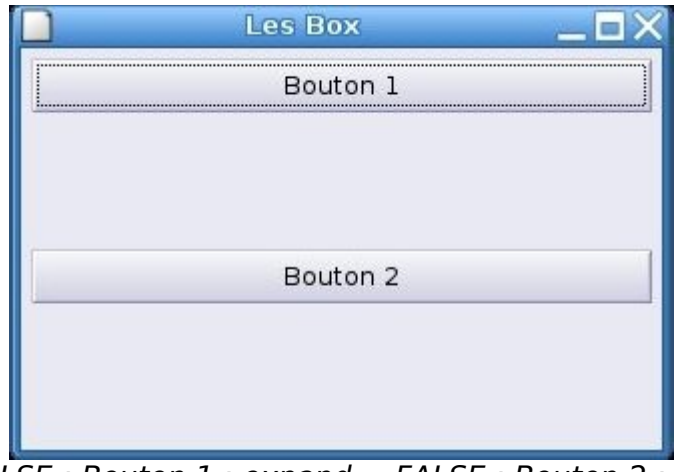
homogeneous = TRUE ; Bouton 1 : fill = FALSE ; Bouton 2 : fill = FALSE



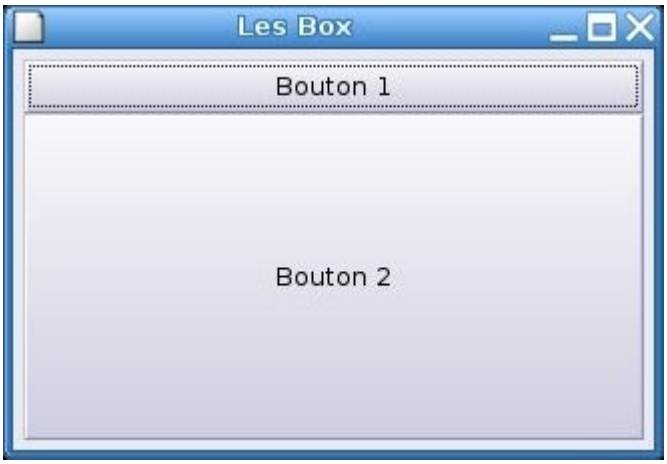
homogeneous = TRUE ; Bouton 1 : fill = FALSE ; Bouton 2 : fill = TRUE



homogeneous = FALSE ; Bouton 1 : expand = FALSE ; Bouton 2 : expand = FALSE



homogeneous = FALSE ; Bouton 1 : expand = FALSE ; Bouton 2 : expand = TRUE - fill = FALSE



homogeneous = FALSE ; Bouton 1 : expand = FALSE ; Bouton 2 : expand = TRUE - fill = TRUE

2.3 Programme exemple

Pour ce chapitre, nous allons insérer quatre boutons dans une GtkHBox et une GtkVBox afin d'étudier les fonctions d'ajout du widget GtkBox.

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

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pHBox;
    GtkWidget *pButton[4];

    gtk_init(&argc,&argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "Les GtkBox");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    /* Creation de la GtkBox verticale */
    pVBox = gtk_vbox_new(TRUE, 0);
    /* Ajout de la GtkVBox dans la fenetre */
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    /* Creation des boutons */
    pButton[0] = gtk_button_new_with_label("Bouton 1");
    pButton[1] = gtk_button_new_with_label("Bouton 2");
    pButton[2] = gtk_button_new_with_label("Bouton 3");
    pButton[3] = gtk_button_new_with_label("Bouton 4");

    /* Ajout de Bouton 1 dans la GtkVBox */
    gtk_box_pack_start(GTK_BOX(pVBox), pButton[0], TRUE, FALSE, 0);

    /* Creation de la box horizontale */
    pHBox = gtk_hbox_new(TRUE, 0);

    /* Ajout de la GtkHBox dans la GtkVBox */
```

```
    gtk_box_pack_start(GTK_BOX(pVBox), pHBox, TRUE, TRUE, 0);

    /* Ajout des boutons 2 et 3 dans la GtkHBox */
    gtk_box_pack_start(GTK_BOX(pHBox), pButton[1], TRUE, TRUE, 0);
    gtk_box_pack_start(GTK_BOX(pHBox), pButton[2], TRUE, FALSE, 0);

    /* Ajout du dernier bouton dans la GtkVBox */
    gtk_box_pack_start(GTK_BOX(pVBox), pButton[3], TRUE, TRUE, 0);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}
```

Résultat

Les Tables

1. Présentation

Dans la famille des containers, voilà sûrement le widget le plus intéressant. En effet il peut être parfois douloureux de placer correctement son interface avec l'utilisation de plusieurs GtkBox. Le widget GtkTable est conçu pour résoudre ce problème car il utilise une grille invisible pour attacher les widgets mais sans pour autant perdre la puissance de GTK+ avec le redimensionnement automatique.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkTable

2. Utilisation de base

Au moment de la création de la GtkTable, nous allons spécifier le nombre de lignes et de colonnes, puis y placer les éléments avec 3 principales caractéristiques :

- la position de départ et de fin de l'élément par rapport aux lignes ;
- la position de départ et fin de l'élément par rapport aux colonnes ;
- la façon de réagir du widget (remplir la zone, agrandir, etc...).

2.1 Création d'une GtkTable

La fonction de création est :

```
GtkWidget* gtk_table_new(guint rows, guint columns, gboolean homogeneous);
```

Les paramètres *rows* et *columns* permettent de définir respectivement le nombre de lignes et de colonnes de la grille. Le paramètre *homogeneous* quant à lui définit, comme pour une GtkBox, si tous les widgets contenus dans la GtkTable utilisent un espace équivalent.

2.2 Insertion d'éléments

La première fonction étudiée est :

```
void gtk_table_attach( GtkTable *table, GtkWidget *child, guint left_attach, guint right_attach,
guint top_attach, guint bottom_attach, GtkAttachOptions xoptions, GtkAttachOptions yoptions,
guint xpadding, guint ypadding);
```

A première vue, cette fonction peut apparaître compliquée, mais elle est en réalité très simple. Le paramètre *child* représente le widget à attacher à la grille, les paramètres *left_attach* et *right_attach*, les positions à gauche et à droite du widget et les paramètres *top_attach* et *bottom_attach*, les positions supérieures et inférieures du widget.

Les paramètres *xoptions* et *yoptions* permettent de spécifier respectivement la façon dont le widget réagit horizontalement et verticalement au redimensionnement de la GtkTable. Ces paramètres peuvent prendre 3 valeurs (que l'on peut associer) :

- GTK_EXPAND : spécifie que cette section de la grille s'étirera pour remplir l'espace disponible ;
- GTK_SHRINK : détermine ce qui se produira s'il y a un espace insuffisant pour répondre à la requête de taille du widget enfant, alors le widget se voit attribué une allocation réduite, ce qui peut entraîner un effet de "bords coupés" ;
- GTK_FILL : spécifie que le widget enfant s'étirera pour remplir l'espace disponible, important que si GTK_EXPAND est défini.

Les deux derniers paramètres *xpadding* et *ypadding* définissent l'espace supplémentaire à ajouter aux bords du widget (à droite et à gauche pour le premier, au-dessus et en dessous pour le second).

La deuxième fonction est :

```
void gtk_table_attach_defaults(GtkTable *table, GtkWidget *child, guint left_attach, guint
right_attach, guint top_attach, guint bottom_attach );
```

Ceci est la version simplifiée de la première fonction car elle définit automatiquement les paramètres *xoptions* et *yoptions* à GTK_EXPAND | GTK_FILL et les paramètres *xpadding* et *ypadding* à 0.

2.3 Modification de la table

Il est possible de changer la taille de la grille après sa création à l'aide de cette fonction :

```
void gtk_table_resize(GtkTable *table, guint rows, guint columns);
```

Le paramètre *table* est la GtkTable à modifier, et les paramètres *rows* et *columns* les nouveaux nombres de lignes et de colonnes.

Ces deux fonctions permettent de changer l'espace d'une ligne ou d'une colonne spécifique :

```
gtk_table_set_row_spacing(GtkTable *table, guint row, guint spacing);
gtk_table_set_col_spacing(GtkTable *table, guint column, guint spacing);
```

La première définit l'espace autour d'une ligne tandis que la deuxième fait la même chose pour une colonne.

Celles-ci ont le même rôle que les deux précédentes fonctions, mais agissent sur l'ensemble de la GtkTable :

```
gtk_table_set_row_spacings(GtkTable *table, guint spacing);
gtk_table_set_col_spacings(GtkTable *table, guint spacing);
```

Et pour connaître ces espacements nous avons quatre fonctions différentes :

```
guint gtk_table_get_row_spacing(GtkTable *table, guint row);
guint gtk_table_get_col_spacing(GtkTable *table, guint column);
guint gtk_table_get_default_row_spacing(GtkTable *table);
guint gtk_table_get_default_col_spacing(GtkTable *table);
```

Les deux premières fonctions permettent de connaître l'espace entre ligne numéro *row* (ou la colonne numéro *column*) et sa suivante. Les deux autres fonctions quant à elles, renvoient la valeur par défaut des espacements, c'est à dire la valeur qui sera

utilisé au prochain ajout d'un widget.

2.4 Programme exemple

Nous allons utiliser le même exemple que dans le chapitre sur les GtkBox, pour bien montrer la différence au niveau du code.

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

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pTable;
    GtkWidget *pButton[4];

    gtk_init(&argc, &argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    gtk_window_set_title(GTK_WINDOW(pWindow), "Les GtkTable");
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    /* Creation et insertion de la table 3 lignes 2 colonnes */
    pTable=gtk_table_new(3,2,TRUE);
    gtk_container_add(GTK_CONTAINER(pWindow), GTK_WIDGET(pTable));

    /* Creation des boutons */
    pButton[0]= gtk_button_new_with_label("Bouton 1");
    pButton[1]= gtk_button_new_with_label("Bouton 2");
    pButton[2]= gtk_button_new_with_label("Bouton 3");
    pButton[3]= gtk_button_new_with_label("Bouton 4");

    /* Insertion des boutons */
    gtk_table_attach(GTK_TABLE(pTable), pButton[0],
        0, 2, 0, 1,
        GTK_EXPAND | GTK_FILL, GTK_EXPAND,
        0, 0);
    gtk_table_attach_defaults(GTK_TABLE(pTable), pButton[1],
        0, 1, 1, 2);
    gtk_table_attach(GTK_TABLE(pTable), pButton[2],
        1, 2, 1, 2,
        GTK_EXPAND, GTK_EXPAND | GTK_FILL,
        0, 0);
    gtk_table_attach_defaults(GTK_TABLE(pTable), pButton[3],
        0, 2, 2, 3);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}
```

Les listes chaînées

1. Présentation

Vous connaissez sûrement déjà les listes chaînées. Elles sont très pratiques, mais il faut s'occuper soit même de la gestion de la mémoire, de chaîner les éléments, à chaque fois que l'on veut ajouter ou supprimer un élément. La bibliothèque GLib? propose des listes chaînées génériques que nous allons pouvoir utiliser dans nos applications Gtk+.

2. La liste chaînée simple : GSList

2.1 Structure

Regardons tout d'abord comment est défini cet objet :

```
struct GSList
{

gpointer data;
struct GSList *next;

};
```

Nous voyons donc qu'elle nous permet de créer la plus simple des liste chaînée, c'est à dire que chaque élément connaît son suivant et rien d'autre.

2.2 Création d'une GSList

Il faut d'abord avoir le pointeur sur le premier élément de la liste. Pour cela, rien de plus simple :

```
GSList *premier = NULL;
```

Ensuite, il n'existe pas de fonction spécifique pour créer la liste, il suffit juste d'ajouter un élément à la liste.

2.3 Ajout d'éléments

Plusieurs fonctions sont disponibles mais nous allons en étudier deux.

```
GSList* g_slist_append (GSList *list, gpointer data);
GSList* g_slist_prepend (GSList *list, gpointer data);
```

La première g_slist_append ajoute un nouvel élément à la fin de la liste, alors que g_slist_prepend l'ajoute au début de la liste. La variable *list* est bien sûr la liste chaînée à laquelle on veut ajouter un élément comportant la donnée *data*. Il suffit de faire un cast pour donner le type de *data*.

La valeur de retour est très importante : elle correspond à l'adresse du premier élément de la liste. En effet, au départ notre élément premier ne pointe nul part

(premier = NULL), il faut donc toujours récupérer cette adresse (surtout dans le cas de g_slist_prepend où le premier élément change).

2.4 Récupérer les données d'une GSList

Là aussi, nous n'allons étudier que deux fonctions :

```
GSList* g_slist_nth (GSList *list, guint n);
gpointer g_slist_nth_data (GSList *list, guint n);
```

Le paramètre *list* est bien sur la liste à laquelle l'élément recherché appartient, et *n* est la position de l'élément dans la liste (le premier élément est à n=0). Avec ces deux fonctions, il suffit juste de connaître la position d'un élément pour récupérer la donnée qu'il contient.

La première fonction renvoie un pointeur sur une variable du type GSList. Il faudra donc utiliser l'opérateur -> pour récupérer la valeur *data*. La deuxième, par contre, renvoie directement la valeur de *data*.

Si la valeur donnée à *n* ne fait pas partie de la liste, la valeur de retour sera NULL. Supposons qu'une variable de type GtkWidget soit stockée dans le premier élément d'une liste nommé liste, et que l'on veuille récupérer ce widget, il faudra alors coder :

```
temp_list = g_slist_nth(liste, 0);
widget = GTK_WIDGET(temp_list->data);
```

ou

```
widget = GTK_WIDGET(g_slist_nth_data(liste, 0));
```

2.5 Suppression d'élément d'une GSList

Pour supprimer un élément définitivement d'une liste, la fonction à utiliser est la suivante :

```
GSList* g_slist_remove (GSList *list, gconstpointer data);
```

Cette fonction cherche le premier élément de la liste contenant la donnée *data* et le supprime. La valeur de retour est là aussi très importante car elle correspond (toujours pareil) au nouveau premier élément de la liste. Si par hasard, plusieurs éléments contiennent la donnée *data*, seul le premier sera supprimé. Dans ce cas, pour tous les supprimer, il faut utiliser cette fonction (dont l'utilisation est identique à la première) :

```
GSList* g_slist_remove_all (GSList *list, gconstpointer data);
```

Pour supprimer une liste entière, la fonction est :

```
void g_slist_free (GSList *list);
```

Avec toutes ces fonctions, nous en savons suffisamment pour pouvoir utiliser une liste simple.

3. La liste doublement chaînée : GList

En plus des GSList, la bibliothèque GLib? propose un autre type de liste chaînée qui est la GList. A la différence des GSList, les GList sont des listes doublement chaînées. En effet en plus de connaître son élément suivant, chaque élément connaît aussi son précédent.

3.1 Structure

Regardons tout d'abord comment est défini cet objet :

```
struct GList
{

gpointer data;
struct GList *next;
struct GList *prev;

};
```

3.2 Création d'une GSList

Il faut d'abord avoir le pointeur sur le premier élément de la liste. Pour cela, rien de plus simple :

```
GList *premier = NULL;
```

Ensuite, il n'existe pas de fonction spécifique pour créer la liste, il suffit juste d'ajouter un élément à la liste.

3.2 Ajout d'éléments

Plusieurs fonctions sont disponibles mais nous allons en étudier deux.

```
GList* g_slist_append (GList *list, gpointer data);
GList* g_slist_prepend (GList *list, gpointer data);
```

La première g_list_append ajoute un nouvel élément à la fin de la liste, alors que g_list_prepend l'ajoute au début de la liste. La variable *list* est bien sûr la liste chaînée à laquelle on veut ajouter un élément comportant la donnée *data*. Il suffit de faire un cast pour donner le type de *data*.

La valeur de retour est très importante : elle correspond à l'adresse du premier élément de la liste. En effet, au départ notre élément premier ne pointe nul part (premier = NULL), il faut donc toujours récupérer cette adresse (surtout dans le cas de g_list_prepend où le premier élément change).

3.4 Récupérer les données d'une GList

Là aussi, nous n'allons étudier que deux fonctions :

GList* g_list_nth (GList *list, guint n);
gpointer g_list_nth_data (GList *list, guint n);

Le paramètre *list* est bien sur la liste à laquelle l'élément recherché appartient, et *n* est la position de l'élément dans la liste (le premier élément est à n=0). Avec ces deux fonctions, il suffit juste de connaître la position d'un élément pour récupérer la donnée qu'il contient.

La première fonction renvoie un pointeur sur une variable du type GList. Il faudra donc utiliser l'opérateur -> pour récupérer la valeur *data*. La deuxième, par contre, renvoie directement la valeur de *data*.

Si la valeur donnée à *n* ne fait pas partie de la liste, la valeur de retour sera NULL. Supposons qu'une variable de type GtkWidget soit stockée dans le premier élément d'une liste nommé liste, et que l'on veuille récupérer ce widget, il faudra alors coder :

```
temp_list = g_list_nth(liste, 0);
widget = GTK_WIDGET(temp_list->data);
```

ou

```
widget = GTK_WIDGET(g_list_nth_data(liste, 0));
```

3.5 Suppression d'élément d'une GList

Pour supprimer un élément définitivement d'une liste, la fonction à utiliser est la suivante :

```
GList* g_list_remove (GList *list, gconstpointer data);
```

Cette fonction cherche le premier élément de la liste contenant la donnée *data* et le supprime. La valeur de retour est là aussi très importante car elle correspond (toujours pareil) au nouveau premier élément de la liste. Si par hasard, plusieurs éléments contiennent la donnée *data*, seul le premier sera supprimé. Dans ce cas, pour tous les supprimer, il faut utiliser cette fonction (dont l'utilisation est identique à la première) :

```
GList* g_list_remove_all (GList *list, gconstpointer data);
```

Pour supprimer une liste entière, la fonction est :

```
void g_list_free (GList *list);
```

Remarque :

Les fonctions g_list_remove() et g_list_remove_all() libèrent la mémoire des données transmises. A contrario, la fonction g_list_free() ne fait que libérer l'espace mémoire utilisée par la liste chaînée. Il faut alors explicitement libérer la mémoire allouée pour les données avant g_list_free(). Une manière simple est élégante de le faire pourrait être :

```
g_list_foreach(GList *list, (GFunc)g_free, NULL);
g_list_free(GList *list);
```

Avec toutes ces fonctions, nous en savons suffisamment pour pouvoir utiliser une liste doublement chaînée.

La saisie de données

1. Présentation

Nous allons maintenant voir comment permettre à l'utilisateur final de saisir des données, qu'il s'agisse de texte ou de valeur numérique. Nous allons utiliser pour cela le widget GtkEntry qui défini une zone de texte (une ligne) dans lequel l'utilisateur peut taper du texte ou alors dans lequel le programme peut afficher une information.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkEntry

2. Utilisation de base

2.1 Création d'une GtkEntry

Pour créer une GtkEntry, nous avons à notre disposition fonction simple d'utilisation :

```
GtkWidget* gtk_entry_new(void);
```

2.2 Récupération de la donnée saisie

Afin de récupérer le texte qui a été saisi par l'utilisateur, il faut utiliser cette fonction :

```
G_CONST_RETURN gchar* gtk_entry_get_text(GtkEntry *entry);
```

Elle permet de récupérer le texte qui a été tapé par l'utilisateur dans le GtkEntry entry. La valeur de retour est de type G_CONST_RETURN gchar*, c'est à dire qu'il faudra récupérer le texte dans une variable de type const gchar*. De plus, il est inutile d'allouer de la mémoire pour la variable qui va recevoir le texte, et donc de ne surtout pas libérer la mémoire car cela rendrait votre application instable.

2.3 Afficher un message dans la zone de texte

Pour afficher un message dans une GtkEntry, la fonction est :

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

Le paramètre *entry* correspond bien sûr au GtkEntry dans lequel on veut insérer le texte *text*.

2.4 Limiter le nombre de caractères

Gtk+ nous offre la possibilité de limiter le nombre de caractères qu'un utilisateur peut saisir avec la fonction :

*void gtk_entry_set_max_length(GtkEntry *entry, gint max);*

Le paramètre *max* correspond bien sûr à limite que nous voulons fixer.

2.5 Programme exemple

Comme exemple, nous allons créer une fenêtre comportant un GtkEntry, un GtkButton et un GtkLabel. Le but sera d'afficher le texte du GtkEntry dans le GtkLabel. Cette opération s'effectuera lorsque l'utilisateur appuie sur la touche ENTREE à la fin de sa saisie (interception du signal "activate") ou lorsqu'il cliquera sur le GtkButton (interception du signal "clicked").

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

struct _MainWindow
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pEntry;
    GtkWidget *pButton;
    GtkWidget *pLabel;
};

typedef struct _MainWindow MainWindow;

void OnUpdate(GtkWidget *pEntry, gpointer data);

int main(int argc, char **argv)
{
    MainWindow *pApp;

    gtk_init(&argc, &argv);

    pApp = g_malloc(sizeof(MainWindow));

    pApp->pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pApp->pWindow), "Le widget GtkEntry");
    gtk_window_set_default_size(GTK_WINDOW(pApp->pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pApp->pWindow), "destroy",
G_CALLBACK(gtk_main_quit), NULL);

    pApp->pVBox = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pApp->pWindow), pApp->pVBox);

    /* Creation du GtkEntry */
    pApp->pEntry = gtk_entry_new();
    /* Insertion du GtkEntry dans la GtkVBox */
    gtk_box_pack_start(GTK_BOX(pApp->pVBox), pApp->pEntry, TRUE, FALSE, 0);

    pApp->pButton = gtk_button_new_with_label("Copier");
    gtk_box_pack_start(GTK_BOX(pApp->pVBox), pApp->pButton, TRUE, FALSE, 0);

    pApp->pLabel = gtk_label_new(NULL);
    gtk_box_pack_start(GTK_BOX(pApp->pVBox), pApp->pLabel, TRUE, FALSE, 0);

    /* Connexion du signal "activate" du GtkEntry */
    g_signal_connect(G_OBJECT(pApp->pEntry), "activate", G_CALLBACK(OnUpdate),
(gpointer) pApp);
```

```
    /* Connexion du signal "clicked" du GtkButton */
    /* La donnee supplementaire est la GtkVBox pVBox */
    g_signal_connect(G_OBJECT(pApp->pButton), "clicked", G_CALLBACK(OnUpdate),
(gpointer*) pApp);

    gtk_widget_show_all(pApp->pWindow);

    gtk_main();

    g_free(pApp);

    return EXIT_SUCCESS;
}

/* Fonction callback execute lors du signal "activate" */
void OnUpdate(GtkWidget *pEntry, gpointer data)
{
    const gchar *sText;
    MainWindow *pApp;

    /* Recuperation de data */
    pApp = (MainWindow*) data;

    /* Recuperation du texte contenu dans le GtkEntry */
    sText = gtk_entry_get_text(GTK_ENTRY(pApp->pEntry));

    /* Modification du texte contenu dans le GtkLabel */
    gtk_label_set_text(GTK_LABEL(pApp->pLabel), sText);
}
```

Résultats :

3. Utiliser la fonction mot de passe

3.1 Visibilité du texte

Généralement, lorsque nous tapons un mot de passe, nous souhaitons que celui-ci reste secret. Le widget GtkEntry permet cela grâce à cette fonction :

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

Il suffit donc de mettre le paramètre *visible* à FALSE pour cacher le texte qui sera entré.
A l'inverse, pour savoir si le texte entré sera visible ou pas, il faut utiliser cette fonction :

```
gboolean gtk_entry_get_visibility(GtkEntry *entry);
```

La valeur de retour sera, bien sûr, égale à TRUE si le texte est visible et à FALSE dans le cas contraire.

3.2 Le caractère affiché

Par défaut, lorsque l'on ajoute du texte à un GtkEntry qui a son paramètre visible à FALSE, GTK+ remplacera toutes les lettres par des '*'. Pour modifier celui-ci, il faut utiliser cette fonction :

*void gtk_entry_set_invisible_char(GtkEntry *entry, gunichar ch);*

Le paramètre *ch* correspond au caractère de remplacement que nous souhaitons. Celui-ci est de type gunichar qui correspond à l'encodage UCS-4. Bien que l'affichage de GTK+ se fasse avec l'encodage UTF-8, cela ne pose aucun problème car cette fois-ci, la conversion est faite automatiquement.

Et pour terminer, la fonction permettant de connaître le caractère de remplacement est :

*gunichar gtk_entry_get_invisble_char(GtkEntry *entry);*

3.3 Programme exemple

Cette fois, nous allons reprendre l'exemple précédent en activant le mode "mot de passe" et en limitant la saisie à huit caractères. Nous modifierons aussi le caractère de remplacement '*' par '$'.

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

struct _MainWindow
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pEntry;
    GtkWidget *pButton;
    GtkWidget *pLabel;
};

typedef struct _MainWindow MainWindow;

void OnUpdate(GtkWidget *pEntry, gpointer data);

int main(int argc, char **argv)
{
    MainWindow *pApp;

    gtk_init(&argc, &argv);

    pApp = g_malloc(sizeof(MainWindow));

    pApp->pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pApp->pWindow), "Le widget GtkEntry");
    gtk_window_set_default_size(GTK_WINDOW(pApp->pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pApp->pWindow), "destroy",
G_CALLBACK(gtk_main_quit), NULL);

    pApp->pVBox = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pApp->pWindow), pApp->pVBox);

    /* Creation du GtkEntry */
    pApp->pEntry = gtk_entry_new();
    /* Limitation du nombre de caracteres */
    gtk_entry_set_max_length(GTK_ENTRY(pApp->pEntry), 8);
    /* Mode mot de passe */
    gtk_entry_set_visibility(GTK_ENTRY(pApp->pEntry), FALSE);
    /* Modification du caractere affiche */
    gtk_entry_set_invisible_char(GTK_ENTRY(pApp->pEntry), '$');
    /* Insertion du GtkEntry dans la GtkVBox */
```

```
    gtk_box_pack_start(GTK_BOX(pApp->pVBox), pApp->pEntry, TRUE, FALSE, 0);

    pApp->pButton = gtk_button_new_with_label("Copier");
    gtk_box_pack_start(GTK_BOX(pApp->pVBox), pApp->pButton, TRUE, FALSE, 0);

    pApp->pLabel = gtk_label_new(NULL);
    gtk_box_pack_start(GTK_BOX(pApp->pVBox), pApp->pLabel, TRUE, FALSE, 0);

    /* Connexion du signal "activate" du GtkEntry */
    g_signal_connect(G_OBJECT(pApp->pEntry), "activate", G_CALLBACK(OnUpdate),
(gpointer) pApp);

    /* Connexion du signal "clicked" du GtkButton */
    /* La donnee supplementaire est la GtkVBox pVBox */
    g_signal_connect(G_OBJECT(pApp->pButton), "clicked", G_CALLBACK(OnUpdate),
(gpointer*) pApp);

    gtk_widget_show_all(pApp->pWindow);

    gtk_main();

    g_free(pApp);

    return EXIT_SUCCESS;
}

/* Fonction callback execute lors du signal "activate" */
void OnUpdate(GtkWidget *pEntry, gpointer data)
{
    const gchar *sText;
    MainWindow *pApp;

    /* Recuperation de data */
    pApp = (MainWindow*) data;

    /* Recuperation du texte contenu dans le GtkEntry */
    sText = gtk_entry_get_text(GTK_ENTRY(pApp->pEntry));

    /* Modification du texte contenu dans le GtkLabel */
    gtk_label_set_text(GTK_LABEL(pApp->pLabel), sText);
}
```

Résultat

Les décorations

1. Présentation

Maintenant, nous connaissons suffisamment de widgets pour créer des fenêtres complexes. Afin d'améliorer l'esthétique des ces fenêtres nous allons voir comment ajouter des décorations (ou séparation) entre les différentes parties de la fenêtre. Il existe deux types de décoration :

- le cadre GtkFrame qui entoure toute une zone de la fenêtre et qui possède un texte permettant de définir la zone ;
- la ligne GtkSeparator? qui divise en deux parties différentes, le widget GtkHSeparator pour les lignes horizontales et le widget GtkVSeparator pour les lignes verticales.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkFrame

GObject -> GtkObject -> GtkWidget -> GtkSeparator? -> GtkHSeparator

GObject -> GtkObject -> GtkWidget -> GtkSeparator? -> GtkVSeparator

2. Le cadre

2.1 Création

Le widget GtkFrame étant très simple, il n'y a qu'une seule fonction de création :

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

Le paramètre *label* est tout simplement le texte qui sera affiché en haut à gauche du cadre (position par défaut).

