GIMP Toolkit: GTK+ v1.2

Alexis Nédélec

Ecole Nationale d'Ingénieurs de Brest
Technopôle Brest-Iroise, Site de la Pointe du Diable
CP 15 29608 BREST Cedex (FRANCE)
e-mail : nedelec@enib.fr

Table des Matières

Introduction

On peut définir GTK comme:

- ▷ une API "Orienté Objet"
- ▷ pour le développement d'IHM graphiques (GUI)
- ▷ sous Licence GNU (LGPL)

Glossaire de Sigles:

- ▷ **GNU** : **G**NU's **N**ot **U**nix ou "Vive le Logiciel Libre !".
- ▷ **GNOME** : **G**NU **N**etwork **O**bject **M**odel **E**nvironment
- ▷ **GIMP** : **G**eneral **I**mage **M**anipulation **P**rogram
- ▷ **GDK** : **G**IMP **D**rawing **K**it
- ▷ **GTK** : **G**IMP **T**ool**K**it

Introduction

Le Projet GNOME:

- ▷ 1997: Miguel de Icaza du "Mexican Autonomous National University"
- ▷ objectifs : développement de logiciels libres (open source)
- ▷ inspiré des développements de KDE (Qt)

GNOME est basé sur un ensemble de librairies existantes

- ▷ `glib`: utilitaire pour la création et manipulation de structures
- ▷ `GTK+`: Boîte à outils pour le développement d'IHM graphiques
- ▷ `ORBit`: Le Broker GNOME (CORBA 2.2) pour la distribution d'objets
- ▷ `Imlib`: pour la manipulation d'images sous X Window et GDK

Introduction

Librairies spécifiques du projet GNOME

- ▷ `libgnome`: utilitaires (non-GUI) de bases pour toute application GNOME
- ▷ `libgnomeui`: extensions de `libgnome` pour le développement de GUI
- ▷ `libgnorba`: services CORBA (activations d'objets, sécurité, ...)
- ▷ `libzvt`: widget terminal (`ZvtTerm`) intégrable dans les applications
- ▷ `libart_lgpl`: pour le rendu graphique (Raph Levien)
- ▷ `gnome-print`: expérimental pour le format d'impression (`libart_lgpl`)
- ▷ `gnome-xml`: analyseur de fichier XML en structure d'arbre et réciproquement
- ▷ `Guile`: gestion du langage de programmation Scheme
- ▷ `Bonobo`: gestion de documents composites (équivalent des OLE)

Introduction

GTK fournit les spécificités nécessaires au développement d'application GNOME

- ▷ spécialisation par héritage
- ▷ infrastructure signal/callbacks
- ▷ ensembles de widgets dérivés de `GtkWidget`

Gnome enrichit cette librairie avec ses propres widgets (`libgnomeui`)

- ▷ gestion de fenêtre (`GnomeApp, GnomeDock`)
- ▷ gestion d'affichage (`GnomeCanvas, GnomePixmap`)
- ▷ fonctions de convénience pour la création de dialogue
- ▷ ...

Introduction

GTK est implémenté en C (portabilité, performance), portages possible vers:

- ▷ C++, Guile, Perl,
- ▷ Python, TOM, Ada,
- ▷ Objective C, Free Pascal, Eiffel

Globalement on retrouvera les widgets de type

- ▷ affichage d'informations
- ▷ gadgets avec interaction
- ▷ containers de widgets

Il existe un générateur d'IHM: **glade** (version 0.5.5)

Introduction

Didacticiel GTK

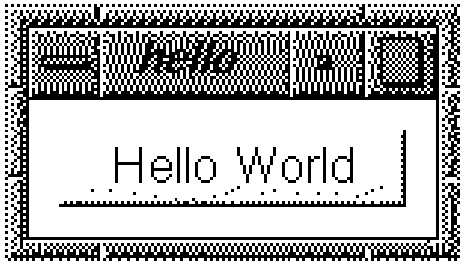
`http://www.gtk.org/tutorial`

- ▷ **"Hello World"**: les débuts en GTK
- ▷ **Widget Overview**: types de widgets GTK et étapes de création
- ▷ **Packing Widgets**: agencement de widgets dans des containers
- ▷ **The Button Widget**: normal, toggle, check et radio
- ▷ **Adjustments**: mécanismes de réarrangement des widgets
- ▷ **Range Widgets**: scrollbars, scale widgets
- ▷ **Miscellaneous Widgets**: labels, tooltips, dialogs, pixmaps,...
- ▷ **Container Widgets**: Paned Window, scrolled window, button boxes

Introduction

- **CList Widget**: rangement multi-colonnes
- **Tree Widget**: organisation hiérarchique des données
- **Menu Widget**: gestion des menus
- **Text Widget**: édition et affichage de textes (cut, paste, multi-click,...)
- **Undocumented Widgets**
- **Setting Widget Attributes**: titre, sensibilité, accélérateurs, ...
- **Timeouts, IO and Idle Functions**
- **Advanced Event and Signal Handling**
- **Managing Selections**: gestion ICCCM d' XWindow
- **Writing Your Own Widgets**

Premier Programme




```
{logname@hostname} gtk-config --cflags
-I/usr/X11R6/include -I/usr/lib/glib/include
{logname@hostname} gtk-config --libs
-L/usr/lib -L/usr/X11R6/lib -lgtk -lgdk -rdynamic -lgmodule -lglib
-ldl -lXext -lX11 -lm
{logname@hostname} make
gcc -c -Wall -c -g `gtk-config --cflags` hello.c
gcc -o hello hello.o  `gtk-config --libs`
{logname@hostname} hello
Hello
{logname@hostname} hello World
Hello World
```

Premier Programme

Définition des fonctions réflexes pour

- ▷ sortir de l'application: `destroy_cb()`
- ▷ interagir sur le widget de l'application: `hello_cb()`

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

void
destroy_cb( GtkWidget *widget, gpointer client_data )
{
  g_print ("destroy and exit application\n");
  gtk_main_quit();
}

void
hello_cb( GtkWidget *widget, gpointer client_data )
{
  char* data = (char *) client_data;
   if (data != NULL ) g_print ("Hello %s \n", data);
  else printf("Hello \n");
}
```

Premier Programme

```
int main( int   argc, char *argv[] )
{
  GtkWidget *window;
  GtkWidget *button;

  gtk_init(&argc, &argv);
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_signal_connect ( GTK_OBJECT (window),
                       "destroy",
                       GTK_SIGNAL_FUNC (destroy_cb),
                       NULL );
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);

  button = gtk_button_new_with_label ("Hello World");
  gtk_signal_connect ( GTK_OBJECT (button),
                       "clicked",
                       GTK_SIGNAL_FUNC (hello_cb),
                       argv[1]);
  gtk_container_add (GTK_CONTAINER (window), button);
  gtk_widget_show (button);
  gtk_widget_show (window);
  /* gtk_widget_show_all(window) */
  gtk_main ();
  return(0);
}
```

Premier Programme

Programmation de base d'une application GTK

- ▷ implémentation des fonctions réflexes: `function_cb()`
- ▷ initialisation de l'application GTK: `gtk_init()`
- ▷ création des widgets de l'application
  - ◇ fenêtre principale: `gtk_window_new()`
  - ◇ boutton-poussoir: `gtk_button_new_with_label()`
- ▷ association réflexe/signal sur les widgets: `gtk_signal_connect()`
- ▷ gestion de widgets dans un container: `GTK_CONTAINER()`
- ▷ affichage de widgets: `gtk_widget_show_all()`, `gtk_widget_show()`

Signaux et Réflexes

"Théorie" des signaux et des réflexes

- ▷ Lorsqu'un widget recoit un événement, il émet un signal approprié
- ▷ les signaux sont commun à tout type de widget (`destroy`)
- ▷ ou spécifiques aux widgets (`clicked` sur un bouton poussoir)

On connecte un signal à un widget `GtkObject` par la fonction

```
gint gtk_signal_connect( GtkObject     *object,
                         gchar         *event_name,
                         GtkSignalFunc  func,
                         gpointer       client_data );
```

Signature d'une fonction réflexe associée au signal:

```
void (*GtkSignalFunc) ( GtkWidget *widget, gpointer client_data );
```

Signaux et Réflexes

Il existe une autre fonction pour connecter des signaux sur des objets

```
gint gtk_signal_connect_object( GtkObject     *object,
                                gchar         *name,
                                GtkSignalFunc  func,
                                GtkObject     *slot_object );
```

Signature d'une fonction réflexe associée au signal:

```
void (*GtkSignalFunc) ( GtkObject *object );
```

Utilisation pour les fonctions n'admettant qu'un seul argument (`GtkObject`),
P.S. : Il en existe un nombre important dans la bibliothèque `GTK`

Signaux et Réflexes

Exploitation des structures d'événements

▷ `GdkEvent`: équivalent de l'union `XEvent` sous XWindow.

Signature des fonctions réflexes associées

