

# Utilisation de GtkUIManager

par Nicolas Joseph ([home](#)) ([Blog](#))

Date de publication : 13 Juillet 2005

Utilisation de GtkUIManager en C pour créer un menu et une barre d'outils.

- I - Introduction
- II - Création d'un menu
  - II-A - GtkUIManager
  - II-B - GtkActionEntry
- III - Création d'une barre d'outils
- IV - Les raccourcis clavier
- V - Conclusion
- VI - Remerciement
- VII - Code source complet

## I - Introduction

Ce tutorial à pour but de expliquer l'utilisation de l'objet *GtkUIManager* qui fait partie des nouveautés de la version 2.4 de GTK+. Son utilisation peut paraître déroutante de prime abord mais en fait elle simplifie grandement la mise en place de menus, de barres d'outils et de raccourcis clavier. Elle comporte de nombreux avantages comme je vais essayer de vous le démontrer tout au long de ce tutorial.

## II - Création d'un menu

Il existe plusieurs possibilités pour créer un menu en GTK+ :

- La méthode manuelle : en utilisant `gtk_menu_new`, `gtk_menu_item_new_with_label...` cette méthode est longue et difficilement maintenable
- En utilisant l'usine à gaz `GtkItemFactory`, mais cette méthode est obsolète
- Et enfin une dernière méthode qui va nous intéresser dans cet article : à l'aide du composant `GtkUIManager`

Cette dernière méthode est en fait divisée en deux parties : l'organisation du menu (sauvegardée dans un fichier XML, géré par l'objet `GtkUIManager`) et la forme (gérée à l'aide de groupes d'actions, par l'intermédiaire de l'objet `GtkActionEntry`).

### II-A - GtkUIManager

Le fichier XML se contente de fournir des informations concernant l'organisation du menu. A titre d'information, voici le fichier DTD qui définit la syntaxe XML à utiliser :

```

<!ELEMENT ui          (menubar|toolbar|popup|accelerator)* >
<!ELEMENT menubar    (menuitem|separator|placeholder|menu)* >
<!ELEMENT menu       (menuitem|separator|placeholder|menu)* >
<!ELEMENT popup      (menuitem|separator|placeholder|menu)* >
<!ELEMENT toolbar    (toolitem|separator|placeholder)* >
<!ELEMENT placeholder (menuitem|toolitem|separator|placeholder|menu)* >
<!ELEMENT menuitem   EMPTY >
<!ELEMENT toolitem   EMPTY >
<!ELEMENT separator  EMPTY >
<!ELEMENT accelerator EMPTY >
<!ATTLIST menubar    name                #IMPLIED
                  action                #IMPLIED >
<!ATTLIST toolbar    name                #IMPLIED
                  action                #IMPLIED >
<!ATTLIST popup      name                #IMPLIED
                  action                #IMPLIED >
<!ATTLIST placeholder name                #IMPLIED
                  action                #IMPLIED >
<!ATTLIST separator  name                #IMPLIED
                  action                #IMPLIED >
<!ATTLIST menu       name                #IMPLIED
                  action                #REQUIRED
                  position (top|bot) #IMPLIED >
<!ATTLIST menuitem   name                #IMPLIED
                  action                #REQUIRED
                  position (top|bot) #IMPLIED >
<!ATTLIST toolitem   name                #IMPLIED
                  action                #REQUIRED
                  position (top|bot) #IMPLIED >
<!ATTLIST accelerator name                #IMPLIED
                  action                #REQUIRED >
    
```

Le fichier XML doit commencer par la balise `<ui>` et se terminer par `</ui>`. A l'intérieur de ces balises, peuvent se trouver des balises `<menubar>`, `<toolbar>`, `<popup>` et `<accelerator>`. La balise `<menubar>` signale le début d'une barre de menu, elle ne prend pas de paramètre. La zone encadrée par les balises `<menubar>` et `</menubar>` peut contenir les balises suivantes :

- `<menuitem>` : il s'agit d'un bouton qui permet d'effectuer une action lorsque l'on clique dessus

- `<separator>` : comme son nom l'indique, il s'agit d'un séparateur
- `<placeholder>` : ?
- `<menu>` : affiche un sous menu lorsque l'on clique dessus (le plus courant)

A part `<separator>`, les deux autres balises ne nécessitent qu'un seul argument qui est le paramètre `action`. Ce paramètre est une chaîne de caractères qui nous servira ensuite à identifier le menu dans la structure `GtkActionEntry`.