2.2 Modification du texte

Il peut arriver que dans votre application, le texte du cadre nécessite une modification. Bien entendu, le widget GtkFrame est fourni avec toutes les fonctions nécessaires.

```
void gtk_frame_set_label(GtkFrame *frame, const gchar *label);
```

Le paramètre *frame* est le widget GtkFrame dont nous voulons modifier le texte, et *label* est le nouveau texte à inscrire. Cette fois encore, il faut utiliser une macro de conversion pour le premier paramètre qui est cette fois GTK_FRAME(). Pour récupérer le texte du GtkFrame, la fonction est :

```
G_CONST_RETURN gchar* gtk_frame_get_label(GtkFrame *frame);
```

2.3 Remplacer le texte par un widget

Les cadres offrent aussi la possibilité de remplacer le texte par un widget quelconque (GtkImage, GtkStockItem?, ...) grâce à cette fonction :

```
gtk_frame_set_label_widget(GtkFrame *frame, GtkWidget *label_widget);
```

Et comme toujours, la fonction permettant de connaître le widget affiché:

```
GtkWidget* gtk_frame_get_label_widget(GtkFrame *frame);
```

2.4 Position du texte

Par défaut, la position du texte est en haut à gauche du cadre, centré en hauteur par rapport à la ligne supérieure. Cela aussi peut être modifié avec cette fonction :

```
void gtk_frame_set_label_align(GtkFrame *frame, gfloat xalign, gfloat yalign);
```

Les valeurs *xalign* et *yalign* doivent être comprises entre 0.0 et 1.0.

Le paramètre *xalign* définit la position horizontale du texte. Une valeur de 0.0 positionne le texte à gauche du cadre, tandis qu'une valeur de 1.0 le positionne à droite. Evidemment, une valeur de 0.5 centrera le texte.

Quant à *yalign*, il permet de définir la position verticale du texte par rapport à la ligne supérieur du cadre. Une valeur de 0.0 mettra le nommeur en dessous de la ligne et une valeur de 1.0 le mettra au-dessus de la ligne.

On peut, bien sûr, connaître les valeurs de positionnement avec les fonctions suivantes :

```
void gtk_frame_get_label_align(GtkFrame *frame, gfloat *xalign, gfloat *yalign);
```

2.5 Style du cadre

Le style du cadre correspond plus à la configuration visuelle des lignes du cadre et plus précisément encore l'ombre des lignes. Cette modification de fait avec cette fonction :

```
void gtk_frame_set_shadow_type(GtkFrame *frame, GtkShadowType type);
```

Le paramètre *type* peut prendre cinq valeurs différentes dont voici la liste avec leurs illustrations :


Type de GtkFrame
GTK_SHADOW_IN


Type de GtkFrame
GTK_SHADOW_OUT


Type de GtkFrame
GTK_SHADOW_ETCHED_IN

Et sans surprise, voici la fonction qui permet de connaître le type des lignes :

```
GtkShadowType gtk_frame_get_shadow_type(GtkFrame *frame);
```

3. Les lignes

Alors cette fois, cela va être très rapide et très simple.

3.1 Création

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

La première fonction crée une ligne horizontale alors que la deuxième crée une ligne verticale.
Il n'y a rien de plus à propos de ce widget.

4. Exemple

4.1 Description

Afin de vous montrer l'avantage visuel de l'utilisation des GtkFrame et GtkSeparator?, nous allons créer une fenêtre qui demande à l'utilisateur de saisir des informations le concernant (nom, prénom, adresse, ...).
Voila à quoi ressemble la fenêtre sans l'utilisation des décorations :

Attention, ce programme ne fait rien du tout, c'est simplement pour montrer la différence entre avec et sans les décorations.

4.2 Programme exemple

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

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pFrame;
    GtkWidget *pVBoxFrame;
    GtkWidget *pSeparator;
    GtkWidget *pEntry;
    GtkWidget *pLabel;
    gchar *sUtf8;

    gtk_init(&argc, &argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

```
    /* On ajoute un espace de 5 sur les bords de la fenetre */
    gtk_container_set_border_width(GTK_CONTAINER(pWindow), 5);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkEntry et GtkSeparator");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    pVBox = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    /* Creation du premier GtkFrame */
    pFrame = gtk_frame_new("Etat civil");
    gtk_box_pack_start(GTK_BOX(pVBox), pFrame, TRUE, FALSE, 0);

    /* Creation et insertion d une boite pour le premier GtkFrame */
    pVBoxFrame = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pFrame), pVBoxFrame);

    /* Creation et insertion des elements contenus dans le premier GtkFrame */
    pLabel = gtk_label_new("Nom :");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    sUtf8 = g_locale_to_utf8("Prénom :", -1, NULL, NULL, NULL);
    pLabel = gtk_label_new(sUtf8);
    g_free(sUtf8);
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    /* Creation d un GtkHSeparator */
    pSeparator = gtk_hseparator_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pSeparator, TRUE, FALSE, 0);

    pLabel = gtk_label_new("Date de naissance :");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    /* Creation du deuxieme GtkFrame */
    pFrame = gtk_frame_new("Domicile");
    gtk_box_pack_start(GTK_BOX(pVBox), pFrame, TRUE, FALSE, 0);

    /* Creation et insertion d une boite pour le deuxieme GtkFrame */
    pVBoxFrame = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pFrame), pVBoxFrame);

    /* Creation et insertion des elements contenus dans le deuxieme GtkFrame */
    pLabel = gtk_label_new("Adresse :");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    pLabel = gtk_label_new("Adresse :");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    pLabel = gtk_label_new("Code postal :");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
```

```
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    pLabel = gtk_label_new("Ville :");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    /* Creation du troisieme GtkFrame */
    sUtf8 = g_locale_to_utf8("Téléphones", -1, NULL, NULL, NULL);
    pFrame = gtk_frame_new(sUtf8);
    g_free(sUtf8);
    gtk_box_pack_start(GTK_BOX(pVBox), pFrame, TRUE, FALSE, 0);

    /* Creation et insertion d une boite pour le troisieme GtkFrame */
    pVBoxFrame = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pFrame), pVBoxFrame);

    /* Creation et insertion des elements contenus dans le troisieme GtkFrame */
    pLabel = gtk_label_new("Domicile");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    pLabel = gtk_label_new("Professionnel");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    pLabel = gtk_label_new("Portable");
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pLabel, TRUE, FALSE, 0);
    pEntry = gtk_entry_new();
    gtk_box_pack_start(GTK_BOX(pVBoxFrame), pEntry, TRUE, FALSE, 0);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}
```

Résultat :


GtkEntry et GtkSeparator
Etat civil
Nom :
Prénom :
Date de naissance :
Domicile
Adresse :
Adresse :
Code postal :
Ville :
Téléphones
Domicile
Professionnel
Portable

Les images

1. Présentation

Nous allons cette fois, essayer de rendre nos fenêtres un peu moins austères. Pour cela, nous allons utiliser des images grâce au widget GtkImage.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkMisc? -> GtkImage

2. Utilisation de base

2.1 Création

Pour ce widget, il y a toute une panoplie de fonction de création. Nous n'allons en étudier que quelques-unes, car certaines font appel à des notions plus complexes. Voici donc les fonctions étudiées :

```
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);
```

La première crée une image mais complètement vide.

La deuxième crée l'image à partir du fichier *filename*. Gtk+ est capable d'utiliser les images qui sont au format PNG, JPEG, TIFF. Le chemin du fichier *filename* peut être relatif ou absolu. Si le chemin spécifié est incorrect ou que le format de l'image est invalide, l'image de retour sera celle-ci présentera une croix rouge sur fond blanc.

La troisième fonction, récupère l'image qui est associée à un objet GtkStockItem? afin de l'afficher. Le paramètre *size* peut prendre sept valeurs différentes pour définir la taille de l'image à afficher :

2.2 Modification de l'image.

Cette étape intervient lorsque vous avez créé une image vide ou lorsque vous voulez changer d'image. Les deux fonctions étudiées ici sont :

*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);*

Les paramètres sont les même que lors de la création d'un widget GtkImage, sauf qu'il faut préciser à quel widget il faut appliquer l'image.

2.3 Programme exemple

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

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pImage;
    GtkWidget *pQuitImage;
    GtkWidget *pQuitBtn;

    gtk_init(&argc, &argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkImage");
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    pVBox = gtk_vbox_new(FALSE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    /* Chargement d'une image a partir d'un fichier */
    pImage = gtk_image_new_from_file("./gtk.png");
    gtk_box_pack_start(GTK_BOX(pVBox), pImage, FALSE, FALSE, 5);

    pQuitBtn = gtk_button_new();
    gtk_box_pack_start(GTK_BOX(pVBox), pQuitBtn, TRUE, FALSE, 5);
    g_signal_connect(G_OBJECT(pQuitBtn), "clicked", G_CALLBACK(gtk_main_quit),
NULL);

    /* Chargement d'une image a partir d'un GtkStockItem */
    pQuitImage = gtk_image_new_from_stock(GTK_STOCK_QUIT,
GTK_ICON_SIZE_LARGE_TOOLBAR);
    gtk_container_add(GTK_CONTAINER(pQuitBtn), pQuitImage);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}
```

Résultat :


GtkImage

Les boîtes de dialogue

1. Présentation

Nous allons dans ce chapitre étudier un style de fenêtre particulier : les boîtes de dialogue. Elles permettent de demander à l'utilisateur des informations ou alors d'afficher des messages. Ces boîtes de dialogue ressemblant fortement à des fenêtres classiques, il est normal que le widget GtkDialog dérive directement de GtkWindow.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkWindow -> GtkDialog
GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkWindow -> GtkDialog -> GtkMessageDialog

1.2 Constitution d'une boîte de dialogue

Comme nous pouvons le voir sur l'image ci-dessus, une boîte de dialogue est constituée de trois éléments :

- une GtkVBox globale (en rouge) qui contiendra tous les widgets affichés ;
- une GtkHSeparator (en vert) qui sert de séparation entre la zone de saisie et la zone des boutons ;
- une GtkHBox (en bleu) qui contiendra les boutons de réponse.

On peut donc ainsi distinguer deux zones différentes :

- la zone de travail au-dessus de la GtkHSeparator ;
- la zone de réponse au-dessous de la GtkHSeparator.

2. La boîte de saisie

La saisie d'information via le widget GtkEntry ne s'effectue presque jamais dans la fenêtre principale mais dans une boîte de dialogue. Nous allons donc voir comment créer une telle boîte.

2.1 Création

La fonction de création est un peu plus complexe que d'habitude :

```
GtkWidget* gtk_dialog_new_with_buttons(const gchar *title, GtkWindow *parent,
GtkDialogFlags flags, const gchar *first_button_text, ...);
```

Le premier paramètre *title* n'est autre que le titre de la boîte de dialogue. Le deuxième paramètre *parent* sert à désigner la fenêtre parente. Cette valeur peut être égale à NULL.

Le troisième paramètre *flags* permet de définir certains paramètres de la boîte de dialogue. Ce paramètre peut prendre trois valeurs différentes :

- GTK_DIALOG_MODAL : créer une boîte de dialogue modale, c'est à dire que tant que l'utilisateur n'a pas cliqué sur un des boutons de réponse, les autres fenêtres seront figées ;
- GTK_DIALOG_DESTROY_WITH_PARENT : la boîte de dialogue est détruite si la fenêtre parente est détruite ;
- GTK_DIALOG_NO_SEPARATOR : la ligne de séparation entre la zone de travail et la zone de réponse n'est pas affichée.

Enfin, le dernier paramètre est plutôt une liste de paramètres permettant de définir les boutons de la boîte de dialogue ainsi que les réponses qui leurs sont associées. Pour chaque bouton que nous voulons ajouter, il faut définir le texte du bouton et le type de réponse qui sera envoyé lorsque nous cliquerons sur le bouton.

Le texte du bouton peut, comme pour tous les boutons, être un texte normal, un texte avec raccourci (Rappel : c'est de la forme "_Quitter") ou même un GtkStockItem?.

La valeur de la réponse, peut être un élément de type GtkResponseType ou bien une valeur entière positive que nous pouvons définir nous-mêmes. Le plus simple étant bien sûr d'utiliser les valeurs classiques définies par le type GtkResponseType dont voici la liste :

- GTK_RESPONSE_NONE ;
- GTK_RESPONSE_REJECT ;
- GTK_RESPONSE_ACCEPT ;
- GTK_RESPONSE_DELETE_EVENT ;
- GTK_RESPONSE_OK ;
- GTK_RESPONSE_CANCEL ;
- GTK_RESPONSE_CLOSE ;
- GTK_RESPONSE_YES ;
- GTK_RESPONSE_NO ;
- GTK_RESPONSE_APPLY ;
- GTK_RESPONSE_HELP.

Une fois que tous les boutons ont été définis, il faut le dire à notre fonction de création. Pour cela, il suffit de terminer la liste des paramètres par NULL.

2.2 Ajout d'éléments dans la zone de travail

La majeure partie des éléments de la boîte de dialogue sont maintenant créés. Il reste

cependant à ajouter les éléments de la zone de travail pour que la boîte soit complète. Une fois que tous les éléments à ajouter dans la zone de travail sont créés, il suffit de les ajouter dans la boîte de dialogue avec la fonction *gtk_box_pack_start*.

La question est maintenant de savoir comment passer la GtkVBox en paramètre à la fonction *gtk_box_pack_start*. Nous allons tout simplement utiliser l'opérateur -> pour l'élément GtkDialog de cette manière :

```
GTK_DIALOG(nom_de_la_boite)->vbox;
```

La boîte de dialogue est maintenant complète.

2.3 Affichage de la boîte de dialogue

L'affichage de la boîte de dialogue comporte deux étapes. Tout d'abord, il faut demander d'afficher tout le contenu de la GtkVBox qui est inclut dans la boîte de dialogue avec la fonction *gtk_widget_show_all*.

Ensuite il faut afficher la boîte de dialogue en elle-même et attendre la réponse de l'utilisateur avec cette fonction :

```
gint gtk_dialog_run(GtkDialog *dialog);
```

Le paramètre de cette fonction étant de type GtkDialog il faut utiliser la macro de conversion GTK_DIALOG().

Cette fonction affiche l'intégralité de la boîte de dialogue, mais rentre aussi dans une boucle récursive qui ne s'arrêtera que lorsque l'utilisateur cliquera sur un des boutons de retour. La valeur de retour correspond à la valeur associée au bouton cliqué. Si, par hasard, l'utilisateur quitte la boîte de dialogue en cliquant sur la croix de celle-ci, gtk_dialog_run renvoie GTK_RESPONSE_NONE.

Une fois la valeur de retour connue, il ne reste plus qu'à agir en conséquence.

2.4 Programme exemple

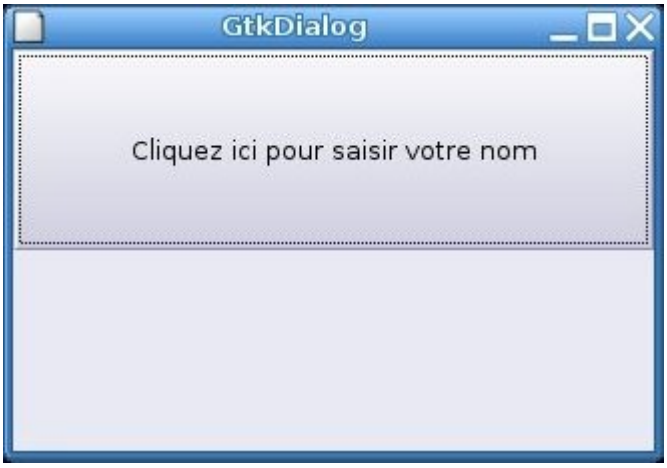
L'exemple de ce chapitre est constitué d'une fenêtre principale dans laquelle nous allons ajouter un GtkButton et un GtkLabel. Lorsque l'utilisateur clique sur le GtkButton, une boîte de dialogue apparaîtra et demandera à l'utilisateur de saisir son nom.
Cette boîte de dialogue sera constituée d'une GtkEntry, d'un GtkButton "OK" et d'un GtkButton "Annuler". Si l'utilisateur clique sur "OK" le contenu de la GtkEntry sera copié dans le GtkLabel de la fenêtre principale tandis que s'il clique sur "Annuler" on affichera "Vous n'avez rien saisi." dans le GtkLabel.

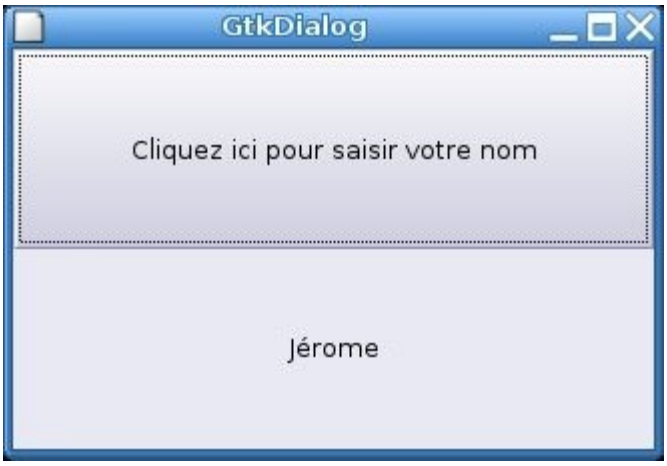
```
#include <stdlib.h>
#include <gtk/gtk.h>

static GtkWidget *pLabel;
static GtkWidget *pWindow;

void LancerBoite(void);

int main(int argc, char **argv)
{
    GtkWidget *pVBox;
    GtkWidget *pButton;
```

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

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkDialog");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    pVBox = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    pButton = gtk_button_new_with_label("Cliquez ici pour saisir votre nom");
    gtk_box_pack_start(GTK_BOX(pVBox), pButton, FALSE, TRUE, 0);

    pLabel = gtk_label_new(NULL);
    gtk_box_pack_start(GTK_BOX(pVBox), pLabel, FALSE, FALSE, 0);

    /* Connexion du signal "clicked" pour ouvrir la boite de dialogue */
    g_signal_connect(G_OBJECT(pButton), "clicked", G_CALLBACK(LancerBoite),
NULL);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}

void LancerBoite(void)
{
    GtkWidget* pBoite;
    GtkWidget* pEntry;
    const gchar* sNom;

    /* Creation de la boite de dialogue */
    /* 1 bouton Valider */
    /* 1 bouton Annuler */
    pBoite = gtk_dialog_new_with_buttons("Saisie du nom",
        GTK_WINDOW(pWindow),
        GTK_DIALOG_MODAL,
        GTK_STOCK_OK,GTK_RESPONSE_OK,
        GTK_STOCK_CANCEL,GTK_RESPONSE_CANCEL,
        NULL);

    /* Creation de la zone de saisie */
    pEntry = gtk_entry_new();
    gtk_entry_set_text(GTK_ENTRY(pEntry), "Saisissez votre nom");
    /* Insertion de la zone de saisie dans la boite de dialogue */
    /* Rappel : paramètre 1 de gtk_box_pack_start de type GtkBox */
    gtk_box_pack_start(GTK_BOX(GTK_DIALOG(pBoite)->vbox), pEntry, TRUE, FALSE,
0);

    /* Affichage des elements de la boite de dialogue */
    gtk_widget_show_all(GTK_DIALOG(pBoite)->vbox);

    /* On lance la boite de dialogue et on recupere la reponse */
    switch (gtk_dialog_run(GTK_DIALOG(pBoite)))
    {
        /* L utilisateur valide */
        case GTK_RESPONSE_OK:
```

```
            sNom = gtk_entry_get_text(GTK_ENTRY(pEntry));
            gtk_label_set_text(GTK_LABEL(pLabel), sNom);
            break;
        /* L utilisateur annule */
        case GTK_RESPONSE_CANCEL:
        case GTK_RESPONSE_NONE:
        default:
            gtk_label_set_text(GTK_LABEL(pLabel), "Vous n'avez rien saisi !");
            break;
    }

    /* Destruction de la boite de dialogue */
    gtk_widget_destroy(pBoite);
}
```

Résultats :





3. La boîte de message

Comme nous venons de le voir, le widget GtkDialog permet d'accélérer la programmation de simple fenêtre de saisie. Pour ce qui est de l'affichage de message, GTK+ offre un nouveau widget qui dérive directement de GtkDialog : le widget GtkMessageDialog. Ce widget permet de créer une boîte de dialogue complète avec une seule fonction.

3.1 Création

L'unique fonction de ce widget est la fonction permettant de créer la boîte de dialogue :

```
GtkWidget* gtk_message_dialog_new (GtkWindow *parent, GtkDialogFlags flags,
GtkMessageType type, GtkButtonsType buttons, const gchar *message_format, ...);
```

Les paramètres *parent* et *flags* sont identiques à ceux du widget GtkDialog, nous ne reviendrons donc pas dessus et allons nous concentrer sur les nouveaux paramètres. Le tout premier *type*, permet de définir le texte qui sera affiché dans la barre de titre de la boîte de dialogue ainsi que l'icône correspondant. Ce paramètre est de type GtkMessageType et peut prendre une des quatre valeurs suivantes :

- GTK_MESSAGE_INFO : titre de la boîte "Information"
- GTK_MESSAGE_WARNING : titre de la boîte "Avertissement"
- GTK_MESSAGE_QUESTION : titre de la boîte "Question"
- GTK_MESSAGE_ERROR : titre de la boîte "Erreur".

Le deuxième nouveau paramètre *buttons*, de type GtkButtonsType, permet de définir les boutons qui seront présents en bas de la boîte de dialogue. Les valeurs autorisées sont les suivantes :

- GTK_BUTTONS_NONE
- GTK_BUTTONS_OK
- GTK_BUTTONS_CLOSE
- GTK_BUTTONS_CANCEL
- GTK_BUTTONS_YES_NO

- GTK_BUTTONS_OK_CANCEL

Et le dernier paramètre *message_format* est tout simplement le texte qui sera affiché à l'intérieur de la boîte de dialogue. Ce texte peut être formaté comme il est possible de le faire avec la fonction *printf*.

3.2 Programme exemple

Nous allons créer une fenêtre comportant deux boutons. Le premier permettra d'afficher les informations habituelles d'une boîte de dialogue "A propos...". Le deuxième offrira la possibilité de quitter le programme, en passant par une demande de confirmation à l'utilisateur.

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

void OnAboutBtn(GtkWidget *pBtn, gpointer data);
void OnQuitBtn(GtkWidget *pBtn, gpointer data);

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pQuitterBtn;
    GtkWidget *pAboutBtn;

    gtk_init(&argc,&argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkMessageDialog");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    pVBox = gtk_vbox_new(TRUE,0);
    gtk_container_add(GTK_CONTAINER(pWindow),pVBox);

    pAboutBtn = gtk_button_new_with_label("A propos...");
    gtk_box_pack_start(GTK_BOX(pVBox), pAboutBtn, TRUE, FALSE,0);
    g_signal_connect(G_OBJECT(pAboutBtn), "clicked", G_CALLBACK(OnAboutBtn),
(GtkWidget*) pWindow);

    pQuitterBtn = gtk_button_new_from_stock (GTK_STOCK_QUIT);
    gtk_box_pack_start(GTK_BOX(pVBox), pQuitterBtn, TRUE, FALSE, 0);
    g_signal_connect(G_OBJECT(pQuitterBtn), "clicked", G_CALLBACK(OnQuitBtn),
(GtkWidget*) pWindow);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}

void OnAboutBtn(GtkWidget *pBtn, gpointer data)
{
    GtkWidget *pAbout;
    gchar *sSite = "http://www.gtk-fr.org";
```

```
    /* Creation de la boite de message */
    /* Type : Information -> GTK_MESSAGE_INFO */
    /* Bouton : 1 OK -> GTK_BUTTONS_OK */
    pAbout = gtk_message_dialog_new (GTK_WINDOW(data),
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_INFO,
        GTK_BUTTONS_OK,
        "Cours GTK+ 2.0\n%s",
        sSite);

    /* Affichage de la boite de message */
    gtk_dialog_run(GTK_DIALOG(pAbout));

    /* Destruction de la boite de message */
    gtk_widget_destroy(pAbout);
}

void OnQuitBtn(GtkWidget* widget, gpointer data)
{
    GtkWidget *pQuestion;

    /* Creation de la boite de message */
    /* Type : Question -> GTK_MESSAGE_QUESTION */
    /* Boutons : 1 OUI, 1 NON -> GTK_BUTTONS_YES_NO */
    pQuestion = gtk_message_dialog_new (GTK_WINDOW(data),
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_QUESTION,
        GTK_BUTTONS_YES_NO,
        "Voulez vous vraiment\nquitter ce programme?");

    /* Affichage et attente d une reponse */
    switch(gtk_dialog_run(GTK_DIALOG(pQuestion)))
    {
        case GTK_RESPONSE_YES:
            /* OUI -> on quitte l application */
            gtk_main_quit();
            break;
        case GTK_RESPONSE_NO:
            /* NON -> on detruit la boite de message */
            gtk_widget_destroy(pQuestion);
            break;
    }
}
```

Résultats :


GtkMessageDialog
A propos...
Quitter


GtkMessageDialog
Information
Cours GTK+ 2.0
http://www.gtk-fr.org
Valider


GtkMessageDialog
Question
Voulez vous vraiment
quitter ce programme?
Non
Oui

Les boutons (Partie 2)

1. Présentation

Nous allons étudier cette fois-ci trois nouveaux types de boutons qui dérivent du widget GtkButton. L'étude de ces widgets sera rapide car ils ne comportent que très peu de fonctions.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkButton -> GtkToggleButton

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkButton -> GtkToggleButton -> GtkCheckButton

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkButton -> GtkToggleButton -> GtkCheckButton -> GtkRadioButton

2. Le bouton poussoir

Le widget GtkToggleButton est un bouton poussoir qui ne peut prendre que deux états: enfoncé ou relâché.

2.1 Création

Comme d'habitude, il n'y a rien de bien compliqué vu que c'est toujours le même principe :

```
GtkWidget* gtk_toggle_button_new(void);
GtkWidget* gtk_toggle_button_new_with_label(const gchar* label);
GtkWidget* gtk_toggle_button_new_with_mnemonics(const gchar* label);
```

La première fonction crée un nouveau bouton vide, alors que la seconde ajoute du texte à l'intérieur et la troisième ajoute en plus un raccourci clavier.

2.2 Etat du bouton

Il peut être intéressant de connaître l'état du bouton pour agir en conséquence. Une fois encore, rien de plus simple on utilise la fonction :