```
void function_cb( GtkWidget *widget,
                  GdkEventAny  *event,
                  gpointer   client_data );
```

Exemple de connexion de signal sur un événement

```
button = gtk_button_new_with_label ("Hello World");
gtk_signal_connect( GTK_OBJECT(button),
                    "button_press_event",
                    GTK_SIGNAL_FUNC(button_press_cb),
                    NULL);
```

Signaux et Réflexes

Liste des événements exploitables

```
button_press_event
button_release_event
motion_notify_event
delete_event
destroy_event
expose_event
key_press_event
key_release_event
enter_notify_event
leave_notify_event
configure_event
focus_in_event
focus_out_event
map_event
unmap_event
property_notify_event
selection_clear_event
selection_request_event
selection_notify_event
proximity_in_event
proximity_out_event
drag_begin_event
drag_request_event
drag_end_event
drop_enter_event
drop_leave_event
drop_data_available_event
other_event
```

Signaux et Réflexes

Déconnexion d'un signal associé à un objet

```
gint gtk_signal_connect( GtkObject *object,
                         gchar *name,
                         GtkSignalFunc func,
                         gpointer client_data );
```

Valeur de retour de la fonction:

▷ identifiant du signal associé à l'objet.

On déconnecte le signal de l'objet par cet identifiant

```
void gtk_signal_disconnect( GtkObject *object, gint id )
```

Déconnexion de tous les signaux associés à un objet

```
void gtk_signal_handlers_destroy( GtkObject *object )
```

Signaux et Réflexes

Déconnexion **temporaire** d'un signal associé à un objet

```
void gtk_signal_handler_block(GtkObject *object, guint handler_id)

void gtk_signal_handler_block_by_func(GtkObject     *object,
                                      GtkSignalFunc  func, gpointer data)

void gtk_signal_handler_block_by_data(GtkObject *object, gpointer data)


void gtk_signal_handler_unblock(GtkObject *object, guint handler_id)

void gtk_signal_handler_unblock_by_func(GtkObject     *object,
                                        GtkSignalFunc  func, gpointer data)

void gtk_signal_handler_unblock_by_data(GtkObject *object, gpointer data)
```

Signaux et Réflexes

Exemple de gestion d'un filtre pour editer des chiffres

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

void filtre (GtkEditable *edit, const gchar* text,
             gint length, gint *pos, gpointer client_data )
{
  int i, new_length = 0;
  gchar *new_text = g_malloc(length);

  for (i=0;i<length ;i++) {
    if (text[i]>'0' && text[i]<'9') {
      new_text[new_length++]= text[i];
    }
    else gdk_beep();
  }
  if ( new_length > 0) {
      gtk_signal_handler_block_by_func( GTK_OBJECT(edit), (GtkSignalFunc) filtre, client_data);
      gtk_editable_insert_text(edit, new_text, new_length, pos);
      gtk_signal_handler_unblock_by_func( GTK_OBJECT(edit), (GtkSignalFunc) filtre, client_data);
  }
  gtk_signal_emit_stop_by_name( GTK_OBJECT(edit), "insert_text");
  g_free(new_text);
}
```

Signaux et Réflexes

```
int main( int argc,char *argv[] )
{
  GtkWidget *window;
  GtkWidget *entry;

  gtk_init(&argc, &argv);
  window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "test signal");
  gtk_container_set_border_width(GTK_CONTAINER(window), 10);
  gtk_signal_connect( GTK_OBJECT(window), "delete_event", (GtkSignalFunc) gtk_exit, NULL);
  entry= gtk_entry_new();
  gtk_signal_connect( GTK_OBJECT(entry), "insert_text", (GtkSignalFunc) filtre, NULL);
  gtk_container_add(GTK_CONTAINER(window), entry);
  gtk_widget_show_all(window);
  gtk_main();
  return 0;
}
```

Description de Widgets

La création de widget se fait suivant les étapes

1. création (instanciation) de widget: `gtk_*_new_*()`
2. connexion des signaux et événements: `gtk_signal_connect()`
3. fixer les attributs du widget, par exemple: `gtk_widget_set_name()`
4. positionnement du widget dans un container:
   - ▷ `gtk_container_add()`
   - ▷ `gtk_box_pack_start()`
5. visualisation du widget: `gtk_widget_show()`

Transtypage

GTK utilise un système de transtypage avec des macros pour

- ▷ tester la possibilité de transtypage
- ▷ effectuer le transtypage

Quelques unes de ces macros

- ▷ `GTK_WINDOW( aWidget )`
- ▷ `GTK_WIDGET( aWidget )`
- ▷ `GTK_SIGNAL_FUNC( aFunction )`
- ▷ `GTK_CONTAINER( aWidget )`
- ▷ `GTK_WIDGET_NO_WINDOW( aWidget )`
- ▷ `GTK_WIDGET_REAlIZED( aWidget )`
- ▷ `GTK_WIDGET_MAPPED( aWidget )`
- ▷ `GTK_WIDGET_VISIBLE( aWidget )`
- ▷ ...

Cycle de Vie

Destruction de widget en cours d'éxécution

- ▷ destruction de widgets de "haut niveau" (conteneurs):
  - ◇ `gtk_widget_destroy( GtkWidget* )`
- ▷ destruction de widgets à l'intérieur d'un conteneur:
  - ◇ destruction complète
    - ○ `gtk_container_remove(GtkWidget* aContainer, GtkWidget* aWidget);`
  - ◇ enlever le widget du container
    - ○ `gtk_widget_ref(GtkWidget*);`
    - ○ `gtk_container_remove(GtkWidget* aContainer, GtkWidget* aWidget);`

Réalisation, mappage, affichage

comme sous X window

- ▷ Une fenêtre X est une `GdkWindow`
- ▷ à un widget GTK correspond une fenêtre `GdkWindow`
- ▷ certain widgets (`GtkLabel...`) n'ont pas de `GdkWindow` associés

Etats d'un `GtkWidget` par rapport à sa fenêtre `GdkWindow` associée

- ▷ Réalisé, le widget est associé à sa fenêtre:
  - ◇ `gtk_widget_realize(GtkWidget*)`
  - ◇ `gtk_widget_unrealize(GtkWidget*)`
- ▷ Mappé, la fenêtre sera rendue visible à l'écran:
  `gtk_widget_map(GtkWidget*), gtk_widget_unmap(GtkWidget*)`
- ▷ rendu visible, ce que l'on utilise généralement
  `gt_widget_show(GtkWidget*), gtk_widget_hide(GtkWidget*)`

Gestion des événements

Gestion automatique de mémoire en sortie d'application

`gtk_main_quit()`

Gestion de boucle principale:

- ▷ `gtk_main()`: lancement de boucle (récursivité)
- ▷ `gtk_main_quit()`: arrêt d'une boucle de traitement d'événements
- ▷ `guint gtk_main_level()`: niveau de récursivité
- ▷ `gtk_main_iteration()`: gestion prioritaire d'un événement
- ▷ `gint gtk_events_pending()`: pour contrôler les événements de la boucle

Hiérarchie de Widgets

```
GtkObject
      +GtkWidget
      | +GtkMisc
      | | +GtkLabel
      | | | +GtkAccelLabel
      | | | `GtkTipsQuery
      | | +GtkArrow
      | | +GtkImage
      | | `GtkPixmap
      | +GtkContainer
      | | +GtkBin
      | | | +GtkAlignment
      | | | +GtkFrame
      | | | | `GtkAspectFrame
      | | | +GtkButton
      | | | | +GtkToggleButton
      | | | | | `GtkCheckButton
      | | | | |   `GtkRadioButton
      | | | | `GtkOptionMenu
      | | | +GtkItem
      | | | | +GtkMenuItem
      | | | | | +GtkCheckMenuItem
      | | | | | | `GtkRadioMenuItem
      | | | | | `GtkTearoffMenuItem
```

Hiérarchie de Widgets

```
| | | | +GtkListItem
| | | | `GtkTreeItem
| | | +GtkWindow
| | | | +GtkColorSelectionDialog
| | | | +GtkDialog
| | | | | `GtkInputDialog
| | | | +GtkDrawWindow
| | | | +GtkFileSelection
| | | | +GtkFontSelectionDialog
| | | | `GtkPlug
| | | +GtkEventBox
| | | +GtkHandleBox
| | | +GtkScrolledWindow
| | | `GtkViewport
| | +GtkBox
| | | +GtkButtonBox
| | | | +GtkHButtonBox
| | | | `GtkVButtonBox
| | | +GtkVBox
| | | | +GtkColorSelection
| | | | `GtkGammaCurve
| | | `GtkHBox
| | |   +GtkCombo
| | |   `GtkStatusbar
```

