

Bien utiliser les GtkComboBox et GtkComboBoxEntry

par [Franck Hecht](#)

Date de publication : 15/04/2007

Dernière mise à jour :

Ce tutoriel vise à vous apprendre comment utiliser correctement les widgets successeurs de GtkCombo de GTK+ à savoir GtkComboBox et GtkComboBoxEntry !

I - Introduction

II - GtkComboBox

II-A - Mise en oeuvre personnalisée

- II-A-1 - Création d'un modèle GtkTreeModel
- II-A-2 - Initialisation du widget
- II-A-3 - Mise en place du gestionnaire de rendu
- II-A-4 - Remplissage du GtkComboBox avec des données
- II-A-5 - Récupérer les données de l'élément sélectionné
- II-A-6 - Suppression de l'élément sélectionné
- II-A-7 - Vider complètement notre GtkComboBox
- II-A-8 - Code source d'un exemple

II-B - Mise en oeuvre prédéfinie

- II-B-1 - Initialisation du widget
- II-B-2 - Remplissage du GtkComboBox avec des données
- II-B-3 - Récupérer le texte de l'élément sélectionné
- II-B-4 - Suppression de l'élément sélectionné
- II-B-5 - Code source d'un exemple

III - GtkComboBoxEntry

III-A - Les petites différences de la mise en oeuvre personnalisée...

- III-A-1 - Pour l'initialisation du widget
- III-A-2 - Pour la création des objets de rendu
- III-A-3 - Pour la suppression de l'élément sélectionné
- III-A-4 - Code source d'un exemple

III-B - Mise en oeuvre prédéfinie

III-C - La zone de texte du GtkComboBoxEntry

- III-C-1 - Comment y accéder ?
- III-C-2 - Comment valider la saisie et l'insérer dans la liste ?
- III-C-3 - Comment bloquer la saisie utilisateur ?

IV - Fonctions diverses

- IV-A - Afficher un élément par défaut
- IV-B - Permettre la mise en fenêtre de la liste déroulante
- IV-C - Afficher les données sur plusieurs colonnes

V - Bonus

VI - Remerciements

I - Introduction

Depuis la version 2.4 de GTK+, le widget GtkCombo a été déprécié, ce qui signifie qu'il ne doit plus être utilisé pour les développements de vos futures applications ! En revanche, deux nouveaux widgets ont fait leur apparition à savoir GtkComboBox et GtkComboBoxEntry.



Je vois souvent et même encore aujourd'hui, des gens utiliser ces nouveaux widgets avec les fonctions de l'ancien widget et parfois même c'est l'inverse qui se produit ! Ce tutoriel va donc vous apprendre à maîtriser ces deux nouveaux widgets de GTK+ et vous apprendrez les quatre manières (*deux par widgets en réalité*) de les mettre en oeuvre. Vous apprendrez également comment les gérer et les personnaliser !

Avant de commencer je voudrais vous rappeler qu'il existe sur developpez.com des forums destinés à GTK+ ainsi qu'un site et une FAQ:

- [Site](#)
- [FAQ](#)
- [Forums](#)

II - GtkComboBox

II-A - Mise en oeuvre personnalisée

La mise en oeuvre personnalisée consiste à créer un GtkComboBox à partir de la fonction [gtk_combo_box_new](#) ou [gtk_combo_box_new_with_model](#)

Cette façon de faire permet en effet d'initialiser un GtkComboBox avec une présentation des données autre que texte. On peut tout comme avec le widget GtkTreeView par exemple créer un magasin de données qui permette aussi bien l'ajout de données numériques, du texte, voir même les deux simultanément et également d'autres types !

Dans cette section nous verrons comment mettre en place un GtkComboBox qui permet l'ajout de données numériques + texte (*donc deux colonnes de données par élément de la liste combo*). Ceci a pour avantage par exemple que lorsque vous récupérez les données de l'élément sélectionné, vous n'avez pas besoin de convertir une chaîne de caractères en nombre, cela peut souvent s'avérer pratique, nous le verrons ensemble !

II-A-1 - Création d'un modèle GtkTreeModel

La création d'un GtkTreeModel se résume en fait par la création d'un GtkListStore ou GtkTreeStore. Ici nous allons utiliser un GtkListStore ce qui paraît être plus approprié pour notre widget. Il nous faut donc pour cela déclarer une variable de type **GtkListStore** et l'initialiser avec la fonction [gtk_list_store_new](#) comme suit:

Création d'un nouveau modèle

```
GtkListStore * p_model = gtk_list_store_new (2, G_TYPE_INT, G_TYPE_STRING);
```

Ici nous avons donc créé un nouveau modèle de données pour notre GtkComboBox qui permet dans l'ordre, de contenir un entier + une chaîne de caractères ! Le premier paramètre de la fonction n'est autre que le nombre de colonnes puis à la suite et dans l'ordre bien sûr, le type de chaque colonne. Les types possibles sont les suivants:

- G_TYPE_NONE
- G_TYPE_INTERFACE
- G_TYPE_CHAR
- G_TYPE_UCHAR
- G_TYPE_BOOLEAN
- G_TYPE_INT
- G_TYPE_UINT
- G_TYPE_LONG
- G_TYPE_ULONG
- G_TYPE_INT64
- G_TYPE_UINT64
- G_TYPE_ENUM
- G_TYPE_FLAGS
- G_TYPE_FLOAT
- G_TYPE_DOUBLE
- G_TYPE_STRING
- G_TYPE_POINTER
- G_TYPE_BOXED
- G_TYPE_PARAM

- G_TYPE_OBJECT
- G_TYPE_GTYPE
- GDK_TYPE_PIXBUF (pour insérer des colonnes d'images)

II-A-2 - Initialisation du widget

L'initialisation du widget se fait soit avec la fonction `gtk_combo_box_new` ou `gtk_combo_box_new_with_model`. La seule différence entre les deux, c'est que la première n'implique pas la création au préalable d'un modèle alors que la seconde l'exige étant donné qu'elle prend en argument ce nouveau modèle ! Ici nous utiliseront la seconde fonction:

Initialisation d'un GtkComboBox avec notre modèle

```
GtkWidget * p_combo = gtk_combo_box_new_with_model (GTK_TREE_MODEL (p_model));
```

Voilà, rien de plus, vous pouvez même l'intégrer dans un conteneur dès que la création du widget est effectuée !