```
gboolean gtk_toggle_button_get_active (GtkToggleButton *toggle_button);
```

Cette dernière nous renvoie TRUE si le bouton est enfoncé et FALSE sinon. Afin de pouvoir utiliser le paramètre *toggle_button* qui est le bouton dont on veut connaître l'état, il faut utiliser la macro GTK_TOGGLE_BUTTON().

Pour modifier l'état du bouton, c'est aussi simple :

```
void gtk_toggle_button_set_active (GtkToggleButton *toggle_button, gboolean is_active);
```

Il suffit de mettre le paramètre *is_active* à TRUE si l'on veut enfoncer le bouton ou à

FALSE pour le relâcher.

2.3 Apparence du bouton

Il existe cependant un troisième état qui n'est pas accessible en cliquant sur le bouton mais par une fonction spécifique. Ce troisième état, vous le connaissez sûrement. Le meilleur exemple est celui des éditeurs de texte :

Vous avez une phrase dans laquelle il y a du texte normal et du texte en gras. Si vous sélectionnez le texte en gras, le bouton permettant justement de le mettre en gras s'enfonce (en général B). Par contre si vous sélectionnez le texte normal, ce même bouton repasse à son état relâché. Venons en au troisième état, qui apparaît lorsque vous sélectionnez la phrase entière. Le bouton ne change pas d'état mais seulement d'aspect, il donne l'impression d'être inactif.

Ce changement d'aspect doit se faire manuellement, et s'effectue avec cette fonction:

*void gtk_toggle_button_set_inconsistent(GtkToggleButton *toggle_button, gboolean setting);*

Il suffit de mettre le paramètre *setting* à TRUE pour donner au bouton l'aspect inactif. Et pour connaître l'aspect du bouton, il y a tout naturellement la fonction :

*gboolean gtk_toggle_button_get_inconsistent(GtkToggleButton *toggle_button);*

2.4 Programme exemple

Nous allons construire une application contenant un GtkToggleButton qui changera d'état (bien sûr) lorsque nous cliquerons dessus, mais aussi lorsque nous cliquerons sur un deuxième bouton (le changement d'état entraînera la modification du label). De plus, il y aura un troisième bouton pour changer l'aspect du bouton (idem, on change aussi le label).

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

void OnToggle(GtkWidget *pToggle, gpointer data);
void OnEtatBtn(GtkWidget *pWidget, gpointer pToggle);
void OnAspectBtn(GtkWidget *pWidget, gpointer pToggle);

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pToggleBtn;
    GtkWidget *pEtatBtn;
    GtkWidget *pAspectBtn;
    GtkWidget *pVBox;
    gchar *sLabel;

    gtk_init(&argc,&argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkToggleButton");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);

    pVBox = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);
```

```
    /* Creation du label du bouton */
    sLabel = g_locale_to_utf8("Etat : Relâché - Aspect : Normal", -1, NULL,
NULL, NULL);
    /* Creation du bouton GtkToggleButton */
    pToggleBtn = gtk_toggle_button_new_with_label(sLabel);
    /* Le label sLabel n'est plus utile */
    g_free(sLabel);

    gtk_box_pack_start(GTK_BOX(pVBox), pToggleBtn, FALSE, FALSE, 0);

    pEtatBtn = gtk_button_new_with_label("CHANGER ETAT");
    gtk_box_pack_start(GTK_BOX(pVBox), pEtatBtn, FALSE, FALSE, 0);

    pAspectBtn = gtk_button_new_with_label("CHANGER ASPECT");
    gtk_box_pack_start(GTK_BOX(pVBox), pAspectBtn, FALSE, FALSE, 0);

    gtk_widget_show_all(pWindow);

    /* Connexion des signaux */
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);
    g_signal_connect(G_OBJECT(pToggleBtn), "toggled", G_CALLBACK(OnToggle),
NULL);
    g_signal_connect(G_OBJECT(pEtatBtn), "clicked", G_CALLBACK(OnEtatBtn),
pToggleBtn);
    g_signal_connect(G_OBJECT(pAspectBtn), "clicked", G_CALLBACK(OnAspectBtn),
pToggleBtn);

    gtk_main();

    return EXIT_SUCCESS;
}

void OnEtatBtn(GtkWidget *pWidget, gpointer pToggle)
{
    gboolean bEtat;

    /* Recuperation de l etat du bouton */
    bEtat = gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(pToggle));

    /* Modification de l etat du bouton */
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(pToggle), (bEtat ^ TRUE));
}

void OnAspectBtn(GtkWidget *pEtatBtn, gpointer pToggle)
{
    gboolean bInconsistent;

    /* Recuperation de l aspect du bouton */
    bInconsistent =
gtk_toggle_button_get_inconsistent(GTK_TOGGLE_BUTTON(pToggle));

    /* Modification de l aspect du bouton */
    gtk_toggle_button_set_inconsistent(GTK_TOGGLE_BUTTON(pToggle),
(bInconsistent ^ TRUE));

    /* On emet le signal "toggle" pour changer le texte du bouton */
    gtk_toggle_button_toggled(GTK_TOGGLE_BUTTON(pToggle));
}
```

```
void OnToggle(GtkWidget *pToggle, gpointer data)
{
    gboolean bEtat;
    gboolean bInconsistent;
    gchar *sLabel;
    gchar *sLabelUtf8;

    /* Recuperation de l etat du bouton */
    bEtat = gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(pToggle));
    /* Recuperation de l aspect du bouton */
    bInconsistent =
gtk_toggle_button_get_inconsistent(GTK_TOGGLE_BUTTON(pToggle));

    /* Construction du label du bouton */
    sLabel = g_strdup_printf("Etat : %s - Aspect : %s",
        bEtat ? "Enfoncé" : "Relâché",
        bInconsistent ? "Modifié" : "Normal");
    /* Encodage du label en UTF8 */
    sLabelUtf8 = g_locale_to_utf8(sLabel, -1, NULL, NULL, NULL);

    /* Modification du label du bouton */
    gtk_button_set_label(GTK_BUTTON(pToggle), sLabelUtf8);

    /* Les chaines sLabel et sLabelUtf8 n'ont plus d'utilite */
    g_free(sLabel);
    g_free(sLabelUtf8);
}
```

Résultat :

3. Les cases à cocher

3.1 Création

Une fois encore, la syntaxe et l'utilisation des fonctions de création restent classiques :

```
GtkWidget* gtk_check_button_new (void);
GtkWidget* gtk_check_button_new_with_label (const gchar *label);
GtkWidget* gtk_check_button_new_with_mnemonic(const gchar *label);
```

Il n'existe pas d'autres fonctions pour ce widget, il faut donc utiliser les fonctions des widgets GtkToggleButton et GtkButton pour récupérer les propriétés du widget (état, aspect, label, ...).

3.2 Programme exemple

Nous n'allons pas ici créer de programme exemple pour ce widget car il suffit juste de remplacer gtk_toggle_button_new_with_label par gtk_check_button_new_with_label dans l'exemple précédent pour obtenir ceci :

4. Les boutons GtkRadioButton

Nous allons maintenant le widget GtkRadioButton qui se différencie des autres boutons par la possibilité d'en grouper plusieurs. De ce fait, lorsque par exemple nous avons un groupe de trois boutons, il n'y en a qu'un seul qui pourra être actif. On pourrait très bien faire cela avec le widget GtkCheckButton mais cela serait beaucoup plus long à programmer.

4.1 Création

Afin de grouper les boutons radio, GTK+ utilise les listes simplement chaînées de GLib. Pour créer le premier bouton radio du groupe, il faut obligatoirement passer par une de ces fonctions :

```
GtkWidget* gtk_radio_button_new (GSList *group);
GtkWidget* gtk_radio_button_new_with_label (GSList *group, const gchar *label);
GtkWidget* gtk_radio_button_new_with_mnemonic(GSList *group, const gchar *label);
```

Au moment de la création, le bouton radio est automatiquement ajouté à la liste *group*. Cependant, ce paramètre n'est pas obligatoire. Nous pouvons très bien mettre NULL comme valeur et cela marchera de la même manière, sauf que nous n'aurons pas de pointeur sur la liste. La valeur de retour de cette fonction sera aussi copiée dans la variable data de la liste chaînée.

Ensuite pour rajouter les autres boutons au groupe, il y a plusieurs possibilités. La première est d'utiliser une des trois fonctions précédentes mais ce n'est pas tout, car autant pour le premier bouton du groupe, il n'est pas nécessaire d'avoir une liste, autant pour les autres boutons cela devient obligatoire. Pour cela, GTK+ nous fournit une fonction qui permet d'obtenir la liste dans laquelle les boutons du groupe sont ajoutés :

```
GSList* gtk_radio_button_get_group(GtkRadioButton *radio_button);
```

Avec cette fonction, nous pouvons donc connaître la liste à laquelle appartient le bouton *radio_button*, ce qui va nous permettre d'ajouter de nouveau bouton au groupe. Mais là où cela se complique, c'est qu'il faut récupérer la liste avant chaque ajout de bouton avec le dernier bouton ajouté comme paramètre. Voici ce que cela donnerai pour un groupe de trois boutons :

```
pRadio1 = gtk_radio_button_new_with_label(NULL, "Radio 1");
pGroup = gtk_radio_button_get_group(GTK_RADIO_BUTTON(pRadio1));
pRadio2 = gtk_radio_button_new_with_label(pGroup, "Radio 2");
pGroup = gtk_radio_button_get_group(GTK_RADIO_BUTTON(pRadio2));
pRadio3 = gtk_radio_button_new_with_label(pGroup, "Radio 3");
```

Ce système peut donc s'avérer lourd lors de la création du groupe, surtout s'il contient un grand nombre de boutons. Heureusement, les concepteurs de GTK+ ont pensé à nous simplifier la vie en ajoutant ces trois fonctions :

```
GtkWidget* gtk_radio_button_new_from_widget(GtkRadioButton *group);
GtkWidget* gtk_radio_button_new_with_label_from_widget(GtkRadioButton *group, const gchar
*label);
GtkWidget* gtk_radio_button_new_with_mnemonic_from_widget(GtkRadioButton *group, const
gchar *label);
```

Cette fois *group* ne correspond pas à la liste mais à un des boutons du groupe. A chaque appel d'une de ces fonctions GTK+ va s'occuper de récupérer correctement la liste à laquelle appartient le bouton *group* et ajouter le nouveau bouton. Cela aura pour conséquence, dans le cas d'un groupe de trois boutons, de réduire le nombre de lignes de code de 5 à 3 (et oui, une ligne de code c'est une ligne de code).

Nous savons maintenant tout ce qu'il faut pour créer un groupe de GtkRadioButton.

4.2 Programme exemple

Le programme exemple se présente sous la forme d'un mini sondage à trois choix :

- Pour ;
- Contre ;
- Sans opinion.

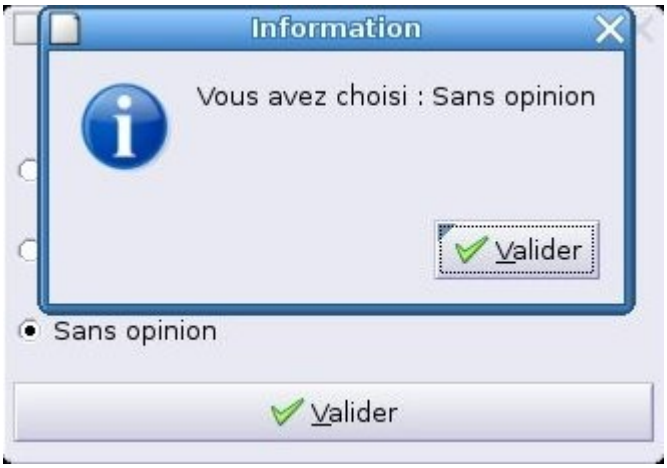
La possibilité de choisir parmi une de ces valeurs est rendue possible par l'utilisation des GtkRadioButton. Un autre bouton (tout ce qu'il y a de plus normal) permet de valider le choix fait par l'utilisateur ainsi que d'afficher le résultat dans une boîte de dialogue.

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

void OnValider(GtkWidget *pBtn, gpointer data);

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pRadio1;
    GtkWidget *pRadio2;
    GtkWidget *pRadio3;
    GtkWidget *pValider;
    GtkWidget *pLabel;

    gtk_init(&argc, &argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkRadioButton");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);

    pVBox = gtk_vbox_new(TRUE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow),pVBox);

    pLabel = gtk_label_new("Votre choix :");
    gtk_box_pack_start(GTK_BOX(pVBox), pLabel, FALSE, FALSE, 0);

    /* Creation du premier bouton radio */
    pRadio1 = gtk_radio_button_new_with_label(NULL, "Pour");
    gtk_box_pack_start(GTK_BOX (pVBox), pRadio1, FALSE, FALSE, 0);
    /* Ajout du deuxieme */
    pRadio2 = gtk_radio_button_new_with_label_from_widget(GTK_RADIO_BUTTON
(pRadio1), "Contre");
    gtk_box_pack_start(GTK_BOX (pVBox), pRadio2, FALSE, FALSE, 0);
    /* Ajout du troisieme */
    pRadio3 = gtk_radio_button_new_with_label_from_widget(GTK_RADIO_BUTTON
(pRadio1), "Sans opinion");
    gtk_box_pack_start(GTK_BOX (pVBox), pRadio3, FALSE, FALSE, 0);

    pValider = gtk_button_new_from_stock(GTK_STOCK_OK);
    gtk_box_pack_start(GTK_BOX (pVBox), pValider, FALSE, FALSE, 0);

    gtk_widget_show_all(pWindow);

    /* Connexion des signaux */
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);
    g_signal_connect(G_OBJECT(pValider), "clicked", G_CALLBACK(OnValider),
```

```
pRadio1);

    gtk_main();

    return EXIT_SUCCESS;
}

void OnValider(GtkWidget *pBtn, gpointer data)
{
    GtkWidget *pInfo;
    GtkWidget *pWindow;
    GSList *pList;
    const gchar *sLabel;

    /* Recuperation de la liste des boutons */
    pList = gtk_radio_button_get_group(GTK_RADIO_BUTTON(data));

    /* Parcours de la liste */
    while(pList)
    {
        /* Le bouton est il selectionne */
        if(gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(pList->data)))
        {
            /* OUI -> on copie le label du bouton */
            sLabel = gtk_button_get_label(GTK_BUTTON(pList->data));
            /* On met la liste a NULL pour sortir de la boucle */
            pList = NULL;
        }
        else
        {
            /* NON -> on passe au bouton suivant */
            pList = g_slist_next(pList);
        }
    }

    /* On recupere la fenetre principale */
    /*
    /* A partir d'un widget, gtk_widget_get_toplevel
    /* remonte jusqu'a la fenetre mere qui nous est
    /* utile pour l'affichage de la boite de dialogue
    */
    pWindow = gtk_widget_get_toplevel(GTK_WIDGET(data));

    pInfo = gtk_message_dialog_new (GTK_WINDOW(pWindow),
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_INFO,
        GTK_BUTTONS_OK,
        "Vous avez choisi : %s", sLabel);

    gtk_dialog_run(GTK_DIALOG(pInfo));

    gtk_widget_destroy(pInfo);
}
```

Résultat :


GtkRadioButton
Votre choix :
Pour
Contre
Sans opinion
Valider


Information
Vous avez choisi : Sans opinion
Valider
Sans opinion
Valider

Les Menus

1. Introduction

Avant toute chose, il faut savoir q'un menu est composé de plusieurs parties distinctes:

- La barre de menu
  - GtkMenuBar
- Le menu lui même
  - GtkMenu
- Les éléments d'un menu
  - GtkMenuItem ;
  - GtkImageMenuItem ;
  - GtkRadioMenuItem ;
  - GtkCheckMenuItem ;
  - GtkSeparatorMenuItem ;
  - GtkTearoffMenuItem.

La barre de menu est l'élément principal qui contiendra des éléments. A chaque élément, nous pouvons associer un menu qui s'ouvrira lorsque l'utilisateur cliquera sur cet élément. Et chaque menu contiendra des éléments qui pourront ouvrir des sous-menus, etc...

En plus d'utiliser tous ces widgets, il faudra aussi le widget GtkMenuShell qui est une classe de base pour les widgets GtkMenuBar et GtkMenu.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkMenuShell -> GtkMenuBar

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkMenuShell -> GtkMenu

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkItem? -> GtkMenuItem

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkItem? -> GtkMenuItem -> GtkImageMenuItem

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkItem? -> GtkMenuItem -> GtkCheckMenuItem

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkItem? -> GtkMenuItem -> GtkCheckMenuItem -> GtkRadioMenuItem

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkItem? -> GtkMenuItem -> GtkSeparatorMenuItem

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkItem? -> GtkMenuItem -> GtkTearoffMenuItem

2. Le principes

2.1 Les différentes étapes

La création d'un menu doit passer par au moins six étapes différentes :

- Etape 1 : création de l'élément GtkMenuBar qui sera la barre de menu;
- Etape 2 : création d'un élément GtkMenu qui sera un menu ;
- Etape 3 : création des éléments GtkMenuItem à insérer dans le menu ;
- Etape 4 : création de l'élément GtkMenuItem qui ira dans l'élément GtkMenuBar;
- Etape 5 : association de cet élément avec le menu créer précédemment ;
- Etape 6 : ajout de l'élément GtkMenuItem dans la barre GtkMenuBar.

Si par la suite, vous souhaitez ajouter d'autres menu, il suffit de recommencer à partir de l'étape 2.

2.2 Etape 1 : Création de l'élément GtkMenuBar

Cette étape, rapide et simple, se résume en l'utilisation d'une seule fonction :

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

2.3 Etape 2 : Création d'un élément GtkMenu

Cette fois aussi, une seule fonction est utilisée :

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

2.4 Etape 3 : Création des éléments GtkMenuItem (ou autres)

Dans un premier temps, il faut créer le GtkMenuItem grâce cette fonction :

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

Maintenant que l'élément est créé, il faut l'insérer dans le menu. Pour cela, il existe plusieurs fonctions possibles, mais nous n'allons en voir que deux :

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

Ces fonctions ne font pas partie du widget GtkMenu mais de GtkMenuShell qui est le widget dont GtkMenu dérive. La première fonction ajoute les éléments de haut en bas alors que la seconde les ajoute de bas en haut. Le paramètre *menu_shell* est le menu dans lequel nous voulons ajouter l'élément et le paramètre *child* est l'élément à ajouter. Pour le premier paramètre, il faudra utiliser la macro de conversion GTK_MENU_SHELL().
Cette étape peut s'avérer longue à coder, car il faudra la répéter pour chaque élément que nous voulons ajouter au menu.

2.5 Etape 4 : Création de l'élément GtkMenuItem qui ira dans l'élément GtkMenuBar

Il suffit d'utiliser une des fonctions de création du widget GtkMenuItem.

2.6 Etape 5 : Association de cet élément avec le menu créé précédemment

Il faut maintenant dire que lorsque l'utilisateur cliquera sur l'élément créer pendant l'étape 4, il faudra ouvrir le menu qui a été créé précédemment. La fonction adéquate est celle-ci :

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

2.7 Etape 6 : Ajout de l'élément GtkMenuItem dans la barre GtkMenuBar

GtkMenuBar dérivant aussi de GtkMenuShell, nous allons utiliser la même fonction que lors de l'étape 3 (gtk_menu_shell_append) pour ajouter le sous-menu au menu principal.

2.8 Programme exemple

Nous allons créer une fenêtre simple avec un menu "Fichier" et un menu "?".

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

void OnQuitter(GtkWidget* widget, gpointer data);
void OnAbout(GtkWidget* widget, gpointer data);

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pMenuBar;
    GtkWidget *pMenu;
    GtkWidget *pMenuItem;

    gtk_init(&argc, &argv);

    /* Creation de la fenetre */
    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkMenu");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    /* Creation de la GtkVBox */
    pVBox = gtk_vbox_new(FALSE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    /**** Creation du menu ****/

    /* ETAPE 1 */
    pMenuBar = gtk_menu_bar_new();
```

```
    /** Premier sous-menu **/
    /* ETAPE 2 */
    pMenu = gtk_menu_new();
    /* ETAPE 3 */
    pMenuItem = gtk_menu_item_new_with_label("Nouveau");
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    pMenuItem = gtk_menu_item_new_with_label("Ouvrir");
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    pMenuItem = gtk_menu_item_new_with_label("Enregistrer");
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    pMenuItem = gtk_menu_item_new_with_label("Fermer");
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    pMenuItem = gtk_menu_item_new_with_label("Quitter");
    g_signal_connect(G_OBJECT(pMenuItem), "activate", G_CALLBACK(OnQuitter),
(GtkWidget*) pWindow);
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);
    /* ETAPE 4 */
    pMenuItem = gtk_menu_item_new_with_label("Fichier");
    /* ETAPE 5 */
    gtk_menu_item_set_submenu(GTK_MENU_ITEM(pMenuItem), pMenu);
    /* ETAPE 6 */
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenuBar), pMenuItem);

    /** Second sous-menu **/
    /* ETAPE 2 */
    pMenu = gtk_menu_new();
    /* ETAPE 3 */
    pMenuItem = gtk_menu_item_new_with_label("A propos de...");
    g_signal_connect(G_OBJECT(pMenuItem), "activate", G_CALLBACK(OnAbout),
(GtkWidget*) pWindow);
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);
    /* ETAPE 4 */
    pMenuItem = gtk_menu_item_new_with_label("?");
    /* ETAPE 5 */
    gtk_menu_item_set_submenu(GTK_MENU_ITEM(pMenuItem), pMenu);
    /* ETAPE 6 */
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenuBar), pMenuItem);

    /* Ajout du menu a la fenetre */
    gtk_box_pack_start(GTK_BOX(pVBox), pMenuBar, FALSE, FALSE, 0);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}

void OnQuitter(GtkWidget* widget, gpointer data)
{
    GtkWidget *pQuestion;

    pQuestion = gtk_message_dialog_new(GTK_WINDOW(data),
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_QUESTION,
        GTK_BUTTONS_YES_NO,
        "Voulez vous vraiment\n"
```

```
        "quitter le programme?");

    switch(gtk_dialog_run(GTK_DIALOG(pQuestion)))
    {
        case GTK_RESPONSE_YES:
            gtk_main_quit();
            break;
        case GTK_RESPONSE_NONE:
        case GTK_RESPONSE_NO:
            gtk_widget_destroy(pQuestion);
            break;
    }
}

void OnAbout(GtkWidget* widget, gpointer data)
{
    GtkWidget *pAbout;

    pAbout = gtk_message_dialog_new (GTK_WINDOW(data),
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_INFO,
        GTK_BUTTONS_OK,
        "Cours GTK+ 2.0\n"
        "http://www.gtk-fr.org");

    gtk_dialog_run(GTK_DIALOG(pAbout));

    gtk_widget_destroy(pAbout);
}
```

Résultat :

3. Les éléments avancées d'un menu

En plus des GtkMenuItem, GTK+ offre cinq éléments de menu additionnels prêts à l'emploi : GtkImageMenuItem, GtkRadioMenuItem, GtkCheckMenuItem, GtkSeparatorMenuItem et GtkTearoffMenuItem.

3.1 Les GtkImageMenuItem

GtkImageMenuItem est un widget qui permet de créer une entrée de menu textuelle avec une icône juste devant.

Pour créer un GtkImageMenuItem, on a à disposition quatre fonctions :

```
GtkWidget* gtk_image_menu_item_new(void);
GtkWidget* gtk_image_menu_item_new_from_stock(const gchar *stock_id, GtkAccelGroup? *accel_group);
GtkWidget* gtk_image_menu_item_new_with_label(const gchar *label);
GtkWidget* gtk_image_menu_item_new_with_mnemonic(const gchar *label);
```

On retrouve encore les mêmes types de fonctions de création, aussi je passe sur l'explication des paramètres. La seule nouveautée est peut être le GtkAccelGroup? *accel_group de gtk_image_menu_item_new_from_stock, qui sert à ajouter l'entrée à un GtkAccelGroup? précédemment créé en utilisant le raccourci par défaut de l'icône

stock.

Maintenant, il s'agit de définir une icône pour notre entrée, pour cela, on a :

```
void gtk_image_menu_item_set_image(GtkImageMenuItem *image_menu_item, GtkWidget *image);
```

Cette fonction permet de définir l'icône (généralement on utilisera un GtkImage) qui sera affichée par l'entrée passée en paramètre. Cette fonction peut aussi servir à remplacer l'icône que gtk_image_menu_item_new_from_stock définit pour l'entrée lors de sa création.

La dernière fonction spécifique des GtkImageMenuItem est :

```
GtkWidget* gtk_image_menu_item_get_image(GtkImageMenuItem *image_menu_item);
```

Elle sert, comme vous l'aurez deviné, à récupérer ce qui sert d'icône à l'entrée.

3.2 Les GtkCheckMenuItem

Le widget GtkCheckMenuItem est en fait un GtkCheckButton qui a été modifié afin de pouvoir êtreinséré dans des menus.

Les GtkCheckMenuItem émettent le signal "toggled" lorsqu'on les (dé)coche.

Les fonctions de création d'un GtkCheckMenuItem sont :

```
GtkWidget* gtk_check_menu_item_new(void);
GtkWidget* gtk_check_menu_item_new_with_label(const gchar *label);
GtkWidget* gtk_check_menu_item_new_with_mnemonic(const gchar *label);
```

Rien de nouveau à l'horizon, alors nous continuons.

Nous avons plusieurs fonctions qui permettent de changer l'état de notre GtkCheckMenuItem par programme :

```
void gtk_check_menu_item_set_active(GtkCheckMenuItem *check_menu_item, gboolean is_active);
void gtk_check_menu_item_set_inconsistent(GtkCheckMenuItem *check_menu_item, gboolean setting);
void gtk_check_menu_item_toggled(GtkCheckMenuItem *check_menu_item);
```

La première fonction pemet de passer le GtkCheckMenuItem dans l'état "coché" si le paramètre *is_active* est TRUE ou dans l'état "non coché" si *is_active* est FALSE. Cette fonction ne permet pas de mettre notre GtkCheckMenuItem dans le troisième état "demi coché", pour cela, il faut utiliser la deuxième fonction, et grâce au paramètre *setting*, on active ou non cet état. La troisième fonction, elle, permet d'alterner entre état "coché" et "non coché" car elle émet en fait le signal "toggled", ce qui a pour effet d'inverser l'état du GtkCheckMenuItem.

Nous disposons également des fonctions associées pour récupérer l'état d'un GtkCheckMenuItem :

```
gboolean gtk_check_menu_item_get_active(GtkCheckMenuItem *check_menu_item);
gboolean gtk_check_menu_item_get_inconsistent(GtkCheckMenuItem *check_menu_item);
```

La première permet de connaître l'état d'un GtkCheckMenuItem, elle renvoie TRUE si il est "coché" ou FALSE sinon.

La deuxième permet juste de savoir si le GtkCheckMenuItem est dans le troisième état "demi coché".

3.3 Les GtkRadioMenuItem

Ici aussi, le widget GtkRadioMenuItem est un GtkRadioButton adapté pour les menus. Les GtkRadioMenuItem héritent des GtkCheckMenuItem, donc tout ce qui a été dit juste avant s'applique aussi ici.

Pour créer un GtkRadioMenuItem, nous pouvons nous servir de :

```
GtkWidget* gtk_radio_menu_item_new(GSList *group);
GtkWidget* gtk_radio_menu_item_new_with_label(GSList *group, const gchar *label);
GtkWidget* gtk_radio_menu_item_new_with_mnemonic(GSList *group, const gchar *label);
```

Ce widget fonctionne de la même manière que le widget GtkRadioButton, et donc le paramètre group de ces fonctions, sert à dire au GtkRadioMenuItem à quel groupe il va appartenir.