Pour résumer, voici un exemple de fichier XML qui sert à créer un menu simple du genre :

```
- Fichier
- Nouveau
- Ouvrir
- Quitter
- Aide
- Aide
- A propos
```

Et voici le fichier XML correspondant :

```
<ui>
  <menubar>
    <menu action="FichierMenuAction">
      <menuitem action="NouveauAction" />
      <menuitem action="OuvrirAction" />
      <menuitem action="QuitterAction" />
    </menu>
    <menu action="AideMenuAction">
      <menuitem action="AideAction" />
      <menuitem action="AproposAction" />
    </menu>
  </menubar>
</ui>
```

Une fois le fichier XML créé, passons du côté du code. Il faut commencer par créer un objet de type de type `GtkUIManager` :

```
GtkUIManager *p_uiManager = gtk_ui_manager_new ();
```

Ensuite, il suffit de faire analyser le fichier XML par GTK+ :

```
gtk_ui_manager_add_ui_from_file
(
  p_uiManager,
  "menu.xml",
  NULL
);
```

Le premier paramètre de la fonction `gtk_ui_manager_add_ui_from_file` est tout naturellement l'objet `GtkUIManager` créé précédemment, ensuite vient le nom du fichier XML contenant l'organisation du menu. Le dernier paramètre, qui peut être NULL, permet de récupérer une éventuelle erreur qui se serait produite lors de l'analyse du fichier, mais ceci dépasse le cadre de cet article.

Pour finir, il faut fournir à `GtkUIManager` une fonction à appeler pour ajouter les widgets dans le menu :

```
g_signal_connect
(
```

```
p_uiManager,  
"add_widget",  
G_CALLBACK( menu_addWidget ),  
p_menuBox  
);
```

Ceci est une utilisation classique de la fonction de gestion des callback de la glib. Seul point important à noter, il faut passer un pointeur sur un objet de type *GtkContainer* qui contiendra le widget nouvellement créé.

La fonction *menu\_addWidget* est des plus simples puisqu'elle se contente d'insérer l'élément et de l'afficher :

```
static void menu_addWidget (GtkUIManager * p_uiManager, GtkWidget * p_widget,  
                           GtkContainer * p_box)  
{  
    gtk_box_pack_start (GTK_BOX (p_box), p_widget, FALSE, FALSE, 0);  
    gtk_widget_show (p_widget);  
    return;  
}
```

## II-B - GtkActionEntry

Après la forme, il faut associer une action pour chaque élément du menu. Ceci va être fait à l'aide de la structure *GtkActionEntry* qui est définie comme ceci :

```
typedef struct  
{  
    const gchar *name;  
    const gchar *stock_id;  
    const gchar *label;  
    const gchar *accelerator;  
    const gchar *tooltip;  
    GCallback callback;  
} GtkActionEntry;
```

Voici le détail de chaque champ :

- *name* : c'est le nom de l'élément du menu, il s'agit du nom spécifié dans le fichier XML par l'intermédiaire du paramètre *action*
- *stock\_id* : ce champ permet de spécifier une image prédéfinie (GTK\_STOCK\_NEW par exemple)
- *label* : c'est le texte qui sera affiché
- *accelerator* : pour spécifier un raccourcis clavier, nous y reviendront dans la suite de cet article
- *tooltip* : ce paramètre ne sert à rien dans le cas d'un menu, il nous sera utile lors de la création de la barre d'outils
- *callback* : c'est l'adresse de la fonction qui va être appelée lors d'un clique sur cet élément

Pour continuer notre exemple, voici la structure correspondant au fichier XML ci-dessus :

```
GtkActionEntry entries[] =  
{  
    { "FichierMenuAction", NULL, "Fichier", NULL, NULL, NULL },  
    { "NouveauAction", GTK_STOCK_NEW, "Nouveau", NULL, NULL, G_CALLBACK( menu_new ) },  
    { "OuvrirAction", GTK_STOCK_OPEN, "Ouvrir", NULL, NULL, G_CALLBACK( menu_open ) },  
    { "QuitterAction", GTK_STOCK_QUIT, "Quitter", NULL, NULL, G_CALLBACK( menu_quit ) },  
    { "AideMenuAction", NULL, "Aide", NULL, NULL, NULL },  
    { "AideAction", GTK_STOCK_HELP, "Aide", NULL, NULL, G_CALLBACK( menu_help ) },  
    { "AproposAction", GTK_STOCK_ABOUT, "A propos", NULL, NULL, G_CALLBACK( menu_about ) }
```

```
};
```

Bien sûr les fonctions callback doivent être définies.