II-A-3 - Mise en place du gestionnaire de rendu

C'est ici que nous allons préciser à notre GtkComboBox comment rendre à l'écran (*dessiner et présenter*) les différentes colonnes de données. Nous allons continuer dans notre exemple actuel et nous allons donc simplement créer deux gestionnaires de rendu de texte. Il existe également d'autres types de rendu soit:

- [GtkCellRendererAccel](#)
- [GtkCellRendererCombo](#)
- [GtkCellRendererPixbuf](#)
- [GtkCellRendererProgress](#)
- [GtkCellRendererSpin](#)
- [GtkCellRendererText](#)
- [GtkCellRendererToggle](#)

La plupart d'entre eux sont en générale utilisés pour rendre des cellules d'un GtkTreeView mais on peut très bien utiliser par exemple les *GtkCellRendererPixbuf* pour afficher une image pour chaque élément d'un GtkComboBox ! Les autres n'ont pas vraiment d'intérêt pour ce type de widget. Nous, nous allons simplement voir comment mettre en place deux objets de rendu de texte:

Mise en place des objets de rendu des cellules

```
GtkCellRenderer * p_cell = NULL;

p_cell = gtk_cell_renderer_text_new ();
gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (p_combo), p_cell, FALSE);
gtk_cell_layout_set_attributes (
    GTK_CELL_LAYOUT (p_combo),
    p_cell, "text", 0,
    NULL
);

p_cell = gtk_cell_renderer_text_new ();
gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (p_combo), p_cell, FALSE);
gtk_cell_layout_set_attributes (
    GTK_CELL_LAYOUT (p_combo),
    p_cell, "text", 1,
    NULL
);
```

Comme vous pourrez le constater, il faut mettre en place un objet par colonne. Vous pouvez également voir qu'il n'est pas nécessaire de garder un pointeur sur chacun d'eux car par la suite nous n'y accéderons plus !

Voici les étapes une par une pour la création et la mise en place d'un objet de rendu:

- 1 Créer un objet de type `GtkCellRender` suivant le type de données à rendre.
- 2 L'insertion de l'objet dans le `GtkComboBox`. Ceci fonctionne en fait comme les `GtkBox` et il existe donc une autre fonction d'ajout qui est `gtk_cell_layout_pack_end` et qui permet l'ajout par la fin !
- 3 Cette dernière étape permet de préciser les attributs pour le rendu des colonnes par rapport à leur type. Ici vu que le type pour un entier n'existe pas, nous utilisons simplement le type `"text"` !

II-A-4 - Remplissage du GtkComboBox avec des données

Le remplissage est une des actions les plus simple à réaliser ! Il nous faut dans un premier temps insérer le nouvel élément dans la liste avec une des fonctions suivantes:

- [gtk_list_store_insert](#)
- [gtk_list_store_insert_before](#)
- [gtk_list_store_insert_after](#)
- [gtk_list_store_insert_with_values](#)
- [gtk_list_store_insert_with_valuesv](#)
- [gtk_list_store_prepend](#)
- [gtk_list_store_append](#)

Ces fonctions, en mettant en place le nouvel élément, initialisent également un objet de type `GtkTreeIter`. Cet objet nous est nécessaire car il contient diverses données comme l'emplacement (*ou le chemin*) exacte où se situe l'élément. Il convient également d'utiliser ce type d'objet directement après son initialisation car il devient très vite invalide, rien que le fait de changer l'élément courant rend invalide un `GtkTreeIter` !

Voici un exemple d'insertion correcte d'un nouvel élément dans notre `GtkComboBox`:

Insertion d'un nouvel élément

```
/* Ajout d'un nouvel element dans le magasin. */
gtk_list_store_append (p_model, & iter);

/* On remplit le nouvel element. */
gtk_list_store_set (
    p_model, & iter,
    0, 10, 1, "Une action...",
    -1
);
```

Ici, nous avons inséré un élément en fin de liste puis nous l'avons rempli. La fonction de remplissage prend dans l'ordre, comme deux premiers paramètres, le pointeur vers le modèle que nous avons créé au début de ce tutoriel et l'adresse du `GtkTreeIter` nouvellement initialisé grâce à la fonction `gtk_list_store_append`. La suite n'est pas plus compliquée, on donne une suite de colonnes à remplir par rapport aux colonnes créées précédemment soit ici, on commence toujours par l'index de la colonne qu'on veut remplir puis le type de la donnée à insérer (`0, 10`) et ainsi de suite. La liste doit être terminée par la valeur `-1` ce qui permet de marquer la fin de la liste !

On utilise également assez souvent le terme de magasin en faisant référence au modèle car c'est ce que c'est, un magasin de données !

II-A-5 - Récupérer les données de l'élément sélectionné

Le plus gros du travail se fait à partir d'ici ! En effet, créer un modèle de données personnalisé implique également

des fonctions de traitement personnelles comme la récupération des données de l'élément courant de notre *GtkComboBox*. Toujours par rapport à notre exemple, le plus simple que nous pourrions faire est de créer une structure contenant les données que nous voulons récupérer. Voici notre structure pour notre exemple:

Structure pour notre modèle de données

```
typedef struct
{
    gint    index;
    gchar * p_text;
}
combo_data_st;
```

Nous stockerons ici ce que contient notre modèle de données, un *entier signé* et une chaîne de caractères. Ici c'est le même comportement que nous avons avec les *GtkTreeIter* à savoir que la structure contenant des données de l'élément sélectionné deviendra très vite invalide !