Le premier GtkRadioMenuItem d'un groupe prendra toujours NULL comme paramètre *group*, de cette façon il va en créer un nouveau. Ensuite, il suffira de récupérer ce paramètre *group* grâce à la fonction suivante et de le passer à un autre GtkRadioMenuItem pour que celui ci fasse partie du même groupe que le premier :

```
GSList* gtk_radio_menu_item_get_group(GtkRadioMenuItem *radio_menu_item);
```

On peut aussi définir ou remplacer le groupe d'un GtkRadioMenuItem aprés coup grâce à :

```
void gtk_radio_menu_item_set_group(GtkRadioMenuItem *radio_menu_item, GSList *group);
```

3.4 Les GtkSeparatorMenuItem

Ce widget permet d'insérer une séparation constituée d'une ligne horizontale dans le menu.
Ce widget ne possède qu'une seule fonction de création :

```
GtkWidget* gtk_separator_menu_item_new(void);
```

3.5 Les GtkTearoffMenuItem

Le widget GtkTearoffMenuItem permet de détacher le menu de sa barre et d'en faire une fenêtre à part entière.

Pour le créer, il faut utiliser la fonction suivante :

```
GtkWidget* gtk_tearoff_menu_item_new(void);
```

Cette fonction va ajouter au menu une ligne en pointillée. Un premier clic sur

l'élément GtkTearoffMenuItem créera une copie du menu dans une fenêtre, et un second clic sur l'élément (que cela soit dans le menu ou dans la fenêtre) supprimera la fenêtre créée.

Pour savoir si le menu est attaché ou détaché, il faut utiliser la fonction :

*gboolean gtk_menu_get_tearoff_state(GtkMenu *menu);*

Cette dernière renverra TRUE si le menu est détaché et FALSE sinon.

3.6 Programme exemple

Nous avons repris l'exemple précédent en modifiant le menu pour y mettre nos nouveaux items et en y ajoutant trois labels pour indiquer l'état des différents items. Il y a aussi trois callbacks supplémentaires pour la mise à jour de nos labels.

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

void OnQuitter(GtkWidget* widget, gpointer data);
void OnAbout(GtkWidget* widget, gpointer data);
void OnRadio(GtkWidget* widget, gpointer data);
void OnTearoff(GtkWidget* widget, gpointer data);
void OnCheck(GtkWidget* widget, gpointer data);

static GtkWidget *pRadioLabel;
static GtkWidget *pCheckLabel;
static GtkWidget *pTearoffLabel;

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pVBox2;
    GtkWidget *pMenuBar;
    GtkWidget *pMenu;
    GtkWidget *pMenuItem;
    GSList *pList;
    gchar *sTempLabel;

    gtk_init(&argc, &argv);

    /* Creation de la fenetre */
    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkMenu");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    /* Creation de la GtkVBox */
    pVBox = gtk_vbox_new(FALSE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    /**** Creation du menu ****/

    /* ETAPE 1 */
    pMenuBar = gtk_menu_bar_new();
    /** Premier sous-menu **/
    /* ETAPE 2 */
    pMenu = gtk_menu_new();
```

```
    /* ETAPE 3 */

    /* GtkTearoffMenuItem */
    pMenuItem = gtk_tearoff_menu_item_new();
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);
    g_signal_connect(G_OBJECT(pMenuItem),"activate",G_CALLBACK(OnTearoff),(gpoin
ter)pMenu);

    /* GtkImageMenuItem */
    pMenuItem = gtk_image_menu_item_new_from_stock(GTK_STOCK_NEW,NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    pMenuItem = gtk_image_menu_item_new_from_stock(GTK_STOCK_OPEN,NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    pMenuItem = gtk_image_menu_item_new_from_stock(GTK_STOCK_SAVE,NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    pMenuItem = gtk_image_menu_item_new_from_stock(GTK_STOCK_CLOSE,NULL);
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    /* GtkSeparatorItem */
    pMenuItem = gtk_separator_menu_item_new();
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    /* GtkRadioMenuItem */
    pMenuItem = gtk_radio_menu_item_new_with_label(NULL, "Radio 1");
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);
    pList = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(pMenuItem));
    /* Il est inutile ici d'utiliser le signal "toggled" */
    g_signal_connect(G_OBJECT(pMenuItem), "activate", G_CALLBACK(OnRadio),
NULL);

    pMenuItem = gtk_radio_menu_item_new_with_label(pList, "Radio 2");
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);
    pList = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(pMenuItem));
    g_signal_connect(G_OBJECT(pMenuItem) ,"activate", G_CALLBACK(OnRadio),
NULL);

    pMenuItem = gtk_radio_menu_item_new_with_label(pList, "Radio 3");
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);
    pList = gtk_radio_menu_item_get_group(GTK_RADIO_MENU_ITEM(pMenuItem));
    g_signal_connect(G_OBJECT(pMenuItem) ,"activate", G_CALLBACK(OnRadio),
NULL);

    /* GtkSeparatorItem */
    pMenuItem = gtk_separator_menu_item_new();
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    /* GtkCheckMenuItem */
    pMenuItem = gtk_check_menu_item_new_with_label("Check");
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);
    g_signal_connect(G_OBJECT(pMenuItem),"toggled",G_CALLBACK(OnCheck),(gpointer
)pMenu);

    /* GtkSeparatorItem */
    pMenuItem = gtk_separator_menu_item_new();
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    pMenuItem = gtk_image_menu_item_new_from_stock(GTK_STOCK_QUIT,NULL);
    g_signal_connect(G_OBJECT(pMenuItem), "activate", G_CALLBACK(OnQuitter),
```

```
(GtkWidget*) pWindow);
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);

    /* ETAPE 4 */
    pMenuItem = gtk_menu_item_new_with_label("Fichier");
    /* ETAPE 5 */
    gtk_menu_item_set_submenu(GTK_MENU_ITEM(pMenuItem), pMenu);
    /* ETAPE 6 */
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenuBar), pMenuItem);

    /** Deuxieme sous-menu **/
    /* ETAPE 2 */
    pMenu = gtk_menu_new();
    /* ETAPE 3 */
    pMenuItem = gtk_menu_item_new_with_label("A propos de...");
    g_signal_connect(G_OBJECT(pMenuItem), "activate", G_CALLBACK(OnAbout),
(GtkWidget*) pWindow);
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenu), pMenuItem);
    /* ETAPE 4 */
    pMenuItem = gtk_menu_item_new_with_label("?");
    /* ETAPE 5 */
    gtk_menu_item_set_submenu(GTK_MENU_ITEM(pMenuItem), pMenu);
    /* ETAPE 6 */
    gtk_menu_shell_append(GTK_MENU_SHELL(pMenuBar), pMenuItem);

    /* Creation de la deuxieme GtkVBox (pour les labels) */
    pVBox2 = gtk_vbox_new(FALSE, 0);

    pRadioLabel = gtk_label_new("Radio 1 est actif");
    gtk_box_pack_start(GTK_BOX(pVBox2), pRadioLabel, TRUE, TRUE, 0);

    sTempLabel = g_locale_to_utf8("Check est décoché", -1, NULL, NULL, NULL);
    pCheckLabel = gtk_label_new(sTempLabel);
    g_free(sTempLabel);
    gtk_box_pack_start(GTK_BOX(pVBox2), pCheckLabel, TRUE, TRUE, 0);

    sTempLabel = g_locale_to_utf8("Menu attaché", -1, NULL, NULL, NULL);
    pTearoffLabel = gtk_label_new(sTempLabel);
    g_free(sTempLabel);
    gtk_box_pack_start(GTK_BOX(pVBox2), pTearoffLabel, TRUE, TRUE, 0);

    /* Ajout du menu a la fenetre */
    gtk_box_pack_start(GTK_BOX(pVBox), pMenuBar, FALSE, FALSE, 0);
    /* Ajout des labels a la fenetre */
    gtk_box_pack_start(GTK_BOX(pVBox), pVBox2, TRUE, TRUE, 0);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}


void OnRadio(GtkWidget* widget, gpointer data)
{
    const gchar *sRadioName;
    gchar *sLabel;

    /* Recuperer le label du bouton radio active */
    sRadioName = gtk_label_get_label(GTK_LABEL(GTK_BIN(widget)->child));
```

```
    sLabel = g_strdup_printf("%s est actif",sRadioName);
    gtk_label_set_label(GTK_LABEL(pRadioLabel), sLabel);
    g_free(sLabel);
}

void OnCheck(GtkWidget* widget, gpointer data)
{
    gboolean bCoche;
    gchar *sLabel;
    gchar *sLabelUtf8;

    /* Savoir si le GtkCheckMenuItem est coche ou non */
    bCoche = gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(widget));

    if(bCoche)
        sLabel = g_strdup("Check est coché");
    else
        sLabel = g_strdup("Check est décoché");

    sLabelUtf8 = g_locale_to_utf8(sLabel, -1, NULL, NULL, NULL);

    gtk_label_set_label(GTK_LABEL(pCheckLabel), sLabelUtf8);
    g_free(sLabel);
    g_free(sLabelUtf8);
}

void OnTearoff(GtkWidget* widget, gpointer data)
{
    gboolean bDetache;
    gchar *sLabel;
    gchar *sLabelUtf8;

    /* Savoir si le menu est detache ou non */
     bDetache = gtk_menu_get_tearoff_state(GTK_MENU(data));

    if(bDetache)
        sLabel = g_strdup("Menu détaché");
    else
        sLabel = g_strdup("Menu attaché");

    sLabelUtf8 = g_locale_to_utf8(sLabel, -1, NULL, NULL, NULL);

    gtk_label_set_label(GTK_LABEL(pTearoffLabel), sLabelUtf8);
    g_free(sLabel);
    g_free(sLabelUtf8);
}

void OnQuitter(GtkWidget* widget, gpointer data)
{
    GtkWidget *pQuestion;

    pQuestion = gtk_message_dialog_new(GTK_WINDOW(data),
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_QUESTION,
        GTK_BUTTONS_YES_NO,
        "Voulez vous vraiment\n"
        "quitter le programme?");

    switch(gtk_dialog_run(GTK_DIALOG(pQuestion)))
    {
```

```
        case GTK_RESPONSE_YES:
            gtk_main_quit();
            break;
        case GTK_RESPONSE_NONE:
        case GTK_RESPONSE_NO:
            gtk_widget_destroy(pQuestion);
            break;
    }
}

void OnAbout(GtkWidget* widget, gpointer data)
{
    GtkWidget *pAbout;

    pAbout = gtk_message_dialog_new (GTK_WINDOW(data),
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_INFO,
        GTK_BUTTONS_OK,
        "Cours GTK+ 2.0\n"
        "http://www.gtk-fr.org");

    gtk_dialog_run(GTK_DIALOG(pAbout));

    gtk_widget_destroy(pAbout);
}
```

Résultat :


GtkMenu
Fichier ?
Fichier
Nouveau
Ouvrir
Enregistrer
Fermer
Radio 1
Radio 2
Radio 3
Check
Quitter
Radio 2 est actif
Check est coché
Menu détaché

La barre d'outils

ATTENTION : beaucoup de fonctions utilisées dans ce tutorial ne doivent plus être utilisées avec les versions récentes de GTK+. Un nouveau tutorial est en cours d'élaboration

1. Présentation

La barre d'outils est très utile pour placer en dessous du menu les commandes les plus utilisées comme faire/défaire, nouveau, ouvrir, sauvegarde, etc. Un widget, GtkToolbar, indispensable pour une bonne application.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkToolbar

2. Utiliser une GtkToolbar

2.1 Création

Comme pour tous les autres widget, c'est très simple :

```
GtkWidget* gtk_toolbar_new(void);
```

2.2 Insertion d'éléments

GtkToolbar étant un container, après l'avoir créée, il va falloir ajouter des widget dedans. Cela se divise en 2 catégories.

- les widgets habituels comme les GtkButton, GtkToggleButton, GtkRadioButton pour lesquels GtkToolbar fourni des fonctions qui s'occupent de la création de ces derniers ;
- les autres widgets que nous pouvons ajouter mais c'est à nous de fournir les widget donc il faudra les créer auparavant.

Voici les premières fonctions qui permettent d'ajouter un bouton avec du texte et (ou) un icône :

```
GtkWidget* gtk_toolbar_append_item (GtkToolbar *toolbar, const char *text, const char *tooltip_text, const char *tooltip_private_text, GtkWidget *icon, GtkSignalFunc? callback, gpointer user_data);
GtkWidget* gtk_toolbar_prepend_item (GtkToolbar *toolbar, const char *text, const char *tooltip_text, const char *tooltip_private_text, GtkWidget *icon, GtkSignalFunc? callback, gpointer user_data);
GtkWidget* gtk_toolbar_insert_item (GtkToolbar *toolbar, const char *text, const char *tooltip_text,const char *tooltip_private_text, GtkWidget *icon, GtkSignalFunc? callback, gpointer user_data, gint position);
```

La première fonction ajoute un bouton à la suite des autres boutons, la seconde l'ajoute en première position et la dernière à une position spécifique.

Pour chaque fonction, le premier paramètre est la barre d'outil dans laquelle on veut ajouter un élément. Il faut comme d'habitude utiliser une macro de conversion qui cette fois est GTK_TOOLBAR().

Ensuite, le paramètre *text* n'est autre que le label du bouton. Le troisième paramètre, *tooltip_text*, est une aide qui s'affichera lorsque l'utilisateur laisse le pointeur de sa souris quelques secondes sur le bouton. Le paramètre suivant, *tooltip_private_text*, n'est plus utilisé car la partie qui gère ce paramètre est obsolète. Il faut donc mettre ce paramètre à NULL.

Le paramètre *icon* est là pour définir l'image qui sera associée au bouton. Les deux paramètres servent à connecter une fonction callback au clic sur le bouton. Le paramètre *callback* est en fait le nom de la fonction callback et *user_data* est la donnée supplémentaire à passer à la fonction callback.

Et pour terminer, la fonction gtk_toolbar_insert_item possède un paramètre supplémentaire *position* qui détermine à quelle position le bouton doit être ajouté. Si cette valeur est soit tros grande, soit négative, cette fonction aura le même effet que gtk_toolbar_append_item.

Ensuite il est possible de créer des éléments selon un type de widget prédéfini, comme GtkRadioButton, GtkToggleButton, GtkButton :

```
GtkWidget* gtk_toolbar_append_element (GtkToolbar *toolbar, GtkToolbarChildType? type,
GtkWidget *widget, const char *text, const char *tooltip_text, const char *tooltip_private_text,
GtkWidget *icon, GtkSignalFunc? callback, gpointer user_data);
GtkWidget* gtk_toolbar_prepend_element (GtkToolbar *toolbar, GtkToolbarChildType? type,
GtkWidget *widget, const char *text, const char *tooltip_text, const char *tooltip_private_text,
GtkWidget *icon, GtkSignalFunc? callback, gpointer user_data);
GtkWidget* gtk_toolbar_insert_element (GtkToolbar *toolbar, GtkToolbarChildType? type,
GtkWidget *widget, const char *text, const char *tooltip_text, const char *tooltip_private_text,
GtkWidget *icon, GtkSignalFunc? callback, gpointer user_data, gint position);
```

La majorité des paramètres de ces fonctions ont été détaillés précédemment, nous n'allons donc étudier que les nouveaux.

Tout d'abord, le paramètre *type* permet de définir le type de widget que nous allons ajouter, et peut prendre une de ces valeurs :

- GTK_TOOLBAR_CHILD_SPACE pour ajouter un espace ;
- GTK_TOOLBAR_CHILD_BUTTON pour ajouter un GtkButton ;
- GTK_TOOLBAR_CHILD_TOGGLEBUTTON pour ajouter un GtkToggleButton ;
- GTK_TOOLBAR_CHILD_RADIOBUTTON pour ajouter un GtkRadioButton ;
- GTK_TOOLBAR_CHILD_WIDGET pour ajouter un widget quelconque.

Il y a ensuite le paramètre *widget* qui doit être utiliser dans deux cas. Le premier cas est bien entendu, si nous ajoutons un élément de type GTK_TOOLBAR_CHILD_WIDGET.

Alors widget sera en fait le widget que nous voulons ajouter.

Le deuxième cas est si nous ajoutons un élément de type GTK_TOOLBAR_CHILD_RADIOBUTTON. Nous avons vu que les GtkRadioButton fonctionnés par groupe, alors dans ce cas, pour grouper les boutons radios le paramètre *widget* doit être un GtkRadioButton ajouté précédemment pour que GTK+ puisse les grouper. Bien sûr, s'il s'agit du premier GtkRadioButton, il faut mettre widget à NULL.

Dans tous les autres cas, le paramètre widget doit obligatoirement être à NULL.

Et maintenant, voila les fonctions pour ajouter n'importe quel type de widget :

```
void gtk_toolbar_append_widget (GtkToolbar *toolbar, GtkWidget *widget, const char
*tooltip_text, const char *tooltip_private_text);
void gtk_toolbar_prepend_widget (GtkToolbar *toolbar, GtkWidget *widget, const char
*tooltip_text, const char *tooltip_private_text);
void gtk_toolbar_insert_widget (GtkToolbar *toolbar, GtkWidget *widget, const char
*tooltip_text, const char *tooltip_private_text, gint position);
```

Cette fois, tout est simple, le paramètre *widget* est le widget que nous voulons ajouter. Il faudra tout de même connecter manuellement les signaux du widget ajouté.

Pour finir, il est possible d'utiliser les GtkStockItem? pour créer un bouton. Voici la fonction qui permet cela :

```
GtkWidget* gtk_toolbar_insert_stock (GtkToolbar *toolbar, const gchar *stock_id, const char
*tooltip_text, const char *tooltip_private_text, GtkSignalFunc? callback, gpointer user_data, gint
position);
```

Le paramètre *stock_id* est tout simplement l'identifiant du GtkStockItem? qui figurera sur le bouton.

2.3 Les espaces

Nous avons déjà vu que les fonctions de type gtk_toolbar_*_element permettaient d'ajouter un espace pour rendre plus claire la barre d'outils. Il existe en plus de cela trois autres fonctions :

```
void gtk_toolbar_append_space(GtkToolbar *toolbar );
void gtk_toolbar_prepend_space(GtkToolbar *toolbar );
void gtk_toolbar_insert_space(GtkToolbar *toolbar, guint pos );
```

Comme d'habitude la première fonction ajoute un espace à la suite des autres éléments, la deuxième au tout début de la barre d'outils et la troisième à une position particulière.

Pour supprimer un espace de la barre d'outils, il existe cette fonction :

```
void gtk_toolbar_remove_space(GtkToolbar *toolbar, guint pos);
```

2.4 Orientation de la barre d'outils

La barre d'outil peut être orienter verticalement ou horizontalement et cela à n'importe quel moment a l'aide de cette fonction :

```
void gtk_toolbar_set_orientation(GtkToolbar *toolbar, GtkOrientation? orientation) ;
```

Le paramètre *orientation* peut prendre deux valeurs :

- GTK_ORIENTATION_HORIZONTAL ;
- GTK_ORIENTATION_VERTICAL.

Et, cette fonction permet de connaître l'orientation de la barre d'outil :

*GtkOrientation? gtk_toolbar_get_oritentation(GtkToolbar *toolbar);*

2.5 Les styles

Il est possible de changer la façon d'afficher certains éléments de la barre d'outils : afficher que le texte, seulement les icônes ou les deux. Voila la fonction qui permet de contrôler cela :

*void gtk_toolbar_set_style(GtkToolbar *toolbar, GtkToolbarStyle? style) ;*

Le paramètre *style* peut prendre 4 valeurs différentes :

- GTK_TOOLBAR_ICONS pour n'afficher que l'icône ;
- GTK_TOOLBAR_TEXT pour n'afficher que le texte ;
- GTK_TOOLBAR_BOTH pour afficher le texte en dessous de l'icône (valeur par défaut) ;
- GTK_TOOLBAR_BOTH_HORIZ pour afficher le texte à côté de l'icône.

Et pour finir, cette fonction permet de connaître le style de la barre d'outil :

*GtkToolbarStyle? gtk_toolbar_get_style(GtkToolbar *toolbar);*

2.6 La taille des icônes

Enfin pour terminer, GTK+ nous offre la possibilité de modifier la taille des icônes de la barre d'outils avec cette fonction :

*void gtk_toolbar_set_icon_size(GtkToolbar *toolbar, GtkIconSize icon_size);*

Le paramètre *icon_size* est du type GtkIconSize que nous avons déjà rencontré dans le chapitre sur les images. Nous allons tout de même rappeler les différentes valeurs possibles qui sont les suivantes :

- GTK_ICON_SIZE_MENU ;
- GTK_ICON_SIZE_SMALL_TOOLBAR ;
- GTK_ICON_SIZE_LARGE_TOOLBAR (valeur par défaut) ;
- GTK_ICON_SIZE_BUTTON ;
- GTK_ICON_SIZE_DND ;
- GTK_ICON_SIZE_DIALOG.

Et bien sûr, pour connaître la taille des icônes, nous avons la fonction :

*GtkIconSize gtk_toolbar_get_icon_size(GtkToolbar *toolbar);*

2.7 Programme exemple

Dans cet exemple, nous allons juste créer une barre d'outils dans laquelle nous allons ajouter quelques boutons dont deux d'entre eux permettront de changer l'orientation de la barre d'outil.

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

void OnChangeOrientation(GtkWidget *widget, gpointer data);
```

```
static GtkWidget *pToolbar = NULL;

int main(int argc, char **argv)
{
   GtkWidget *pWindow;
   GtkWidget *pVBox;

   gtk_init(&argc, &argv);

   /* Creation de la fenetre */
   pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(pWindow), "GtkToolbar");
   gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
   g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

   pVBox = gtk_vbox_new(FALSE, 0);
   gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

   /* Creation de la barre d'outils */
   pToolbar = gtk_toolbar_new();
   gtk_box_pack_start(GTK_BOX(pVBox), pToolbar, FALSE, FALSE, 0);

   /* Creation a partir de stock */
   gtk_toolbar_insert_stock(GTK_TOOLBAR(pToolbar),
      GTK_STOCK_NEW,
      "Nouveau",
      NULL,
      NULL,
      NULL,
      -1);
   gtk_toolbar_insert_stock(GTK_TOOLBAR(pToolbar),
      GTK_STOCK_OPEN,
      "Ouvrir",
      NULL,
      NULL,
      NULL,
      -1);
   gtk_toolbar_insert_stock(GTK_TOOLBAR(pToolbar),
      GTK_STOCK_SAVE,
      "Enregistrer",
      NULL,
      NULL,
      NULL,
      -1);
   gtk_toolbar_insert_stock(GTK_TOOLBAR(pToolbar),
      GTK_STOCK_QUIT,
      "Fermer",
      NULL,
      G_CALLBACK(gtk_main_quit),
      NULL,
      -1);

   /* Insertion d'un espace */
   gtk_toolbar_append_space(GTK_TOOLBAR(pToolbar));

   /* Creation a partir de stock */
   gtk_toolbar_insert_stock(GTK_TOOLBAR(pToolbar),
      GTK_STOCK_GO_FORWARD,
      "Horizontale",
```

```
      NULL,
      G_CALLBACK(OnChangeOrientation),
      GINT_TO_POINTER(GTK_ORIENTATION_HORIZONTAL),
      -1);
   gtk_toolbar_insert_stock(GTK_TOOLBAR(pToolbar),
      GTK_STOCK_GO_DOWN,
      "Verticale",
      NULL,
      G_CALLBACK(OnChangeOrientation),
      GINT_TO_POINTER(GTK_ORIENTATION_VERTICAL),
      -1);

   /* Modification de la taille des icones */
   gtk_toolbar_set_icon_size(GTK_TOOLBAR(pToolbar),
      GTK_ICON_SIZE_BUTTON);
   /* Affichage uniquement des icones */
   gtk_toolbar_set_style(GTK_TOOLBAR(pToolbar),
      GTK_TOOLBAR_ICONS);

   gtk_widget_show_all(pWindow);

   gtk_main();

   return EXIT_SUCCESS;
}

void OnChangeOrientation(GtkWidget *widget, gpointer data)
{
   /* Modification de l'orientation */
   gtk_toolbar_set_orientation(GTK_TOOLBAR(pToolbar),
      GPOINTER_TO_INT(data));
}
```

Résultat :


GtkToolbar
Horizontale

La barre d'état

1. Présentation

La barre d'état peut avoir plusieurs utilités. Tout d'abord lorsqu'un programme est en train d'effectuer une action, elle peut servir à signaler à l'utilisateur ce que le programme fait. Elle peut aussi servir à expliquer l'utilité d'un bouton (ou d'un élément du menu).

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBox -> GtkHBox -> GtkStatusbar

2. Utilisation de base

2.1 Création

La création de ce widget est aussi simple que pour tous les autres widgets. Il suffit d'utiliser cette fonction :

```
GtkWidget *gtk_statusbar_new(void);
```

2.2 Identification

Regardons tout d'abord comment fonctionne une GtkStatusBar. Les messages sont mis dans une pile, l'élément le plus haut de la pile étant affiché. Mais avant de pouvoir ajouter un message dans la pile, il faut que Gtk+ sache qui a envoyé le message. En effet pour la barre d'état, nous avons la possibilité de définir quelle partie (ou quel module) du programme a envoyé un message.

Donc avant de pouvoir insérer des messages, il faut créer un ou plusieurs identifiant suivant le cas avec cette fonction :

```
guint gtk_statusbar_get_context_id (GtkStatusbar *statusbar, const gchar
*context_description);
```

Cette fonction va automatiquement ajouter un contexte, et renvoyer la valeur correspondante. Par la suite cet identifiant est à utiliser pour ajouter ou enlever des éléments à la pile.

2.3 Ajout de message

Pour ajouter un élément, il n'y a qu'une seule fonction :

```
guint gtk_statusbar_push (GtkStatusbar *statusbar, guint context_id, const gchar *text);
```

Le paramètre *statusbar* est bien sûr la barre d'état dans laquelle nous voulons ajouter un message, le paramètre *context_id* la valeur correspondante au contexte qui a été créé et *text* le texte qu'il faut afficher dans la barre d'état.

Après l'appel de cette fonction, le message est automatiquement ajouté en haut de la pile et est affiché dans la barre d'état. La valeur de retour correspond à l'identifiant du message dans la pile qui peut être utile pour sa suppression.

2.4 Suppression d'un message

Pour supprimer un message de la pile, nous avons cette fois deux fonctions différentes:

```
void gtk_statusbar_pop (GtkStatusbar *statusbar, guint context_id);
void gtk_statusbar_remove (GtkStatusbar *statusbar, guint context_id, guint message_id);
```

La première fonction enlève l'élément le plus haut placé dans la pile provenant de la partie du programme ayant l'identifiant *context_id*. Alors que la deuxième fonction enlève l'élément en fonction de son identifiant et celui de son contexte.

2.5 Programme exemple

Nous allons construire une application comportant deux boutons et barre d'état. Le but de cette application et d'afficher un message dans la barre d'état lorsque la souris survole un des boutons.