Hiérarchie de Widgets

```
| | +GtkCList
| | | `GtkCTree
| | +GtkFixed
| | +GtkNotebook
| | | `GtkFontSelection
| | +GtkPaned
| | | +GtkHPaned
| | | `GtkVPaned
| | +GtkLayout
| | +GtkList
| | +GtkMenuShell
| | | +GtkMenuBar
| | | `GtkMenu
| | +GtkPacker
| | +GtkSocket
| | +GtkTable
| | +GtkToolbar
| | `GtkTree
| +GtkCalendar
| +GtkDrawingArea
| | `GtkCurve
```

Hiérarchie de Widgets

```
| +GtkEditable
| | +GtkEntry
| | | `GtkSpinButton
| | `GtkText
| +GtkRuler
| | +GtkHRuler
| | `GtkVRuler
| +GtkRange
| | +GtkScale
| | | +GtkHScale
| | | `GtkVScale
| | `GtkScrollbar
| |   +GtkHScrollbar
| |   `GtkVScrollbar
| +GtkSeparator
| | +GtkHSeparator
| | `GtkVSeparator
| +GtkPreview
| `GtkProgress
|   `GtkProgressBar
+GtkData
| +GtkAdjustment
| `GtkTooltips
`GtkItemFactory
```

Widget sans fenêtre

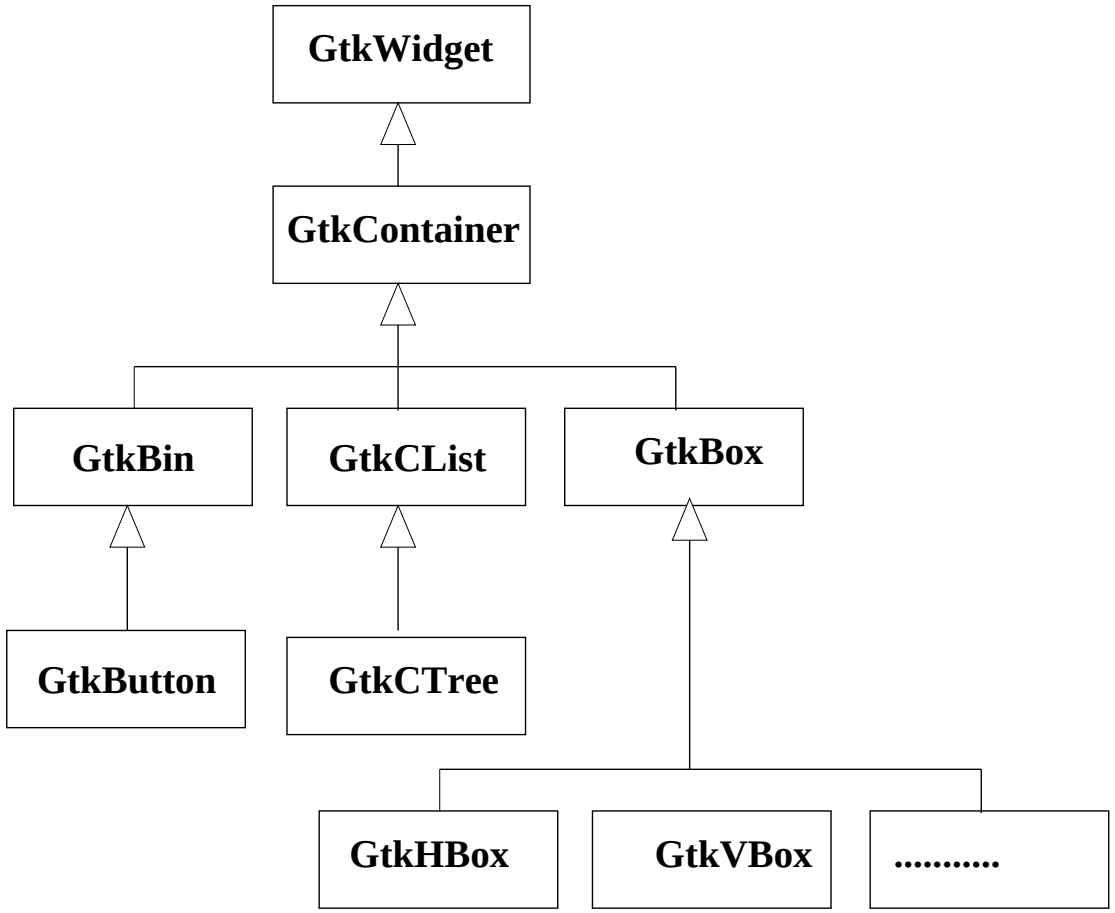
Les widgets suivants ne sont pas associés à des fenêtres

`GtkAlignment`
`GtkArrow`
`GtkBin`
`GtkBox`
`GtkImage`
`GtkItem`
`GtkLabel`
`GtkPixmap`
`GtkScrolledWindow`
`GtkSeparator`
`GtkTable`
`GtkAspectFrame`
`GtkFrame`
`GtkVBox`
`GtkHBox`
`GtkVSeparator`
`GtkHSeparator`

Pour capter des événements sur ce type de widget

▷ utiliser des `EventBox`

Containers de Widget



classe `GtkBox`

Dans le programme précédent

▷ `gtk_container_add (GTK_CONTAINER (window), button);`

Si plusieurs widgets, comment les agencer dans une fenêtre ?

▷ utilisation de containers: `BOX, TABLE, CLIST ...`

Container en ligne / colonne:

▷ `GtkHbox` : sur une ligne, gestion de la largeur des descendants

▷ `GtkVbox` : sur une colonne, gestion de la hauteur des descendants

Création du container

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

Arguments:

▷ `homogeneous`: largeur ou hauteur identique pour les widgets

▷ `spacing`: espacement entre widgets (avec leur "emballage")

classe `GtkBox`

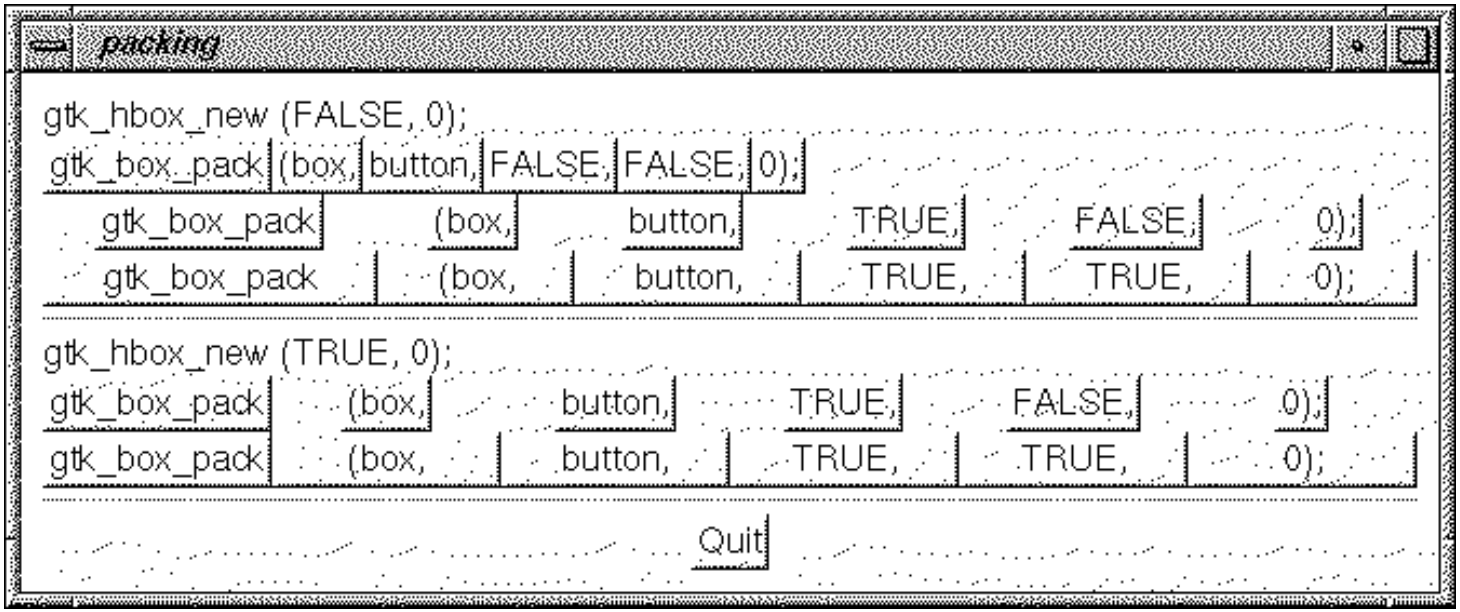
Insertion de descendants à l'intérieur du container

```
void gtk_box_pack_start( GtkBox    *box, GtkWidget *child,
                         gboolean expand, gboolean fill,
                         gint     padding );
```