Il nous faut donc désormais notre fonction qui ira récupérer les données de l'élément courant, la voici:

Fonction qui récupère les données de l'élément courant

```
combo_data_st get_active_data (GtkComboBox * p_combo)
{
    GtkTreeModel * p_model = NULL;
    GtkTreeIter  iter;
    combo_data_st p_st;

    /* 1-. On recupere le modele qu'on a cree. */
    p_model = gtk_combo_box_get_model (p_combo);

    /* 2-. On recupere le GtkTreeIter de l'element courant. */
    if (gtk_combo_box_get_active_iter (p_combo, & iter))
    {
        /*
         * 3-.
         * On recupere les donnees de l'element courant a savoir
         * un entier et une chaine de caracteres.
         */
        gtk_tree_model_get (
            p_model,
            & iter,
            0, & p_st.index,
            1, & p_st.p_text,
            -1
        );
    }

    return p_st;
}
```

Qu'avons-nous ici... Nous possédons maintenant une fonction permettant de récupérer les données de l'élément courant de notre *GtkComboBox*. Cette fonction prend comme il se doit, une pointeur sur notre widget et renvoie une structure de données par rapport à celle que nous avons définie plus haut.

Décortiquons les actions qu'elle effectue pour récupérer nos données:

- 1 On commence par récupérer le *GtkTreeModel* de notre *GtkComboBox* à savoir tout simplement notre modèle *GtkListStore* !
- 2 Il nous faut également récupérer le *GtkTreeIter* de l'élément courant afin de pouvoir en récupérer les données qu'il contient, c'est ce que nous faisons au sein de la condition **if**. En effet, la fonction que nous utilisons ici renvoie une valeur de type **gboolean**, il nous est donc possible de facilement l'insérer dans une condition

II-A-6 - Suppression de l'élément sélectionné

La suppression de l'élément courant se fait à peu près de la même façon que la récupération des données. Les étapes **1** et **2** décrites ci-dessus sont donc identiques, seule la dernière change car celle-ci aura pour effet la suppression, voyons tout de suite comment créer une telle fonction:

Suppression de l'élément sélectionné de notre GtkComboBox

```
void remove_active_item (GtkComboBox * p_combo)
{
    GtkTreeModel * p_model = NULL;
    GtkTreeIter   iter;

    /* On recupere le modele qu'on a cree. */
    p_model = gtk_combo_box_get_model (p_combo);

    /* On recupere le GtkTreeIter de l'element courant. */
    if (gtk_combo_box_get_active_iter (p_combo, & iter))
    {
        /* On supprime l'element courant. */
        gtk_list_store_remove (GTK_LIST_STORE (p_model), & iter);
    }
}
```

Comme on pouvait s'y attendre, seule la dernière étape change, nous appelons ici la fonction [gtk_list_store_remove](#) afin de supprimer l'élément courant d'après son *GtkTreeIter*.

II-A-7 - Vider complètement notre GtkComboBox

C'est sans doute la chose la plus facile car il n'y a pas plus de deux instructions. Pour vider complètement notre *GtkComboBox*, nous utiliserons la fonction [gtk_list_store_clear](#) Il faut bien sûr récupérer au préalable un pointeur sur notre modèle de données comme nous l'avons fait dans les deux dernières sections mais c'est tout !

II-A-8 - Code source d'un exemple

Voici un code source complet mettant en application ce que nous avons vu dans cette section. Vous pouvez [télécharger](#) le code source complet (*avec un projet Code::Blocks Linux*) présenté ci-dessous:

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

/*
 * La structure concue pour contenir toutes les donnees de
 * l'element courant de notre GtkComboBox d'apres le modele
 * que nous lui avons cree.
 */
typedef struct
{
    gint    index;
    gchar * p_text;
}
combo_data_st;

/*
 * Fonction qui recupere les donnees de l'element courant affiche.
 */
static combo_data_st get_active_data (GtkComboBox * p_combo)
{
    GtkTreeModel * p_model = NULL;
    GtkTreeIter   iter;
```

```

combo_data_st    p_st;

/* On recupere le modele qu'on a cree. */
p_model = gtk_combo_box_get_model (p_combo);

/* On recupere le GtkTreeIter de l'element courant. */
if (gtk_combo_box_get_active_iter (p_combo, & iter))
{
    /*
     * On recupere les donnees de l'element courant a savoir
     * un entier et une chaine de caracteres.
     */
    gtk_tree_model_get (
        p_model,
        & iter,
        0, & p_st.index,
        1, & p_st.p_text,
        -1
    );
}

return p_st;
}

/*
 * Fonction qui supprime l'element courant du GtkComboBox.
 */
static void remove_active_item (GtkComboBox * p_combo)
{
    GtkTreeModel * p_model = NULL;
    GtkTreeIter   iter;

    /* On recupere le modele qu'on a cree. */
    p_model = gtk_combo_box_get_model (p_combo);

    /* On recupere le GtkTreeIter de l'element courant. */
    if (gtk_combo_box_get_active_iter (p_combo, & iter))
    {
        /* On supprime l'element courant. */
        gtk_list_store_remove (GTK_LIST_STORE (p_model), & iter);
    }
}

/*
 * Callback des GtkButton.
 */
static void cb_show (GtkWidget * p_wid, gpointer p_data)
{
    GtkComboBox * p_combo = p_data;
    combo_data_st p_st;

    p_st = get_active_data (p_combo);
    g_printf ("Element courant : %d%s\n", p_st.index, p_st.p_text);