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

static GtkWidget *pStatusBar;

void OnExitBtnEnter(GtkWidget *pButton, gpointer iContextId);
void OnAboutBtnEnter(GtkWidget *pButton, gpointer iContextId);
void ClearStatus(GtkWidget *pButton, gpointer iContextId);

int main(int argc, char **argv)
{
    GtkWidget* pWindow;
    GtkWidget* pVBox;
    GtkWidget* pExitButton;
    GtkWidget *pAboutButton;
    guint iContextId1;
    guint iContextId2;

    gtk_init(&argc, &argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkStatusbar");
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    pVBox=gtk_vbox_new(FALSE,5);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    pExitButton=gtk_button_new_with_label("Quitter");
    gtk_box_pack_start(GTK_BOX(pVBox), pExitButton, TRUE, FALSE, 5);
    g_signal_connect(G_OBJECT(pExitButton), "clicked",
G_CALLBACK(gtk_main_quit), NULL);

    pAboutButton=gtk_button_new_with_label("A propos...");
    gtk_box_pack_start(GTK_BOX(pVBox), pAboutButton, TRUE, FALSE, 5);
```

```
    /* Creation de la barre d'etat */
    pStatusBar = gtk_statusbar_new();

    gtk_box_pack_end(GTK_BOX(pVBox), pStatusBar, FALSE, FALSE, 0);

    /* Creation des contextes */
    iContextId1 = gtk_statusbar_get_context_id(GTK_STATUSBAR(pStatusBar),
"ExitMsg");
    iContextId2 = gtk_statusbar_get_context_id(GTK_STATUSBAR(pStatusBar),
"AboutMsg");

    g_signal_connect(G_OBJECT(pExitButton), "enter", G_CALLBACK(OnExitBtnEnter),
        GINT_TO_POINTER(iContextId1));
    g_signal_connect(G_OBJECT(pAboutButton), "enter",
G_CALLBACK(OnAboutBtnEnter),
        GINT_TO_POINTER(iContextId2));
    g_signal_connect(G_OBJECT(pExitButton), "leave", G_CALLBACK(ClearStatus),
        GINT_TO_POINTER(iContextId1));
    g_signal_connect(G_OBJECT(pAboutButton), "leave", G_CALLBACK(ClearStatus),
        GINT_TO_POINTER(iContextId2));

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}

void OnExitBtnEnter(GtkWidget *pButton, gpointer iContextId)
{
    /* Ajout d'un message */
    gtk_statusbar_push (GTK_STATUSBAR (pStatusBar), GPOINTER_TO_INT(iContextId),
"Quitter l'application");
}

void OnAboutBtnEnter(GtkWidget *pButton, gpointer iContextId)
{
    /* Ajout d'un message */
    gtk_statusbar_push (GTK_STATUSBAR (pStatusBar), GPOINTER_TO_INT(iContextId),
"Informations");
}

void ClearStatus(GtkWidget *pButton, gpointer iContextId)
{
    /* Suppression d'un message */
    gtk_statusbar_pop(GTK_STATUSBAR(pStatusBar), GPOINTER_TO_INT(iContextId));
}
```

Résultat :


GtkStatusbar
Quitter
A propos...
Quitter l'application


GtkStatusbar
Quitter
A propos...
Informations

La sélection de valeurs numériques

1. Présentation

Nous allons dans ce chapitre étudier trois widgets différents qui vont nous permettre de définir une valeur numérique sans avoir à la saisir au clavier. Le premier widget que nous allons étudier est GtkScrollbar qui se dérive en deux widgets différents, un qui est vertical et l'autre horizontal. Puis nous étudierons le widget GtkScale qui lui aussi se décline en un widget vertical et un horizontal. Et nous terminerons avec le widget GtkSpinButton.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkEntry -> GtkSpinButton

GObject -> GtkObject -> GtkWidget -> GtkRange -> GtkScale -> GtkHScale

GObject -> GtkObject -> GtkWidget -> GtkRange -> GtkScale -> GtkVScale

GObject -> GtkObject -> GtkWidget -> GtkRange -> GtkScrollbar -> GtkHScrollbar

GObject -> GtkObject -> GtkWidget -> GtkRange -> GtkScrollbar -> GtkVScrollbar

2. Pré requis

Ces trois widgets permettent de choisir une valeur numérique à l'intérieur d'une fourchette de valeurs bien définies. Le fonctionnement similaire de ces trois widgets est rendu possible par l'utilisation de l'objet GtkAdjustment?. Nous allons donc tout d'abord acquérir les notions nécessaires sur ce widget pour comprendre les widgets de sélection.

2.1 Structure de l'objet GtkAdjustment?.

```
struct _GtkAdjustment
{
GtkObject parent_instance;
gdouble lower;
gdouble upper;
gdouble value;
gdouble step_increment;
gdouble page_increment;
gdouble page_size;
}
```

La propriété *parent_instance* nous dit tout simplement que l'objet GtkAdjustment? dérive directement de GtkObject. Les valeurs *lower* et *upper* déterminent respectivement la valeur minimale et la valeur maximale que peut prendre le widget qui utilise cet objet. La valeur *value* donne la valeur actuelle telle qu'elle est définie par le widget. Les valeurs *step_increment* et *page_increment* définissent de combien value sera modifiée à chaque fois que l'utilisateur demande d'augmenter ou de diminuer la valeur. Nous verrons la différence entre ces deux valeurs avec l'étude des widgets GtkScrollbar et GtkScale. La dernière propriété *page_size* définit la taille d'une

page, mais là aussi cela deviendra plus clair avec l'étude des widgets qui va suivre.

2.2 Les fonctions de bases

La première fonction de base est bien sûr la fonction qui permet de créer un tel objet :

```
GtkObject* gtk_adjustment_new (gdouble value, gdouble lower, gdouble upper, gdouble step_increment, gdouble page_increment, gdouble page_size);
```

Nous retrouvons ici tous les paramètres définis précédemment.

Ensuite pour modifier ou récupérer la valeur actuelle de l'objet, il y a ces deux fonctions :

```
void gtk_adjustment_set_value(GtkAdjustment? *adjustment, gdouble value);
gdouble gtk_adjustment_get_value(GtkAdjustment? *adjustment);
```

Bien entendu la première fonction change le widget utilisateur pour que sa position représente la valeur value, et la deuxième fonction récupère cette valeur.

Et pour terminer voici la fonction qui permet de modifier les valeurs limites de l'objet :

```
void gtk_adjustment_clamp_page (GtkAdjustment? *adjustment, gdouble lower, gdouble upper);
```

Nous avons vu l'essentiel des fonctions de l'objet GtkAdjustment?, mais en règle générale, il n'est pas nécessaire de les utiliser, car les widgets qui nécessitent cet objet, ont leurs propres fonctions d'accès et de modification des paramètres.

3. Le widget GtkScrollbar

Comme nous l'avons dit dans la présentation, ce widget se décline en deux variantes différentes :

- une horizontale avec le widget GtkHScrollbar ;
- une verticale avec le widget GtkVScrollbar.

Mais nous allons voir que mise à part les fonctions de création, toutes les autres opérations se font avec les mêmes fonctions quel que soit le widget utilisé.

3.1 Création

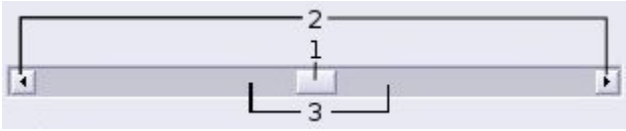
Voici donc les deux fonctions de création disponibles :

```
GtkWidget* gtk_vscrollbar_new(GtkAdjustment? *adjustment);
GtkWidget* gtk_hscrollbar_new(GtkAdjustment? *adjustment);
```

La première fonction crée donc une GtkVScrollbar et la seconde une GtkHScrollbar. De plus, nous voyons qu'il faut obligatoirement un GtkAdjustment? pour définir les paramètres du widget. Il va donc falloir dans un premier temps créer un GtkAdjustment? et définir ses valeurs.

Pour ce qui est des valeurs *lower*, *upper*, et *value*, il n'y a pas de problème. Par contre pour les valeurs *step_increment* et *page_increment*, il faut connaître un peu le fonctionnement de ce widget. Regardons d'abord à quoi ressemble un widget

GtkHScrollbar.



La zone 1 est en fait l'afficheur de la valeur actuelle prise par la GtkScrollbar. Elle permet aussi de sélectionner une valeur en le déplaçant à l'aide de la souris. Les deux zones 2 permettent de modifier la valeur de la zone 1. Un clic de souris sur une de ces zones modifiera la valeur de plus ou moins la valeur de *step_increment*. Les zones 3 ont le même objectif que les zones 2 mais cette fois avec une modification de valeur de *page_increment*.

Il reste encore le dernier paramètre à configurer. Ce dernier, *page_size*, détermine la taille de la zone 1. Si, par exemple le widget GtkScrollbar a une valeur minimale de 0 et une valeur maximale de 100, et nous définissons *page_size* comme ayant une valeur de 50, la zone 1 du widget aura une longueur égale à la moitié de celle du widget. Le dernier point sur ce paramètre est qu'il détermine aussi la valeur qu'il faut donner à *upper*. Ce widget fonctionnant en utilisant des pages, la valeur *upper* n'est jamais atteinte. Si l'on veut pouvoir choisir une valeur entre 0 et 100 avec un page_size égale à 50, il faudra mettre upper à 150 soit le *upper* théorique plus *page_size*.

En effet, ce widget n'a pas comme vocation première de permettre de sélectionner des valeurs numériques mais plutôt l'affichage d'autre widget dont la hauteur (ou la largeur) est supérieure à la zone d'affichage. Ainsi lorsque la barre de sélection est complètement à gauche, la valeur de la GtkScrollbar est bien de 0, mais elle considère qu'elle affiche une zone allant de 0 à *page_size* (50 dans notre exemple). De ce fait, si la barre de sélection est complètement à droite la valeur de la GtkScrollbar sera égale à upper-page_size et considèrera qu'elle affiche une zone allant de upper-page_size à upper (50 à 100 dans notre exemple). Pour conclure, il faut toujours donner à *upper* la valeur maximale que nous souhaitons pouvoir sélectionner plus la valeur de page_size.

Nous savons maintenant comment configurer le GtkAdjustment? afin de créer proprement une GtkScrollbar.

3.2 La gestion des valeurs

Nous allons maintenant traiter les fonctions de gestion d'une GtkScrollbar. Pour cela, nous n'allons pas utiliser les widgets GtkHScrollbar et GtkVScrollbar car il n'y a pour chaque widget qu'une seule fonction de création, ni le widget GtkScrollbar car il ne possède aucune fonction, mais le widget GtkRange dont dérive toute cette panoplie.

Tout d'abord pour fixer la valeur d'une GtkScrollbar, nous avons à notre disposition la fonction suivante :

```
void gtk_range_set_value(GtkRange *range, gdouble value);
```

Bien sûr le premier paramètre est la GtkScrollbar dont nous voulons fixer la valeur. Il faut pour cela utiliser la macro de conversion GTK_RANGE(). Le deuxième paramètre est la valeur que nous voulons donner à notre GtkScrollbar.

A l'inverse, la fonction nous permettant de connaître la valeur de la GtkScrollbar est :

*gdouble gtk_range_get_value(GtkRange *range);*

Ensuite le widget GtkRange nous offre la possibilité de modifier les bornes du widget ainsi que les différents pas avec ces deux fonctions :

*void gtk_range_set_increments (GtkRange *range, gdouble step, gdouble page);*
*void gtk_range_set_range (GtkRange *range, gdouble min, gdouble max);*

La première fonction modifie les pas de modification. Les paramètres *step* et *page* correspondent aux paramètres *step_increment* et *page_increment* du GtkAdjustment? associé au widget.

La deuxième fonction permet de modifié les valeurs minimale et maximale que le widget peut prendre. Là encore, il faut penser au problème de la valeur maximale pour bien configurer max.

Une autre possibilité pour modifier ces paramètres est de créer un nouveau GtkAdjustment? et de l'associer à la GtkScrollbar avec cette fonction :

*void gtk_range_set_adjustment (GtkRange *range, GtkAdjustment? *adjustment);*

3.3 Programme exemple

Notre premier exemple de ce chapitre va nous permettre de sélection les valeurs RGB que nous souhaitons affecter. La couleur formée par ces valeurs ne sera pas affichée car nous nous contenterons d'afficher ces valeurs dans un label.

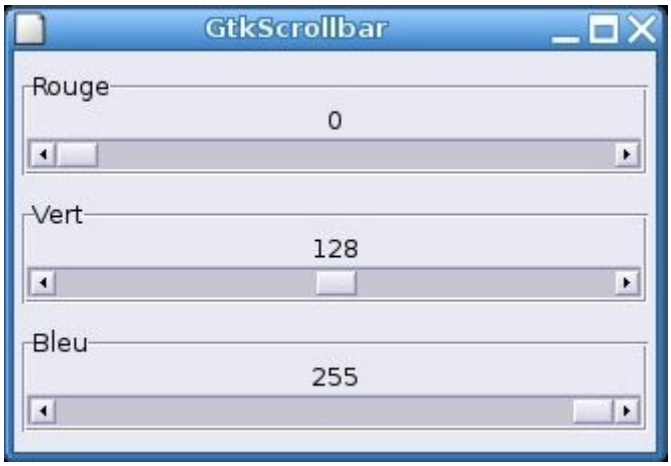
Pour modifier les valeurs des labels en fonction des modifications des GtkScrollbar, nous allons capturer le signal "value-changed" de chaque GtkScrollbar.

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

void OnScrollbarChange(GtkWidget *pWidget, gpointer data);

int main(int argc,char **argv)
{
   GtkWidget* pWindow;
   GtkWidget *pMainVBox;
   GtkWidget *pFrame;
   GtkWidget *pColorBox;
   GtkWidget *pLabel;
   GtkWidget *pScrollbar;
   GtkObject *Adjust;

   gtk_init(&argc,&argv);

   pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(pWindow), "GtkScrollbar");
   gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
   gtk_container_set_border_width(GTK_CONTAINER(pWindow), 4);

   pMainVBox = gtk_vbox_new(TRUE, 0);
   gtk_container_add(GTK_CONTAINER(pWindow), pMainVBox);

   pFrame = gtk_frame_new("Rouge");
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);
   pColorBox = gtk_vbox_new(TRUE, 0);
```

```
   gtk_container_add(GTK_CONTAINER(pFrame), pColorBox);

   /* Label d'affichage de valeur R*/
   pLabel = gtk_label_new("0");
   gtk_box_pack_start(GTK_BOX(pColorBox), pLabel, FALSE, FALSE, 0);
   /* Creation d un GtkAdjustment */
   Adjust = gtk_adjustment_new(0, 0, 256, 1, 10, 1);
   /* Creation d une scrollbar horizontale*/
   pScrollbar = gtk_hscrollbar_new(GTK_ADJUSTMENT(Adjust));
   gtk_box_pack_start(GTK_BOX(pColorBox), pScrollbar, TRUE, TRUE, 0);
   /* Connexion du signal pour modification de l affichage */
   g_signal_connect(G_OBJECT(pScrollbar), "value-changed",
      G_CALLBACK(OnScrollbarChange), (GtkWidget*)pLabel);

   /* Idem pour G */
   pFrame = gtk_frame_new("Vert");
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);
   pColorBox = gtk_vbox_new(TRUE, 0);
   gtk_container_add(GTK_CONTAINER(pFrame), pColorBox);

   pLabel = gtk_label_new("0");
   gtk_box_pack_start(GTK_BOX(pColorBox), pLabel, FALSE, FALSE, 0);
   Adjust = gtk_adjustment_new(0, 0, 256, 1, 10, 1);
   pScrollbar = gtk_hscrollbar_new(GTK_ADJUSTMENT(Adjust));
   gtk_box_pack_start(GTK_BOX(pColorBox), pScrollbar, TRUE, TRUE, 0);
   g_signal_connect(G_OBJECT(pScrollbar), "value-changed",
      G_CALLBACK(OnScrollbarChange), (GtkWidget*)pLabel);

   /* Idem pour B */
   pFrame = gtk_frame_new("Bleu");
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);
   pColorBox = gtk_vbox_new(TRUE, 0);
   gtk_container_add(GTK_CONTAINER(pFrame), pColorBox);

   pLabel = gtk_label_new("0");
   gtk_box_pack_start(GTK_BOX(pColorBox), pLabel, FALSE, FALSE, 0);
   Adjust = gtk_adjustment_new(0, 0, 256, 1, 10, 1);
   pScrollbar = gtk_hscrollbar_new(GTK_ADJUSTMENT(Adjust));
   gtk_box_pack_start(GTK_BOX(pColorBox), pScrollbar, TRUE, TRUE, 0);
   g_signal_connect(G_OBJECT(pScrollbar), "value-changed",
      G_CALLBACK(OnScrollbarChange), (GtkWidget*)pLabel);

   gtk_widget_show_all(pWindow);

   g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

   gtk_main();

   return EXIT_SUCCESS;
}

void OnScrollbarChange(GtkWidget *pWidget, gpointer data)
{
   gchar* sLabel;
   gint iValue;

   /* Recuperation de la valeur de la scrollbar */
   iValue = gtk_range_get_value(GTK_RANGE(pWidget));
   /* Creation du nouveau label */
   sLabel = g_strdup_printf("%d", iValue);
```

```
    /* Modification du label */
    gtk_label_set_text(GTK_LABEL(data), sLabel);
    /* Liberation memoire */
    g_free(sLabel);
}
```

Résultat



4. Le widget GtkScale

Les widgets GtkHScale et GtkVScale ont un fonctionnement quasi identique aux widget GtkHScrollbar et GtkVScrollbar mais sont plus simple d'utilisation. Un des avantages de ce widget est qu'il offre la possibilité d'afficher tout seul sa valeur.

4.1 Création

Comme pour les GtkScrollbar, nous pouvons créer des GtkScale à l'aide de GtkAdjustment? avec ces fonctions :

```
GtkWidget* gtk_vscale_new(GtkAdjustment? *adjustment);
GtkWidget* gtk_hscale_new (GtkAdjustment? *adjustment);
```

La première fonction crée donc un GtkScale vertical alors la deuxième un GtkScale horizontal.
Mais pour ne pas avoir à créer un objet GtkAdjustment?, les créateurs de GTK+ ont pensé à nous fournir une fonction qui n'en a pas besoin :

```
GtkWidget* gtk_vscale_new_with_range (gdouble min, gdouble max, gdouble step);
GtkWidget* gtk_hscale_new_with_range (gdouble min, gdouble max, gdouble step);
```

Avec ces quatre fonctions, la signification de *step* (ou *step_increment* si l'on utilise un GtkAdjustment?) est un peu différente car le widget GtkScale n'a pas de bouton fléché comme GtkScrollbar. Cette valeur est utilisée pour un déplacement à l'aide des touches fléchées lorsque le widget a le focus. De plus, si nous n'utilisons pas de GtkAdjustment?, la valeur de *page_increment* est égale à dix fois celle de *step*.

4.2 La gestion des valeurs.

Pour récupérer ou fixer les différentes valeurs d'un widget GtkScale il faut, comme pour le widget GtkScrollbar, utiliser les fonctions du widget GtkRange qui ont été décrites plus haut. Nous n'allons donc pas revenir dessus.

4.3 Affichage de la valeur sous forme de label.

Comme nous l'avons dit dans la présentation de ce widget, il s'occupe seul d'afficher et de mettre à jour un label qui affiche la valeur du widget. Par défaut, le label est affiché au-dessus du GtkScale. Etudions les quelques fonctions qui nous sont fournis pour gérer cet affichage.

```
void gtk_scale_set_draw_value (GtkScale *scale, gboolean draw_value);
gboolean gtk_scale_get_draw_value (GtkScale *scale);
```

La première fonction nous permet de décider si nous voulons qu'un label s'affiche ou pas. Pour cela il suffit de mettre le paramètre *draw_value* à TRUE si nous souhaitons qu'il s'affiche ou FALSE si nous ne le souhaitons pas. En ce qui concerne le premier paramètre, il faut utiliser la macro de conversion GTK_SCALE().

La deuxième fonction permet de savoir si le label est affiché ou pas.

Ensuite nous avons la possibilité de gérer le nombre de chiffre après la virgule qui sont affichés :

```
void gtk_scale_set_digits (GtkScale *scale, gint digits);
gint gtk_scale_get_digits (GtkScale *scale);
```

Dans la première fonction, *digit* est bien sûr le nombre de chiffre après la virgule à afficher.

Et pour terminer, nous allons voir comment gérer la position du label par rapport à la barre de sélection :

```
void gtk_scale_set_value_pos (GtkScale *scale, GtkPositionType? pos);
GtkPositionType? gtk_scale_get_value_pos (GtkScale *scale);
```

Le paramètre *pos* (de la première fonction) peut prendre une de ces quatre valeurs :

- GTK_POS_LEFT pour le mettre à gauche ;
- GTK_POS_RIGHT pour le mettre à droite ;
- GTK_POS_TOP pour le mettre au-dessus ;
- GTK_POS_BOTTOM pour le mettre en dessous.

4.4 Exemple.

Nous allons reprendre l'exemple précédent que nous allons transformer pour l'utilisation de widget GtkHScale ce qui le rendra plus simple. Cependant, pour utiliser les fonctions du widget GtkScale, le premier GtkHScale aura un label au dessus, le second aura un label en dessous et le troisième pas du tout.

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

```
int main(int argc,char **argv)
{
   GtkWidget* pWindow;
   GtkWidget *pMainVBox;
   GtkWidget *pFrame;
   GtkWidget *pLabel;
   GtkWidget *pScale;

   gtk_init(&argc,&argv);

   pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(pWindow), "GtkScale");
   gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
   gtk_container_set_border_width(GTK_CONTAINER(pWindow), 4);

   pMainVBox = gtk_vbox_new(TRUE, 0);
   gtk_container_add(GTK_CONTAINER(pWindow), pMainVBox);

   pFrame = gtk_frame_new("Rouge");
   /* Creation du widget GtkHScale */
   pScale = gtk_hscale_new_with_range(0, 255, 1);
   gtk_container_add(GTK_CONTAINER(pFrame), pScale);
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);

   pFrame = gtk_frame_new("Vert");
   pScale = gtk_hscale_new_with_range(0, 255, 1);
   /* Position du label en dessous */
   gtk_scale_set_value_pos(GTK_SCALE(pScale), GTK_POS_BOTTOM);
   gtk_container_add(GTK_CONTAINER(pFrame), pScale);
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);

   pFrame = gtk_frame_new("Bleu");
   pScale = gtk_hscale_new_with_range(0, 255, 1);
   /* Pas d'affichage du label */
   gtk_scale_set_draw_value(GTK_SCALE(pScale), FALSE);
   gtk_container_add(GTK_CONTAINER(pFrame), pScale);
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);

   gtk_widget_show_all(pWindow);

   g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

   gtk_main();

   return EXIT_SUCCESS;
}
```

Résultat

5. Le widget GtkSpinButton

Ce widget est un peu différent des deux précédents dans le sens ou il s'agit en fait d'un widget GtkEntry et de deux GtkButton assemblés pour former un tout. De plus il permet de sélectionner une valeur soit en utilisant les boutons soit en entrant directement la valeur au clavier. Mais voyons tout cela en détail.

5.1 Création

Il existe là aussi deux fonctions de créations :

```
GtkWidget* gtk_spin_button_new (GtkAdjustment? *adjustment, gdouble climb_rate, guint digits);
GtkWidget* gtk_spin_button_new_with_range (gdouble min, gdouble max, gdouble step);
```

La première fonction permet de créer un GtkSpinButton à partir d'un GtkAdjustment?, mais il faut cependant préciser le paramètre *climb_rate* qui correspond au paramètre *step_increment* du GtkAdjustment? et *digits* qui représente le nombre de chiffre après la virgule de la valeur que nous allons pouvoir sélectionner.

La deuxième fonction permet de se passer de GtkAdjustment? car nous fixons les valeurs minimales et maximales (paramètres *min* et *max*) ainsi que la valeur de modification avec le paramètre *step*. En ce qui concerne le nombre de chiffres après la virgule, il sera identique à celui de *step*.

Pour ce widget, le clavier permet aussi de modifier sa valeur. Les touches fléchées HAUT et BAS permettent de modifier la valeur de *step_increment* (dans le cas de l'utilisation d'un GtkAdjustment?) ou de *step*. De plus les touches PAGE_UP et PAGE_DOWN permettent de modifier la valeur de *page_increment* avec une création utilisant un GtkAdjustment? ou de step*10 dans l'autre cas.

5.2 Gestion des valeurs

Voyons tout d'abord comment récupérer ou modifier la valeur en cours d'un GtkSpinButton à l'aide d'une de ces trois fonctions :

*void gtk_spin_button_set_value (GtkSpinButton *spin_button, gdouble value);*
*gdouble gtk_spin_button_get_value (GtkSpinButton *spin_button);*
*gint gtk_spin_button_get_value_as_int(GtkSpinButton *spin_button);*

La première fonction permet donc de modifier la valeur d'un GtkSpinButton. Il faut pour le premier paramètre utiliser la macro de conversion GTK_SPIN_BUTTON().

La deuxième fonction permet de récupérer la valeur d'un GtkSpinButton tout simplement.

La dernière fonction a le même but que la deuxième mais celle-ci renvoie un entier alors que la deuxième renvoie un double.

Nous pouvons nous occuper du nombre de chiffre après la virgule avec ces deux fonctions :

*guint gtk_spin_button_get_digits (GtkSpinButton *spin_button);*
*void gtk_spin_button_set_digits (GtkSpinButton *spin_button, guint digits);*

La première fonction permettant de connaître le nombre de chiffre après la virgule alors que la seconde permet de le modifier.

Et pour terminer voici les fonctions qui permettent de modifier les bornes des valeurs :

*void gtk_spin_button_get_range (GtkSpinButton *spin_button, gdouble *min, gdouble *max);*
*void gtk_spin_button_set_range (GtkSpinButton *spin_button, gdouble min, gdouble max);*

Comme d'habitude, la première fonction permet de connaître les bornes du GtkSpinButton et la seconde permet de les modifier.

Mais bien sûr, tout cela peut se modifier à l'aide des GtkAdjustment? avec ces deux fonctions :

GtkAdjustment? gtk_spin_button_get_adjustment(GtkSpinButton *spin_button);*
*void gtk_spin_button_set_adjustment (GtkSpinButton *spin_button, GtkAdjustment? *adjustment);*

5.3 Exemple

Le programme exemple sera identique aux deux premiers, c'est à dire qu'il va permettre de sélectionner une valeur RGB à l'aide de trois widget GtkSpinButton.

