FACULTÉ DES SCIENCES ET TECHNIQUES
DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Ingénierie Logicielle et Intégration
des Systèmes Informatiques

Compte rendu

Création des macros && Gestion du fichier

Filière Ingénieur :
Ingénierie Logicielle et Intégration des Systèmes Informatiques

Réalisé par :
DAOUI NAIMA
ADHAIM LOKMANE
OUTGOUGA Jalal eddine
ZADDI Abdelmajid
Encadré par :
Prof. Bekkhoucha

2022/2023

Table de matières

Introduction :

La première partie de ce rapport est consacrée aux structures de données et aux macros dans le cadre de la programmation avec la bibliothèque GTK3. Nous allons voir comment utiliser ces structures de données et ces macros pour créer des interfaces utilisateur efficaces et interactives.
La deuxième partie de ce rapport se concentre sur la gestion d'un fichier XML pour charger et sauvegarder les propriétés des éléments de l'interface utilisateur. Nous allons voir comment utiliser les bibliothèques de manipulation XML pour lire et écrire des fichiers XML.
En combinant ces deux parties, nous allons créer des interfaces utilisateur complexes et dynamiques avec la possibilité de charger et de sauvegarder les propriétés des éléments de l'interface utilisateur dans un fichier XML

Partie 1 - Les structures et les les macros :

I- Container :

1-Le Box :

✓ La structure de données

```
typedef struct
{

  GtkWidget* box;
  /* un booléen indiquant si les éléments  contenus dans la
     boîte doivent être de la même taille*/
  gboolean homogeneous ;
  /*un entier indiquant l'espace à ajouter entre les éléments
    Contenus dans la boîte */
 gint spacing ;
 int orientation;//ORIENTATION_VERTICAL0;ORIENTATION_HORIZONTAL 1
 gint x;//position x
 gint y;// position Y
 }Box;
```

✓ L'initialisation des propriétés

La fonction commence par créer une nouvelle GtkBox en utilisant l'orientation fournie en entrée. Ensuite, elle stocke les informations fournies dans les champs correspondants de la structure Box. Enfin, elle configure la GtkBox avec les paramètres homogeneous et spacing en utilisant la fonction gtk_box_set_homogeneous.
La fonction ne renvoie aucune sortie, elle initialise simplement la structure Box fournie en entrée.

```
/* Entrées:
     - box: un pointeur vers une structure Box pré-allouée
   Sorties : un pointeur vers une structure Box
   Rôle:
     - Initialise la structure Box en créant une nouvelle GtkBox avec
      des valeurs par défaut
 */
```

```
Box initDefaultbox(Box box)
          {
             box = initB(box);
             vb->homogeneous =TRUE;
             vb->spacing = 0;
             vb->x = 0;
             vb->y = 0;
          }
/* Entrées:
        - box: un pointeur vers une structure Box
   Sorties : un pointeur vers une structure Box pré-allouée
   Rôle :
         Alloue de la mémoire pour une structure de type box*
   */

Box* initB(Box* box)
 {
         box = (Box*)malloc(sizeof(Box));
         return box;
 }
```

✓ La création du box

La fonction prend en entrée un pointeur vers une structure Box contenant les paramètres nécessaires à la création d'une boîte. Elle crée ensuite une nouvelle boîte avec les paramètres spécifiés. Si l'orientation de la boîte est horizontale, elle la définit en conséquence. Enfin, elle définit l'homogénéité de la boîte en fonction du paramètre correspondant dans la structure Box. La fonction renvoie le widget de la boîte nouvellement créée .

```
       /*
        -Entrées: vb Pointeur vers une structure Box contenant les paramètres de
         la boîte à créer.
        -Sorties: GtkWidget* Le widget de la boîte nouvellement créée.
        -Rôle: Crée une nouvelle boîte GTK en fonction des paramètres d'une
         structure Box donnée.
       */

 GtkWidget* creer_box(Box* vb)
 {
   GtkWidget* box = gtk_box_new(GTK_ORIENTATION_VERTICAL, vb->spacing);
   if (vb->orientation == 1)
gtk_orientable_set_orientation(GTK_ORIENTABLE(box), GTK_ORIENTATION_HORIZONTAL);
  gtk_box_set_homogeneous(GTK_BOX(box), vb->homogeneous);
   return box;
 }
```

✓ Un exemple de création d'un box

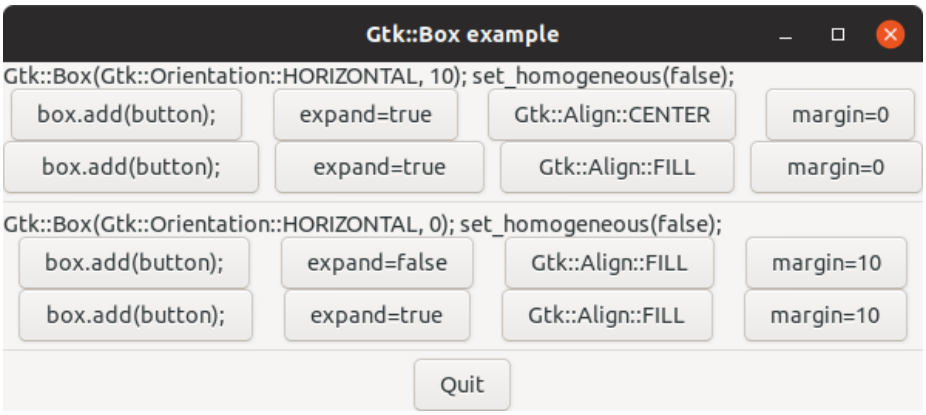


Figure 0 : Un exemple de création de box

II- Les fenêtres :

✓ La structure de données :

```
// Définition de la structure MaFenetre
typedef struct
{
    GtkWidget *wind ;              // Pointeur vers l'objet de fenêtre GTK
    gchar *title;                 // Titre de la fenêtre
    gint posX;                   // Position horizontale de la fenêtre
    gint posY;                  // Position verticale de la fenêtre
    gint rouge;                // Valeur de rouge pour la couleur de fond
    gint vert;                 // Valeur de vert pour la couleur de fond
    gint bleu;                 // Valeur de bleu pour la couleur de fond
    gint Hauteur;              // Hauteur de la fenêtre
    gint Largeur;              // Largeur de la fenêtre
     gboolean isResizable;    // la fenêtre est redimensionnable ou non
    gchar *iconfile;          // Chemin d'accès à l'icône de la fenêtre
    gboolean isFullscreen;    // la fenêtre est en mode plein écran ou non
    gboolean isDecorated;     //  la fenêtre a des décorations ou non
    gboolean isDeletable;     // fenêtre peut être fermée ou non
} MaFenetre;
```

Cette structure est conçue pour stocker les propriétés principales d'une fenêtre GTK, y compris sa position, sa taille, sa couleur de fond, son icône et diverses autres options d'affichage. Ces informations sont généralement utilisées pour initialiser et configurer une fenêtre dans un programme GTK.

✓ L'initialisation des propriétés :

```
MaFenetre *Init_default_window_prop(MaFenetre *wind)
{
    wind = initF(wind);
    wind->iconfile = (char*)"prebot.ico";
    strcpy(wind->iconfile,"prebot.ico");
    wind->wind = NULL;
    wind->isResisable = TRUE;
    wind->isDecorated = TRUE
    wind->isFullscreen=FALSE;
```

```
    wind->isDeletable= TRUE;

    wind->Largeur = 450;
    wind->Hauteur = 250;
    wind->title = (char*)"default title";
    wind->posX=0;
    wind->posY=0;
    wind->rouge=65535;
    wind->bleu=65535;
    wind->vert=65535;
    return wind;
}
```

La fonction **Init_default_window_prop** prend en entrée un pointeur vers une structure MaFenetre et initialise ses membres avec des valeurs par défaut.

Voici une description détaillée de chaque étape de la fonction :

La fonction appelle la fonction initF pour initialiser la structure avec des valeurs nulles ou par défaut.
Elle définit le chemin d'accès à l'icône de la fenêtre sur "prebot.ico".
Elle initialise le pointeur wind->wind à NULL.
Elle définit le booléen isResizable à TRUE, ce qui signifie que la fenêtre peut être redimensionnée.
Elle définit le booléen isDecorated à TRUE, ce qui signifie que la fenêtre est décorée avec des bordures et des boutons de commande.
Elle définit le booléen isFullscreen à FALSE, ce qui signifie que la fenêtre n'est pas en mode plein écran.
Elle définit le booléen isDeletable à TRUE, ce qui signifie que la fenêtre peut être fermée.
Elle définit la largeur de la fenêtre sur 450 pixels.
Elle définit la hauteur de la fenêtre sur 250 pixels.
Elle définit le titre de la fenêtre sur "default title".
Elle définit la position horizontale de la fenêtre à 0.
Elle définit la position verticale de la fenêtre à 0.
Elle définit les valeurs de rouge, vert et bleu pour la couleur de fond de la fenêtre à 65535 (qui correspond au blanc).
Enfin, la fonction renvoie le pointeur wind mis à jour avec les valeurs par défaut pour les propriétés de la fenêtre.