    (void) p_wid;
}

static void cb_remove (GtkWidget * p_wid, gpointer p_data)
{
    GtkComboBox * p_combo = p_data;

    remove_active_item (p_combo);
    (void) p_wid;
}

int main (int argc, char ** argv)
{
    GtkWidget * p_win    = NULL;
    GtkWidget * p_vbox   = NULL;

```

```

GtkWidget * p_button[2];
GtkWidget * p_combo      = NULL;

gtk_init (& argc, & argv);

/*
 * Creation de la fenetre principale.
 */
p_win = gtk_window_new          (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title           (GTK_WINDOW (p_win), "GtkComboBox...");
gtk_window_set_default_size    (GTK_WINDOW (p_win), 250, 150);
gtk_container_set_border_width (GTK_CONTAINER (p_win), 5);
gtk_window_set_position       (GTK_WINDOW (p_win), GTK_WIN_POS_CENTER);

/* Ajout du conteneur principal et des boutons. */
p_vbox = gtk_vbox_new          (TRUE, 5);
gtk_container_add              (GTK_CONTAINER (p_win), p_vbox);

p_button[0] = gtk_button_new_with_label ("Recuperer l'element courant...");
gtk_box_pack_start (GTK_BOX (p_vbox), p_button[0], TRUE, TRUE, 2);
p_button[1] = gtk_button_new_with_label ("Supprimer l'element courant...");
gtk_box_pack_start (GTK_BOX (p_vbox), p_button[1], TRUE, TRUE, 2);

/*
 * Creation d'un GtkComboBox avec un GtkTreeModel personnalise.
 */
{
    GtkListStore      * p_model = NULL;
    GtkCellRenderer  * p_cell  = NULL;
    GtkTreeIter iter;
    gint i = 0;

    /*
     * Creation d'un modele pour le GtkComboBox. Ce modele permettra d'afficher
     * sur une premiere colonne un chiffre et a cote un texte.
     */
    p_model = gtk_list_store_new (2, G_TYPE_INT, G_TYPE_STRING);

    /*
     * Creation d'un GtkComboBox avec son modele.
     */
    p_combo = gtk_combo_box_new_with_model (GTK_TREE_MODEL (p_model));
    gtk_box_pack_start (GTK_BOX (p_vbox), p_combo, TRUE, TRUE, 0);

    /*
     * On met en place un gestionnaire de rendu des cellules. Il faut en
     * creer un par colonne et suivant le type de chaque donnees.
     *
     * Ici, on a deux colonnes de texte, vu que les colonne d'entiers
     * n'existent pas on prend ca en compte comme du texte.
     */
    p_cell = gtk_cell_renderer_text_new ();
    gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (p_combo), p_cell, FALSE);
    gtk_cell_layout_set_attributes (
        GTK_CELL_LAYOUT (p_combo),
        p_cell, "text", 0,
        NULL
    );

    p_cell = gtk_cell_renderer_text_new ();
    gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (p_combo), p_cell, FALSE);
    gtk_cell_layout_set_attributes (
        GTK_CELL_LAYOUT (p_combo),
        p_cell, "text", 1,
        NULL
    );

    /*
     * Remplissage du magasin.
     */
    for (i = 0; i < 10; i++)
    {

```

```

    /* Ajout d'un nouvel element dans le magasin. */
    gtk_list_store_append (p_model, & iter);

    /* On remplit le nouvel element. */
    gtk_list_store_set (
        p_model, & iter,
        0, i + 1, 1, " - Element...",
        -1
    );
}

/*
 * On affiche un element par default. L'index commence à zero.
 */
gtk_combo_box_set_active (GTK_COMBO_BOX (p_combo), 2);
}

/*
 * On definit les callback :
 */
g_signal_connect (
    G_OBJECT (p_win), "destroy", G_CALLBACK (gtk_main_quit), NULL
);
g_signal_connect (
    G_OBJECT (p_button[0]), "clicked", G_CALLBACK (cb_show), p_combo
);
g_signal_connect (
    G_OBJECT (p_button[1]), "clicked", G_CALLBACK (cb_remove), p_combo
);

gtk_widget_show_all (p_win);
gtk_main ();

return 0;
}

```

II-B - Mise en oeuvre prédéfinie

Cette mise en oeuvre du widget permet de mettre en place le plus simplement possible un GtkComboBox contenant uniquement du texte comme information, chose la plus courante pour ce type de widget ! Des fonctions supplémentaires permettant également de vous faciliter la tâche ont été de ce fait ajoutées, voyons tout cela...

II-B-1 - Initialisation du widget

L'initialisation se fait tout simplement avec la fonction `gtk_combo_box_new_text` comme ceci:

Initialisation d'un GtkComboBox en mode texte

```
GtkWidget * p_combo = gtk_combo_box_new_text ();
```

II-B-2 - Remplissage du GtkComboBox avec des données

Si vous créez un *GtkComboBox* de cette manière, vous devez également utiliser les fonctions qui sont prévues pour cette version du widget. Vous avez le choix entre ces différentes fonctions:

- [gtk_combo_box_append_text](#) (*ajout en fin de liste*)
- [gtk_combo_box_insert_text](#) (*ajout à la position donnée*)
- [gtk_combo_box_prepend_text](#) (*ajout en début de liste*)

Voici par exemple comment ajouter 6 éléments dans notre liste:

Insertion de données texte