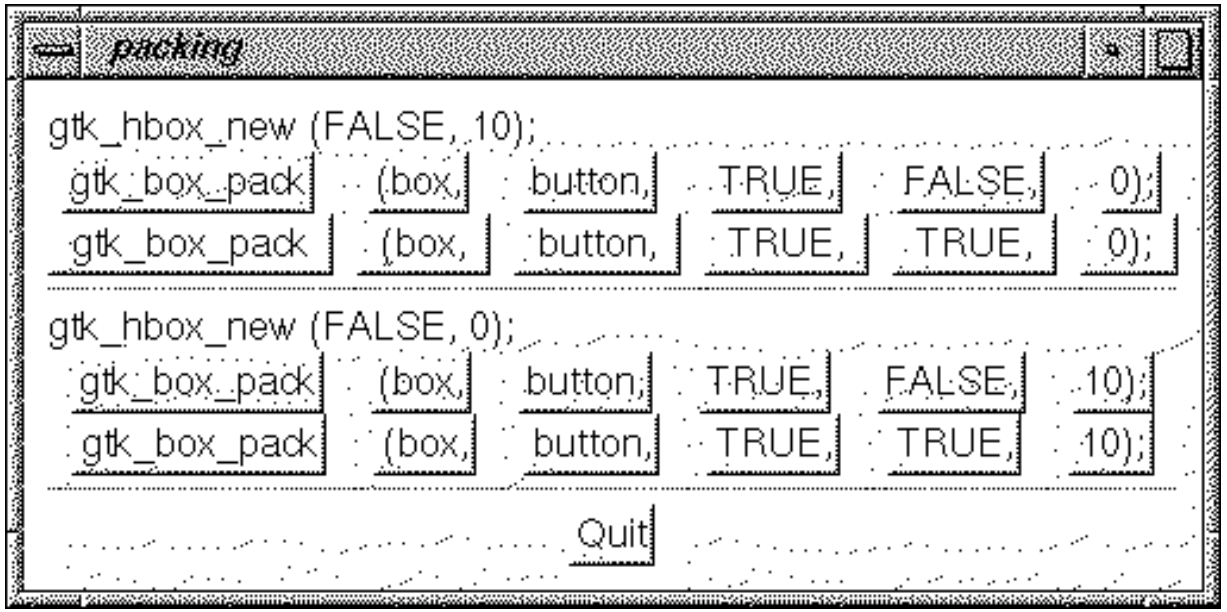
Arguments :

- ▷ `box,child` : le container utilisé et l'objet à insérer
- ▷ `expand` : booléen, pour occuper l'espace restant
- ▷ `fill` : espace occupé par l'enfant (`TRUE`) ou l' "emballage" (`FALSE`) ne sert que si `expand` a la valeur `TRUE`
- ▷ `padding` : largeur d' "emballage" de l'enfant (de chaque coté)

classe `GtkBox`



classe `GtkBox`



classe `GtkTable`

Création de TABLES: agencement de widgets en cellules

```
GtkWidget *gtk_table_new( guint nrows, guint ncolumns,
                          gboolean homogeneous );
```

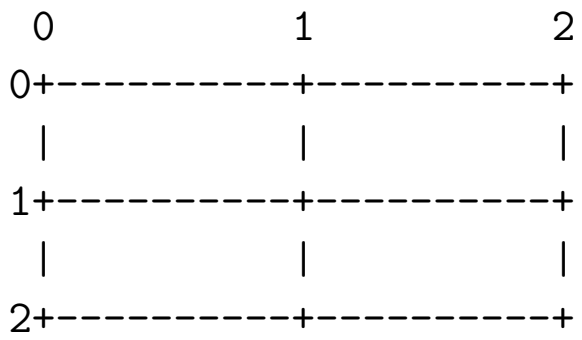
Arguments:

▷ `nrows`: nombre de lignes

▷ `ncolumns`: nombre de colonnes

▷ `homogeneous` : si `TRUE`, les widgets auront la taille du plus grand

Exemple:



classe `GtkTable`

Placement de widgets dans des TABLES

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

classe `GtkTable`

Options de placements (`xoptions, yoptions`)

- ▷ `GTK_EXPAND, GTK_FILL, GTK_SHRINK`

`GTK_SHRINK`: si espace insuffisant, le widget à insérer est reduit.
Il existe des fonctions pour gérer les options par defaut

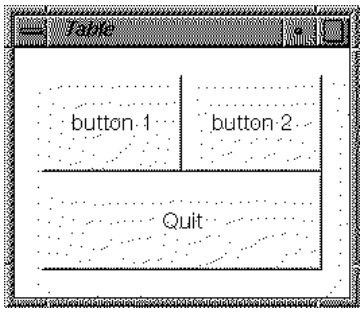
```
void gtk_table_attach_defaults( GtkTable  *table,
                                GtkWidget *widget,
                                gint       left_attach,
                                gint       right_attach,
                                gint       top_attach,
                                gint       bottom_attach );
```

qui affecteront les valeurs suivantes aux widgets à insérer

- ▷ `GTK_FILL | GTK_EXPAND`
- ▷ `xpadding = ypadding = 0`

Containers de Widget

Programme de création de table :




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

void
function_cb( GtkWidget *widget, gpointer client_data )
{
    char *data =  (char *) client_data;
    g_print ("Hello again - %s was pressed\n", data);
}
gint
delete_ev( GtkWidget *widget,GdkEvent  *event, gpointer   data )
{
    gtk_main_quit ();
    return(FALSE);
}
```

Containers de Widget

```
int main( int   argc, char *argv[] ) {
  GtkWidget *window;
  GtkWidget *button;
  GtkWidget *table;

  gtk_init (&argc, &argv);
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "Table");

  gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                      GTK_SIGNAL_FUNC (delete_ev), NULL);
  gtk_container_set_border_width (GTK_CONTAINER (window), 20);

  table = gtk_table_new (2, 2, TRUE);
  gtk_container_add (GTK_CONTAINER (window), table);

  button = gtk_button_new_with_label ("button 1");
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (function_cb), (gpointer) "button 1");
```

Containers de Widget

```
  gtk_table_attach_defaults (GTK_TABLE(table), button, 0, 1, 0, 1);
  gtk_widget_show (button);

  button = gtk_button_new_with_label ("button 2");
  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (function_cb), (gpointer) "button 2");
  gtk_table_attach_defaults (GTK_TABLE(table), button, 1, 2, 0, 1);
  gtk_widget_show (button);

  button = gtk_button_new_with_label ("Quit");

  gtk_signal_connect (GTK_OBJECT (button), "clicked",
                      GTK_SIGNAL_FUNC (delete_event), NULL);