En somme, cette fonction est utile pour initialiser rapidement une structure MaFenetre avec des valeurs par défaut pour ses propriétés de fenêtre, ce qui peut être utile pour éviter la répétition de code et faciliter la création de nouvelles fenêtres avec des propriétés communes.

✓ La génération de la fenêtre :

```
void fenetre(MaFenetre *wind)
{
    wind->wind = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (wind->wind), wind->title);

    gtk_window_set_icon_from_file(wind->wind,wind->iconfile,NULL);
```

```
    gtk_widget_modify_bg(wind->wind, GTK_STATE_NORMAL, &(GdkColor){1, wind->rouge, wind->vert,wind->bleu});

    if (wind->isFullscreen) gtk_window_fullscreen(wind->wind);

    if(!wind->isDecorated) gtk_window_set_decorated(wind->wind,FALSE);
    if (!wind->isDeletable) gtk_window_set_deletable(wind->wind, FALSE);
    gtk_window_set_resizable(wind->wind,wind->isResisable);

    gtk_widget_set_size_request (wind->wind,wind->Largeur,  wind->Hauteur);

    gtk_window_move(wind->wind, wind->posX, wind->posY);

}
```

La fonction fenetre prend en entrée un pointeur vers une structure MaFenetre et crée une fenêtre GTK avec les propriétés spécifiées dans la structure.Voici une description détaillée de chaque étape de la fonction :

La fonction crée une nouvelle fenêtre en appelant la fonction gtk_window_new.

Elle définit le titre de la fenêtre en utilisant la valeur de wind->title avec la fonction gtk_window_set_title.

Elle définit l'icône de la fenêtre en utilisant la valeur de wind->iconfile avec la fonction gtk_window_set_icon_from_file.

Elle définit la couleur de fond de la fenêtre en utilisant les valeurs de rouge, vert et bleu stockées dans la structure wind avec la fonction gtk_widget_modify_bg.

Si la fenêtre doit être en mode plein écran, elle appelle la fonction gtk_window_fullscreen pour la mettre en plein écran.

Si la fenêtre ne doit pas avoir de décorations, elle appelle la fonction gtk_window_set_decorated avec l'argument FALSE pour la désactiver.

Si la fenêtre ne doit pas pouvoir être fermée, elle appelle la fonction gtk_window_set_deletable avec l'argument FALSE pour la désactiver.

Elle définit si la fenêtre est redimensionnable ou non en appelant la fonction gtk_window_set_resizable avec la valeur de wind->isResizable.

Elle définit la taille de la fenêtre en appelant la fonction gtk_widget_set_size_request avec les valeurs de wind->Largeur et wind->Hauteur.

Elle définit la position de la fenêtre en appelant la fonction gtk_window_move avec les valeurs de wind->posX et wind->posY.

En somme, cette fonction crée une nouvelle fenêtre GTK avec les propriétés spécifiées dans la structure MaFenetre. Elle permet de personnaliser facilement les propriétés de la fenêtre en utilisant la structure, ce qui peut rendre le code plus modulaire et plus facile à comprendre.

✓ <u>Un exemple de création d'une fenêtre :</u>

Figure 1 : Un exemple d'une fenêtre

III- Les boutons :

1- Les boutons simples

1.1-La structure de données

```
/* La structure des données pour un boutton simple */
typedef struct prButton
{
      GtkWidget* button;// l'objet button
      GtkWidget* racine; // le conteneur
      gchar label[20];        // label du button
      gchar stock_icon[20];   // button du stock
      char nom[20];           // nom du button
      int mnemonic;           // mnemonic
      int type_special;       // button special
      char image[20] ;
      struct Position* pos;   // position du button

 } prSimpleBut;

typedef struct Position
{
     int x;  // position sur l'abscisse en pixel
     int y;  // position sur l'ordonnée en pixel
} prPos;
```

1.2-L'initialisation des propriétés

```
 /*
  Entrées : Une structure boutton simple
  Sorties : Pas de valeur de retour
  Rôle : Cette fonction permet d'initialiser tous les champs d'une   */
  Structure "prSimpleBut" à des valeurs par défaut
*/


void init_button(prSimpleBut* button)
{
    button->button = NULL;
    button->racine = NULL;
    memset(button->label, 0, sizeof(button->label));
    memset(button->stock_icon, 0, sizeof(button->stock_icon));
    memset(button->nom, 0, sizeof(button->nom));
    button->mnemonic = 0;
    button->type_special = 0;
    memset(button->image, 0, sizeof(button->image));
  button->pos = NULL;
}
```

1.3-La création d'un bouton simple

```
/*
  Entrées: Un pointeur vers une structure prSimpleBut contenant les informations
  nécessaires à la création et à la configuration du bouton.
  Sorties: Un pointeur vers un GtkWidget correspondant au bouton créé.
  Rôle: Cette fonction crée un bouton en utilisant un label, un bouton du stock
  ou une image, en fonction des informations contenues dans la structure
```

```
   prSimpleBut. Elle configure ensuite les différentes propriétés du bouton  telles
   que le nom, la sensibilité, la taille, etc. en fonction des champs de la
   structure.
   Enfin, elle ajoute le bouton au conteneur spécifié dans la structure prSimpleBut
   et renvoie un pointeur vers le GtkWidget correspondant au bouton créé.

*/
GtkWidget* create_button(prSimpleBut* button)
{
    GtkWidget* gtk_button;

    if (strlen(button->stock_icon) > 0) { // création avec un bouton du stock
        gtk_button = gtk_button_new_from_stock(button->stock_icon);
    }
    else if (strlen(button->image) > 0) { // création avec une image
        GtkWidget* image = gtk_image_new_from_file(button->image);
        gtk_button = gtk_button_new();
        gtk_container_add(GTK_CONTAINER(gtk_button), image);
    }
    else { // création avec un label
        gtk_button = gtk_button_new_with_label(button->label);
    }

    // configuration des propriétés du bouton
    gtk_widget_set_name(gtk_button, button->nom);
    gtk_widget_set_sensitive(gtk_button, TRUE);
gtk_button_set_use_stock(GTK_BUTTON(gtk_button),strlen(button->stock_icon)>0);
    gtk_button_set_image_position(GTK_BUTTON(gtk_button), GTK_POS_TOP);
    gtk_button_set_always_show_image(GTK_BUTTON(gtk_button), TRUE);
    gtk_button_set_focus_on_click(GTK_BUTTON(gtk_button), TRUE);
    gtk_widget_set_size_request(gtk_button, -1, -1);
    gtk_button_set_label(GTK_BUTTON(gtk_button), button->label);
    gtk_button_set_use_underline(GTK_BUTTON(gtk_button), TRUE);
    if (button->mnemonic != 0)
   {

gtk_label_set_mnemonic_widget(GTK_LABEL(gtk_bin_get_child(GTK_BIN(gtk_button))),
gtk_button);
 gtk_widget_add_mnemonic_label(gtk_button, button->mnemonic);
   }

 // ajout du bouton au conteneur
gtk_layout_put(GTK_LAYOUT(button->racine),
                                        gtk_button,button->pos->x,button->pos->y);
    button->button = gtk_button;
    return gtk_button;
}
```

1.4-Un exemple de création d'un bouton simple

Figure 2 : Un exemple d'un bouton simple

2- Les boutons radios :

2.1-La structure de données

```
/* La structure des données pour un boutton simple */
typedef struct

  {
      GtkWidget* button; // le widget du radio bouton
      char label[20]; // le label du radio bouton
      char nom[20]; // le nom du groupe de radio boutons
      int mnemonic; // la touche d'accès rapide
      gboolean active; // l'état actif ou inactif du radio bouton
      prPos* pos; // la position du radio bouton dans la fenêtre

  }RadioButton;
```

2.2-L'initialisation des propriétés :

- La fonction initialise le champ "button" à NULL, qui est un pointeur vers un objet
- graphique de type bouton radio.
- La fonction initialise les champs "label" et "nom" à une chaîne de caractères vide.
- La fonction initialise le champ "mnemonic" à 0, qui représente un caractère de
- raccourci clavier.
- La fonction initialise le champ "active" à FALSE, qui représente l'état actif ou inactif du
- bouton radio.
- La fonction initialise le champ "pos" à NULL, qui est un pointeur vers la position du
- bouton radio dans l'interface graphique.

```
/*
    Entrées : un pointeur vers une structure RadioButton.
    Sorties : aucune.
    Rôle : cette fonction initialise les différents champs de la
    structure RadioButton à des valeurs par défaut.
*/


void init_radio_button(RadioButton * button)
   {
       button->button = NULL;
       strcpy(button->label,"");
       strcpy(button->nom,"");
       button->mnemonic = 0;
       button->active = FALSE;
       button->pos = NULL;
    }
```