5.4 Programme exemple

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

int main(int argc,char **argv)
 {
   GtkWidget* pWindow;
   GtkWidget *pMainVBox;
   GtkWidget *pFrame;
   GtkWidget *pSpin;

   gtk_init(&argc,&argv);

   pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(pWindow), "GtkSpinButton");
```

```
   gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
   gtk_container_set_border_width(GTK_CONTAINER(pWindow), 4);

   pMainVBox = gtk_vbox_new(TRUE, 0);
   gtk_container_add(GTK_CONTAINER(pWindow), pMainVBox);

   pFrame = gtk_frame_new("Rouge");
   /* Creation du widget GtkSpinButton */
   pSpin = gtk_spin_button_new_with_range(0, 255, 1);
   gtk_container_add(GTK_CONTAINER(pFrame), pSpin);
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);

   pFrame = gtk_frame_new("Vert");
   pSpin = gtk_spin_button_new_with_range(0, 255, 1);
   gtk_container_add(GTK_CONTAINER(pFrame), pSpin);
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);

   pFrame = gtk_frame_new("Bleu");
   pSpin = gtk_spin_button_new_with_range(0, 255, 1);
   gtk_container_add(GTK_CONTAINER(pFrame), pSpin);
   gtk_box_pack_start(GTK_BOX(pMainVBox), pFrame, FALSE, FALSE, 0);

   gtk_widget_show_all(pWindow);

   g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

   gtk_main();

   return EXIT_SUCCESS;
}
```

Résultat

La barre de progression

1. Présentation

Parfois un programme doit faire de gros calculs, et il est utile d'avertir l'utilisateur sur l'avancement du calcul afin qu'il soit au courant et ne pense pas que le programme est bloqué. Le widget GtkProgressBar permet donc d'afficher la progression d'une opération quelconque.

1.2 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkProgress? -> GtkProgressBar

2. Utilisation de base

2.1 Création

Ce widget étant comme tous les autres widgets de GTK+ sa fonction de création est très simple :

```
GtkWidget* gtk_progress_bar_new(void)
```

2.2 Modification de la position

Pour fixer la position de la GtkProgressBar, la fonction est :

```
void gtk_progress_bar_set_fraction(GtkProgressBar *pbar, gdouble fraction);
```

Le paramètre *pbar* est la barre de progression et le paramètre *fraction* est la valeur que l'on veut donner à la barre de progression. Cette valeur doit être comprise entre 0.0 (0%) et 1.0 (100%).

Pour connaître la position de la GtkProgressBar, la fonction est tout simplement :

```
gdouble gtk_progress_bar_get_fraction(GtkProgressBar *pbar);
```

La valeur de retour est bien entendu comprise entre 0.0 et 1.0.

2.3 Orientation de la barre de progression

La barre de progression peut être soit verticale, soit horizontale. De plus pour chaque position, l'on peut définir le sens de progression, et tout cela avec cette fonction :

```
void gtk_progress_bar_set_orientation(GtkProgressBar *pbar, GtkProgressBarOrientation orientation);
```

Le paramètre *orientation* peut prendre quatre valeurs possible :

- GTK_PROGRESS_LEFT_TO_RIGHT : la barre de progression évolue de gauche à droite ;

- GTK_PROGRESS_RIGHT_TO_LEFT : la barre de progression évolue de droite à gauche ;
- GTK_PROGRESS_BOTTOM_TO_TOP : la barre de progression évolue de bas en haut ;
- GTK_PROGRESS_TOP_TO_BOTTOM : la barre de progression évolue de haut en bas.

Au contraire pour connaître son sens de progression, il faut utiliser la fonction :

*GtkProgressBarOrientation gtk_progress_bar_get_orientation(GtkProgressBar *pbar);*

2.4 Programme exemple

Notre premier exemple, très simple, sera fait d'une fenêtre (bien entendu) avec à l'intérieur une barre de progression (qui évoluera de droite à gauche) et un bouton qui permettra de faire avancer la barre de progression de 10%. Et lorsque la barre de progression sera à 100%, nous la réinitialiserons à 0%.

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

void OnButton(GtkWidget *pWidget, gpointer data);

int main(int argc,char **argv)
{
   GtkWidget* pWindow;
   GtkWidget *pMainVBox;
   GtkWidget *pProgress;
   GtkWidget *pButton;

   gtk_init(&argc,&argv);

   pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(pWindow), "GtkProgressBar");
   gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
   gtk_container_set_border_width(GTK_CONTAINER(pWindow), 4);

   pMainVBox = gtk_vbox_new(TRUE, 0);
   gtk_container_add(GTK_CONTAINER(pWindow), pMainVBox);

   /* Creation de la barre de progression */
   pProgress = gtk_progress_bar_new();
   gtk_progress_bar_set_orientation(GTK_PROGRESS_BAR(pProgress),
GTK_PROGRESS_RIGHT_TO_LEFT);
   gtk_box_pack_start(GTK_BOX(pMainVBox), pProgress, TRUE, FALSE, 0);

   pButton = gtk_button_new_with_label("Ajouter 10%");
   gtk_box_pack_start(GTK_BOX(pMainVBox), pButton, TRUE, FALSE, 0);
   g_signal_connect_swapped(G_OBJECT(pButton), "clicked", G_CALLBACK(OnButton),
pProgress);

   gtk_widget_show_all(pWindow);

   g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

   gtk_main();
```

```
   return EXIT_SUCCESS;
}

void OnButton(GtkWidget *pWidget, gpointer data)
{
   gdouble dFraction;

   /* Recuperation de la valeur de la barre de progression */
   dFraction = gtk_progress_bar_get_fraction(GTK_PROGRESS_BAR(pWidget));

   dFraction += 0.1;

   if(dFraction > 1.0)
      dFraction = 0.0;

   /* Modification de la valeur de la barre de progression */
   gtk_progress_bar_set_fraction(GTK_PROGRESS_BAR(pWidget), dFraction);
}
```

Résultat

3. Utilisation dans une boucle

3.1 Problème

Si nous modifions légèrement le programme précédent pour insérer une boucle qui fera avancer automatiquement la barre de progression (par exemple une boucle de 2000 itérations) cela ne fonctionnera pas. La seule raison pour laquelle cela ne fonctionnera pas est que GTK+ ne reprend pas la main car le programme reste dans la boucle for alors GTK+ ne peut pas mettre à jour les widgets affichés.

3.2 Solution

La solution, est de dire clairement à GTK+ qu'il doit remettre à jour grâce à la fonction suivante :

```
gboolean gtk_main_iteration (void);
```

Cette fonction permet de faire, comme son nom l'indique, une seule itération. Donc à

ce moment là GTK+ va reprendre la main puis la rendre aussitôt, si bien qu'il pourra mettre à jour les widgets. Il suffit donc d'ajouter cette fonction après *gtk_progress_bar_set_fraction* pour faire fonctionner correctement notre programme.

Ce problème étant réglé, nous allons faire face à un deuxième problème, ou plutôt pseudo-problème. En général, lorsque le programme fait un gros calcul, l'application doit être "bloquer" pendant ce temps. Donc tant que le traitement n'est pas fini, il faut éviter que l'utilisateur ne puisse change des données. Prenons le cas d'un correcteur d'orthographe, disons qu'il le fasse automatiquement. Pendant qu'il vérifie l'orthographe il serai bête que l'utilisateur puisse modifier le texte, ce qui fausserai alors toute la correction. Pourquoi cet exemple? Et bien le fait de rendre la main a GTK+ lui donne le pouvoir de traiter d'autres évènements, comme un clic de sourie et autres. Donc à tout moment pendant ce calcul l'utilisateur peut modifier quelque chose (ici pas grand choses si ce n'est que re-cliquer sur le bouton de démarrage de la boucle), donc il faut pouvoir l'empêcher de faire une quelconque action.

Nous allons pouvoir bloquer l'utilisateur à l'aide de ces deux fonctions :

*void gtk_grab_add (GtkWidget *widget);*
*void gtk_grab_remove (GtkWidget *widget);*

Nous allons faire une description rapide des ces deux fonctions à l'aide d'un exemple. Prenons le cas ou l'utilisateur fait dans une application (de dessin par exemple) une sélection par glissement, quand il quitte la zone pour atterrir à coté voir en dehors de la fenêtre, la sélection continue, et bien c'est parce que l'application se focalise sur la fenêtre de sélection, c'est un peut ce que fait gtk_grab_add. Nous lui donnons un widget et seuls les évènements de ce widget seront traiter par GTK+, et cela tant que gtk_grab_remove n'a pas été invoqué. Si bien que quand l'utilisateur fait une sélection et qu'il passe au-dessus d'un autre widget, il est ignorer.

Voilà maintenant nous avons tout pour que pendant la boucle la barre de progression soit remise à jour et sans que l'utilisateur ne puisse cliquer ailleurs.

3.3 Programme exemple

Nous réutilisons donc l'exemple précédant en modifiant la fonction du bouton qui sert maintenant à démarrer la progression de la barre qui sera effectuer dans la fonction callback.

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

void OnButton(GtkWidget *pWidget, gpointer data);

int main(int argc,char **argv)
{
   GtkWidget* pWindow;
   GtkWidget *pMainVBox;
   GtkWidget *pProgress;
   GtkWidget *pButton;

   gtk_init(&argc,&argv);

   pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_title(GTK_WINDOW(pWindow), "GtkProgressBar");
   gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
   gtk_container_set_border_width(GTK_CONTAINER(pWindow), 4);
```

```
   pMainVBox = gtk_vbox_new(TRUE, 0);
   gtk_container_add(GTK_CONTAINER(pWindow), pMainVBox);

   /* Creation de la barre de progression */
   pProgress = gtk_progress_bar_new();
   gtk_box_pack_start(GTK_BOX(pMainVBox), pProgress, TRUE, FALSE, 0);

   pButton = gtk_button_new_from_stock(GTK_STOCK_REFRESH);
   gtk_box_pack_start(GTK_BOX(pMainVBox), pButton, TRUE, FALSE, 0);
   g_signal_connect_swapped(G_OBJECT(pButton), "clicked", G_CALLBACK(OnButton),
pProgress);

   gtk_widget_show_all(pWindow);

   g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

   gtk_main();

   return EXIT_SUCCESS;
}

void OnButton(GtkWidget *pWidget, gpointer data)
{
   gdouble dFraction;
   gint i;
   gint iTotal = 2000;

   /* Initialisation */
   gtk_progress_bar_set_fraction(GTK_PROGRESS_BAR(pWidget), 0.0);

   /* Ici on grab sur la barre de progression pour 2 raisons : */
   /* - cela evite a GTK+ de regarder tous les evenements ce qui rend plus
rapide */
   /* l'utilisation de gtk_main_iteration() */
   /* - on empeche toute action de l'utilisateur */
   gtk_grab_add(pWidget);

   for(i = 0 ; i < iTotal ; ++i)
   {
      dFraction = (gdouble)i / (gdouble)iTotal;

      /* Modification de la valeur de la barre de progression */
      gtk_progress_bar_set_fraction(GTK_PROGRESS_BAR(pWidget), dFraction);

      /* On donne la main a GTK+ */
      gtk_main_iteration ();
   }

   /* On supprime le grab sur la barre de progression */
   gtk_grab_remove(pWidget);
}
```

Résultat


GtkProgressBar
Rafraîchir

La sélection des fichiers

1. Présentation

Pour qu'un utilisateur puisse sélectionner un fichier pour des actions comme la sauvegarde ou l'ouverture, nous allons utiliser les widgets GtkFileChooserDialog et GtkFileChooser.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkWindow -> GtkDialog -> GtkFileChooserDialog

GInterface? -> GtkFileChooser

2. Utilisation de base

2.1 Création

Nous n'allons étudier qu'une seule fonction de création sur les deux que propose GTK+ :

```
GtkWidget* gtk_file_chooser_dialog_new(const gchar *title, GtkWindow *parent,
GtkFileChooserAction? action, const gchar *first_button_text, ...);
```

Avec cette fonction nous allons donc ainsi créer la boîte de dialogue qui va nous demander de sélectionner un nom de fichier (ou un répertoire) pour effectuer une opération de sauvegarde ou autre.

Le premier paramètre *title* est le titre de la fenêtre, le deuxième paramètre *parent* permet de définir la fenêtre parente.

Le troisième paramètre *action* permet de définir le comportement de la boîte de dialogue. Ce paramètre peut prendre quatre valeurs différentes :

- ***GTK_FILE_CHOOSER_ACTION_OPEN*** pour ouvrir un fichier ;
- ***GTK_FILE_CHOOSER_ACTION_SAVE*** pour sauver un fichier ;
- ***GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER*** pour sélectionner un répertoire ;
- ***GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER*** pour pouvoir créer ou renommer un répertoire.

Ensuite le fonctionnement est le même que pour les boîte de dialogue classique, il faut ajouter les boutons présents dans la boîte de sélection ainsi que les valeurs de retour, le tout se terminant par un NULL pour valider la liste des paramètres de la fonction.

2.2 Affichage

Comme nous venons de le voir, le widget GtkFileChooserDialog dérive de GtkDialog, c'est donc tout naturellement que l'affichage de la boîte de sélection va se faire avec cette fonction :

*gint gtk_dialog_run (GtkDialog *dialog);*

La valeur de retour sera très importante pour savoir, si l'utilisateur à sélectionné un nom de fichier ou pas.

2.3 Récupération du chemin

Encore une fois, nous allons utiliser les propriétés du widget GtkDialog. Si la valeur de retour de *gtk_dialog_run* est *GTK_RESPONSE_OK*, alors un nom de fichier a été sélectionné, et il faut donc récupérer sa valeur avec la fonction suivante :

gchar gtk_file_chooser_get_filename(GtkFileChooser *chooser);*

La paramètre *chooser* correspond en fait la boîte de sélection précédemment créée (widget GtkFileChooserDialog).

Attention : GTK+ alloue de la mémoire pour la valeur de retour de cette fonction. Cette mémoire doit être libérée avec un *g_free* lorsque cette valeur n'est plus utile.

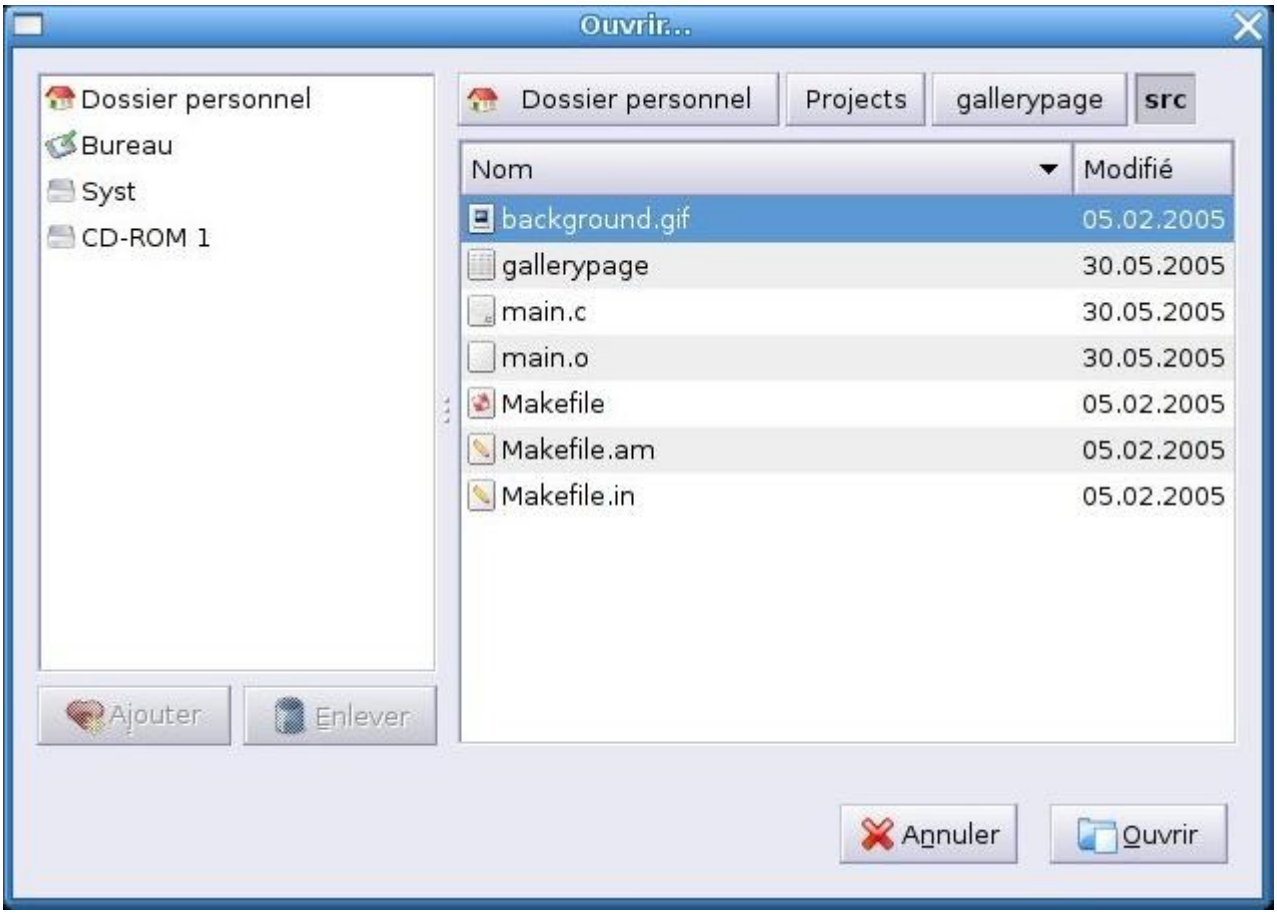
2.4 Programme exemple

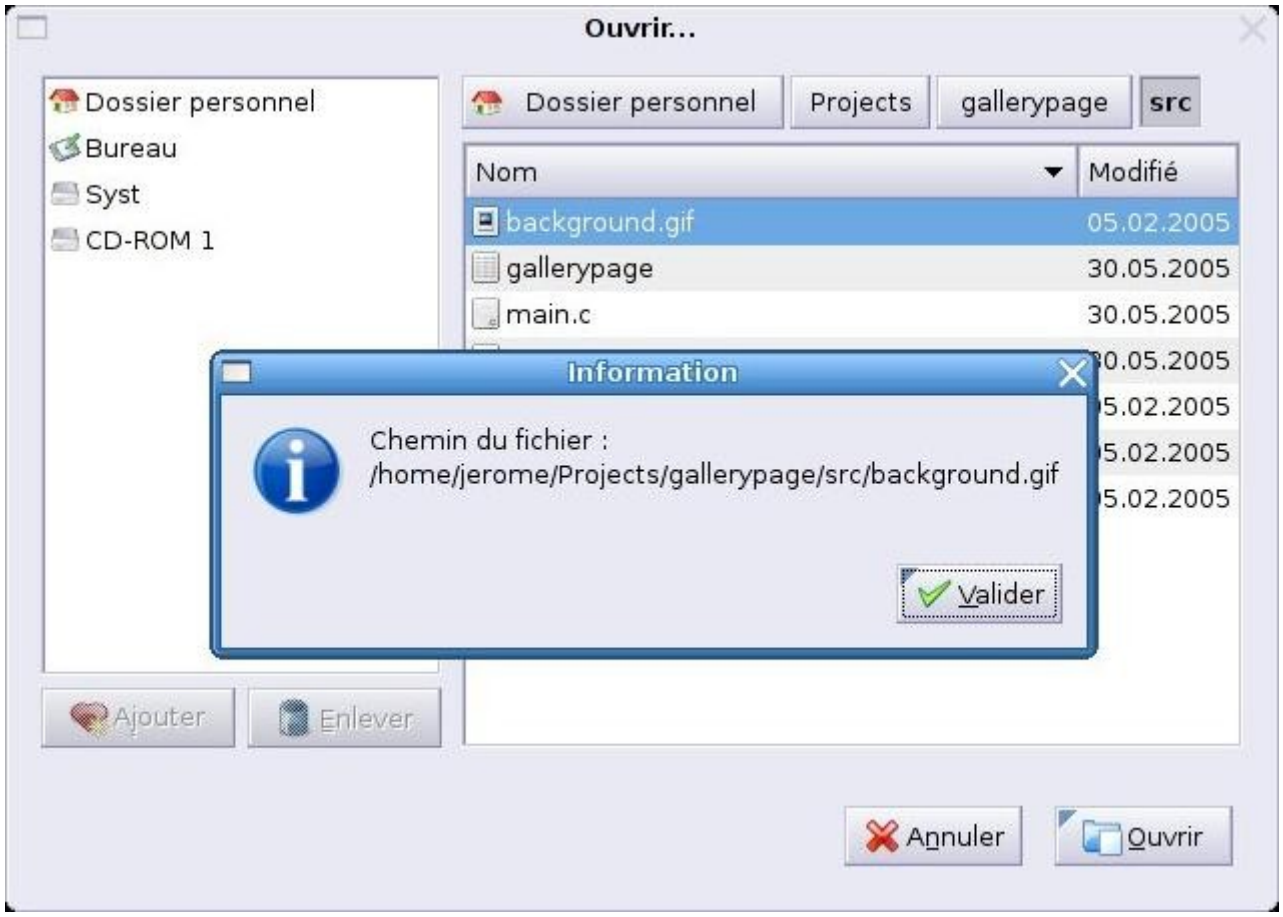
Nous allons créer une fenêtre avec un bouton nous permettant d'ouvrir une GtkFileChooserDialog. Une fois le fichier sélectionné, nous affichons son chemin dans une boîte de dialogue.

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

void OnButton(GtkWidget *pWidget, gpointer data);

int main(int argc,char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pButton;

    gtk_init(&argc,&argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkFileSelection");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);

    pButton=gtk_button_new_with_mnemonic("_Explorer...");
    gtk_container_add(GTK_CONTAINER(pWindow), pButton);

    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);
    g_signal_connect(G_OBJECT(pButton), "clicked", G_CALLBACK(OnButton), NULL);

    gtk_widget_show_all(pWindow);
    gtk_main();
    return EXIT_SUCCESS;
}

void OnButton(GtkWidget *pWidget, gpointer data)
{
    GtkWidget *pFileSelection;
    GtkWidget *pDialog;
    GtkWidget *pParent;
    gchar *sChemin;
```

```
    pParent = GTK_WIDGET(data);

    /* Creation de la fenetre de selection */
    pFileSelection = gtk_file_chooser_dialog_new("Ouvrir...",
        GTK_WINDOW(pParent),
        GTK_FILE_CHOOSER_ACTION_OPEN,
        GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
        GTK_STOCK_OPEN, GTK_RESPONSE_OK,
        NULL);
    /* On limite les actions a cette fenetre */
    gtk_window_set_modal(GTK_WINDOW(pFileSelection), TRUE);

    /* Affichage fenetre */
    switch(gtk_dialog_run(GTK_DIALOG(pFileSelection)))
    {
        case GTK_RESPONSE_OK:
            /* Recuperation du chemin */
            sChemin =
gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(pFileSelection));
            pDialog = gtk_message_dialog_new(GTK_WINDOW(pFileSelection),
                GTK_DIALOG_MODAL,
                GTK_MESSAGE_INFO,
                GTK_BUTTONS_OK,
                "Chemin du fichier :\n%s", sChemin);
            gtk_dialog_run(GTK_DIALOG(pDialog));
            gtk_widget_destroy(pDialog);
            g_free(sChemin);
            break;
        default:
            break;
    }
    gtk_widget_destroy(pFileSelection);
}
```

Résultats


Ouvrir...
Dossier personnel
Bureau
Syst
CD-ROM 1
Ajouter
Enlever
Dossier personnel
Projects
gallerypage
src
Nom
Modifié
background.gif 05.02.2005
gallerypage 30.05.2005
main.c 30.05.2005
main.o 30.05.2005
Makefile 05.02.2005
Makefile.am 05.02.2005
Makefile.in 05.02.2005
Annuler
Ouvrir


Ouvrir...
Dossier personnel
Bureau
Syst
CD-ROM 1
Dossier personnel
Projects
gallerypage
src
Nom
Modifié
background.gif
05.02.2005
gallerypage
30.05.2005
main.c
30.05.2005
Information
Chemin du fichier :
/home/jerome/Projects/gallerypage/src/background.gif
Valider
Ajouter
Enlever
Annuler
Ouvrir

Les fenêtres avec barres de défilement

1. Introduction

Nous allons cette fois voir un widget qui permet d'ajouter des barres de défilement au widget qui sera son enfant. Il s'agit d'un widget container qui s'appelle GtkScrolledWindow.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkBin? -> GtkScrolledWindow

2. Utilisation de base

2.1 Création

Comme vous pouvez vous y attendre, la fonction de création est comme toutes les autres, c'est à dire prévisible.

```
GtkWidget* gtk_scrolled_window_new(GtkAdjustment? *hadjustment, GtkAdjustment? *vadjustment);
```

Les paramètres *hadjustment* et *vadjustment* permettent de définir les propriétés des barres de défilement. L'objet GtkAdjustment? permet dans notre cas, de définir (entre autres) la valeur minimale que peut prendre la barre de défilement et le pas de celle-ci.

Mais à moins de connaître exactement tous ces paramètres, il vaut mieux laisser faire Gtk, et mettre *hadjustment* et *vadjustment* à NULL.

2.2 Ajout du widget enfant.

Pour cela, il y a deux cas différents. Si par exemple le widget enfant a la capacité d'avoir ses propres barres de défilements, il suffit d'utiliser la fonction ***gtk_container_add***.

Par contre, dans le cas ou le widget ne possède pas cette capacité, il faudra utiliser cette fonction :

```
void gtk_scrolled_window_add_with_viewport(GtkScrolledWindow *scrolled_window, GtkWidget *child);
```

Avec cette fonction, Gtk va créer ce qul s'appelle un viewport (qui peut avoir des barres de défilement) et va insérer l'enfant dans ce viewport.

Seuls les widgets GtkTreeView, GtkTextView, et GtkLayout? ont la possibilité d'avoir des barres défilements. Pour les autres widgets (comme les GtkHBox, GtkVBox, ...) il faudra utiliser la deuxième fonction.

2.3 Affichage des barres de défilement.

Par défaut, les barres de défilement s'affichent automatiquement (que ce soit la barre horizontale ou verticale). Il se peut que, par exemple, vous n'ayez besoin que de la barre verticale. Dans ce cas vous ne voulez sûrement pas avoir de barre de défilement horizontale. La solution à votre problème est cette fonction :

```
void gtk_scrolled_window_set_policy (GtkScrolledWindow *scrolled_window, GtkPolicyType? hscroll_policy, GtkPolicyType? vscroll_policy);
```

Les paramètres *hscroll_policy* et *vscroll_policy* peuvent prendre trois paramètres :

- GTK_POLICY_ALWAYS : la barre de défilement est toujours affichée.
- GTK_POLICY_AUTOMATIC : la barre de défilement n'est affichée que si elle est utile.
- GTK_POLICY_NEVER : la barre de défilement n'est jamais affichée.

Au contraire, pour savoir si elles sont affichées ou pas, il y a la fonction :

```
void gtk_scrolled_window_get_policy (GtkScrolledWindow *scrolled_window, GtkPolicyType? *hscroll_policy, GtkPolicyType? *vscroll_policy);
```

2.4 Construction du programme exemple.

Notre exemple est très simple. Il consiste uniquement à afficher 10 labels les uns au-dessus des autres dans une fenêtre qui n'excèdera pas une taille de 320x200. Voyons à quoi ressemble notre fenêtre si elle n'utilise pas les barres de défilement.

Et voila le résultat, la fenêtre ne respecte pas la condition TAILLE = 320x200. Il faut utiliser le widget GtkScrolledWindow.

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

```
int main(int argc, char* argv[])
{
        GtkWidget* pWindow;
        GtkWidget* pBox;
        GtkWidget *pScrollbar;
        int i;

        gtk_init(&argc, &argv);

        pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
        gtk_window_set_title(GTK_WINDOW(pWindow), "GtkScrolledWindow");
        g_signal_connect(G_OBJECT(pWindow),"destroy",G_CALLBACK(gtk_main_quit),0
);

        pScrollbar = gtk_scrolled_window_new(NULL, NULL);
        gtk_container_add(GTK_CONTAINER(pWindow),pScrollbar);

        pBox=gtk_vbox_new(FALSE,5);

        gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW(pScrollbar),
pBox);

        gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(pScrollbar),
GTK_POLICY_NEVER, GTK_POLICY_ALWAYS);

        for(i = 1 ; i <= 10 ; ++i)
        {
                GtkWidget *pLabel;
                char sTexte[10];

                g_sprintf(sTexte, "Label %d", i);

                pLabel = gtk_label_new(sTexte);

                gtk_box_pack_start(GTK_BOX(pBox), pLabel, FALSE, FALSE, 5);
        }

        gtk_widget_show_all(pWindow);

        gtk_main();

        return EXIT_SUCCESS;
}
```

Résultats


GtkScrolledWindow
Label 1
Label 2
Label 3
Label 4
Label 5
Label 6


GtkScrolledWindow
Label 5
Label 6
Label 7
Label 8
Label 9
Label 10

Les zones de texte

1. Présentation

Nous avons appris à saisir une ligne tapée par l'utilisateur grâce à GtkEntry mais comment fait-on pour saisir plusieurs lignes (comme le font les traitements de texte, etc.). Et bien nous utilisons non pas un mais trois widgets !