```
gtk_combo_box_append_text (GTK_COMBO_BOX (p_combo), "Element ... 4");
gtk_combo_box_append_text (GTK_COMBO_BOX (p_combo), "Element ... 5");
gtk_combo_box_append_text (GTK_COMBO_BOX (p_combo), "Element ... 6");
gtk_combo_box_prepend_text (GTK_COMBO_BOX (p_combo), "Element ... 3");
gtk_combo_box_prepend_text (GTK_COMBO_BOX (p_combo), "Element ... 2");
gtk_combo_box_prepend_text (GTK_COMBO_BOX (p_combo), "Element ... 1");
```

Dans les trois première lignes, nous insérons les éléments l'un en-dessous de l'autre et dans les trois lignes suivantes, nous insérons les éléments toujours en début de liste soit l'un au-dessus de l'autre !

II-B-3 - Récupérer le texte de l'élément sélectionné

C'est par une simple fonction que nous pouvons récupérer le texte de l'élément sélectionné, il s'agit de la fonction [gtk_combo_box_get_active_text](#)

Récupérer le texte de l'élément sélectionné

```
gchar * p_text = gtk_combo_box_get_active_text (GTK_COMBO_BOX (p_combo));
```

Cette fonction renvoie une chaîne de caractères nouvellement allouée ce qui implique que vous devrez libérer cet espace mémoire vous même avec la fonction [g_free](#) lorsque vous en aurez plus besoin !

Cette fonction n'est disponible qu'à partir de la version 2.6 de GTK+. Une fonction alternative peut être mise en place, voir ci-dessous !

Si vous possédez une version antérieure à la version 2.6 de GTK+, il vous faut faire par vos propres moyens pour pouvoir récupérer le texte de l'élément sélectionné ! Voici une implémentation d'une fonction permettant de faire ceci:

```
#if GTK_CHECK_VERSION (2, 6, 0)
#define combo_box_active_get_text (combo_box) gtk_combo_box_get_active_text (combo_box)
#else
char *combo_box_active_get_text (GtkComboBox *combo_box)
{
    gchar *s_text = NULL;
    gboolean b_ret = FALSE;
    GtkTreeIter iter;

    g_return_val_if_fail (combo_box != NULL, s_text);

    b_ret = gtk_combo_box_get_active_iter (combo_box, &iter);
    if (b_ret)
    {
        GtkTreeModel *p_model = NULL;

        p_model = gtk_combo_box_get_model (combo_box);
        if (p_model != NULL)
        {
            gtk_tree_model_get (p_model, &iter, 0, &s_text, -1);
        }
    }
    return s_text;
}
#endif
```

Ce code fonctionne ainsi... Si la version est égale ou supérieure à la version 2.6.0 de GTK+ alors, c'est la fonction officielle de GTK+ qui sera appelée et dans le cas contraire, ce sera la fonction [combo_box_active_get_text](#) qui sera utilisée !

II-B-4 - Suppression de l'élément sélectionné

La suppression reste également facile mais nécessite un peu plus d'effort. En effet, il faut tout d'abord récupérer l'index de l'élément courant avec la fonction [gtk_combo_box_get_active](#) pour pouvoir le supprimer car la fonction de suppression [gtk_combo_box_remove_text](#) exige l'index de ce dernier !

Voici comment mettre ceci en place facilement:

Supprimer l'élément sélectionné

```
gint index = -1;

index = gtk_combo_box_get_active (GTK_COMBO_BOX (p_combo));
gtk_combo_box_remove_text (GTK_COMBO_BOX (p_combo), index);
```

II-B-5 - Code source d'un exemple

Voici un code source complet mettant en application ce que nous avons vu dans cette section. Vous pouvez [télécharger](#) le code source complet (avec un projet `Code::Blocks Linux`) présenté ci-dessous:

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

/*
 * Callback des GtkButton.
 */
static void cb_show (GtkWidget * p_wid, gpointer p_data)
{
    GtkComboBox * p_combo = p_data;
    gchar * p_text = NULL;

    p_text = gtk_combo_box_get_active_text (GTK_COMBO_BOX (p_combo));
    g_printf ("%s\n", p_text);
    g_free (p_text);

    (void) p_wid;
}

static void cb_remove (GtkWidget * p_wid, gpointer p_data)
{
    GtkComboBox * p_combo = p_data;
    gint index = -1;

    index = gtk_combo_box_get_active (GTK_COMBO_BOX (p_combo));
    gtk_combo_box_remove_text (GTK_COMBO_BOX (p_combo), index);

    (void) p_wid;
}

int main (int argc, char ** argv)
{
    GtkWidget * p_win = NULL;
    GtkWidget * p_vbox = NULL;
    GtkWidget * p_button[2];
    GtkWidget * p_combo = NULL;

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

```

/*
 * Creation de la fenetre principale.
 */
p_win = gtk_window_new          (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title           (GTK_WINDOW (p_win), "GtkComboBox...");
gtk_window_set_default_size    (GTK_WINDOW (p_win), 250, 150);
gtk_container_set_border_width (GTK_CONTAINER (p_win), 5);
gtk_window_set_position        (GTK_WINDOW (p_win), GTK_WIN_POS_CENTER);

/* Ajout du conteneur principal et des boutons. */
p_vbox = gtk_vbox_new          (TRUE, 5);
gtk_container_add              (GTK_CONTAINER (p_win), p_vbox);

p_button[0] = gtk_button_new_with_label ("Recuperer l'element courant...");
gtk_box_pack_start            (GTK_BOX (p_vbox), p_button[0], TRUE, TRUE, 2);
p_button[1] = gtk_button_new_with_label ("Supprimer l'element courant...");
gtk_box_pack_start            (GTK_BOX (p_vbox), p_button[1], TRUE, TRUE, 2);

/*
 * Creation d'un GtkComboBox predefinit et ajout dans le conteneur.
 */
p_combo = gtk_combo_box_new_text ();
gtk_box_pack_end              (GTK_BOX (p_vbox), p_combo, TRUE, TRUE, 2);