2.3-La création d'un bouton simple

```
/*
  Entrées:
    -Un objet GtkLayout* layout qui représente le conteneur dans lequel le
     bouton radio sera créé.
    -Un pointeur RadioButton* button qui représente les informations
     nécessaires à la création du bouton radio.
  Sorties:
     Un objet GtkWidget* qui représente le bouton radio créé.
```

```
        Rôle:
          La fonction create_radio_button() crée un bouton radio en utilisant
          les informations fournies par le pointeur button

       */



GtkWidget* create_radio_button(GtkLayout* layout, RadioButton* button)
{
 GtkWidget* radio_button=gtk_radio_button_new_with_mnemonic(NULL, button->label);

 gtk_widget_set_name(radio_button, button->nom);

 gtk_widget_set_sensitive(radio_button, TRUE);

 gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(radio_button), button->active);

 if (button->mnemonic != 0)
  {

gtk_label_set_mnemonic_widget(GTK_LABEL(gtk_bin_get_child(GTK_BIN(radio_button)))
, radio_button);

 gtk_widget_add_mnemonic_label(radio_button, button->mnemonic);
 }
if (button->pos != NULL)
{
gtk_layout_put(GTK_LAYOUT(layout),radio_button,button->pos->x,button->pos->y);
}
 button->button = radio_button;

 return radio_button;

 }
```

2.4-Un exemple de création d'un bouton radio

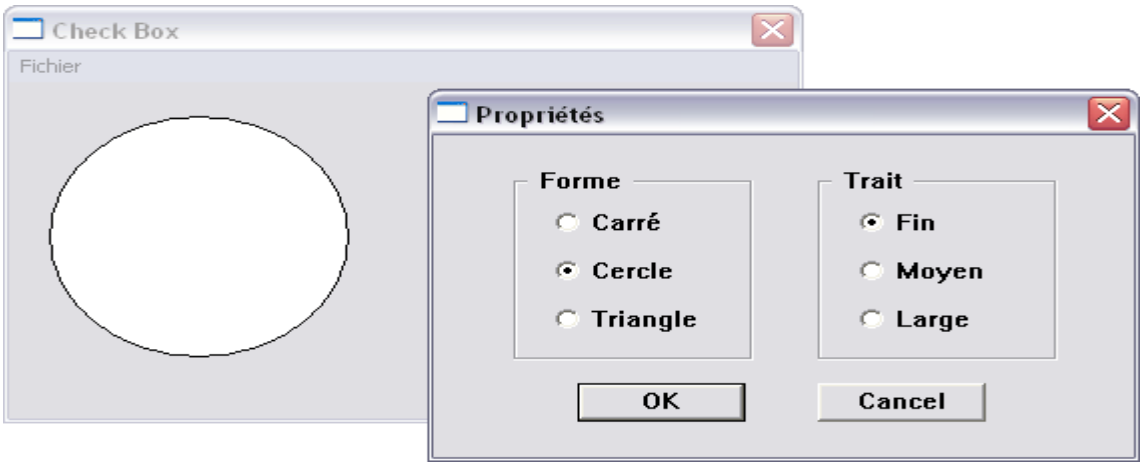


Figure 3 : Un exemple d'un bouton radio

3- Les check boutons :

3.1-La structure de données

```c
typedef struct
       {
           GtkWidget* button; // le widget du check bouton
           char label[20]; // le label du check bouton
           char nom[20]; // le nom du check bouton
           int mnemonic; // la touche d'accès rapide
           gboolean active; // l'état actif ou inactif du check bouton
           prPos* pos; // la position du check bouton dans la fenêtre
           int width;//largeur
           int height;//hauteur

       }CheckButton;
```

3.2-L'initialisation des propriétés

```c
/*
    Entrées: une structure CheckButton
    Sorties: aucune
    Rôle: initialiser les différents champs de la structure CheckButton en
    leur  attribuant une valeur par défaut.
*/
    void init_check_button(CheckButton *check_button)
     {
              check_button->button = NULL;
              memset(check_button->label, 0, sizeof(check_button->label));
              memset(check_button->nom, 0, sizeof(check_button->nom));
              check_button->mnemonic = 0;
              check_button->height = 50 ;
              check_button->width = 100;
              check_button->active = FALSE;
              check_button->pos = NULL;
      }
```

3.3-La création d'un checkbutton :

```c
/*

 Entrées: la fonction prend en entrée un pointeur vers une structure CheckButton
  contenant les informations nécessaires pour la création d'un bouton "check"
  (case à cocher) et un objet GtkLayout* layout qui représente le conteneur dans
lequel le check_bouton sera créé.
Sorties: la fonction retourne un pointeur vers le bouton "check" créé avec les
paramètres spécifiés dans la structure CheckButton.
Rôle: la fonction crée un bouton "check" (case à cocher) à partir des
informations spécifiées dans la structure CheckButton.
Les actions suivantes sont effectuées :

        o  Création du bouton "check" avec l'étiquette spécifiée dans la
           structure.
        o  Attribution d'un nom au bouton, tel que spécifié dans la structure.
        o  Définition de l'état actif ou inactif du bouton, tel que spécifié
           dans la structure.
        o  Définition de la taille du bouton, telle que spécifiée dans la
           structure.
        o  Alignement du bouton à gauche en haut de son conteneur.
        o  Si un raccourci clavier est spécifié dans la structure, il est
           configuré pour activer la case à cocher.
```

- Le pointeur vers le bouton créé est stocké dans la structure CheckButton.

*/

```
GtkWidget* create_check_button(CheckButton* cb)
   {
      GtkWidget* check_button = gtk_check_button_new_with_mnemonic(cb->label);
      gtk_widget_set_name(check_button, cb->nom);
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check_button), cb->active);
    gtk_widget_set_size_request(check_button, cb->width, cb->height);
    // Ajout du CheckButton dans le GtkLayout à la position spécifiée
    gtk_layout_put(GTK_LAYOUT(layout), check_button, cb.pos->x, cb.pos->y);
    gtk_widget_set_halign(check_button, GTK_ALIGN_START);
    gtk_widget_set_valign(check_button, GTK_ALIGN_START);
    if (cb->mnemonic != 0)
{

gtk_label_set_mnemonic_widget(GTK_LABEL(gtk_bin_get_child(GTK_BIN(check_button)))
,check_button);
 gtk_widget_add_accelerator(check_button,"activate",
gtk_accel_group_get_default(),
                           cb->mnemonic, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
  }
   return check_button;

}//fin fonction
```

3.4-Un exemple de création d'un checkbutton

Figure 4 : Un exemple d'un check bouton

IV. Les Menus :

1-La structure de données :

```
typedef struct mennu
  {
```

```
                    GtkWidget* menu;//l'objet menu
                    struct item* fils;//liste de ses fils
          }Menu;

      typedef struct item
       {
                    GtkWidget* item;
                    gchar* titre;//le titre
                    struct mennu* menu;
                    struct item* next;//l'élément suivant
       }Item;
```

2-L'initialisation des propriétés :

```
/*
   Entrées : un pointeur vers une structure de type Menu
   Sorties : un pointeur vers la structure de type Menu initialisée
   Rôle : alloue dynamiquement de l'espace pour la structure Menu et initialise
ses membres à des valeurs par défauts.
*/
  Menu* initMenu(Menu* menu)
    {
           menu = (Menu*)malloc(sizeof(Menu));
           menu->menu = NULL;
           menu->fils = NULL;
          return menu;
    }

  /*
      Entrées : un pointeur vers une structure de type Item
      Sorties : un pointeur vers la structure de type Item initialisée
      Rôle : alloue dynamiquement de l'espace pour la structure Item et
initialise ses membres à des valeurs par défaut.
  */



 Item* initItem(Item* item)
    {
                    item = (Item*)malloc(sizeof(Item));
                    item->item = NULL;
                    item->titre = NULL;
                    item->menu = NULL;
                    item->next = NULL;
             return item;
   }
```