- GtkTextBuffer : ce widget non-graphique sert à stocker les données. C'est tout simplement un buffer !
- GtkTextView : ce widget sert à afficher le contenu d'un GtkTextBuffer. Il s'occupe de tout ce qui est graphique : il gère les évènements de l'utilisateur, les insertions de caractères, etc.
- GtkTextIter : ce widget non-graphique désigne une position dans GtkTextBuffer. Il permet de manipuler GtkTextBuffer.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkTextView
GObject -> GtkTextBuffer

GtkTextIter

2. Utilisation de base

2.1 Création

Pour créer un GtkTextView, nous avons la possibilité d'utiliser une des ces deux fonctions :

```
GtkWidget* gtk_text_view_new(void);
GtkWidget* gtk_text_view_new_with_buffer(GtkTextBuffer *buffer);
```

La première fonction crée un GtkTextView vierge, c'est à dire qu'elle s'occupe de la création du GtkTextBuffer qui sera utilisée pour l'affichage. La deuxième fonction permet de définir à l'avance le GtkTextBuffer qui sera utilisé (qu'il soit vide ou non) pour l'affichage à l'aide du paramètre *buffer*.

Maintenant que notre GtkTextView est créée, il ne reste plus qu'à l'insérer dans un container pour l'afficher.

2.2 Accéder au contenu de GtkTextBuffer

Dans un premier temps, avant de pouvoir accéder au contenu d'un GtkTextBuffer, il faut avoir une référence sur cet élément. Si lors de la création du GtkTextView nous avons spécifié un GtkTextBuffer déjà existant, il n'y a aucun problème, par contre si nous avons utiliser la première fonction de création, il va falloir récupérer l'adresse du GtkTextBuffer avec cette fonction :

GtkTextBuffer gtk_text_view_get_buffer(GtkTextView *text_view);*

Maintenant nous pouvons récupérer le texte du GtkTextBuffer avec cette fonction :

gchar gtk_text_buffer_get_text(GtkTextBuffer *buffer, const GtkTextIter *start, const GtkTextIter *end, gboolean include_hidden_chars);*

Le premier paramètre *buffer* n'est rien d'autre que le buffer du GtkTextView sur lequel l'on est en train de travail. Les paramètres *start* et *end* marquent respectivement le début et la fin du texte que l'on veut récupérer. Et le dernier paramètre *include_hidden_chars* permet de définir si l'on veut ou pas récupérer les caractères invisibles (par exemple les GtkTextTag? ou les GtkTextMark?).

Comme nous pouvons le voir d'après la fonction précédente, nous allons utiliser des GtkTextIter pour délimiter le texte que nous voulons récupérer. Les GtkTextIter donne en fait la position donnée, à un moment donné, du curseur, d'un caractère ou autre dans le buffer. Il faut cependant faire attention que dès que le GtkTextBuffer est modifié, un GtkTextIter qui pointé à un endroit précis du GtkTextBuffer n'est plus valide même si sa position n'a théoriquement pas changé.

Nous allons terminer par étudier deux fonctions permettant d'obtenir le GtkTextIter marquant le début et la fin d'un GtkTextBuffer :

*void gtk_text_buffer_get_start_iter (GtkTextBuffer *buffer, GtkTextIter *iter);*
*void gtk_text_buffer_get_end_iter(GtkTextBuffer *buffer, GtkTextIter *iter);*

La première fonction permet d'obtenir le GtkTextIter marquant le début du GtkTextBuffer et la deuxième fonction celui permettant d'en marquer la fin. La paramètre *iter* est ici une valeur de retour.

2.3 Programme exemple

Nous allons créer une fenêtre contenant un GtkTextView et un GtkButton. Lorsque l'on clique sur le GtkButton, l'intégralité du GtkTextView est alors affiché dans une GtkDialog.

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

void OnShow(GtkWidget *pWidget, gpointer *data);

int main(int argc, char* argv[])
{
        GtkWidget* pWindow;
        GtkWidget* pBox;
        GtkWidget* pTextView;
        GtkWidget* pButton;

        gtk_init(&argc, &argv);
        pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
        gtk_window_set_title(GTK_WINDOW(pWindow), "GtkTextView");
        g_signal_connect(G_OBJECT(pWindow), "destroy",
G_CALLBACK(gtk_main_quit), NULL);

        pBox = gtk_vbox_new(FALSE, 5);
        gtk_container_add(GTK_CONTAINER(pWindow), pBox);
```

```
        pTextView = gtk_text_view_new();
        gtk_box_pack_start(GTK_BOX(pBox), pTextView, TRUE, TRUE, 0);

        pButton=gtk_button_new_with_label("Afficher");
        gtk_box_pack_start(GTK_BOX(pBox),pButton,FALSE,FALSE,0);
        g_signal_connect(G_OBJECT(pButton), "clicked", G_CALLBACK(OnShow),
(gpointer) pTextView);

        gtk_widget_show_all(pWindow);

        gtk_main();

        return EXIT_SUCCESS;
}

void OnShow(GtkWidget *pWidget, gpointer *data)
{
        GtkWidget *pDialog;
        GtkWidget *pTextView;
        GtkTextBuffer* pTextBuffer;
        GtkTextIter iStart;
        GtkTextIter iEnd;
        gchar* sBuffer;

        pTextView = GTK_WIDGET(data);

        /* On recupere le buffer */
        pTextBuffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(pTextView));
        /* On recupere l'origine du buffer */
        gtk_text_buffer_get_start_iter(pTextBuffer, &iStart);
        /* On recupere la fin du buffer */
        gtk_text_buffer_get_end_iter(pTextBuffer, &iEnd);

        /* On copie le contenu du buffer dans une variable */
        sBuffer = gtk_text_buffer_get_text(pTextBuffer, &iStart, &iEnd, TRUE);

        /* On affiche le texte dans une boite de dialogue. */
        pDialog = gtk_message_dialog_new(NULL,
                GTK_DIALOG_MODAL,
                GTK_MESSAGE_INFO,
                GTK_BUTTONS_OK,
                "Votre texte :\n%s", sBuffer);

        gtk_dialog_run(GTK_DIALOG(pDialog));
        gtk_widget_destroy(pDialog);

        /* On libere la memoire */
        g_free(sBuffer);
}
```

Résultat

3. Afficher le contenu d'un fichier

Maintenant, nous allons utiliser ce que nous avons vu dans les deux précédents chapitres afin de créer une application ouvrant un fichier et affichant son contenu.

3.1 Construction de l'application

Dans un premier temps, nous allons utiliser le widget GtkScrolledWindow pour y

insérer notre GtkTextView, puis nous allons demander à l'utilisateur de choisir un nom de fichier à l'aide du widget GtkFileChooser.

Une fois le nom du fichier connu, nous allons utiliser une fonction de la GLib pour ouvrir et lire le contenu du fichier. Cela ce fait à l'aide de cette unique fonction :

*gboolean g_file_get_contents(const gchar *filename, gchar **contents, gsize *length, GError error);*

Le paramètre *filename* est le nom du fichier à ouvrir. Tous les autres paramètres sont en fait des valeurs de retour. Le paramètre *contents* est la variable dans laquelle va être insérer le contenu du fichier, le paramètre *length* est la longueur en octets de la variable *contents*. Et pour finir le paramètre *error* est la variable dans laquelle on pourra obtenir des informations sur l'erreur qui a eu lieu. Cette fonction renvoie TRUE si tout c'est bien passé et FALSE sinon. Attention, le paramètre *contents* doit avoir la valeur NULL avant d'utiliser cette fonction, car elle alloue elle-même la mémoire pour cette variable. De ce fait lorsque la variable *contents* ne sera plus utilisée, il ne faut pas oublier de libérer la mémoire avec un ***g_free***.

Nous allons ensuite vider le contenu du GtkTextBuffer à l'aide de cette fonction :

*void gtk_text_buffer_delete(GtkTextBuffer *buffer, GtkTextIter *start, GtkTextIter *end);*

Et pour finir nous allons ajouter notre texte dans le buffer avec cette fonction :

*void gtk_text_buffer_insert(GtkTextBuffer *buffer, GtkTextIter *iter, const gchar *text, gint len);*

Le paramètre *text* est le texte que nous allons ajouter à notre GtkTextBuffer à la position donnée par le paramètre *iter*. Le paramètre *len* défini la longueur de *text*. Cette variable peut être mise à -1.

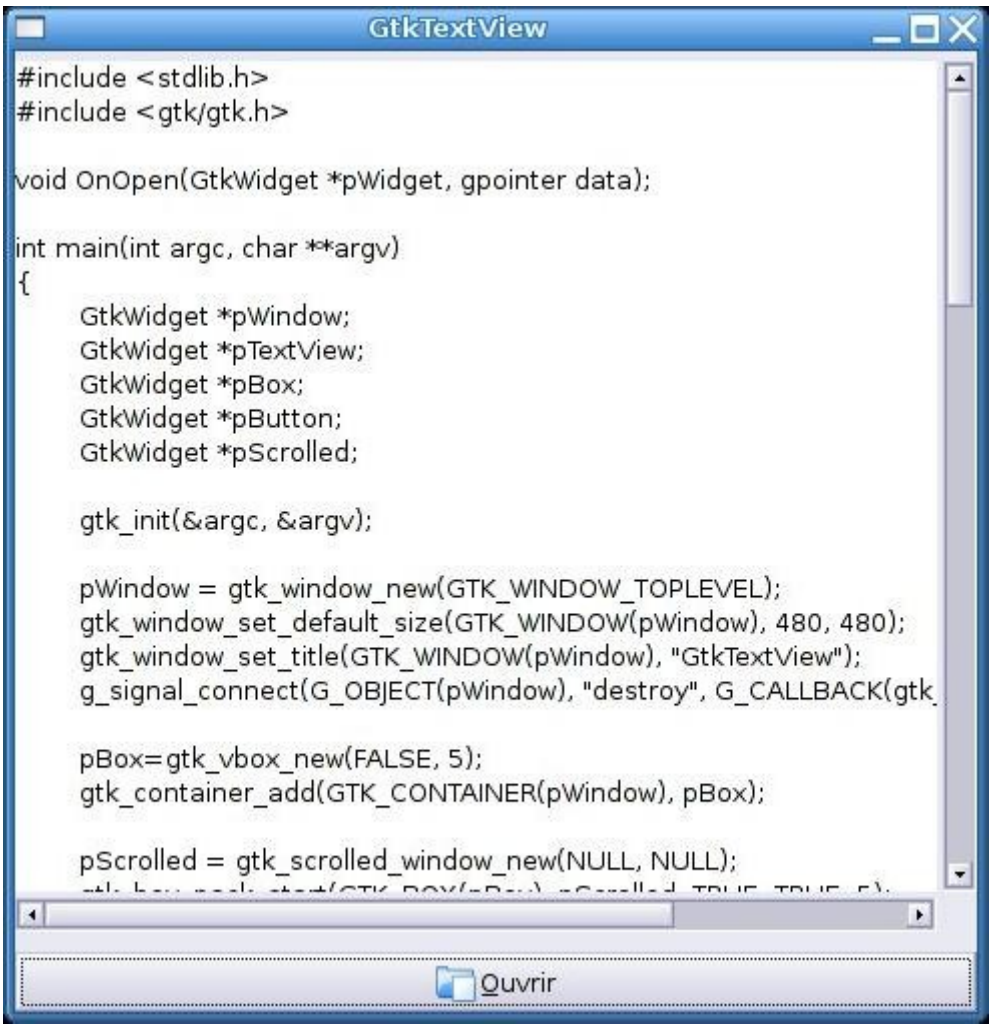
3.2 Programme exemple

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

void OnOpen(GtkWidget *pWidget, gpointer data);

int main(int argc, char **argv)
{
        GtkWidget *pWindow;
        GtkWidget *pTextView;
        GtkWidget *pBox;
        GtkWidget *pButton;
        GtkWidget *pScrolled;

        gtk_init(&argc, &argv);

        pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        gtk_window_set_default_size(GTK_WINDOW(pWindow), 480, 480);
        gtk_window_set_title(GTK_WINDOW(pWindow), "GtkTextView");
        g_signal_connect(G_OBJECT(pWindow), "destroy",
G_CALLBACK(gtk_main_quit), NULL);

        pBox=gtk_vbox_new(FALSE, 5);
        gtk_container_add(GTK_CONTAINER(pWindow), pBox);

        pScrolled = gtk_scrolled_window_new(NULL, NULL);
```

```
        gtk_box_pack_start(GTK_BOX(pBox), pScrolled, TRUE, TRUE, 5);

        pTextView=gtk_text_view_new();
        gtk_container_add(GTK_CONTAINER(pScrolled), pTextView);
        gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(pScrolled),
                GTK_POLICY_AUTOMATIC,
                GTK_POLICY_AUTOMATIC);

        pButton=gtk_button_new_from_stock(GTK_STOCK_OPEN);
        gtk_box_pack_start(GTK_BOX(pBox), pButton, FALSE, FALSE, 0);
        g_signal_connect(G_OBJECT(pButton), "clicked", G_CALLBACK(OnOpen),
(gpointer)pTextView);

        gtk_widget_show_all(pWindow);

        gtk_main();

        return EXIT_SUCCESS;
}


void OnOpen(GtkWidget *pWidget, gpointer data)
{
        GtkWidget *pFileChooser;
        GtkTextBuffer *pTextBuffer;
        GtkTextIter iStart, iEnd;
        gchar *sFile, *sBuffer;

        /* Creation de la fenetre de selection */
        pFileChooser = gtk_file_chooser_dialog_new("Ouvrir...",
                NULL,
                GTK_FILE_CHOOSER_ACTION_OPEN,
                GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
                GTK_STOCK_OPEN, GTK_RESPONSE_OK,
                NULL);
        /* On limite les actions a cette fenetre */
        gtk_window_set_modal(GTK_WINDOW(pFileChooser), TRUE);

        /* Affichage fenetre */
        switch(gtk_dialog_run(GTK_DIALOG(pFileChooser)))
        {
                case GTK_RESPONSE_OK:
                        /* Recuperation du chemin */
                        sFile =
gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(pFileChooser));
                        sBuffer = NULL;
                        g_file_get_contents(sFile, &sBuffer, NULL, NULL);
                        g_free(sFile);
                        break;
                default:
                        break;
        }
        gtk_widget_destroy(pFileChooser);

        /* Recuperation du buffer */
        pTextBuffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(data));
        /* Suppression des données du buffer */
        gtk_text_buffer_get_start_iter(pTextBuffer, &iStart);
        gtk_text_buffer_get_end_iter(pTextBuffer, &iEnd);
        gtk_text_buffer_delete(pTextBuffer, &iStart, &iEnd);
        /* Affichage du fichier */
```

```
        gtk_text_buffer_get_start_iter(pTextBuffer, &iStart);
        gtk_text_buffer_insert(pTextBuffer, &iStart, sBuffer, -1);
        g_free(sBuffer);
}
```

Résultat



Les pages à onglets

1. Présentation

Nous allons étudier dans ce chapitre les pages à onglets qui sont souvent utilisés dans les fenêtres de configuration. Pour cela nous allons utiliser le widget GtkNotebook qui dérive du widget GtkContainer.

1.1 Hiérarchie

GObject -> GtkObject -> GtkWidget -> GtkContainer -> GtkNotebook

2. Utilisation de base

2.1 Création

Pour ne pas changer la création du widget en lui-même est très simple :

```
GtkWidget* gtk_notebook_new(void);
```

Maintenant que notre widget est créé, nous allons lui ajouter des pages.

2.2 Insertion de pages

Nous avons là aussi trois fonctions qui restent très classiques :

```
void gtk_notebook_append_page (GtkNotebook *notebook, GtkWidget *child, GtkWidget
*tab_label);
void gtk_notebook_prepend_page (GtkNotebook *notebook, GtkWidget *child, GtkWidget
*tab_label);
void gtk_notebook_insert_page (GtkNotebook *notebook, GtkWidget *child, GtkWidget
*tab_label, gint position);
```

La première fonction ajoute une nouvelle page à onglet à la suite des autres, la deuxième fonction l'ajoute au début, et la dernière l'ajoute à une position particulière. Le premier paramètre, *notebook*, est bien sûr le GtkNotebook dans lequel nous voulons ajouter la page. Il faudra utiliser la macro de conversion GTK_NOTEBOOK(). Le second, *child*, est le widget qui sera insérer dans la nouvelle page, et le troisième (*tab_label*) le widget qui sera affiché dans l'onglet.

La troisième fonction demande un paramètre supplémentaire (*position*) qui est la position à laquelle il faut ajouter la page. Si la valeur n'est pas correcte (négative ou trop grande) la page sera ajoutée à la suite des autres.

2.3 Gestion des pages

Nous allons maintenant voir les fonctions qui permettent de connaître le nombre de page, la page en cours ainsi que d'autres fonctions.

Tout d'abord, la fonction suivante permet de connaître le nombre total de pages contenus dans le GtkNotebook :

```
gint gtk_notebook_get_n_pages(GtkNotebook *notebook);
```

Ensuite, pour connaître le numéro de la page courante, nous avons cette fonction :

```
gint gtk_notebook_get_current_page (GtkNotebook *notebook);
```

Par contre pour connaître le numéro d'une autre page que la courante, il faudra utiliser cette fonction :

```
gint gtk_notebook_page_num (GtkNotebook *notebook, GtkWidget *child);
```

Cette fonction permet d'obtenir le numéro de la page contenant le widget *child*, ce qui impose donc d'avoir garder une trace de ce widget pour pouvoir utiliser cette fonction. Avec un numéro de page, nous allons pouvoir récupérer un pointeur sur le widget qui est contenu dans la page avec cette fonction :

```
GtkWidget* gtk_notebook_get_nth_page (GtkNotebook *notebook, gint page_num);
```

Ou bien nous allons pouvoir tout simplement supprimer la page :

```
void gtk_notebook_remove_page (GtkNotebook *notebook, gint page_num);
```

Et pour terminer, voici trois fonctions de navigation :

```
void gtk_notebook_next_page (GtkNotebook *notebook);
void gtk_notebook_prev_page (GtkNotebook *notebook);
void gtk_notebook_set_current_page (GtkNotebook *notebook, gint page_num);
```

La première fonction passe de la page courante à la page suivante. Si la dernière page est déjà affichée, il ne se passera rien.

La deuxième fonction, elle, passe de la page courante à la page précédente. Bien entendu, si la page courante est déjà la première cette fonction n'aura aucun effet. La dernière fonction quant à elle passe directement à une page précise. Si la valeur de *page_num* est négative, nous passerons à la dernière page, par contre si cette valeur est trop grande, aucune action ne sera faite.

2.4 Gestion des labels

Nous allons maintenant voir comment modifier ou récupérer le label d'une page. Attention pour chacune de ces fonctions, il faut connaître le widget contenu dans la page.
Tout d'abord, pour récupérer le label de la page, il existe deux fonctions différentes :

```
G_CONST_RETURN gchar* gtk_notebook_get_tab_label_text(GtkNotebook *notebook,
GtkWidget *child);
GtkWidget* gtk_notebook_get_tab_label (GtkNotebook *notebook, GtkWidget *child);
```

La première fonction renvoie directement le label de la page sous la forme d'un gchar*. Cette fonction ne fonctionnera correctement uniquement si le label de la page est un GtkLabel, car comme nous l'avons lors de l'étude des fonctions d'insertion des pages, il est possible de définir le label comme autre chose qu'un GtkLabel. Dans ce cas, il faut utiliser la deuxième fonction qui elle renvoie le widget qui a été utilisé lors de la création.

Pour définir un nouveau label pour une page, il existe là aussi deux fonctions :

*void gtk_notebook_set_tab_label_text (GtkNotebook *notebook, GtkWidget *child, const gchar *tab_text);*
*void gtk_notebook_set_tab_label (GtkNotebook *notebook, GtkWidget *child, GtkWidget *tab_label);*

La première fonction permet de définir un label de type GtkLabel. Cette fonction s'occupe de la création du GtkLabel et de son insertion dans l'onglet de la page. La deuxième fonction permet d'insérer n'importe quel widget dans l'onglet, comme pour les fonctions de création. Dans les deux cas, si les paramètres *tab_text* ou *tab_label* sont définis comme NULL, le label de la page sera "Page y", y étant le numéro de la page.

2.5 Modification des propriétés des onglets

Nous allons maintenant voir comment modifier trois des propriétés d'un GtkNotebook. Tout d'abord voyons comment gérer la position des onglets :

*void gtk_notebook_set_tab_pos (GtkNotebook *notebook, GtkPositionType? pos);*
*GtkPositionType? gtk_notebook_get_tab_pos (GtkNotebook *notebook);*

La première fonction permet de modifier la position des onglets. C'est le paramètre pos qui définit sa position, et il doit prendre une des valeurs suivantes :

- GTK_POS_LEFT pour les mettre à gauche ;
- GTK_POS_RIGHT pour les mettre à droite ;
- GTK_POS_TOP pour les mettre en haut ;
- GTK_POS_BOTTOM pour les mettre en bas.

La deuxième fonction permet au contraire de connaître la position des onglets. La valeur de retour est obligatoirement une des quatre valeurs précédentes. Ensuite nous allons pouvoir définir si les onglets doivent s'afficher ou pas avec ces fonctions :

*void gtk_notebook_set_show_tabs (GtkNotebook *notebook, gboolean show_tabs);*
*gboolean gtk_notebook_get_show_tabs (GtkNotebook *notebook);*

Si nous voulons modifier ce paramètre, il faut utiliser la première fonction en mettant le paramètre *show_tabs* à TRUE pour les afficher et à FALSE dans le cas contraire. La deuxième fonction permet quant à elle de connaître l'état de cette propriété. Et enfin, pour terminer nous allons voir comment ajouter deux boutons de navigation à la fin des onglets, pour le cas où il y aurait trop d'onglets à afficher :

*void gtk_notebook_set_scrollable (GtkNotebook *notebook, gboolean scrollable);*
*gboolean gtk_notebook_get_scrollable (GtkNotebook *notebook);*

Si nous mettons le paramètre scrollable de la première fonction à TRUE, deux boutons de navigation apparaîtront et une partie seulement des onglets sera affichée. Pour accéder aux autres onglets il faudra utiliser ces nouveaux boutons. Bien entendu, la deuxième fonction permet de savoir si les boutons de navigation sont présents ou pas.

2.6 Programme exemple

Nous allons créer un programme qui insèrera un GtkNotebook de 8 pages dans une

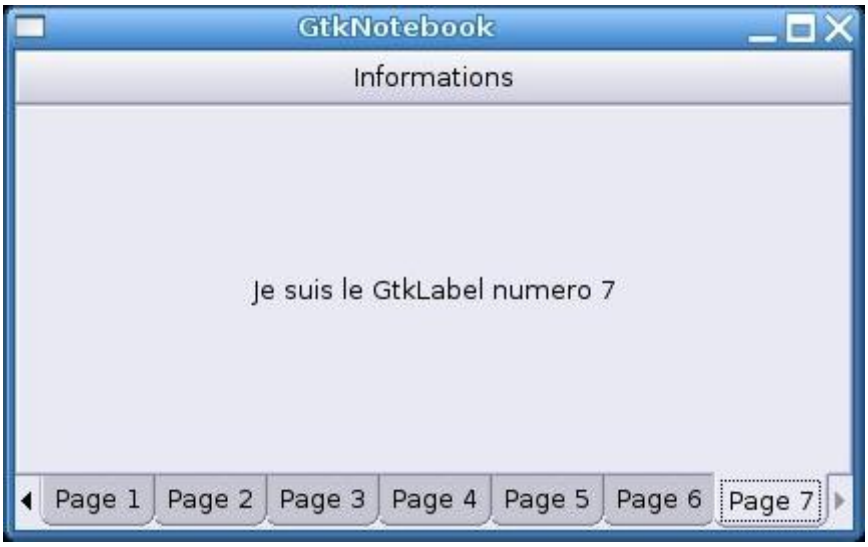
fenêtre. Chaque page contiendra uniquement un label "Je suis le GtkLabel numero x". Il y a aussi un bouton qui permettra d'afficher les informations de la page sélectionnée dans une boîte de dialogue.