  gtk_table_attach_defaults (GTK_TABLE(table), button, 0, 2, 1, 2);
  gtk_widget_show (button);
  gtk_widget_show (table);
  gtk_widget_show (window);
  gtk_main ();
  return 0;
}
```

Entrées de Texte

Il existe différents widgets pour faire une saisie de texte

- ▷ `GtkEditable`: classe de base pour les propriétés communes
- ▷ `GtkEntry`: texte sur une seule ligne (jusqu'à 2047 caractères)
- ▷ `GtkAdjustment`: pour se placer dans un texte
- ▷ `GtkSpinButton`: pour entrer une valeur comprise entre deux bornes
- ▷ `GtkText`: texte multilignes (jusqu'à 4×10^9 caractères)
- ▷ `GtkCombo`: une `GtkEntry` et une `GtkList`

GtkEditable

Classe abstraite regroupant les propriétés générales de ce type de widgets:

- ▷ insertion / destruction de texte
- ▷ affichage du texte courant
- ▷ état de la zone de saisie (éditable ou non)
- ▷ position du curseur texte
- ▷ sélection de texte

Insertion de texte:

- ▷ `gtk_editable_insert_text( GtkEditable *editable,`
  `const gchar *text,`
  `gint length, gint *position)`

Signification des paramètres:

- ▷ `length=-1`: insertion de la totalité du texte sans connaître sa longueur
- ▷ `pos`: position du curseur de texte après l'insertion

GtkEditable

Destruction, récupération de texte:

- `gtk_editable_delete_text( GtkEditable *editable, gint start, gint stop)`
- `gchar *gtk_editable_get_chars( GtkEditable *editable, gint start, gint stop)`

Signification des paramêtres:

- `stop=-1`: destruction, récupération jusqu'à la fin du texte
- la chaîne retournée devra-être libéré (`gfree()`) une fois tilisée

Sélection de texte dans un programme:

- `gtk_editable_select_region( GtkEditable *editable, gint start, gint stop)`

La sélection avec la souris aura le même effet

GtkEditable

Gestion classique des opérations de sélection de texte:

- `gtk_editable_cut_clipboard( GtkEditable *editable )`
- `gtk_editable_copy_clipboard( GtkEditable *editable )`
- `gtk_editable_paste_clipboard( GtkEditable *editable )`

Positionnement du curseur:

- `gtk_editable_set_position( GtkEditable *editable, gint position )`
- `gint gtk_editable_get_position( GtkEditable *editable )`

Signification du paramètre **position** et du retour de fonction:

- nombre de caractères depuis le début du texte

Mode d'édition:

- `gtk_editable_set_editable( GtkEditable *editable, gboolean editable)`

GtkEditable

Signaux et signatures des réflexes associés au entrées de texte:

- **changed**, pour tout changement dans le texte
  - `callback(GtkEditable *editable, gpointer data)`
- `cut_clipboard, copy_clipboard, paste_clipboard`:
  - `callback(GtkEditable *editable, gpointer data)`
- `insert_text`:
  -
```
callback(GtkEditable *editable,
         const gchar *text,
         gint length, gint *position,
         gpointer data)
```
- `delete_text`:
  -
```
callback(GtkEditable *editable,
         gint start, gint stop, gpointer data)
```

GtkEntry

Gestion de texte sur une ligne (caractères visible ou non).
Fonction de création (2047 caractères par défaut).

- `GtkWidget *gtk_entry_new()`
- `GtkWidget *gtk_entry_new_with_max_length( guint16 max )`

Modification de la longueur maximale

- `gtk_entry_set_max_length( guint16 max )`

Insertion, récupération de texte:

- `gtk_entry_set_text( GtkEntry *entry, const gchar *text )`
- `gchar *text gtk_entry_get_text( GtkEntry *entry )`

Insertion dans du texte

- `gtk_entry_prepend_text( GtkEntry *entry, const gchar *text )`
- `gtk_entry_append_text( GtkEntry *entry, const gchar *text )`

GtkEntry

Positionnement de curseur, sélection et visbilité de texte

- ▷ `gtk_entry_set_position( GtkEntry *entry, gint position)`
- ▷ `gtk_entry_select_region( GtkEntry *entry, gint start, gint stop)`
- ▷ `gtk_entry_set_visibility( GtkEntry *entry, gboolean visible)`

Exemple d'utilisation en mode éditable et visible

GtkEntry

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

void enter_callback( GtkWidget *widget, GtkWidget *entry )
{
  gchar *entry_text;
  entry_text = gtk_entry_get_text(GTK_ENTRY(entry));
  printf("Entry contents: %s\n", entry_text);
}

void entry_toggle_editable( GtkWidget *checkbutton, GtkWidget *entry )
{
  gtk_entry_set_editable(GTK_ENTRY(entry), GTK_TOGGLE_BUTTON(checkbutton)->active);
}

void entry_toggle_visibility( GtkWidget *checkbutton, GtkWidget *entry )
{
  gtk_entry_set_visibility(GTK_ENTRY(entry), GTK_TOGGLE_BUTTON(checkbutton)->active);
}
```

GtkEntry

```
int main( int   argc, char *argv[] ) {
    GtkWidget *window;
    GtkWidget *vbox, *hbox;
    GtkWidget *entry;
    GtkWidget *button;
    GtkWidget *check;
/* GtkWindow: the Toplevel window */
    gtk_init (&argc, &argv);
    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_widget_set_usize( GTK_WIDGET (window), 200, 100);
    gtk_window_set_title(GTK_WINDOW (window), "GTK Entry");
    gtk_signal_connect(GTK_OBJECT (window), "delete_event", (GtkSignalFunc) gtk_exit, NULL);
 /* GtkVBox: container for GtkEntry (text), GtkHBox (editable, visible) and  GtkButton (close)*/
    vbox = gtk_vbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (vbox);
/* GtkEntry: The Entry text */
    entry = gtk_entry_new_with_max_length (50);
    gtk_signal_connect(GTK_OBJECT(entry), "activate", GTK_SIGNAL_FUNC(enter_callback), entry);
    gtk_entry_set_text (GTK_ENTRY (entry), "hello");
    gtk_entry_append_text (GTK_ENTRY (entry), " world");
    gtk_entry_select_region (GTK_ENTRY (entry), 0, GTK_ENTRY(entry)->text_length);
    gtk_box_pack_start (GTK_BOX (vbox), entry, TRUE, TRUE, 0);
    gtk_widget_show (entry);
```

GtkEntry

```
/* GtkHBox: container for 2 GtkCheckButton (editable, visible) */
    hbox = gtk_hbox_new (FALSE, 0);
    gtk_container_add (GTK_CONTAINER (vbox), hbox);
    gtk_widget_show (hbox);
    check = gtk_check_button_new_with_label("Editable");
    gtk_box_pack_start (GTK_BOX (hbox), check, TRUE, TRUE, 0);
    gtk_signal_connect (GTK_OBJECT(check), "toggled", GTK_SIGNAL_FUNC(entry_toggle_editable), entry);
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
    gtk_widget_show (check);
    check = gtk_check_button_new_with_label("Visible");
    gtk_box_pack_start (GTK_BOX (hbox), check, TRUE, TRUE, 0);
    gtk_signal_connect (GTK_OBJECT(check), "toggled", GTK_SIGNAL_FUNC(entry_toggle_visibility), entry);
    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
    gtk_widget_show (check);
/* add to the GtkVBox container an exit  GtkButton (close) */
    button = gtk_button_new_with_label ("Close");
    gtk_signal_connect_object (GTK_OBJECT (button),"clicked",GTK_SIGNAL_FUNC(gtk_exit),GTK_OBJECT (window));
    gtk_box_pack_start (GTK_BOX (vbox), button, TRUE, TRUE, 0);
    GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
    gtk_widget_grab_default (button);
    gtk_widget_show (button);
    gtk_widget_show(window);
    gtk_main();
    return(0);
}
```

GtkAdjustment

Les ajustements ne sont pas des `GtkWidget` mais des `GtkObjet`.

- ▷ utilisation lorsque l'on veut manipuler une valeur
- ▷ comprise entre 2 bornes
- ▷ modifiable par crans (incrément de pas, de page)

Deux groupes d'utilisation d'ajustements sur des widgets

- ▷ nécessitant des unités spécifiques pour les valeurs
  - ◇ `GtkScrolledWindow, GtkText ...`
- ▷ utlisant des valeurs arbitraires ajustés directement par l'utilisateur
  - ◇ `GtkScale, GtkSpinButton...`

Utilisable pour connecter des ajustements de widgets entre eux:

```
...
  text = gtk_text_new (NULL, NULL);
  vscrollbar = gtk_vscrollbar_new (GTK_TEXT(text)->vadj);
...
```

GtkAdjustment

Création et modification

- `GtkWidget *gtk_adjustment_new( gfloat value, gfloat lower, gfloat upper, gfloat step_increment, gfloat page_increment, gfloat page_size )`
- `gtk_adjustment_set_value( GtkAdjustment *hadj, GtkAdjustment *vadj)`

Signaux et signatures des réflexes associés au ajustements:

- `disconnect`: lorsque qu'il est détaché d'un widget
- `value_changed`: lorsque la valeur associée est modifiée
  - `callback(GtkAdjustment *adj, gpointer data)`

GtkText

Le widget de la bibliothèque GTK le plus complet de ce domaine

- ▷ affichage de texte sur un nombre (il)limité de lignes
- ▷ utilisation d'un ajustement vertical (l'horizontal est en cours!)
- ▷ gestion de couleurs et de fonts
- ▷ utilisation de commandes clavier d'édition (emacs)

Création d'un texte, non éditable par défaut:

- ▷ `GtkWidget *gtk_text_new( GtkAdjustment *hadj, GtkAdjustment *vadj)`
- ▷ conseil d'utilisation: `gtk_text_new( NULL, NULL);`

Mode dédition:

- ▷ `gtk_text_set_editable( GtkText *gtkText, gboolean editable)`

GtkText

Insertion de texte (le widget se redessine à chaque insertion):

▷ `gtk_text_insert( GtkText *gtkText, GdkFont *font, GdkColor *fg, GdkColor *bg, const char *text, gint length)`

s'il faut insérer plusieurs bouts de texte:

▷ avant insertions: `gtk_text_freeze( GtkText *gtkText )`

▷ après insertions: `gtk_text_thaw( GtkText *gtkText )`

Gestion du curseur de texte:

▷ `gtk_text_set_point( GtkText *gtkText, guint index )`

▷ `guint gtk_text_get_point( GtkText *gtkText )`

Suppression de texte à partir de la position du curseur:

▷ `gtk_text_backward_delete( GtkText *gtkText, guint length )`

▷ `gtk_text_forward_delete( GtkText *gtkText, guint length )`

GtkText

Gestion des affichages de “phrases” sur plusieurs lignes

▷ `gtk_text_set_line_wrap(GtkText *gtkText, gboolean line_wrap)`

Par défaut le mode est `TRUE`:

▷ fin de ligne non-visible affichée sur les lignes suivantes

▷ coupure à n’importe quel endroit du texte

Pour couper une ligne entre deux mots (booléen à `TRUE`):

▷ `gtk_text_set_word_wrap(GtkText *gtkText, gboolean word_wrap)`

Longueur totale du texte en caractères:

▷ `guint gtk_text_get_length( GtkText *gtkText )`

GtkText

Raccourcis clavier sur lignes

- ▷ `Ctrl-A`: se placer en début de ligne
- ▷ `Ctrl-E`: se placer en fin de ligne
- ▷ `Ctrl-N`: se placer à la ligne suivante
- ▷ `Ctrl-P`: se placer à la ligne précédente
- ▷ `Ctrl-K`: efface jusqu'à la fin de ligne
- ▷ `Ctrl-P`: efface toute la ligne

Raccourcis clavier sur mots

- ▷ `Alt-F`: se placer au mot suivant
- ▷ `Alt-B`: se placer au mot précédent
- ▷ `Ctrl-W`: efface un mot à gauche du curseur
- ▷ `Alt-D`: efface un mot à droite du curseur

GtkText

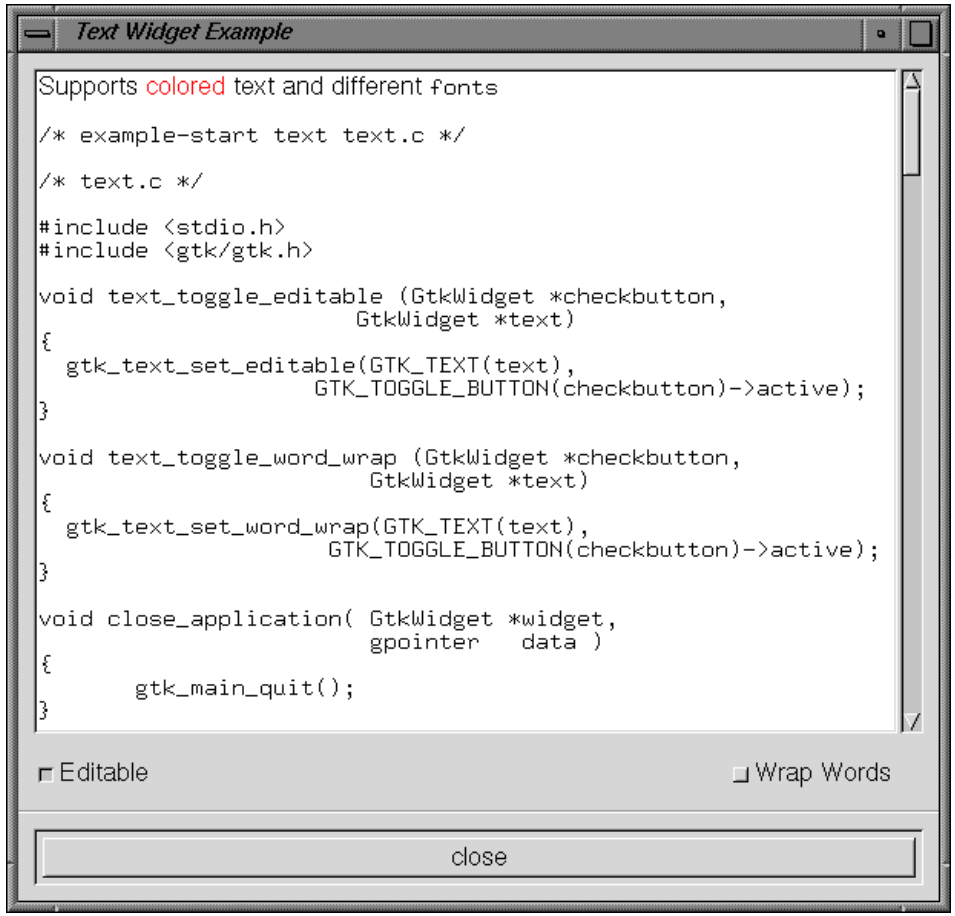
Raccourcis clavier sur caractères

- ▷ `Ctrl-B`: se placer en arrière d'un caractère
- ▷ `Ctrl-F`: se placer en avant d'un caractère
- ▷ `Ctrl-H`: efface un caractère en avant
- ▷ `Ctrl-W`: efface un caractère en arrière
- ▷ `Alt-B`: se placer au mot précédent
- ▷ `Ctrl-W`: efface un mot à gauche du curseur
- ▷ `Alt-D`: efface un mot à droite du curseur

Raccourcis clavier sur presse-papiers

- ▷ `Ctrl-X`: Couper la sélection vers le presse-papiers
- ▷ `Ctrl-C`: Copier la sélection vers le presse-papiers
- ▷ `Ctrl-V`: insérer le contenu du presse-papiers

GtkText


Text Widget Example
Supports colored text and different fonts
/* example-start text text.c */
/* text.c */
#include <stdio.h>
#include <gtk/gtk.h>
void text_toggle_editable (GtkWidget *checkbutton,
GtkWidget *text)
{
gtk_text_set_editable(GTK_TEXT(text),
GTK_TOGGLE_BUTTON(checkbutton)->active);
}
void text_toggle_word_wrap (GtkWidget *checkbutton,
GtkWidget *text)
{
gtk_text_set_word_wrap(GTK_TEXT(text),
GTK_TOGGLE_BUTTON(checkbutton)->active);
}
void close_application(GtkWidget *widget,
gpointer data)
{
gtk_main_quit();
}
Editable
Wrap Words
close

GtkText

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

void text_toggle_editable (GtkWidget *checkbutton, GtkWidget *text)
{
  gtk_text_set_editable(GTK_TEXT(text), GTK_TOGGLE_BUTTON(checkbutton)->active);
}

void text_toggle_word_wrap (GtkWidget *checkbutton, GtkWidget *text)
{
  gtk_text_set_word_wrap(GTK_TEXT(text), GTK_TOGGLE_BUTTON(checkbutton)->active);
}

void close_application( GtkWidget *widget, gpointer   data )
{
       gtk_main_quit();
}
```

GtkText

```
int main( int argc, char *argv[] ) {
  GtkWidget *window;
  GtkWidget *box1, *box2, *hbox;
  GtkWidget *button, *check, *separator;
  GtkWidget *table, *vscrollbar;
  GtkWidget *text;
  GdkColormap *cmap;
  GdkColor color;
  GdkFont *fixed_font;
  FILE *infile;
  gtk_init (&argc, &argv);
/* GtkWindow: the Toplevel window */
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_widget_set_usize (window, 600, 500);
  gtk_window_set_policy (GTK_WINDOW(window), TRUE, TRUE, FALSE);
  gtk_signal_connect (GTK_OBJECT (window), "destroy", GTK_SIGNAL_FUNC(close_application), NULL);
  gtk_window_set_title (GTK_WINDOW (window), "Text Widget Example");
  gtk_container_set_border_width (GTK_CONTAINER (window), 0);
/* GtkVbox: container for GtkVBox (box2)*/
  box1 = gtk_vbox_new (FALSE, 0);
  gtk_container_add (GTK_CONTAINER (window), box1);
  gtk_widget_show (box1);
```

GtkText

```
/* GtkVbox: container for GtkTable (table)*/
  box2 = gtk_vbox_new (FALSE, 10);
  gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
  gtk_box_pack_start (GTK_BOX (box1), box2, TRUE, TRUE, 0);
  gtk_widget_show (box2);
/* GtkTable:  container for GtkText and GtkScrollbar */
  table = gtk_table_new (2, 2, FALSE);
  gtk_table_set_row_spacing (GTK_TABLE (table), 0, 2);
  gtk_table_set_col_spacing (GTK_TABLE (table), 0, 2);
  gtk_box_pack_start (GTK_BOX (box2), table, TRUE, TRUE, 0);
  gtk_widget_show (table);
/* Create the GtkText widget */
  text = gtk_text_new (NULL, NULL);
  gtk_text_set_editable (GTK_TEXT (text), TRUE);
  gtk_table_attach (GTK_TABLE (table), text, 0, 1, 0, 1,
                    GTK_EXPAND | GTK_SHRINK | GTK_FILL, GTK_EXPAND | GTK_SHRINK | GTK_FILL, 0, 0);
  gtk_widget_show (text);
/* Add a vertical scrollbar to the GtkText widget with the text GtkAdjustment */
  vscrollbar = gtk_vscrollbar_new (GTK_TEXT (text)->vadj);
  gtk_table_attach (GTK_TABLE (table), vscrollbar, 1, 2, 0, 1,
                    GTK_FILL, GTK_EXPAND | GTK_SHRINK | GTK_FILL, 0, 0);
  gtk_widget_show (vscrollbar);
```

GtkText