3. La génération de la fenêtre :

```
     /*
         Entrées :
                - un pointeur vers une variable de type Item
                - une chaîne de caractères pour le titre du menu item
         Sorties :
           - un pointeur vers la même variable Item initialisée
         Rôle :
           - Alloue dynamiquement une nouvelle variable de type Item et
            initialise ses champs, y compris la création d'un nouveau
            gtk_menu_item à partir du titre passé en paramètre.
     */
```

```
        Item* creeItem(Item* item, gchar* titre)
           {
                    item = initItem(item);
                    item->titre = titre;
                    item->item = gtk_menu_item_new_with_label(titre);
                    return item;
            }

         /*
              Entrées :
                 - un pointeur vers une variable de type Menu
              Sorties :
                 - un pointeur vers la même variable Menu initialisée
              Rôle :
               - Alloue dynamiquement une nouvelle variable de type Menu et
                 initialise son champ menu, y compris la création d'un nouveau
                 gtk_menu_bar.
         */
                 Menu* creemenubar(Menu* menu)
                 {
                     menu = initMenu(menu);
                     menu->menu = gtk_menu_bar_new();
                     return menu;
                 }


/*
 * Entrées : un pointeur de type Menu.
 * Sortie : un pointeur de type Menu initialisé.
 * Rôle : allouer de l'espace mémoire pour le pointeur et initialiser le champ
menu.
 */




       Menu* creemenu(Menu* menu)
          {
                    menu = initMenu(menu);
                    menu->menu = gtk_menu_new();
                    return menu;
            }


 /*
 * Entrée : un pointeur de type Menu, un pointeur de type Item.
 * Sortie : un pointeur de type Menu modifié.
 * Rôle : ajouter l'objet de type Item à la liste des fils du Menu et à la barre
de menus.
 */


 Menu* Ajouteraumenu(Menu* menu, Item* item)
    {
           if (menu->fils)item->next = menu->fils->next;
           menu->fils = item;
           gtk_menu_shell_append(GTK_MENU_SHELL(menu->menu), menu->fils->item);
           return menu;
    }

/*
```

```
     * Entrée : un pointeur de type Item.
     * Sortie : un pointeur de type Item initialisé avec un sous-menu.
     * Rôle : allouer de l'espace mémoire pour le champ menu de l'objet Item,
initialiser
*/

    Item* Creesousmenu(Item* item)
      {
          if (!item) return NULL;
          item->menu = creemenu(item->menu);
          gtk_menu_item_set_submenu(GTK_MENU_ITEM(item->item), item->menu->menu);
          return item;
       }
```

3 Un exemple de création d'un menu :

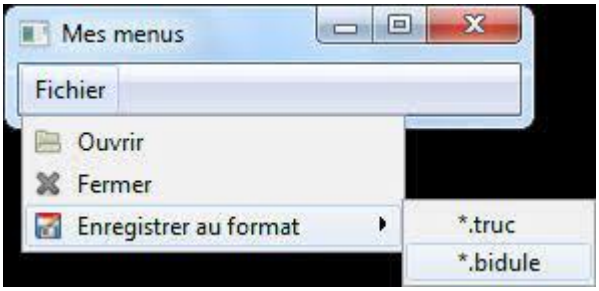


Figure 4 : Un exemple d'un menu

XI. L'image :

1-La structure de données :

```
typedef struct
{
    GtkWidget* image;
    gchar *file;
    gint x;
    gint y;

}Image;
```

2-L'initialisation des propriétés :

```
Image *initImage(Image *image){
    image = (Image*)malloc(sizeof(Image));
    image->image = NULL;
    image->file = (char*)"";
    image->x = NULL;
    image->y = NULL;
    return image;
}
```

4. La génération de l'image :

```
void CreateImage(Image *image)
{
    image->image = gtk_image_new_from_file(image->file);
    gtk_widget_set_size_request(image->image, image->x, image->y);
}
```

XII. Scale :

1-La structure de données :

```
typedef struct {
    GtkWidget *scale;
    gint min_value;
    gint max_value;
    gint step_increment;
    gint page_increment;
    gint value;
    gint digits;
    gboolean draw_value;
    GtkOrientation orientation;
} ScaleProperties;
```

2-L'initialisation des propriétés :

```
ScaleProperties* init_scale(ScaleProperties *scale ){
        scale = (ScaleProperties*)malloc(sizeof(ScaleProperties));
        scale->min_value = 0.0;
        scale->max_value = 100.0;
        scale->step_increment = 1.0;
        scale->page_increment = 10.0;
        scale->value = 50.0;
        scale->digits = 1;
        scale->draw_value = TRUE;
        scale->orientation = GTK_ORIENTATION_HORIZONTAL;
        return ((ScaleProperties*)scale);
}
```

4. La génération du scale :

```
void Create_scale(ScaleProperties *scale) {
    scale->scale = gtk_scale_new_with_range(scale->orientation, scale->min_value,
scale->max_value, scale->step_increment);
    gtk_scale_set_digits(GTK_SCALE(scale->scale), scale->digits);
    gtk_scale_set_draw_value(GTK_SCALE(scale->scale), scale->draw_value);
    gtk_range_set_increments(GTK_RANGE(scale->scale), scale->step_increment,
scale->page_increment);
    gtk_range_set_value(GTK_RANGE(scale->scale), scale->value);

}
```

LES ENTREES :

1-La structure de données :

```
typedef struct
{
  GtkWidget* entry;//widget d'une entrée
  int len;;//la longueur d'entrée.
  int width;//largeur.
  gboolean visible;//le texte est visible ou pas.
  char invisible_car;//le caractère d'un texte invisible.
  gboolean editable;///on peut saisir par clavier.
  const char* text_to_set;// texte à saisir.
  const char* text_to_get;//texte à récupérer.
   }Entry;
```

2-Initialisation de l'entrée:

```
/*
 Cette fonction permet d'initialiser les champs d'une entrée par des valeurs
qu'on va utiliser
par défauts si on souhaite pas modifier par la suite.
*/


void initentry(Entry* e)
    {

          e->visible=TRUE;
          strcpy(e->text_to_set,"");
          e->editable=TRUE;
    }
```

3-creation de l'entrée:

```
/*
  Cette fonction permet de créer une entrée à partir d'une structure passée en
paramètres.
*/
GtkWidget *init_entry(Entry*e)
{
    GtkWidget* entry;
    entry=gtk_entry_new();
    gtk_widget_set_size_request(entry,e->len,e->width);
    gtk_entry_set_visibility(GTK_ENTRY(entry),e->visible);
    if (!e->visible) {
        gtk_entry_set_invisible_char(GTK_ENTRY(entry),e->invis_car);
    }
    if (if(e->text_to_set) {

        gtk_editable_set_editable(GTK_EDITABLE(entry),e->editable);    }
                          }
return (GtkWidget*)entry;

}
```

TEXTVIEW :

1-La structure de données :

```
typedef struct
        {
           char foreground[26];
           int   left_margin ;
           int   size;
           char  style[26]; //italic,oblique,normal
           char  weight[26];//normal,gras,light
           char background[26];

           }format;
```

```
typedef struct
 {
  GtkTextBuffer * *buffer; //le buffer associé au textview.
  format f;
  char  texte[100]; //italic,oblique,normal
  GtkTextIter iter;// l'itération au niveau du buffer.

 }TextView;
```

2-Initialisation de TextView :

```
/*
  Cette fonction permet d'initialiser les champs d'un textview par des valeurs
qu'on va utiliser
par défauts si on souhaite pas modifier par la suite.
*/
void inittextebuf(TexteView* txt)
    {
        strcpy(txt->t.background,"red");
        strcpy(txt->t.foreground,"blue");
        strcpy(txt->t.style,"italic");
        strcpy(txt->t.weight,"gras");
        strcpy(txt->text,"HELLO GTK ");
        txt->t.size=45;
        txt->t.left_margin=45;
  }
```

3- Creation de TextView :

```
/*
  Cette fonction permet de créer un textview à partir d'une structure passer en
paramètre en
affectant le style et le couleur ... du texte de notre structure.
*/

GtkWidget * *create_textView(TextView view)
          {
  GtkWidget *tv;
  view.buffer=gtk_text_buffer_new(NULL);
  tv=gtk_text_view_new_with_buffer(view.buffer);

  ///*******INITIALISATION***********///
init_format_textbuf(&view);
gtk_text_buffer_create_tag(view.buffer,"lmarg","left_margin",view.t.left_margin,N
ULL);
```

```
 ///*******choix du style********///
if(!strcmp(view.t.style,"italic"))
gtk_text_buffer_create_tag(view.buffer,"style","style",PANGO_STYLE_ITALIC,NULL);
if(!strcmp(view.t.style,"oblique"))
gtk_text_buffer_create_tag(view.buffer,"style","style",PANGO_STYLE_OBLIQUE,NULL);
///*********choix du weight*******///
if(!strcmp(view.t.weight,"gras"))
gtk_text_buffer_create_tag(view.buffer,"weight","weight",PANGO_WEIGHT_BOLD,NULL);
if(!strcmp(view.t.weight,"light"))
gtk_text_buffer_create_tag(view.buffer,"weight","weight",PANGO_WEIGHT_LIGHT,NULL;
///*******size********///
gtk_text_buffer_create_tag(view.buffer,"size","size",view.t.size,NULL);
///********color**********************///
gtk_text_buffer_create_tag(view.buffer,"foreground","foreground",view.t.foregroud
,NULL);
///*******background********///
gtk_text_buffer_create_tag(view.buffer,"background","background",view.t.backgroun
d,NULL)
gtk_text_buffer_get_iter_at_offset(view.buffer,&view.iter,0);

return (GtkWidget*)tv
}
```

Label:

1-La structure de données :

```
typedef struct
{

char style[10];///0:italic,1:oblique,3:normal

char font[26];///police d'écriture

char taille[4];///size du texte

char weight[15];///ultralight,light,normal,bold,ultrabold,heavy

char color[26];///couleur du texte

char background[26];///couleur du fond

char underline[10];///single, double, low ou none

char underline_color[26]; ///couleur du soulignement

char barrer_txt[6]; ///true ou false

char color_bar[26]; ///couleur du

char select; //0:non ou 1:oui

char alignement; //0:left,1:right,2:center, ou 3:fill

}forme;
```

typedef struct

{

GtkWidget Label; ///pointeur sur le label*

char texte[255]; ///le texte à afficher

Forme lab; //formatage du texte

}Lab_txt;