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

void OnButton(GtkWidget *pButton, gpointer data);

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pNotebook;
    GtkWidget *pButton;
    gint i;

    gtk_init(&argc,&argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkNotebook");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    pVBox = gtk_vbox_new(FALSE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    pButton = gtk_button_new_with_label("Informations");
    gtk_box_pack_start(GTK_BOX(pVBox), pButton, FALSE, FALSE, 0);

    /* Creation du GtkNotebook */
    pNotebook = gtk_notebook_new();
    gtk_box_pack_start(GTK_BOX(pVBox), pNotebook, TRUE, TRUE, 0);
    /* Position des onglets : en bas */
    gtk_notebook_set_tab_pos(GTK_NOTEBOOK(pNotebook), GTK_POS_BOTTOM);
    /* Ajout des boutons de navigation */
    gtk_notebook_set_scrollable(GTK_NOTEBOOK(pNotebook), TRUE);

    for(i = 0 ; i < 8 ; ++i)
    {
        GtkWidget *pLabel;
        GtkWidget *pTabLabel;
        gchar *sLabel;
        gchar *sTabLabel;

        sLabel = g_strdup_printf("Je suis le GtkLabel numero %d", i);
        sTabLabel = g_strdup_printf("Page %d", i);

        /* Creation des differents GtkLabel */
        pLabel = gtk_label_new(sLabel);
        pTabLabel = gtk_label_new(sTabLabel);

        /* Insertion de la page */
        gtk_notebook_append_page(GTK_NOTEBOOK(pNotebook), pLabel, pTabLabel);

        g_free(sLabel);
        g_free(sTabLabel);
    }

    g_signal_connect(G_OBJECT(pButton), "clicked", G_CALLBACK(OnButton),
```

```
pNotebook);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}

void OnButton(GtkWidget *pButton, gpointer data)
{
    GtkWidget *pDialog;
    GtkWidget *pChild;
    gint iPageNum;
    const gchar *sLabel;
    const gchar *sTabLabel;
    gchar *sDialogText;

    /* Recuperation de la page active */
    iPageNum = gtk_notebook_get_current_page(GTK_NOTEBOOK(data));
    /* Recuperation du widget enfant */
    pChild = gtk_notebook_get_nth_page(GTK_NOTEBOOK(data), iPageNum);

    /* Recuperation du label */
    sLabel = gtk_label_get_text(GTK_LABEL(pChild));
    /* Recuperation du label de l'onglet */
    sTabLabel = gtk_notebook_get_tab_label_text(GTK_NOTEBOOK(data), pChild);

    /* Creation du label de la boite de dialogue */
    sDialogText = g_strdup_printf("C'est la page %d\n"
        "Le label est \"%s\"\n"
        "Le label de l'onglet est \"%s\"\n",
        iPageNum,
        sLabel,
        sTabLabel);

    pDialog = gtk_message_dialog_new (NULL,
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_INFO,
        GTK_BUTTONS_OK,
        sDialogText);

    gtk_dialog_run(GTK_DIALOG(pDialog));

    gtk_widget_destroy(pDialog);

    g_free(sDialogText);
}
```

Résultats :


GtkNotebook
Informations
Je suis le GtkLabel numero 0
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6


GtkNotebook
Informations
Je suis le GtkLabel numero 7
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7


GtkNotebook
Informations
Information
C'est la page 5
Le label est "Je suis le GtkLabel numero 5"
Le label de l'onglet est "Page 5"
Valider
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7

3. Ajouter un menu de navigation

Nous allons maintenant voir comment ajouter un menu qui apparaîtra lorsque nous ferons un clic droit de la souris sur une page du GtkNotebook. Cela ne se fait pas par le biais du widget GtkMenu mais directement avec les fonctions de GtkNotebook.

2.1 Création

Pour avoir un GtkNotebook avec un menu de navigation, trois nouvelles fonctions de création sont disponibles :

```
void gtk_notebook_append_page_menu (GtkNotebook *notebook, GtkWidget *child, GtkWidget *tab_label, GtkWidget *menu_label);
void gtk_notebook_prepend_page_menu (GtkNotebook *notebook, GtkWidget *child, GtkWidget *tab_label, GtkWidget *menu_label);
void gtk_notebook_insert_page_menu (GtkNotebook *notebook, GtkWidget *child, GtkWidget *tab_label, GtkWidget *menu_label, gint position);
```

La majorité des paramètres de ces trois fonctions sont identiques sauf pour le paramètre *menu_label* qui est le widget qui sera affiché dans le menu de navigation. Le fonctionnement des ces fonctions est le même que pour les fonctions sans menu. Ensuite pour activer ou désactiver la possibilité d'afficher le menu il faut utiliser une de ces fonctions :

```
void gtk_notebook_popup_enable (GtkNotebook *notebook);
void gtk_notebook_popup_disable (GtkNotebook *notebook);
```

La première fonction permet d'afficher le menu, et la deuxième au contraire empêche le menu de s'afficher.

2.2 Gestion des labels

Comme pour les onglets, il existe des fonctions pour gérer les labels du menu. Pour récupérer le widget ou directement le texte du menu, nous avons ces deux fonctions :

```
GtkWidget* gtk_notebook_get_menu_label (GtkNotebook *notebook, GtkWidget *child);
G_CONST_RETURN gchar* gtk_notebook_get_menu_label_text(GtkNotebook *notebook, GtkWidget *child);
```

La première fonction renvoie donc le widget, quel qu'il soit, qui est à l'intérieur du menu alors que la seconde renvoie le texte du menu si l'élément est de type GtkLabel. Et pour changer le label du menu, il existe ces deux fonctions :

```
void gtk_notebook_set_menu_label (GtkNotebook *notebook, GtkWidget *child, GtkWidget *menu_label);
void gtk_notebook_set_menu_label_text(GtkNotebook *notebook, GtkWidget *child, const gchar *menu_text);
```

La première accepte n'importe quel type de widget, tandis que la seconde crée elle-même un widget de type GtkLabel avec comme texte la valeur du paramètre menu_text.

2.3 Programme Exemple

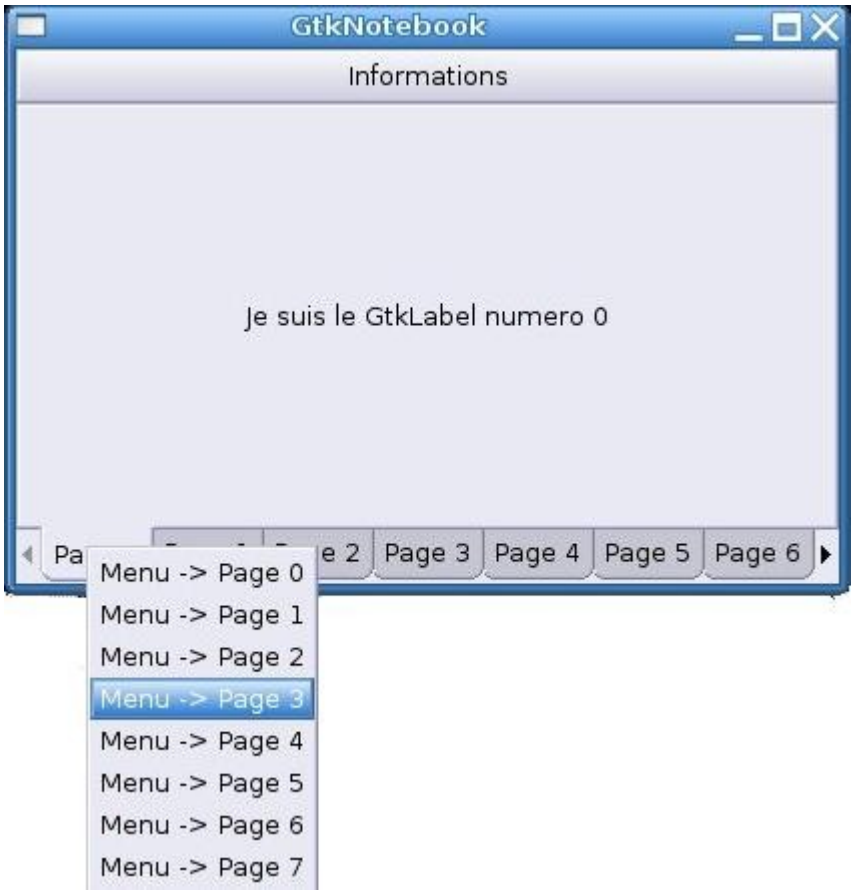
Nous allons reprendre le même exemple que pour la première partie en rajoutant le menu de navigation.

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

void OnButton(GtkWidget *pButton, gpointer data);

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pVBox;
    GtkWidget *pNotebook;
    GtkWidget *pButton;
    gint i;

    gtk_init(&argc,&argv);

    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkNotebook");
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    pVBox = gtk_vbox_new(FALSE, 0);
    gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

    pButton = gtk_button_new_with_label("Informations");
    gtk_box_pack_start(GTK_BOX(pVBox), pButton, FALSE, FALSE, 0);

    /* Creation du GtkNotebook */
    pNotebook = gtk_notebook_new();
    gtk_box_pack_start(GTK_BOX(pVBox), pNotebook, TRUE, TRUE, 0);
    /* Position des onglets : en bas */
    gtk_notebook_set_tab_pos(GTK_NOTEBOOK(pNotebook), GTK_POS_BOTTOM);
    /* Ajout des boutons de navigation */
    gtk_notebook_set_scrollable(GTK_NOTEBOOK(pNotebook), TRUE);

    for(i = 0 ; i < 8 ; ++i)
    {
        GtkWidget *pLabel;
        GtkWidget *pTabLabel;
        GtkWidget *pMenuLabel;
        gchar *sLabel;
        gchar *sTabLabel;
        gchar *sMenuLabel;

        sLabel = g_strdup_printf("Je suis le GtkLabel numero %d", i);
        sTabLabel = g_strdup_printf("Page %d", i);
        sMenuLabel = g_strdup_printf("Menu -> Page %d", i);

        /* Creation des differents GtkLabel */
        pLabel = gtk_label_new(sLabel);
        pTabLabel = gtk_label_new(sTabLabel);
        pMenuLabel = gtk_label_new(sMenuLabel);

        /* Insertion de la page */
        gtk_notebook_append_page_menu(GTK_NOTEBOOK(pNotebook), pLabel,
pTabLabel, pMenuLabel);
```

```
        g_free(sLabel);
        g_free(sTabLabel);
        g_free(sMenuLabel);
    }
    /* Activation du menu popup */
    gtk_notebook_popup_enable(GTK_NOTEBOOK(pNotebook));

    g_signal_connect(G_OBJECT(pButton), "clicked", G_CALLBACK(OnButton),
pNotebook);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}

void OnButton(GtkWidget *pButton, gpointer data)
{
    GtkWidget *pDialog;
    GtkWidget *pChild;
    gint iPageNum;
    const gchar *sLabel;
    const gchar *sTabLabel;
    const gchar *sMenuLabel;
    gchar *sDialogText;

    /* Recuperation de la page active */
    iPageNum = gtk_notebook_get_current_page(GTK_NOTEBOOK(data));
    /* Recuperation du widget enfant */
    pChild = gtk_notebook_get_nth_page(GTK_NOTEBOOK(data), iPageNum);

    /* Recuperation du label */
    sLabel = gtk_label_get_text(GTK_LABEL(pChild));
    /* Recuperation du label de l'onglet */
    sTabLabel = gtk_notebook_get_tab_label_text(GTK_NOTEBOOK(data), pChild);
    /* Recuperation du label du menu pop-up */
    sMenuLabel = gtk_notebook_get_menu_label_text(GTK_NOTEBOOK(data), pChild);

    /* Creation du label de la boite de dialogue */
    sDialogText = g_strdup_printf("C'est la page %d\n"
        "Le label est \"%s\"\n"
        "Le label de l'onglet est \"%s\"\n"
        "Le label du menu est \"%s\"\n",
        iPageNum,
        sLabel,
        sTabLabel,
        sMenuLabel);

    pDialog = gtk_message_dialog_new (NULL,
        GTK_DIALOG_MODAL,
        GTK_MESSAGE_INFO,
        GTK_BUTTONS_OK,
        sDialogText);

    gtk_dialog_run(GTK_DIALOG(pDialog));

    gtk_widget_destroy(pDialog);

    g_free(sDialogText);
```

```
}
```

Résultats :


GtkNotebook
Informations
Je suis le GtkLabel numero 0
Page 3
Page 4
Page 5
Page 6
Menu -> Page 0
Menu -> Page 1
Menu -> Page 2
Menu -> Page 3
Menu -> Page 4
Menu -> Page 5
Menu -> Page 6
Menu -> Page 7


GtkNotebook
Informations
Information
C'est la page 3
Le label est "Je suis le GtkLabel numero 3"
Le label de l'onglet est "Page 3"
Le label du menu est "Menu -> Page 3"
Valider
Page 0
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6

Les listes et arbres

1. Présentation

Nous allons étudier dans ce chapitre comment afficher des données sous forme de liste ou d'arbre. L'utilisation de ce widget se décompose en deux parties distinctes :

- la création du modèle ;
- la gestion de l'affichage.

La création du modèle se fait soit par l'utilisation de GtkListStore pour créer une liste, soit par GtkTreeStore pour la création d'un arbre. Ces deux objets stockent dans un premier temps la structure des données, c'est-à-dire le nombre de colonnes, le type d'affichage (texte, image, case à cocher) et stockent dans un second temps les données qui seront affichées par la suite.

Une fois le modèle créé, nous pouvons passer à la partie affichage des données. Le widget principal est bien sûr GtkTreeView qui utilise dans un premier temps l'objet GtkTreeViewColumn qui servira à définir chaque colonne du widget. Et enfin pour chaque colonne il faut définir le type d'affichage à l'aide des objets GtkCellRendererText (pour afficher du texte), GtkCellRendererPixbuf (pour afficher une image) et GtkCellRendererToggle (pour afficher une case à cocher).

2. Création d'une liste

Pour créer un modèle de type liste, nous allons utiliser l'objet GtkListStore. La création du modèle s'effectue en trois étapes :

- la création de l'objet GtkListStore ;
- l'ajout d'une entrée dans la liste (contenant) ;
- l'insertion des données dans l'entrée (contenu).

2.1 Création

La création de l'objet GtkListStore s'effectue simplement avec cette fonction :

```
GtkListStore* gtk_list_store_new(gint n_columns, ...);
```

Avec cette fonction, nous définissons le nombre de colonnes qui vont être créées (paramètre *n_columns*) ainsi que le type que le type de donnée que va contenir chaque colonne.

2.2 Insertion d'éléments

La première étape de l'insertion consiste à créer une nouvelle entrée (ligne) à la liste. Pour cela, nous allons utiliser l'objet GtkTreeIter. Cet objet permet de savoir à quel endroit nous nous positionnons dans la liste. La création d'une nouvelle ligne va donner une nouvelle position dont nous avons besoin pour entrer les données. Voici la liste des fonctions qui permettent de créer une nouvelle entrée :

```
void gtk_list_store_append(GtkListStore *list_store, GtkTreeIter *iter);
void gtk_list_store_prepend(GtkListStore *list_store, GtkTreeIter *iter);
void gtk_list_store_insert(GtkListStore *list_store, GtkTreeIter *iter, gint position);
void gtk_list_store_insert_before(GtkListStore *list_store, GtkTreeIter *iter, GtkTreeIter *sibling);
void gtk_list_store_insert_after(GtkListStore *list_store, GtkTreeIter *iter, GtkTreeIter *sibling);
```

Pour chacune de ces fonctions, le premier paramètre *list_store* est la liste qui a été crée précédemment et le deuxième paramètre *iter*, est la position de la ligne nouvellement crée.

Les deux premières fonctions sont les plus couramment utilisées lors de la création de la liste. La première fonction crée une nouvelle ligne à la fin de la liste tandis que la deuxième fonction l'ajoute au début de la liste.

Les trois fonctions suivantes permettent d'introduire une nouvelle ligne à une position bien précise. La troisième fonction l'ajoute par rapport à une valeur numérique (paramètre *position*), la quatrième fonction l'ajoute avant une ligne bien définie par son itération *sibling* et la cinquième fonction l'ajoute après une ligne définie de la même manière.

Il faut maintenant définir le contenu de la nouvelle ligne, en utilisant cette unique fonction :

```
void gtk_list_store_set(GtkListStore *list_store, GtkTreeIter *iter, ...);
```

Les paramètres *list_store* et *iter* sont bien sûr la liste et la ligne pour laquelle nous voulons spécifier les données. A la suite de ces deux paramètres il suffit d'ajouter les différentes données avec dans l'ordre la colonne de la donnée suivie de sa valeur. Et pour terminer, il faut dire à cette fonction que la liste des données est terminée simplement en ajoutant -1 en dernier paramètre.

La liste est maintenant prête à l'affichage.

3. Création d'un arbre

Nous allons maintenant voir les légères différences qu'il y a entre la création d'une liste et la création d'un arbre à l'aide de l'objet GtkTreeStore. La principale différence est que chaque ligne peut avoir une ligne parente et une ou plusieurs lignes enfants. Nous n'avons donc plus affaire à une simple succession de lignes, mais à une organisation imbriquée de succession de lignes.

3.1 Création

La fonction de création est très ressemblante à celle de l'objet GtkListStore :

```
GtkTreeStore* gtk_tree_store_new(gint n_columns, ...);
```

Bien entendu, le paramètre *n_columns* est le nombre de colonne de chaque ligne de l'arbre, suivi des différents types des données.

3.2 Insertion d'éléments

Cette fois aussi les fonctions sont presque identiques à celles fournies pour les listes :

```
void gtk_tree_store_append(GtkTreeStore *tree_store, GtkTreeIter *iter, GtkTreeIter *parent);
void gtk_tree_store_prepend(GtkTreeStore *tree_store, GtkTreeIter *iter, GtkTreeIter *parent);
void gtk_tree_store_insert(GtkTreeStore *tree_store, GtkTreeIter *iter, GtkTreeIter *parent, gint position);
void gtk_tree_store_insert_before(GtkTreeStore *tree_store, GtkTreeIter *iter, GtkTreeIter *parent, GtkTreeIter *sibling);
void gtk_tree_store_insert_after(GtkTreeStore *tree_store, GtkTreeIter *iter, GtkTreeIter *parent, GtkTreeIter *sibling);
```

La grande nouveauté, avec toutes ces fonctions, est la présence d'un paramètre supplémentaire, *parent*, de type GtkTreeIter, qui n'est autre que la ligne dont notre nouvel élément sera l'enfant. Si notre nouvelle ligne n'a pas de parent, ce paramètre sera à NULL, sinon il faudra mettre l'itération correspondant à la ligne parente.

Et pour finir, il faut définir les valeurs de chaque colonne grâce à cette fonction :

```
void gtk_tree_store_set(GtkTreeStore *tree_store, GtkTreeIter *iter, ...);
```

De la même façon que pour les éléments de type GtkListStore, la liste de valeurs doit se terminer par -1.

4. Affichage du widget GtkTreeView

Maintenant que le modèle du widget GtkTreeView a été créé, il faut gérer l'affichage de celui-ci. Cela va se dérouler en deux étapes distinctes :

- création de la vue ;
- création des colonnes.

Le déroulement des ces différentes étapes est identique que cela soit avec un modèle de type GtkListStore qu'avec un modèle de type GtkTreeStore.

4.1 Création de la vue

Il existe deux fonctions de création du widget GtkTreeView :

```
GtkWidget* gtk_tree_view_new(void);
GtkWidget* gtk_tree_view_new_with_model(GtkTreeModel? *model);
```

La première fonction crée une vue vide, alors que la deuxième fonction créera la vue à partir du modèle spécifié. Que l'on utilise un GtkListStore ou un GtkTreeStore, il faudra utiliser la macro GTK_TREE_MODEL pour spécifier le paramètre model.

Si la première fonction est utilisée pour la création de la vue, il faudra tout de même donner à la vue un modèle bien précis, pour que lors de l'affichage GTK+ sache quelles données il doit afficher, avec cette fonction :

```
void gtk_tree_view_set_model(GtkTreeView *tree_view, GtkTreeModel? *model);
```

4.2 Création des colonnes

Lors de cette étape, nous allons introduire deux nouveaux objets. Tout d'abord l'objet GtkCellRenderer? et ses dérivés, puis l'objet GtkTreeViewColumn. Nous allons donc maintenant voir comment ajouter les différentes colonnes à notre vue et aussi de quelle manière doit être fait le rendu. La manière dont les cases d'une colonne sont

rendues se défini à l'aide des objets dérivés de GtkCellRenderer? :

- GtkCellRendererText pour afficher du texte ;
- GtkCellRendererToggle pour afficher une case à cocher ou un bouton radio ;
- GtkCellRendererPixbuf pour afficher une image.

Pour créer un de ces objets, nous avons les fonctions suivantes :

```
GtkCellRenderer?* gtk_cell_renderer_text_new(void);
GtkCellRenderer?* gtk_cell_renderer_toggle_new(void);
GtkCellRenderer?* gtk_cell_renderer_pixbuf_new(void);
```

Une fois le GtkCellRenderer? créé, nous pouvons modifier différents paramètres à l'aide de cette fonction :

```
void g_object_set(gpointer object, const gchar *first_property_name, ...);
```

Le premier paramètre, *object*, est l'objet que l'on veut modifier. Il faut utiliser pour ce paramètre la macro G_OBJECT(). Puis, les paramètres suivants vont par couple paramètre/valeur pour modifier par exemple la couleur de fond de la case ou autre. Une fois tous les paramètres à modifier entrés, il faut ajouter NULL pour dire que la liste des modifications à apporter est terminée.

Maintenant que nous savons comment les cases d'une colonne doivent être rendues, nous allons voir comment créer une colonne et comment l'ajouter à la vue. Pour créer une colonne, nous avons cette fonction :

```
GtkTreeViewColumn* gtk_tree_view_column_new_with_attributes(const gchar *title,
GtkCellRenderer? *cell, ...);
```

Le premier paramètre, *title*, est le texte qui sera affiché en haut de la colonne dans une ligne bien spécifique. Le second paramètre, *cell*, est simplement l'objet GtkCellRenderer? que nous venons juste de créer, définissant le rendu de chaque case de la colonne.

Ensuite, il faut à nouveau définir le type de la colonne ainsi que le numéro de la colonne dans le modèle. Les types de colonnes sont "text" pour du texte, "active" pour une case à cocher, "pixbuf" pour une image.

Et pour terminer, comme souvent dans ce type de fonction, il faut ajouter NULL à la suite de tous les paramètres pour dire que la liste est terminée.

Il ne nous reste plus qu'à ajouter la colonne à la vue avec l'une de ces fonctions :

```
gint gtk_tree_view_append_column(GtkTreeView *tree_view, GtkTreeViewColumn *column);
gint gtk_tree_view_insert_column(GtkTreeView *tree_view, GtkTreeViewColumn *column, gint
position);
```

La première fonction ajoute la colonne à la suite des autres, tandis que la seconde l'ajoute à la position *position* ou à la suite des autres si la valeur de position est invalide. La macro GTK_TREE_VIEW() est à utiliser pour le premier paramètre.

4.3 Programmes Exemples

Nous allons créer deux exemples pour montrer l'utilisation du widget GtkTreeView. Dans le premier exemple, nous allons créer une liste avec simplement une colonne

avec du texte et une colonne comportant une case à cocher. Le deuxième exemple, utilisera quant à lui un arbre avec une première colonne affichant une image, et du texte dans la deuxième colonne.

```
#include <stdlib.h>
#include <gtk/gtk.h>
#include <glib/gprintf.h>

enum {
    TEXT_COLUMN,
    TOGGLE_COLUMN,
    N_COLUMN
};

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pListView;
    GtkWidget *pScrollbar;
    GtkListStore *pListStore;
    GtkTreeViewColumn *pColumn;
    GtkCellRenderer *pCellRenderer;
    gchar *sTexte;
    gint i;

    gtk_init(&argc, &argv);

    /* Creation de la fenetre principale */
    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkTreeView et GtkListStore");
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    /* Creation du modele */
    pListStore = gtk_list_store_new(N_COLUMN, G_TYPE_STRING, G_TYPE_BOOLEAN);

    sTexte = g_malloc(12);

    /* Insertion des elements */
    for(i = 0 ; i < 10 ; ++i)
    {
        GtkTreeIter pIter;

        g_sprintf(sTexte, "Ligne %d\0", i);

        /* Creation de la nouvelle ligne */
        gtk_list_store_append(pListStore, &pIter);

        /* Mise a jour des donnees */
        gtk_list_store_set(pListStore, &pIter,
            TEXT_COLUMN, sTexte,
            TOGGLE_COLUMN, TRUE,
            -1);
    }

    g_free(sTexte);

    /* Creation de la vue */
    pListView = gtk_tree_view_new_with_model(GTK_TREE_MODEL(pListStore));
```

```
    /* Creation de la premiere colonne */
    pCellRenderer = gtk_cell_renderer_text_new();
    pColumn = gtk_tree_view_column_new_with_attributes("Titre",
        pCellRenderer,
        "text", TEXT_COLUMN,
        NULL);

    /* Ajout de la colonne à la vue */
    gtk_tree_view_append_column(GTK_TREE_VIEW(pListView), pColumn);

    /* Creation de la deuxieme colonne */
    pCellRenderer = gtk_cell_renderer_toggle_new();
    pColumn = gtk_tree_view_column_new_with_attributes("CheckBox",
        pCellRenderer,
        "active", TOGGLE_COLUMN,
        NULL);

    /* Ajout de la colonne à la vue */
    gtk_tree_view_append_column(GTK_TREE_VIEW(pListView), pColumn);

    /* Ajout de la vue a la fenetre */
    pScrollbar = gtk_scrolled_window_new(NULL, NULL);
    gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(pScrollbar),
        GTK_POLICY_AUTOMATIC,
        GTK_POLICY_AUTOMATIC);
    gtk_container_add(GTK_CONTAINER(pScrollbar), pListView);
    gtk_container_add(GTK_CONTAINER(pWindow), pScrollbar);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}
```

Résultats :


```
#include <stdlib.h>
#include <gtk/gtk.h>
#include <glib/gprintf.h>

enum {
    BMP_COLUMN,
    TEXT_COLUMN,
    N_COLUMN
};

/* Utilisateurs Visual C++ : il faut ajouter gdk_pixbuf-2.0.lib dans les options
du linker */

int main(int argc, char **argv)
{
    GtkWidget *pWindow;
    GtkWidget *pTreeView;
    GtkWidget *pScrollbar;
    GtkTreeStore *pTreeStore;
    GtkTreeViewColumn *pColumn;
    GtkCellRenderer *pCellRenderer;
    GdkPixbuf *pPixBufA;
    GdkPixbuf *pPixBufB;
    gchar *sTexte;
    gint i;

    gtk_init(&argc, &argv);

    /* Creation de la fenetre principale */
    pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
    gtk_window_set_title(GTK_WINDOW(pWindow), "GtkTreeView et GtkTreeStore");
    g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit),
NULL);

    /* Creation du modele */
    pTreeStore = gtk_tree_store_new(N_COLUMN, GDK_TYPE_PIXBUF, G_TYPE_STRING);

    sTexte = g_malloc(16);

    /* Chargement des images */
    pPixBufA = gdk_pixbuf_new_from_file("/usr/share/pixmaps/gnome-computer.png",
NULL);
    pPixBufB = gdk_pixbuf_new_from_file("/usr/share/pixmaps/gnome-folder.png",
```

```
NULL);

    /* Insertion des elements */
    for(i = 0 ; i < 10 ; ++i)
    {
        GtkTreeIter pIter;
        GtkTreeIter pIter2;
        gint j;

        g_sprintf(sTexte, "Ordinateur %d", i);

        /* Creation de la nouvelle ligne */
        gtk_tree_store_append(pTreeStore, &pIter, NULL);

        /* Mise a jour des donnees */
        gtk_tree_store_set(pTreeStore, &pIter,
            BMP_COLUMN, pPixBufA,
            TEXT_COLUMN, sTexte,
            -1);

        for(j = 0 ; j < 2 ; ++j)
        {
            g_sprintf(sTexte, "Repertoire %d", j);

            /* Creation de la nouvelle ligne enfant */
            gtk_tree_store_append(pTreeStore, &pIter2, &pIter);

            /* Mise a jour des donnees */
            gtk_tree_store_set(pTreeStore, &pIter2,
                BMP_COLUMN, pPixBufB,
                TEXT_COLUMN, sTexte,
                -1);
        }
    }

    g_free(sTexte);

    g_object_unref(pPixBufA);
    g_object_unref(pPixBufB);

    /* Creation de la vue */
    pTreeView = gtk_tree_view_new_with_model(GTK_TREE_MODEL(pTreeStore));

    /* Creation de la premiere colonne */
    pCellRenderer = gtk_cell_renderer_pixbuf_new();
    pColumn = gtk_tree_view_column_new_with_attributes("Image",
        pCellRenderer,
        "pixbuf", BMP_COLUMN,
        NULL);

    /* Ajout de la colonne à la vue */
    gtk_tree_view_append_column(GTK_TREE_VIEW(pTreeView), pColumn);

    /* Creation de la deuxieme colonne */
    pCellRenderer = gtk_cell_renderer_text_new();
    pColumn = gtk_tree_view_column_new_with_attributes("Label",
        pCellRenderer,
        "text", TEXT_COLUMN,
        NULL);

    /* Ajout de la colonne à la vue */
```

```
    gtk_tree_view_append_column(GTK_TREE_VIEW(pTreeView), pColumn);

    /* Ajout de la vue a la fenetre */
    pScrollbar = gtk_scrolled_window_new(NULL, NULL);
    gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(pScrollbar),
        GTK_POLICY_AUTOMATIC,
        GTK_POLICY_AUTOMATIC);
    gtk_container_add(GTK_CONTAINER(pScrollbar), pTreeView);
    gtk_container_add(GTK_CONTAINER(pWindow), pScrollbar);

    gtk_widget_show_all(pWindow);

    gtk_main();

    return EXIT_SUCCESS;
}
```

Résultats :

Copyright

Licence

GNU Free Documentation License

Version 1.1, March 2000

59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated,

as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX? input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript?, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated

as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the

Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical

Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.