Une fois la structure *GtkActionEntry* renseignée, il faut l'associer au fichier XML. Pour se faire, on commence par créer un objet de type *GtkActionGroup* :

```
p_actionGroup = gtk_action_group_new( "menuActionGroup" );
```

La fonction *gtk\_action\_group\_new* attend comme seul paramètre le nom du groupe, comme nous ne nous en servons pas par la suite, le choix du nom n'est pas très important, par contre GTK en a besoin. Ensuite nous allons utiliser la structure *GtkActionEntry* pour créer les actions correspondantes et les ajouter au groupe d'actions. Lors de cette étape, les fonctions callback seront associées à l'événement "activate" ainsi qu'aux raccourcis clavier. Pour faire ceci, il suffit d'appeler la fonction *gtk\_action\_group\_add\_actions* :

```
GtkActionGroup *p_actionGroup = gtk_action_group_add_actions( p_actionGroup, entries, G_N_ELEMENTS( entries ), NULL );
```

Le premier paramètre est la structure de groupe d'actions à créer, suivie d'un tableau contenant toutes les structure de type *GtkActionEntry* à regrouper. Ensuite, c'est le nombre d'éléments du tableau qui est nécessaire; la macro *G\_N\_ELEMENTS* permet de retrouver ce nombre facilement, il faut cependant noter qu'elle ne fonctionne qu'avec les tableaux statiques. Et enfin un pointeur sur une donnée utilisateur à passer aux fonctions callback. Il ne reste plus qu'à associer le groupe d'actions à la structure *GtkUIManager* :

```
gtk_ui_manager_insert_action_group( p_uiManager, p_actionGroup, 0 );
```

Les deux premiers paramètres viennent d'être expliqués en détail. Le troisième, quant à lui, permet de spécifier la position à laquelle les actions sont insérées dans le groupe (puisque nous n'en avons qu'une, on la met en première position).

### III - Création d'une barre d'outils

Pour créer une barre d'outils, c'est comme pour le menu sauf qu'il y a pas une ligne de code à ajouter ! Eh oui, c'est tout l'intérêt de cette méthode, si vous avez le menu, en deux secondes vous obtenez la barre d'outils. Pour cela, il suffit de modifier le fichier XML (donc aucune recompilation n'est nécessaire) en utilisant les balises `<toolbar>` et `<toolitem>` comme nous avons utilisé précédemment `<menu>` et `<menuitem>` :

```
<toolbar>
  <toolitem action="NouveauAction" />
  <toolitem action="OuvrirAction" />
  <toolitem action="QuitterAction" />
  <separator />
  <toolitem action="AideAction" />
  <toolitem action="AproposAction" />
</toolbar>
```

Dans cette exemple, j'ai ajouté un séparateur pour illustrer son utilisation. En utilisant les mêmes noms pour le paramètre `action`, nous retrouvons les mêmes attributs que pour le menu grâce à la structure `GtkActionEntry`. Il est maintenant temps de vous expliquer le rôle du champ `tooltip` de cette dernière : il permet d'ajouter une description qui apparaît lorsque le curseur de la souris reste un certain temps immobile sur le bouton. Voici donc le contenu de la nouvelle structure `GtkActionEntry` :

```
GtkActionEntry entries[] =
{
  { "FichierMenuAction", NULL, "Fichier", NULL, NULL, NULL },
  { "NouveauAction", GTK_STOCK_NEW, "Nouveau", NULL, "Nouveau", G_CALLBACK( menu_new ) },
  { "OuvrirAction", GTK_STOCK_OPEN, "Ouvrir", NULL, "Ouvrir", G_CALLBACK( menu_open ) },
  { "QuitterAction", GTK_STOCK_QUIT, "Quitter", NULL, "Quitter", G_CALLBACK( menu_quit ) },
  { "AideMenuAction", NULL, "Aide", NULL, NULL, NULL },
  { "AideAction", GTK_STOCK_HELP, "Aide", NULL, "Aide", G_CALLBACK( menu_help ) },
  { "AproposAction", GTK_STOCK_ABOUT, "A propos", NULL, "About", G_CALLBACK( menu_about ) }
};
```

## IV - Les raccourcis clavier

Dans la structure *GtkActionEntry*, il y a un champ *accelerator* qui est destiné à contenir un raccourci clavier pour le menu correspondant. Le raccourci doit être spécifié sous forme d'une chaîne de caractères au format accepté par la fonction *gtk\_accelerator\_parse()*, c'est-à-dire commençant par ou moins l'une des balises suivantes :