2-Initialisation du Label :

```
/*
   Cette fonction permet d'initialiser les champs d'un label.
Elle prend en arguments un pointeur sur le label et elle initialise
le format du texte par des valeurs standard.
*/


void init_label(label_txt* l)
    {
        strcpy(l->texte,"\Nhello gtk\n!!!");
        strcpy(l->lab.background, "white");
        strcpy(l->lab.color, "vert");
        strcpy(l->lab.barrer_txt, "false");
        strcpy(l->lab.color_bar, "red");
        strcpy(l->lab.font, "calibri");
        strcpy(l->lab.taille, "20");
        strcpy(l->lab.underline, "none");
        strcpy(l->lab.underline_color, "black");
        strcpy(l->lab.style, "normal");
        strcpy(l->lab.weight, "normal");
        l->lab.alignement = '3';
        l->lab.select = '1';
 }
```

X.comobox

Une structure de données ComboBoxProperties est définie pour stocker les propriétés du GtkComboBox, telles que les éléments, la sensibilité, le modèle et le rendu. Une fonction create_combobox est définie pour créer un GtkComboBox avec les propriétés fournies.
Une fonction Init_comobox est également définie pour initialiser une instance de ComboBoxProperties avec des valeurs par défaut.
Dans la fonction principale, GTK est initialisé et la structure ComboBoxProperties est initialisée avec des éléments "A", "B" et "C". Un GtkComboBox est ensuite créé en appelant la fonction create_combobox avec la structure ComboBoxProperties. Enfin, la fenêtre GTK est créée et le GtkComboBox est ajouté à la fenêtre.

```
typedef struct node {
    gchar *value;
    struct node *next;
} Node;

void add_element(Node **head, const gchar *value) {
    Node *new_node = g_new(Node, 1);
    new_node->value = g_strdup(value);
    new_node->next = NULL;
    if (*head == NULL) {
        *head = new_node;
    } else {
        Node *last_node = *head;
        while (last_node->next != NULL) {
            last_node = last_node->next;
        }
        last_node->next = new_node;
    }
}

typedef struct {
    Node *items;
    gboolean is_editable;
    gboolean has_entry;
    GtkTreeModel *model;
    GtkCellRenderer *renderer;
    gint active_index;
    gboolean is_sensitive;
    gint height;
    gint width;
} ComboBoxProperties;


GtkWidget* create_combobox(ComboBoxProperties *properties) {
    GtkListStore *store = gtk_list_store_new(1, G_TYPE_STRING);
    GtkTreeIter iter;
    if (properties->items) {
        Node *iterator = properties->items;
        while (iterator) {
            gtk_list_store_append(store, &iter);
            gtk_list_store_set(store, &iter, 0, iterator->value, -1);
            iterator = iterator->next;
        }
    }
    GtkWidget *combo = gtk_combo_box_new_with_model(GTK_TREE_MODEL(store));
    gtk_combo_box_set_entry_text_column(GTK_COMBO_BOX(combo), 0);
    gtk_combo_box_set_active(GTK_COMBO_BOX(combo), properties->active_index);
    gtk_combo_box_set_model(GTK_COMBO_BOX(combo), GTK_TREE_MODEL(store));
```

```
    gtk_widget_set_sensitive(combo, properties->is_sensitive);
    gtk_cell_renderer_set_visible(properties->renderer, TRUE);
    gtk_cell_layout_pack_start(GTK_CELL_LAYOUT(combo), properties->renderer,
TRUE);
        gtk_widget_set_size_request(combo, properties->width, properties-
>height);


    gtk_cell_layout_set_attributes(GTK_CELL_LAYOUT(combo), properties->renderer,
"text", 0, NULL);

    return combo;
}
ComboBoxProperties Init_comobox()
{
        ComboBoxProperties properties = {NULL, FALSE, FALSE, NULL,
gtk_cell_renderer_text_new(), -1, TRUE,100, 100};
        return properties;

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


    ComboBoxProperties properties=Init_comobox();
    // Ajout des éléments à la liste
    add_element(&(properties.items), "A");
        add_element(&(properties.items), "B");
            add_element(&(properties.items), "C");


    //    add_element(&(properties.active_indexitems), "Deuxième élément");

    // Création du combobox
    GtkWidget *combo = create_combobox(&properties);


    // Affichage de la fenêtre
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

    gtk_container_add(GTK_CONTAINER(window), combo);
    gtk_widget_show_all(window);

    // Boucle principale de GTK
    gtk_main();
}
```

Partie II -La gestion d'un fichier XML

1-Principe de traitement du fichier :

Le principe de fonctionnement de ce code consiste à lire un fichier XML contenant la description d'une interface graphique, puis à générer dynamiquement la fenêtre correspondante avec les widgets décrits dans le fichier. La fonction parsefile est appelée avec le nom du fichier XML en paramètre et commence par ouvrir le fichier en utilisant la fonction xmlParseFile de la bibliothèque libxml2. Si l'ouverture du fichier échoue, un message d'erreur est affiché et la fonction s'arrête.

Ensuite, la racine du document XML est extraite en utilisant la fonction xmlDocGetRootElement, et si cette racine est un élément de type "window", la fonction crée une nouvelle fenêtre avec les propriétés par défaut à l'aide de la fonction Init_default_window_prop, lit les propriétés de la fenêtre à partir des attributs de l'élément "window" avec la fonction ReadWindow, puis crée les widgets enfants avec la fonction CreateWidgets. Les widgets enfants sont lus récursivement en parcourant tous les nœuds enfants de la racine en utilisant une boucle for.

Dans la boucle for, chaque nœud enfant est analysé pour déterminer de quel type de widget il s'agit en utilisant une série de conditions if qui comparent le nom de l'élément à une chaîne de caractères correspondant au nom du widget. Si le nœud enfant est un widget, les fonctions appropriées sont appelées pour lire les propriétés du widget à partir des attributs XML et créer le widget en utilisant les fonctions de création de GTK+ correspondantes. Enfin, les widgets sont liés les uns aux autres en utilisant la fonction linkwidgets pour les disposer correctement dans la fenêtre parente.

En résumé, la lecture du fichier XML est effectuée en parcourant l'arborescence de nœuds avec une boucle for, en déterminant le type de widget de chaque nœud enfant avec une série de conditions if, puis en lisant les propriétés de chaque widget à partir des attributs XML correspondants avec les fonctions de lecture correspondantes. Enfin, les widgets sont créés et liés les uns aux autres pour former la fenêtre complète.

2-Fonction de manipulations

2.1-Container :

2.1.1-Le box :

```
Box* ReadBox(xmlNodePtr node, Box* box)
{
   if (xmlGetProp(node, (xmlChar*)"orientation") != NULL)
    {
            box->orientation = (char*)xmlGetProp(node, (xmlChar*)"orientation");
    }
   if (xmlGetProp(node, (xmlChar*)"spacing") != NULL)
    {
      box->spacing = atoi((char*)xmlGetProp(node, (xmlChar*)"spacing"));
    }
   if (xmlGetProp(node, (xmlChar*)"x") != NULL)
   {
       box->posx = atoi((char*)xmlGetProp(node, (xmlChar*)"x"));
   }
   if (xmlGetProp(node, (xmlChar*)"y") != NULL)
   {
        box->posy = atoi((char*)xmlGetProp(node, (xmlChar*)"y"));
   }
   if (xmlGetProp(node, (xmlChar*)" homogeneous ") != NULL)
```

```
   {
    box-> homogeneous = atoi((char*)xmlGetProp(node, (xmlChar*)" homogeneous "));
   }


   return box;
 }//fin fonction
```

2.2- CreateWidgets :