```
  cmap = gdk_colormap_get_system();
  color.red = 0xffff;
  color.green = 0;
  color.blue = 0;
  if (!gdk_color_alloc(cmap, &color)) g_error("couldn't allocate color");
  fixed_font = gdk_font_load ("-misc-fixed-medium-r-*-*-*-140-*-*-*-*-*-*");
  gtk_widget_realize (text); /* Realizing a widget creates a window for it, ready to insert some text */
  gtk_text_freeze (GTK_TEXT (text)); /* Freeze the text widget, ready for multiple updates */
/* Insert some colored text */
  gtk_text_insert (GTK_TEXT (text), NULL, &text->style->black, NULL, "Supports ", -1);
  gtk_text_insert (GTK_TEXT (text), NULL, &color, NULL, "colored ", -1);
  gtk_text_insert (GTK_TEXT (text), NULL, &text->style->black, NULL, "text and different ", -1);
  gtk_text_insert (GTK_TEXT (text), fixed_font, &text->style->black, NULL, "fonts\n\n", -1);
  infile = fopen("text.c", "r");
  if (infile) {
    char buffer[1024];
    int nchars;
        while (1) {
        nchars = fread(buffer, 1, 1024, infile);
        gtk_text_insert (GTK_TEXT (text), fixed_font, NULL, NULL, buffer, nchars);
        if (nchars < 1024) break;
    }
    fclose (infile);
  }
```

GtkText

```
/* Thaw the text widget, allowing the updates to become visible */
  gtk_text_thaw (GTK_TEXT (text));
/* add to the GtkVBox (box2) a GtkHButtonBox (hbox) */
  hbox = gtk_hbutton_box_new ();
  gtk_box_pack_start (GTK_BOX (box2), hbox, FALSE, FALSE, 0);
  gtk_widget_show (hbox);
/* add to the GtkHButtonBox (hbox) 2 GtkCheckButton (Editable, Wrap Words)  */
  check = gtk_check_button_new_with_label("Editable");
  gtk_box_pack_start (GTK_BOX (hbox), check, FALSE, FALSE, 0);
  gtk_signal_connect (GTK_OBJECT(check), "toggled", GTK_SIGNAL_FUNC(text_toggle_editable), text);
  gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
  gtk_widget_show (check);
  check = gtk_check_button_new_with_label("Wrap Words");
  gtk_box_pack_start (GTK_BOX (hbox), check, FALSE, TRUE, 0);
  gtk_signal_connect (GTK_OBJECT(check), "toggled", GTK_SIGNAL_FUNC(text_toggle_word_wrap), text);
  gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), FALSE);
  gtk_widget_show (check);
```

GtkText

```
/* add to GTkVBox (box1) a GtKSeparator */
  separator = gtk_hseparator_new ();
  gtk_box_pack_start (GTK_BOX (box1), separator, FALSE, TRUE, 0);
  gtk_widget_show (separator);
/* add to GTkVBox (box1)  a GtkVbox container for GtkButton (Close) */
  box2 = gtk_vbox_new (FALSE, 10);
  gtk_container_set_border_width (GTK_CONTAINER (box2), 10);
  gtk_box_pack_start (GTK_BOX (box1), box2, FALSE, TRUE, 0);
  gtk_widget_show (box2);
  button = gtk_button_new_with_label ("close");
  gtk_signal_connect (GTK_OBJECT (button), "clicked", GTK_SIGNAL_FUNC(close_application), NULL);
  gtk_box_pack_start (GTK_BOX (box2), button, TRUE, TRUE, 0);
  GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
  gtk_widget_grab_default (button);
  gtk_widget_show (button);
  gtk_widget_show (window);
  gtk_main ();
  return(0);
}
```

`GtkSpinButton`

Widget pour entrer une valeur entre deux bornes, constitué :

- ▷ d'une entrée de texte éditable (`GtkEntry`) pour afficher une valeur
- ▷ de 2 flèches (`GtkArrow` verticaux) d'incrémentation, décrémentation
- ▷ d'un `GtkAdjustment` associé
  - ◇ l'incrément de pas pour un click à gauche sur une des flèches
  - ◇ l'incrément de page pour un click au milieu sur une des flèches
  - ◇ valeur maximale ou minimale affiché pour un click droit

GtkSpinButton

Création et modification

- `GtkWidget *gtk_spin_button_new( GtkAdjustment *adj, gfloat acceleration, guint decimales )`
- `gtk_spin_button_configure( GtkSpinButton *spin, GtkAdjustment *adj, gfloat acceleration, guint decimales )`

Signification des paramètres:

- `acceleration`: sur l'intervalle $[0.0, 1.0]$, du défilement constant (0.0) à une accélération de plus en plus rapide 1.0)
- `decimales`: nombre de chiffres àprès la virgule

GtkSpinButton

Modificateurs et observateurs

- `gtk_spin_button_set_adjustment(GtkSpinButton *spin, GtkAdjustment *adj)`
- `GtkAdjustment* gtk_spin_button_get_adjustment(GtkSpinButton *spin)`
- `gtk_spin_button_set_digits(GtkSpinButton *spin, guint decimales)`
- `gtk_spin_button_set_value(GtkSpinButton *spin, gfloat value)`
- `gfloat gtk_spin_button_get_value_as_float(GtkSpinButton *spin)`
- `gint gtk_spin_button_get_value_as_int(GtkSpinButton *spin)`

GtkSpinButton

Modification des valeurs

▷ `gtk_spin_button_spin( GtkSpinButton *spin, GtkSpinType direction, gfloat increment )`

La valeur du paramètre pouvant-être mise à jour:

▷ `GTK_SPIN_STEP_FORWARD, GTK_SPIN_STEP_BACKWARD`: valeur de pas d'incrémentation dans `increment`

▷ `GTK_SPIN_PAGE_FORWARD, GTK_SPIN_PAGE_BACKWARD`: valeur d'incrémentation de page dans `increment`

▷ `GTK_SPIN_HOME, GTK_SPIN_END`: adapté à la valeur minimale, maximale de l'ajustement

▷ `GTK_SPIN_USER_DEFINED`: définie par l'utilisateur

GtkSpinButton

Mises à jours

▷ `gtk_spin_button_set_update_policy(`
`GtkSpinButton *spin, GtkSpinButtonUpdatePolicy policy)`

La valeur du paramètre de mise à jour:

▷ `GTK_UPDATE_ALWAYS`: mise à jour quelque soit la valeur rentrée

▷ `GTK_UPDATE_IF_VALID`: si valeur valide pour l'ajustement utilisé.

Ne prendre en compte que les chiffres (`FALSE` par défaut):

▷ `gtk_spin_button_set_numeric(GtkSpinButton *spin,`
`gboolean numeric )`

GtkSpinButton

Pour boucler sur les valeurs maximale ou minimale (`FALSE` par défaut):

▷ `gtk_spin_button_set_wrap(GtkSpinButton *spin, gboolean wrap)`

Arrondir au pas d'incrément le plus proche (`FALSE` par défaut):

▷ `gtk_spin_button_set_snap_to_ticks(GtkSpinButton *spin, gboolean snap)`

Utilisation des différents types d'ombrages:

▷ `gtk_spin_button_set_shwdow_type(GtkSpinButton *spin, GtkShadowType shadow)`

La valeur d'ombrage pouvant-être:

▷ `GTK_SHADOW_IN, GTK_SHADOW_OUT`

▷ `GTK_SHADOW_ETCHED_IN, GTK_SHADOW_ETCHED_OUT`

GtkCombo

Les combo box sont composées:

- ▷ d' une entrée de texte (`GtkEntry`)
- ▷ d'une liste sélectionnable (`GtkEntry`)
- ▷ affichable par une `GtkArrow`

Structure d'un `GtkCombo`

```
struct _GtkCombo {
            GtkHBox hbox;
            GtkWidget *entry;
            GtkWidget *button;
            GtkWidget *popup;
            GtkWidget *popwin;
            GtkWidget *list;
            ...  };
```

GtkCombo

Création de combo box, dont la liste sera vide

▷ `GtkWidget *gtk_combo_new( void )`

Insertion de chaîne dans l'entrée de texte associée:

▷ `gtk_entry_set_text( GTK_COMBO(combo)->entry), ‘‘my string’’);`

Insertion de chaîne dans la liste associée:

▷ `void gtk_combo_set_popdown_strings( GtkCombo *combo,`
`GList *strings )`

Utilisation de la bibliothèque **GLib** pour les listes chaînées