- <Control>
- <Shift>
- <Alt>
- <Release> pour le relâchement d'une touche

Ces balises doivent être suivies, soit par une lettre qui correspond à une touche, soit pour les touches fonction de F1 jusqu'à F12. Voici la structure *GtkActionEntry* définitive :

```
GtkActionEntry entries[] =
{
  { "FichierMenuAction", NULL, "Fichier", NULL, NULL, NULL },
  { "NouveauAction", GTK_STOCK_NEW, "Nouveau", "<Control>N", "Nouveau", G_CALLBACK(
menu_new ) },
  { "OuvrirAction", GTK_STOCK_OPEN, "Ouvrir", "<Control>O", "Ouvrir", G_CALLBACK(
menu_open ) },
  { "QuitterAction", GTK_STOCK_QUIT, "Quitter", "<control>Q", "Quitter", G_CALLBACK(
menu_quit ) },
  { "AideMenuAction", NULL, "Aide", NULL, NULL, NULL },
  { "AideAction", GTK_STOCK_HELP, "Aide", "<Release>F1", "Aide", G_CALLBACK(
menu_help ) },
  { "AproposAction", GTK_STOCK_ABOUT, "A propos", "<Control>A", "About", G_CALLBACK(
menu_about ) }
};
```

## V - Conclusion

Cette méthode de création de menu fait partie des nouveautés de la version 2.4 de GTK+. En plus d'être simple à utiliser, facilement lisible et donc maintenable, elle permet, avec un minimum d'efforts, de générer des menus dynamiquement lors de l'exécution (et plus uniquement en dur dans le code).

## VI - Remerciement

Merci à Beusse pour la relecture attentive de cet article.

## VII - Code source complet

Voici le code complet pour créer un menu à l'aide de *GtkUIManager*. Cette méthode nécessite au minimum GTK+-2.4, cependant, cet exemple utilise des fonctionnalités présentes uniquement à partir de la version 2.6 (tout ce qui concerne le menu "A propos").

### Archive zip

#### menu.xml

```
<ui>
  <menubar>
    <menu action="FichierMenuAction">
      <menuitem action="NouveauAction" />
      <menuitem action="OuvrirAction" />
      <menuitem action="QuitterAction" />
    </menu>
    <menu action="AideMenuAction">
      <menuitem action="AideAction" />
      <menuitem action="AproposAction" />
    </menu>
  </menubar>
  <toolbar>
    <toolitem action="NouveauAction" />
    <toolitem action="OuvrirAction" />
    <toolitem action="QuitterAction" />
    <separator />
    <toolitem action="AideAction" />
    <toolitem action="AproposAction" />
  </toolbar>
</ui>
```

#### main.c

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

static void menu_new (void);
static void menu_open (void);
static void menu_quit (void);
static void menu_help (void);
static void menu_about (void);
static void menu_addWidget (GtkUIManager *, GtkWidget *, GtkContainer *);

int main (int argc, char **argv)
{
  GtkWidget *p_window = NULL;
  GtkWidget *p_vBox = NULL;
  GtkUIManager *p_uiManager = NULL;
  GtkActionGroup *p_actionGroup = NULL;
  GtkActionEntry entries[] = {
    {"FichierMenuAction", NULL, "Fichier", NULL, NULL, NULL},
    {"NouveauAction", GTK_STOCK_NEW, "Nouveau", "<Control>N", "Nouveau",
     G_CALLBACK (menu_new)},
    {"OuvrirAction", GTK_STOCK_OPEN, "Ouvrir", "<Control>O", "Ouvrir",
     G_CALLBACK (menu_open)},
    {"QuitterAction", GTK_STOCK_QUIT, "Quitter", "<control>Q", "Quitter",
     G_CALLBACK (menu_quit)},
    {"AideMenuAction", NULL, "Aide", NULL, NULL, NULL},
    {"AideAction", GTK_STOCK_HELP, "Aide", "<Release>F1", "Aide",
     G_CALLBACK (menu_help)},
    {"AproposAction", GTK_STOCK_ABOUT, "A propos", "<Control>A", "About",
     G_CALLBACK (menu_about)}
  };
  /* Initialisation */
```

## main.c