La fonction CreateWidgets est utilisée pour parcourir l'arbre XML et créer les widgets appropriés en fonction des nœuds XML correspondants. Chaque nœud est traité dans une condition if différente qui appelle une fonction de création de widget appropriée. Par exemple, si le nœud est un élément XML "window", la fonction ReadWindow est appelée pour lire les propriétés de la fenêtre à partir du fichier XML, puis la fonction fenetre est appelée pour créer la fenêtre en utilisant les propriétés lues

```
void CreateWidgets(xmlNodePtr node, GtkWidget *parent)
{
    GtkWidget *fils,*group,*radiobut;
    radiobut = gtk_radio_button_new(NULL);
    MaFenetre *wind;
    Box *box;
    Menu *menu;
    prSimpleBut *buttonsimple;
    CheckButton *buttoncheck;
    RadioButton *buttonradio;
    Image *image;
    Label *label;
    TextView *view;
    ScaleProperties *scale;
    Entry *entry;
    for (xmlNodePtr child = node->children; child != NULL; child = child->next) {
        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"menu") == 0) {
            menu = create_menu_bar_from_xml(child);
            fils = menu->menu;
        }
        if(child->type == XML_ELEMENT_NODE && strcmp((char *)child->name, "box")
== 0){

            box = initDefaultbox(box);
            box = ReadBox(child,box);
            BoxC(box);
            fils = box->box;
            CreateWidgets(child,box->box);
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"window") == 0)
        {
            wind = Init_default_window_prop(wind);
            wind = ReadWindow(child,wind);
            fenetre(wind);
            fils = wind->wind;
            CreateWidgets(child,wind->wind);
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"simplebutton") == 0)
        {
            buttonsimple = parse_button(child,buttonsimple);
            create_button(buttonsimple);
```

```
           // g_print("%s",buttonsimple.stock_icon);
            fils = buttonsimple->button;
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"checkbutton") == 0)
        {
            buttoncheck = parse_check_button(child,buttoncheck);
            create_check_button(buttoncheck);

            fils = buttoncheck->button;
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"radiobutton") == 0)
        {
            buttonradio = parse_radio_button(child,buttonradio);
            create_radio_button(buttonradio,radiobut);
            fils = buttonradio->button;
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"image") == 0)
        {
            image = initImage(image);
            image = ReadImage(child,image);
            CreateImage(image);
            fils = image->image;
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"label") == 0)
        {
            label = initLabel(label);
            label = Readlabel(child,label);
            CreateLabel(label);
            fils = label->label;
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"textview") == 0)
        {
            view = init_format_textbuf(view);
            view = ReadTextview(child,view);
            create_textView(view);
            fils = view->textview;
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"scale") == 0)
        {
            scale = init_scale(scale);
            scale = ReadScale(child,scale);
            Create_scale(scale);
            fils = scale->scale;
        }

        if (child->type == XML_ELEMENT_NODE && strcmp((char *)child->name,
"entry") == 0)
        {
            entry = init_default_entry(entry);
            entry = ReadEntry(child,entry);
            CreateEntry(entry);
```

```
            fils = entry->entry;
        }


        if (strcmp((char *)child->name, "text") != 0) linkwidgets(parent,fils);
    }
}

void parsefile(const char *filename)
{
    xmlDocPtr doc = xmlParseFile(filename);
    if (doc == NULL) {
        g_warning("Failed to parse XML file %s", filename);
        return ;
    }
    xmlNodePtr root = xmlDocGetRootElement(doc);
    if (root->type == XML_ELEMENT_NODE && strcmp((char *)root->name, "window") ==
    {
        //GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
        MaFenetre *wind;
        wind = Init_default_window_prop(wind);
        wind = ReadWindow(root,wind);
        fenetre(wind);
        CreateWidgets(root,wind->wind);
        gtk_widget_show_all(wind->wind);
    }
    else g_print("Window not found");
    xmlFreeDoc(doc);
}

Une fois que les widgets sont créés, la fonction linkwidgets est appelée pour les
lier à leur parent en utilisant les fonctions de la bibliothèque GTK+

void linkwidgets(GtkWidget *parent,GtkWidget *child)
{
    if (GTK_IS_WINDOW(parent)) gtk_container_add(GTK_CONTAINER(parent), child);

    if(GTK_IS_BOX(parent))gtk_box_pack_start(GTK_BOX(parent), child, FALSE,
FALSE, 0);
}
```

Cette fonction lit un fichier XML à l'aide de la bibliothèque libxml2 et recherche un élément "window" dans le document XML. Si l'élément est trouvé, la fonction crée une fenêtre GTK, la configure avec les propriétés par défaut et les propriétés lues dans le document XML, crée des widgets pour la fenêtre à partir des données du document XML, affiche la fenêtre et libère la mémoire utilisée par le document XML. Si l'élément "window" n'est pas trouvé, la fonction affiche un message indiquant que la fenêtre n'a pas été trouvée.

```
void parsefile(const char *filename)
{
   xmlDocPtr doc = xmlParseFile(filename);
   if (doc == NULL) {
      g_warning("Failed to parse XML file %s", filename);
      return ;
   }
   xmlNodePtr root = xmlDocGetRootElement(doc);
```

```
  if (root->type == XML_ELEMENT_NODE && strcmp((char *)root->name, "window") == 0)
  {
     //GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
     MaFenetre *wind;
     wind = Init_default_window_prop(wind);
     wind = ReadWindow(root,wind);
     fenetre(wind);
     CreateWidgets(root,wind->wind);
     gtk_widget_show_all(wind->wind);
  }
  else g_print("Window not found");
  xmlFreeDoc(doc);
}
```

2.3- Les boutons :

2.3.1-les boutons simples

```
prPos* parse_position(xmlNode* node)
{
    prPos* pos = malloc(sizeof(prPos));
    xmlNode* child;
    for (child = node->children; child != NULL; child = child->next)
    {
        if (child->type == XML_ELEMENT_NODE)
        {
            if (!strcmp((char*)child->name, "x"))
            {
                xmlChar* content = xmlNodeGetContent(child);
                pos->x = atoi((char*)content);
                xmlFree(content);
            }
            else if (!strcmp((char*)child->name, "y"))
            {
                xmlChar* content = xmlNodeGetContent(child);
                pos->y = atoi((char*)content);
                xmlFree(content);
            }
        }
    }
    return pos;
}
```

```
  prSimpleBut parse_button(xmlNode* node, prSimpleBut button)
{

    init_button(&button);
    for (xmlAttr* attr = node->properties; attr != NULL; attr = attr->next) {
        if (strcmp((char*)attr->name, "label") == 0) {
            strcpy(button.label, (char*)attr->children->content);
        }
        else if (strcmp((char*)attr->name, "stock_icon") == 0) {
            strcpy(button.stock_icon, (char*)attr->children->content);
        }
```

```
        else if (strcmp((char*)attr->name, "name") == 0) {
            strcpy(button.nom, (char*)attr->children->content);
        }
        else if (strcmp((char*)attr->name, "mnemonic") == 0) {
            button.mnemonic = *(char*)attr->children->content;
        }
        else if (strcmp((char*)attr->name, "type") == 0) {
            button.type_special = atoi((char*)attr->children->content);
        }
        else if (strcmp((char*)attr->name, "image") == 0) {
            strcpy(button.image, (char*)attr->children->content);
        }
        else if (strcmp((char*)attr->name, "x") == 0) {
            int x = atoi((char*)attr->children->content);
            if (button.pos == NULL) {
                button.pos = malloc(sizeof(prPos));
            }
            button.pos->x = x;
        }
        else if (strcmp((char*)attr->name, "y") == 0) {
            int y = atoi((char*)attr->children->content);
            if (button.pos == NULL) {
                button.pos = malloc(sizeof(prPos));
            }
            button.pos->y = y;
        }
    }

    return button;
}
```

3.3.2-les boutons radios

```
 RadioButton* parse_radio_button(xmlNode* node, RadioButton* button)
 {
     button = init_radio_button(button);
    for (xmlAttr* attr = node->properties; attr != NULL; attr = attr->next)
    {
           if (strcmp((char*)attr->name, "label") == 0)
            {
                     strcpy(button->label, (char*)attr->children->content);
            }
            else if (strcmp((char*)attr->name, "x") == 0)
                 {
                     button->x = atoi((char*)attr->children->content);
                 }
                else if (strcmp((char*)attr->name, "y") == 0)
                     {
                         button->y = atoi((char*)attr->children->content);
                     }
                    else if (strcmp((char*)attr->name, "active") == 0)
                        {
                        if (strcmp((char*)attr->children->content, "yes") == 0)
                          button->active = TRUE;
                       if (strcmp((char*)attr->children->content, "no") == 0)
                            button->active = FALSE;
                      }// if (strcmp((char*)attr->name, "active") == 0)
                 }//fin for
           return button;
     }
```

2.3.3-les check boutons

```
void parse_check_button(xmlNode* node, CheckButton* check_button)
{
        xmlNode* child;
        for (child = node->children; child != NULL; child = child->next)
        {
             if (child->type == XML_ELEMENT_NODE)
             {
                if (!strcmp((char*)child->name, "label"))
                  {
                      xmlChar* content = xmlNodeGetContent(child);
                      strncpy(check_button->label, (char*)content,
                      sizeof(check_button->label) - 1);
                      xmlFree(content);
                   }
                else if (!strcmp((char*)child->name, "name"))
                      {
                              xmlChar* content = xmlNodeGetContent(child);
                              strncpy(check_button->nom, (char*)content,
                              sizeof(check_button->nom) - 1);
                              xmlFree(content);
                      }

                       else if (!strcmp((char*)child->name, "mnemonic"))
                             {
                               xmlChar* content = xmlNodeGetContent(child);
                               check_button->mnemonic = content[0];
                               xmlFree(content);
                             }
                           else if (!strcmp((char*)child->name, "position"))
                                {
                               check_button->pos = parse_position(child);
                                }
                       }//if
                   }//for
     }//fin fonction
```