/*
 * Insertion de donnees dans la liste du widget.
 */
gtk_combo_box_append_text     (GTK_COMBO_BOX (p_combo), "Element ... 4");
gtk_combo_box_append_text     (GTK_COMBO_BOX (p_combo), "Element ... 5");
gtk_combo_box_append_text     (GTK_COMBO_BOX (p_combo), "Element ... 6");
gtk_combo_box_prepend_text    (GTK_COMBO_BOX (p_combo), "Element ... 3");
gtk_combo_box_prepend_text    (GTK_COMBO_BOX (p_combo), "Element ... 2");
gtk_combo_box_prepend_text    (GTK_COMBO_BOX (p_combo), "Element ... 1");

/*
 * On affiche un element par default. L'index commence à zero.
 */
gtk_combo_box_set_active      (GTK_COMBO_BOX (p_combo), 2);

/*
 * On definit les callback :
 */
g_signal_connect (
    G_OBJECT (p_win), "destroy", G_CALLBACK (gtk_main_quit), NULL
);
g_signal_connect (
    G_OBJECT (p_button[0]), "clicked", G_CALLBACK (cb_show), p_combo
);
g_signal_connect (
    G_OBJECT (p_button[1]), "clicked", G_CALLBACK (cb_remove), p_combo
);

gtk_widget_show_all (p_win);
gtk_main ();

return 0;
}

```

III - GtkComboBoxEntry

III-A - Les petites différences de la mise en oeuvre personnalisée...

La mise en oeuvre avec des colonnes personnalisées d'un *GtkComboBoxEntry* change un petit peu par rapport au *GtkComboBox*. Ce que nous allons voir ici sont les petites différences de quelques actions et comportements mais sinon tout ce que nous avons vu jusque là est également applicable au *GtkComboBoxEntry*

III-A-1 - Pour l'initialisation du widget

L'initialisation de ce widget dérivé se fait par ses propres fonctions de création soit:

- [gtk_combo_box_entry_new](#)
- [gtk_combo_box_entry_new_with_model](#)
- [gtk_combo_box_entry_new_text](#)

Ici nous utiliserons la fonction *gtk_combo_box_entry_new_with_model* car elle est un peu particulière ! En effet, nous pouvons remarquer un second argument qui peut vous paraître étrange, même en lisant la description. C'est l'index de la colonne contenant le texte que votre widget affichera dans la zone de texte, c'est un widget de type *GtkEntry*.

Si nous conservons le même modèle que dans l'exemple précédent, notre colonne de texte est la colonne d'indice **1** et donc notre initialisation du widget se fera comme suit:

Initialisation d'un GtkComboBoxEntry avec un modèle

```
GtkWidget * p_combo = gtk_combo_box_entry_new_with_model (GTK_TREE_MODEL (p_model), 1);
```

III-A-2 - Pour la création des objets de rendu

Si nous reprenons l'exemple précédent soit une colonne de texte et une pour des valeurs entières, nous devons simplement mettre un seul objet de rendu de cellule pour la colonne contenant les valeurs numériques ! *Pourquoi ? (c'est sans doute la question que vous vous posez...)* Tout simplement car le widget contenant un *GtkEntry*, gère lui même la colonne de texte comme on a dû la préciser dans la section ci-dessus ! Mettre également un objet de rendu pour cette colonne reviendrait à afficher dans ce cas deux fois la même donnée, à part cela rien d'autre ne change comme vous pouvez le remarquer dans l'exemple ci-dessous:

Objet de rendu pour seulement la colonne des entiers.

```
p_cell = gtk_cell_renderer_text_new ();
gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (p_combo), p_cell, FALSE);
gtk_cell_layout_set_attributes (
    GTK_CELL_LAYOUT (p_combo),
    p_cell, "text", 0,
    NULL
);
```

III-A-3 - Pour la suppression de l'élément sélectionné

Le code de la fonction de suppression de l'élément sélectionné que nous avons fait dans le chapitre sur le *GtkComboBox* est identique à un point près ! En effet, la seule chose en plus qu'il va falloir gérer soi même, c'est le vidage de la zone de texte, ce qui ne se fait pas automatiquement dans notre cas. Pour accéder au widget

GtkEntry de notre *GtkComboBoxEntry*, il va falloir utiliser quelques macros comme cela est expliqué dans la référence officielle de GTK+ pour la description de la fonction *gtk_combo_box_entry_new_with_model* que nous avons utilisée plus haut:

*Creates a new GtkComboBoxEntry which has a GtkEntry as child and a list of strings as popup. You can get the GtkEntry from a GtkComboBoxEntry using **GTK_ENTRY (GTK_BIN (combo_box_entry)->child)**. To add and remove strings from the list, just modify model using its data manipulation API.*

Il va de soi qu'il faut utiliser des fonctions relatives au widget *GtkEntry* pour faire ce genre d'actions comme effacer la zone de saisie, voici donc un exemple sur la façon dont on s'y prend:

Effacement de l'élément courant d'un GtkComboBoxentry

```

/*
 * Fonction qui supprime l'element courant du GtkComboBox.
 */
static void remove_active_item (GtkComboBox * p_combo)
{
    GtkTreeModel * p_model = NULL;
    GtkTreeIter   iter;

    /* On recupere le modele qu'on a cree. */
    p_model = gtk_combo_box_get_model (p_combo);

    /* On recupere le GtkTreeIter de l'element courant. */
    if (gtk_combo_box_get_active_iter (p_combo, & iter))
    {
        /* On supprime l'element courant. */
        gtk_list_store_remove (GTK_LIST_STORE (p_model), & iter);