```
gtk_init (&argc, &argv);
/* Creation de la fenetre principale */
p_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title (GTK_WINDOW (p_window),
    "Utilisation de GtkUIManager");
g_signal_connect (G_OBJECT (p_window), "destroy",
    G_CALLBACK (gtk_main_quit), NULL);
/* Creation d'une vbox */
p_vBox = gtk_vbox_new (FALSE, 0);
gtk_container_add (GTK_CONTAINER (p_window), p_vBox);
/* Creation du menu */
p_uiManager = gtk_ui_manager_new ();
p_actionGroup = gtk_action_group_new ("menuActionGroup");
gtk_action_group_add_actions (p_actionGroup, entries,
    G_N_ELEMENTS (entries), NULL);
gtk_ui_manager_insert_action_group (p_uiManager, p_actionGroup, 0);
gtk_ui_manager_add_ui_from_file (p_uiManager, "menu.xml", NULL);
g_signal_connect
    (p_uiManager, "add_widget", G_CALLBACK (menu_addWidget), p_vBox);
/* Boucle principale */
gtk_widget_show_all (p_window);
gtk_main ();
/* destruction */
return (EXIT_SUCCESS);
}

static void menu_addWidget (GtkUIManager * p_uiManager, GtkWidget * p_widget,
    GtkContainer * p_box)
{
    gtk_box_pack_start (GTK_BOX (p_box), p_widget, FALSE, FALSE, 0);
    gtk_widget_show (p_widget);
    return;
}

static void menu_new (void)
{
    GtkWidget *p_dialog = NULL;

    p_dialog = gtk_message_dialog_new
        (NULL, GTK_DIALOG_MODAL, GTK_MESSAGE_INFO, GTK_BUTTONS_OK, "Nouveau");
    gtk_dialog_run (GTK_DIALOG (p_dialog));
    gtk_widget_destroy (p_dialog);
    return;
}

static void menu_open (void)
{
    GtkWidget *p_dialog = NULL;

    p_dialog = gtk_message_dialog_new
        (NULL, GTK_DIALOG_MODAL, GTK_MESSAGE_INFO, GTK_BUTTONS_OK, "Ouvrir");
    gtk_dialog_run (GTK_DIALOG (p_dialog));
    gtk_widget_destroy (p_dialog);
    return;
}

static void menu_quit (void)
{
    gtk_main_quit ();
    return;
}

static void menu_help (void)
{
    GtkWidget *p_dialog = NULL;

    p_dialog = gtk_message_dialog_new
```

## main.c

```
(NULL, GTK_DIALOG_MODAL, GTK_MESSAGE_INFO, GTK_BUTTONS_OK, "Aide");
gtk_dialog_run (GTK_DIALOG (p_dialog));
gtk_widget_destroy (p_dialog);
return;
}

static void menu_about (void)
{
    GError *error = NULL;
    GIOChannel *channel = NULL;

    channel = g_io_channel_new_file ("licence.txt", "r", &error);
    if (channel != NULL)
    {
        char *text = NULL;

        if (g_io_channel_read_to_end (channel, &text, 0, &error) ==
            G_IO_STATUS_NORMAL)
        {
            GtkWidget *p_dialog = gtk_about_dialog_new ();

            gtk_about_dialog_set_name (GTK_ABOUT_DIALOG (p_dialog),
                                     "Exemple d'utilisation de GtkUIManager");
            gtk_about_dialog_set_copyright (GTK_ABOUT_DIALOG (p_dialog),
                                           "Nicolas JOSEPH");
            gtk_about_dialog_set_license (GTK_ABOUT_DIALOG (p_dialog), text);
            gtk_about_dialog_set_website (GTK_ABOUT_DIALOG (p_dialog),
                                         "http://nicolasj.developpez.com");

            gtk_dialog_run (GTK_DIALOG (p_dialog));
            g_free (text);
            if (g_io_channel_shutdown (channel, FALSE, &error) ==
                G_IO_STATUS_ERROR)
            {
                GtkWidget *p_dialog = NULL;

                p_dialog = gtk_message_dialog_new
                    (NULL,
                     GTK_DIALOG_MODAL,
                     GTK_MESSAGE_INFO, GTK_BUTTONS_OK, error->message);
                gtk_dialog_run (GTK_DIALOG (p_dialog));
                gtk_widget_destroy (p_dialog);
            }
        }
    }
    if (error != NULL)
    {
        GtkWidget *p_dialog = NULL;

        p_dialog = gtk_message_dialog_new
            (NULL,
             GTK_DIALOG_MODAL, GTK_MESSAGE_INFO, GTK_BUTTONS_OK, error->message);
        gtk_dialog_run (GTK_DIALOG (p_dialog));
        gtk_widget_destroy (p_dialog);
    }
    return;
}
```