2.4- Les menus :

```
static void create_menu_items(xmlNodePtr node, Menu* menu)
{
  for (xmlNodePtr child = node->children; child != NULL; child = child->next)
  {
  if (child->type == XML_ELEMENT_NODE && strcmp((char*)child->name,"item")== 0)
     {
       Item* item = creeItem(item, (char*)xmlGetProp(child, (xmlChar*)"label"));
       menu = Ajouteraumenu(menu, item);
       if (child->children != NULL)
        {
               item = Creesousmenu(item);
               create_menu_items(child, item->menu);
        }
     } //IF
  }//for
}//fin fonction
Menu* create_menu_bar_from_xml(xmlNodePtr node)
    {
                   Menu* menu = creemenubar(menu);
                   create_menu_items(node, menu);
                   return menu;
     }
```

les entrée :

```
Entry *ReadEntry(xmlNodePtr node,Entry *entry){
    if(xmlGetProp(node, (xmlChar *)"editable") != NULL){
        if(strcmp((char *)xmlGetProp(node, (xmlChar *)"editable"),"TRUE")==0)
entry->editable = TRUE ;
        if(strcmp((char *)xmlGetProp(node, (xmlChar *)"editable"),"FALSE")==0)
entry->editable = FALSE ;
    }
    if(xmlGetProp(node, (xmlChar *)"visible") != NULL){
        if(strcmp((char *)xmlGetProp(node, (xmlChar *)"visible"),"TRUE")==0)
entry->visible = TRUE ;
        if(strcmp((char *)xmlGetProp(node, (xmlChar *)"visible"),"FALSE")==0)
entry->visible = FALSE ;
    }
    if(xmlGetProp(node, (xmlChar *)"x") != NULL){
        entry->x = atoi((char *)xmlGetProp(node, (xmlChar *)"x"));
    }
    if(xmlGetProp(node, (xmlChar *)"y") != NULL){
        entry->y = atoi((char *)xmlGetProp(node, (xmlChar *)"y"));
    }
    if(xmlGetProp(node, (xmlChar *)"text") != NULL){
        entry->text_to_set = (char *)xmlGetProp(node, (xmlChar *)"text");
    }
    if(xmlGetProp(node, (xmlChar *)"invisiblechar") != NULL){
        gchar *tmp = (char *)xmlGetProp(node, (xmlChar *)"invisiblechar");
        entry->invisible_car = tmp[0];
    }
    return entry;
}
```

La fenêtre :

```
MaFenetre *Init_default_window_prop(MaFenetre *wind)
{
    wind = initF(wind);
    wind->iconfile = (char*)"prebot.ico";
    //strcpy(wind->iconfile,"prebot.ico");
    wind->wind = NULL;
    wind->isResisable = TRUE;//il est possible de changer la
//taille de la fenêtre.
    wind->isDecorated = TRUE;//la fenêtre est décorée.
    wind->isFullscreen=FALSE;//la fenêtre n'occupe pas toute
//l'écran.
    wind->isDeletable= TRUE;

    wind->Largeur = 450;
    wind->Hauteur = 250;
    wind->title = (char*)"default title";
    //strcpy(wind->title,"default title");
    wind->posX=0;
    wind->posY=0;
    wind->rouge=65535;
    wind->bleu=65535;
    wind->vert=65535;

    return wind;
}//fin de la fonction Init_default_window_prop.

void fenetre(MaFenetre *wind)
{
    wind->wind = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title (GTK_WINDOW (wind->wind), wind->title);
```

```
    gtk_window_set_icon_from_file(wind->wind,wind->iconfile,NULL);
    gtk_widget_modify_bg(wind->wind, GTK_STATE_NORMAL, &(GdkColor){1, wind-
>rouge, wind->vert,wind->bleu});

    if (wind->isFullscreen) gtk_window_fullscreen(wind->wind);

    if(!wind->isDecorated) gtk_window_set_decorated(wind->wind,FALSE);
    if (!wind->isDeletable) gtk_window_set_deletable(wind->wind, FALSE);
    gtk_window_set_resizable(wind->wind,wind->isResisable);

    gtk_widget_set_size_request (wind->wind,wind->Largeur,  wind->Hauteur);

    gtk_window_move(wind->wind, wind->posX, wind->posY);

}

MaFenetre *ReadWindow(xmlNodePtr node,MaFenetre *wind)
{
  if(xmlGetProp(node, (xmlChar *)"title") != NULL){
    wind->title = (char *)xmlGetProp(node, (xmlChar *)"title");
  }
  if(xmlGetProp(node, (xmlChar *)"icon") != NULL){
    wind->iconfile = (char *)xmlGetProp(node, (xmlChar *)"icon");
  }
  if(xmlGetProp(node, (xmlChar *)"height") != NULL){
    wind->Hauteur = atoi((char *)xmlGetProp(node, (xmlChar *)"height"));
  }
  if(xmlGetProp(node, (xmlChar *)"width") != NULL){
    wind->Largeur = atoi((char *)xmlGetProp(node, (xmlChar *)"width"));
  }
  if(xmlGetProp(node, (xmlChar *)"posX") != NULL){
    wind->posX = atoi((char *)xmlGetProp(node, (xmlChar *)"posX"));
  }
  if(xmlGetProp(node, (xmlChar *)"posY") != NULL){
    wind->posY = atoi((char *)xmlGetProp(node, (xmlChar *)"posY"));
  }
  if(xmlGetProp(node, (xmlChar *)"red") != NULL){
    wind->rouge = atoi((char *)xmlGetProp(node, (xmlChar *)"red"));
  }
  if(xmlGetProp(node, (xmlChar *)"green") != NULL){
    wind->vert = atoi((char *)xmlGetProp(node, (xmlChar *)"green"));
  }
  if(xmlGetProp(node, (xmlChar *)"blue") != NULL){
    wind->bleu = atoi((char *)xmlGetProp(node, (xmlChar *)"blue"));
  }
  if(xmlGetProp(node, (xmlChar *)"resizable") != NULL){
    if(strcmp((char *)xmlGetProp(node, (xmlChar *)"resizable"),"TRUE")==0) wind-
>isResisable = TRUE ;
    if(strcmp((char *)xmlGetProp(node, (xmlChar *)"resizable"),"FALSE")==0) wind-
>isResisable = FALSE ;
  }
  if(xmlGetProp(node, (xmlChar *)"fullscreen") != NULL){
    if(strcmp((char *)xmlGetProp(node, (xmlChar *)"fullscreen"),"TRUE")==0) wind-
>isFullscreen = TRUE ;
    if(strcmp((char *)xmlGetProp(node, (xmlChar *)"fullscreen"),"FALSE")==0)
wind->isFullscreen = FALSE ;
  }
  if(xmlGetProp(node, (xmlChar *)"decorated") != NULL){
    if(strcmp((char *)xmlGetProp(node, (xmlChar *)"decorated"),"TRUE")==0) wind-
>isDecorated = TRUE ;
    if(strcmp((char *)xmlGetProp(node, (xmlChar *)"decorated"),"FALSE")==0) wind-
>isDecorated = FALSE ;
  }
```

```
  if(xmlGetProp(node, (xmlChar *)"deletable") != NULL){
    if(strcmp((char *)xmlGetProp(node, (xmlChar *)"deletable"),"TRUE")==0) wind-
>isDeletable = TRUE ;
    if(strcmp((char *)xmlGetProp(node, (xmlChar *)"deletable"),"FALSE")==0) wind-
>isDeletable = FALSE ;
  }
  return wind;
}
```

Label:

```
Label *Readlabel(xmlNodePtr node,Label *label){
    if(xmlGetProp(node, (xmlChar *)"justify") != NULL){
        label->justify = (char *)xmlGetProp(node, (xmlChar *)"justify");
    }
    if(xmlGetProp(node, (xmlChar *)"text") != NULL){
        label->text = (char *)xmlGetProp(node, (xmlChar *)"text");
    }
    if(xmlGetProp(node, (xmlChar *)"y") != NULL){
        label->y = atoi((char *)xmlGetProp(node, (xmlChar *)"y"));
    }
    if(xmlGetProp(node, (xmlChar *)"x") != NULL){
        label->x = atoi((char *)xmlGetProp(node, (xmlChar *)"x"));
    }
    return label;
}
```

textview:

```
TextView *ReadTextview(xmlNodePtr node,TextView *txt){

    if(xmlGetProp(node, (xmlChar *)"style") != NULL){
        txt->style = (char *)xmlGetProp(node, (xmlChar *)"style");
    }
    if(xmlGetProp(node, (xmlChar *)"weight") != NULL){
        txt->weight = (char *)xmlGetProp(node, (xmlChar *)"weight");
    }
    if(xmlGetProp(node, (xmlChar *)"background") != NULL){
        txt->background = (char *)xmlGetProp(node, (xmlChar *)"background");
    }
    if(xmlGetProp(node, (xmlChar *)"foreground") != NULL){
        txt->foreground = (char *)xmlGetProp(node, (xmlChar *)"foreground");
    }
    if(xmlGetProp(node, (xmlChar *)"text") != NULL){
        txt->text = (char *)xmlGetProp(node, (xmlChar *)"text");
    }
    if(xmlGetProp(node, (xmlChar *)"size") != NULL){
        txt->size = atoi((char *)xmlGetProp(node, (xmlChar *)"size"));
    }
    if(xmlGetProp(node, (xmlChar *)"left_margin") != NULL){
        txt->left_margin = atoi((char *)xmlGetProp(node, (xmlChar
*)"left_margin"));
    }

    return txt;
}
```