        /* Il faut aussi vider la zone de texte. */
        gtk_entry_set_text (GTK_ENTRY (GTK_BIN (p_combo)->child), "");
    }
}

```

III-A-4 - Code source d'un exemple

Voici un code source complet mettant en application ce que nous avons vu dans cette section. Vous pouvez [télécharger](#) le code source complet (avec un projet *Code::Blocks Linux*) présenté ci-dessous:

```

#include <gtk/gtk.h>
#include <glib/gprintf.h>

/*
 * La structure concue pour contenir toutes les donnees de
 * l'element courant de notre GtkComboBox d'apres le modele
 * que nous lui avons cree.
 */
typedef struct
{
    gint    index;
    gchar * p_text;
}
combo_data_st;

/*
 * Fonction qui recupere les donnees de l'element courant affiche.
 */
static combo_data_st get_active_data (GtkComboBox * p_combo)
{
    GtkTreeModel * p_model = NULL;
    GtkTreeIter   iter;
    combo_data_st p_st;

```

```

/* On recupere le modele qu'on a cree. */
p_model = gtk_combo_box_get_model (p_combo);

/* On recupere le GtkTreeIter de l'element courant. */
if (gtk_combo_box_get_active_iter (p_combo, & iter))
{
    /*
     * On recupere les donnees de l'element courant a savoir
     * un entier et une chaine de caracteres.
     */
    gtk_tree_model_get (
        p_model,
        & iter,
        0, & p_st.index,
        1, & p_st.p_text,
        -1
    );
}

return p_st;
}

/*
 * Fonction qui supprime l'element courant du GtkComboBox.
 */
static void remove_active_item (GtkComboBox * p_combo)
{
    GtkTreeModel * p_model = NULL;
    GtkTreeIter   iter;

    /* On recupere le modele qu'on a cree. */
    p_model = gtk_combo_box_get_model (p_combo);

    /* On recupere le GtkTreeIter de l'element courant. */
    if (gtk_combo_box_get_active_iter (p_combo, & iter))
    {
        /* On supprime l'element courant. */
        gtk_list_store_remove (GTK_LIST_STORE (p_model), & iter);

        /* Il faut aussi vider la zone de texte. */
        gtk_entry_set_text (GTK_ENTRY (GTK_BIN (p_combo)->child), "");
    }
}

/*
 * Callback des GtkButton.
 */
static void cb_show (GtkWidget * p_wid, gpointer p_data)
{
    GtkComboBox * p_combo = p_data;
    combo_data_st p_st;

    p_st = get_active_data (p_combo);
    g_printf ("Element courant : %s%d\n", p_st.p_text, p_st.index);

    (void) p_wid;
}

static void cb_remove (GtkWidget * p_wid, gpointer p_data)
{
    GtkComboBox * p_combo = p_data;

    remove_active_item (p_combo);
    (void) p_wid;
}

int main (int argc, char ** argv)
{
    GtkWidget * p_win      = NULL;
    GtkWidget * p_vbox     = NULL;

```

```

GtkWidget * p_button[2];
GtkWidget * p_combo      = NULL;

gtk_init (& argc, & argv);

/*
 * Creation de la fenetre principale.
 */
p_win = gtk_window_new          (GTK_WINDOW_TOPLEVEL);
gtk_window_set_title           (GTK_WINDOW (p_win), "GtkComboBox...");
gtk_window_set_default_size    (GTK_WINDOW (p_win), 250, 150);
gtk_container_set_border_width (GTK_CONTAINER (p_win), 5);
gtk_window_set_position        (GTK_WINDOW (p_win), GTK_WIN_POS_CENTER);

/* Ajout du conteneur principal et des boutons. */
p_vbox = gtk_vbox_new          (TRUE, 5);
gtk_container_add              (GTK_CONTAINER (p_win), p_vbox);

p_button[0] = gtk_button_new_with_label ("Recuperer l'element courant...");
gtk_box_pack_start (GTK_BOX (p_vbox), p_button[0], TRUE, TRUE, 2);
p_button[1] = gtk_button_new_with_label ("Supprimer l'element courant...");
gtk_box_pack_start (GTK_BOX (p_vbox), p_button[1], TRUE, TRUE, 2);

/*
 * Creation d'un GtkComboBoxEntry avec un GtkTreeModel personnalis e.
 */
{
    GtkListStore      * p_model = NULL;
    GtkCellRenderer  * p_cell   = NULL;
    GtkTreeIter iter;
    gint i = 0;

    /*
     * Creation d'un model pour le GtkComboBoxEntry. Ce model permettra
     * d'afficher sur une premiere colonne un chiffre et a cote un texte.
     */
    p_model = gtk_list_store_new (2, G_TYPE_INT, G_TYPE_STRING);

    /*
     * Creation d'un GtkComboBoxEntry avec son model.
     */
    p_combo = gtk_combo_box_entry_new_with_model (GTK_TREE_MODEL (p_model), 1);
    gtk_box_pack_start (GTK_BOX (p_vbox), p_combo, TRUE, TRUE, 0);

    /*
     * Objet de rendu pour seulement la seconde colonne.
     */
    p_cell = gtk_cell_renderer_text_new ();
    gtk_cell_layout_pack_start (GTK_CELL_LAYOUT (p_combo), p_cell, FALSE);
    gtk_cell_layout_set_attributes (
        GTK_CELL_LAYOUT (p_combo),
        p_cell, "text", 0,
        NULL
    );

    /*
     * Remplissage du magasin.
     */
    for (i = 0; i < 10; i++)
    {
        /* Ajout d'un nouvel element dans le magasin. */
        gtk_list_store_append (p_model, & iter);

        /* On remplit le nouvel element. */
        gtk_list_store_set (
            p_model, & iter,
            0, i + 1, 1, "Element... ",
            -1
        );
    }
}

```

```

    /*
     * On affiche un element par default. L'index commence à zero.
     */
    gtk_combo_box_set_active (GTK_COMBO_BOX (p_combo), 2);
}