Scale:

```
ScaleProperties* ReadScale(xmlNodePtr node,ScaleProperties *scale){
    if(xmlGetProp(node, (xmlChar *)"draw_value") != NULL){
        if(strcmp((char *)xmlGetProp(node, (xmlChar *)"draw_value"),"TRUE")==0)
scale->draw_value = TRUE ;
        if(strcmp((char *)xmlGetProp(node, (xmlChar *)"draw_value"),"FALSE")==0)
scale->draw_value = FALSE ;
    }
    if(xmlGetProp(node, (xmlChar *)"orientation") != NULL){
        if(strcmp((char *)xmlGetProp(node, (xmlChar
*)"orientation"),"vertical")==0) scale->orientation = GTK_ORIENTATION_VERTICAL ;
        if(strcmp((char *)xmlGetProp(node, (xmlChar
*)"orientation"),"horizontal")==0) scale->orientation =
GTK_ORIENTATION_HORIZONTAL ;
        g_print("%s",(char *)xmlGetProp(node, (xmlChar *)"orientation"));
    }
    if(xmlGetProp(node, (xmlChar *)"min_value") != NULL){
        scale->min_value = atoi((char *)xmlGetProp(node, (xmlChar
*)"min_value"));
    }
    if(xmlGetProp(node, (xmlChar *)"max_value") != NULL){
        scale->max_value = atoi((char *)xmlGetProp(node, (xmlChar
*)"max_value"));
    }
    if(xmlGetProp(node, (xmlChar *)"step_increment") != NULL){
        scale->step_increment = atoi((char *)xmlGetProp(node, (xmlChar
*)"step_increment"));
    }
    if(xmlGetProp(node, (xmlChar *)"page_increment") != NULL){
        scale->page_increment = atoi((char *)xmlGetProp(node, (xmlChar
*)"page_increment"));
    }
    if(xmlGetProp(node, (xmlChar *)"value") != NULL){
        scale->value = atoi((char *)xmlGetProp(node, (xmlChar *)"value"));
    }
    if(xmlGetProp(node, (xmlChar *)"digits") != NULL){
        scale->digits = atoi((char *)xmlGetProp(node, (xmlChar *)"digits"));
    }
    return scale;
}
```

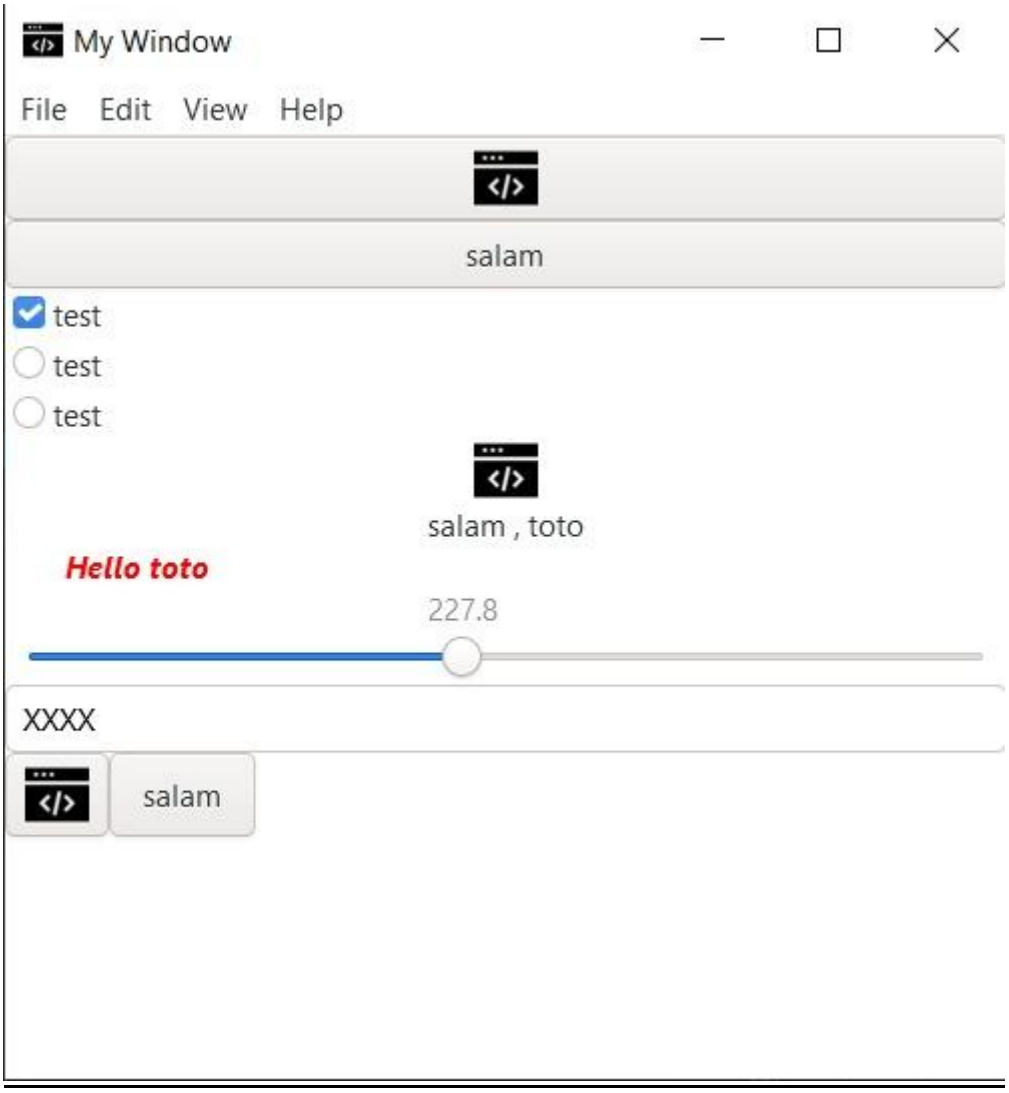
Image:

```
Image *ReadImage(xmlNodePtr node,Image *image){
    if(xmlGetProp(node, (xmlChar *)"file") != NULL){
        image->file = (char *)xmlGetProp(node, (xmlChar *)"file");
    }
    if(xmlGetProp(node, (xmlChar *)"height") != NULL){
        image->y = atoi((char *)xmlGetProp(node, (xmlChar *)"height"));
    }
    if(xmlGetProp(node, (xmlChar *)"width") != NULL){
        image->x = atoi((char *)xmlGetProp(node, (xmlChar *)"width"));
    }
    return image;
}
```

3-Exemple du fichier

```
<?xml version="1.0" encoding="UTF-8"?>
<window title="My Window" width="500" height="500" posX="0" posY ="0" Red="65535"
Green="65535" Blue="65535" icon="prebot.ico">
   <box orientation="vertical">
      <menu>
         <item label="File">
            <item label="New"/>
            <item label="Open"/>
            <item label="Save"/>
            <item label="Save As..."/>
            <separator/>
            <item label="Quit"/>
         </item>
         <item label="Edit">
            <item label="Cut"/>
            <item label="Copy"/>
            <item label="Paste"/>
            <item label="Delete"/>
         </item>
         <item label="View">
            <item label="Toolbar"/>
            <item label="Status Bar"/>
         </item>
         <item label="Help">
            <item label="About"/>
         </item>
      </menu>
      <simplebutton image="prebot.ico"/>
      <simplebutton label="salam"/>
      <checkbutton label="test" active="yes"/>
      <radiobutton label="test" />
      <radiobutton label="test" />
      <image file="prebot.ico" width="0" height="0"/>
      <label text="salam , toto" justify="right"/>
      <textview text="Hello toto" weight="gras" size="20" background="white"
forground="red"/>
      <scale orientation="horizontal" min_value="0" max_value="500" value="250"/>
      <entry text="test" editable="FALSE" visible="FALSE" invisiblechar="X"/>
      <box orientation="horizontal">
         <simplebutton image="prebot.ico"/>
         <simplebutton label="salam"/>
      </box>
   </box>
</window>
```

3-Exemple d’exécution :



Conclusion :

En conclusion, ce projet montre comment utiliser la bibliothèque GTK+ pour créer des interfaces utilisateur graphiques à partir de fichiers XML. Il peut être utilisé comme point de départ pour développer des applications graphiques en C., nous pouvons maintenant créer n’importe quelle interface graphique en choisissant soit la création directe par les fonctions ou par le fichier.