/*
 * On definit les callback :
 */
g_signal_connect (
    G_OBJECT (p_win), "destroy", G_CALLBACK (gtk_main_quit), NULL
);
g_signal_connect (
    G_OBJECT (p_button[0]), "clicked", G_CALLBACK (cb_show), p_combo
);
g_signal_connect (
    G_OBJECT (p_button[1]), "clicked", G_CALLBACK (cb_remove), p_combo
);

gtk_widget_show_all (p_win);
gtk_main ();

return 0;
}

```

III-B - Mise en oeuvre prédéfinie

Tout comme le widget *GtkComboBox*, *GtkComboBoxEntry* possède également une fonction qui nous permet de l'initialiser par défaut avec une seule donnée texte soit la fonction [gtk_combo_box_entry_new_text](#). Les fonctions décrites plus haut pour l'insertion de texte dans un widget de type *GtkComboBox* peuvent également être utilisées ici !

III-C - La zone de texte du GtkComboBoxEntry

III-C-1 - Comment y accéder ?

Comme nous l'avons vu dans la section *III-A-3*, il faut utiliser plusieurs macros pour accéder au widget enfant du *GtkComboBoxEntry* soit:

Accéder aux GtkEntry d'un GtkComboBoxEntry

```
GtkWidget * p_entry = GTK_ENTRY (GTK_BIN (combo_box_entry)->child);
```

III-C-2 - Comment valider la saisie et l'insérer dans la liste ?

Le widget étant un *GtkEntry*, il faut donc utiliser les fonctions et signaux de ce widget précis. Dans notre cas, il faut gérer le signal **"activate"** de ce widget ! Voici un exemple:

Valider le GtkEntry du GtkComboBoxEntry avec la touche Entrée

```

/*
 * Enregistrement du signal "activate" pour le GtkEntry de notre
 * liste GtkComboBoxEntry.
 */
g_signal_connect (
    G_OBJECT (GTK_BIN (p_combo)->child),
    "activate",
    G_CALLBACK (entry_activate),
    (gpointer) p_combo
);

```

Valider le GtkEntry du GtkComboBoxEntry avec la touche Entrée

```
...
/*
 * Callback qui recupere le contenu de la zone de texte et l'insere
 * dans la liste du GtkComboBoxEntry passe dans le second argument.
 */
void entry_activate (GtkEntry * p_entry, gpointer p_data)
{
    GtkComboBox * p_combo = p_data;

    gtk_combo_box_append_text (
        GTK_COMBO_BOX (p_combo),
        gtk_entry_get_text (GTK_ENTRY (p_entry))
    );
}
```

Cet exemple prétend que le GtkComboBoxEntry est initialisé avec la fonction `gtk_combo_box_entry_new_text` !

III-C-3 - Comment bloquer la saisie utilisateur ?

Pour empêcher la saisie par l'utilisateur sur la zone de texte du *GtkComboBoxEntry*, il faut comme précédemment passer directement par le widget *GtkEntry*. Le widget *GtkEntry* étant une sorte de [GtkEditable](#), nous pouvons utiliser la fonction [gtk_editable_set_editable](#) pour arriver à nos fins soit l'exemple suivant bloque la zone de saisie en lecture seule:

Empêcher la saisie utilisateur sur le GtkEntry du GtkComboBoxEntry

```
gtk_editable_set_editable (GTK_EDITABLE (GTK_BIN (p_combo)->child), FALSE);
```

IV - Fonctions diverses

Voici diverses petites fonctions qui peuvent être utilisées sur les deux widgets soit *GtkComboBox* et *GtkComboBoxEntry*, certaines d'entre elles peuvent s'avérer plus ou moins pratique tandis que d'autres sont plus des gadgets visuel, à vous de choisir, en voici quelques-unes...

IV-A - Afficher un élément par défaut

Il est possible d'afficher un élément par défaut dans un *GtkComboBox* et *GtkComboBoxEntry*. Pour y parvenir, il vous faut utiliser la fonction [gtk_combo_box_set_active](#)

IV-B - Permettre la mise en fenêtre de la liste déroulante

Certains d'entre vous se poseront très certainement des questions mais vous avez sans doute déjà dû voir ceci avec les menus. Rappelez-vous, les petits traits tout en haut que l'on peut rajouter et si on clique dessus ça ferme le menu et le met dans une petite fenêtre... C'est exactement ceci, c'est aussi connu sous le nom de "*tearoffs*"!

Cela peut se faire avec la fonction [gtk_combo_box_set_add_tearoffs](#). Si vous mettez le second argument sur **TRUE** vous affichez donc les *tearoffs* ! Voici une illustration de ce que cela peut donner:



IV-C - Afficher les données sur plusieurs colonnes

Il vous est également possible d'afficher les éléments sur plusieurs colonnes, ce qui peut s'avérer très pratique lorsque vous disposez beaucoup d'éléments dans une liste déroulante. Vous pouvez réaliser ceci grâce à la fonction [gtk_combo_box_set_wrap_width](#)

Voici une illustration de ce que cela donne:



V - Bonus

Hé oui vous avez droit un petit bonus de ma part. Je vous offre ici un exemple d'un *GtkComboBox* pouvant afficher des images et du texte ! Cela peut s'avérer assez pratique comme par exemple faire une liste d'actions, embellir un peu le tout avec des images peut être intéressant !

Voici ce que cela donne en image:



Je vous offre même le code source complet (*avec un projet Code::Blocks Linux*) pour que vous puissiez voir comment mettre en place un *GtkComboBox* de cette manière, suivez le [lien](#) !

VI - Remerciements

Un très grand merci à [troumad](#) et à [wichtounet](#) pour la relecture attentive et la correction de ce tutoriel !