```
GList *glist=NULL;
glist = g_list_append(glist, "string 1");
glist = g_list_append(glist, "string 2");
glist = g_list_append(glist, "string 3");
glist = g_list_append(glist, "string 4");
gtk_combo_set_popdown_strings( GTK_COMBO(combo), glist) ;
```

GtkCombo

Utilisation des touches clavier (haut et bas) pour le défilement de la liste:

▷ `gtk_combo_set_use_arrows( GtkCombo *combo, gboolean flag )`

par défaut la valeur est `TRUE` sinon le focus sera donnée au widget suivant
En début ou fin de liste, par défaut, le focus est donné au widget précédent ou suivant.

▷ `gtk_combo_set_use_arrows_always(( GtkCombo *combo, flag )`

permet de redonner le focus sur le défilement de liste dans la combo box.

▷ `gtk_combo_set_use_arrows( GtkCombo *combo, gboolean flag )`

par défaut la valeur est `TRUE` sinon le focus sera donnée au widget suivant
Recherche dans la liste lors d'une entrée de texte:

▷ `gtk_combo_set_case_sensitive( GtkCombo *combo, gboolean flag )`

`FALSE`, par défaut, recherchera indifféremment majuscules et minuscules

Les Listes: `GtkCList`

Gestion de listes en multi-colonnes:

- ▷ création de listes avec ou sans titres
- ▷ insertion/ destruction de lignes
- ▷ insertion/ récupération de contenu de cellule
- ▷ vérification du type contenu dans une cellule
- ▷ insertion, visualisation de titre de colonnes
- ▷ interaction avec les titres de colonnes
- ▷ visualisation dans une cellule
- ▷ justification dans une colonne
- ▷ positionnement sur une ligne
- ▷ gestion de données associées à une cellule
- ▷ gestion de signal sur une cellule
- ▷ gestion de sélection sur une cellule

Les Listes: `GtkCList`

Création de listes avec ou sans titres

- ▷ `GtkWidget* gtk_clist_new_with_titles(gint columns, gchar* titles[])`
- ▷ `GtkWidget* gtk_clist_new(gint columns)`

Insertion/ destruction de ligne dans une liste

- ▷ `int gtk_clist_prepend(GtkCList *clist, gchar *text[])`
- ▷ `int gtk_clist_append(GtkCList *clist, gchar *text[])`
- ▷ `void gtk_clist_insert(GtkCList* clist, gint row, gchar *text[])`
- ▷ `void gtk_clist_remove(GtkCList* clist, gint row)`

La valeur de retour indique le numéro de ligne inséré

Les Listes: `GtkCList`

Insertion, récupération de contenu de cellule

- `gtk_clist_set_text(GtkCList *clist,gint row, gint col, const gchar* text)`
- `gtk_clist_set_pixmap(GtkCList *clist,gint row, gint col,`
  `GdkPixmap *pixmap, GdKBitmap *mask)`
- `gtk_clist_set_pixtext( GtkCList *clist,gint row, gint col,`
  `gchar *text, guint8 spacing,`
  `GdkPixmap *pixmap, GdKBitmap *mask)`
- `int gtk_clist_get_text(GtkCList *clist,gint row, gint col, gchar **text)`
- `int gtk_clist_get_pixmap(GtkCList *clist,gint row, gint col,`
  `GdkPixmap **pixmap, GdkBitmap **mask)`
- `int gtk_clist_get_pixtext( GtkCList *clist,gint row, gint col,`
  `gchar **text, guint8 *spacing,`
  `GdkPixmap **pixmap, GdkBitmap **mask)`

Les Listes: `GtkCList`

Vérification de type contenu dans une cellule (`GtkCellType`):

- ▷ `GtkCellType gtk_clist_get_cell_type(GtkCList *clist, gint row, gint col)`
  - ◇ `GTK_CELL_EMPTY, GTK_CELL_TEXT, GTK_CELL_PIXMAP`
  - ◇ `GTK_CELL_PIXTEXT, GTK_CELL_WIDGET`

Insertion et visualisation de titres de colonnes

- ▷ `gtk_clist_set_column_title(GtkCList *clist, gint column, gchar *title)`
- ▷ `gtk_clist_column_titles_widget(GtkCList *clist, gint column, GtkWidget *widget)`
- ▷ `gtk_clist_column_titles_show(GtkCList *clist)`
- ▷ `gtk_clist_column_titles_hide(GtkCList *clist)`

Les Listes: `GtkCList`

Interaction avec les titres de colonnes

- ▷ `gtk_clist_column_titles_active(GtkCList *clist)`
- ▷ `gtk_clist_column_titles_passive(GtkCList *clist)`
- ▷ `gtk_clist_column_title_active(GtkCList *clist, gint column)`
- ▷ `gtk_clist_column_title_passive(GtkCList *clist, gint column)`

Visualisation de cellules

- ▷ `gtk_clist_set_column_width(GtkCList *clist, gint column, gint width)`
- ▷ `gtk_clist_set_row_height(GtkCList *clist, gint column, gint height)`
- ▷ `GtkVisibility gtk_clist_row_is_visible(GtkCList* clist, gint row)`
  - ◇ `GTK_VISIBILITY_NONE, GTK_VISIBILITY_PARTIAL, GTK_VISIBILITY_FULL`
- ▷ `gtk_clist_row_set_foreground(GtkCList* clist, gint row, GdkColor *color)`
- ▷ `gtk_clist_row_set_background(GtkCList* clist, gint row, GdkColor *color)`

Les Listes: `GtkCList`

Justification dans une colonne et positionnement dans une liste

- ▷ `gtk_clist_set_column_justification(GtkCList *clist, gint column, GtkJustification justification)`
  - ◇ `GTK_JUSTIFY_LEFT, GTK_JUSTIFY_RIGHT`
  - ◇ `GTK_JUSTIFY_CENTER, GTK_JUSTIFY_FILL`
- ▷ `gtk_clist_moveto(GtkCList *clist, gint column, gint width, gfloat row_align, gfloat col_align)`
  - ◇ `row_align`: défini sur l'intervalle $[0.0, 1.0]$, en haut ou dans le bas de la fenêtre scrollable
  - ◇ `col_align`: idem pour colonnes de gauche à droite.

Les Listes: `GtkCList`

Données associées à une cellule

- ▷ `gtk_clist_set_row_data(GtkCList* clist, gint row, gpointer data)`
- ▷ `gpointer gtk_clist_get_row_data(GtkCList* clist, gint row)`

Gestion de sélection de ligne dans une liste

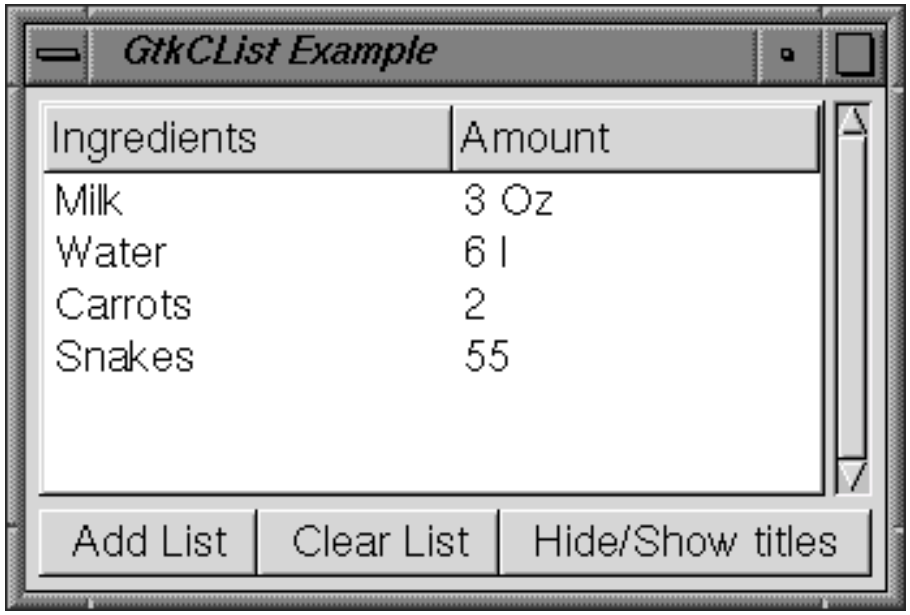
- ▷ `gtk_clist_select_row(GtkCList* clist, gint row, gint column)`
- ▷ `gtk_clist_unselect_row(GtkCList* clist, gint row, gint column)`
- ▷ `gtk_clist_get_selection_info(GtkCList*`
  `clist, gint x, gint, y`
  `gint *row, *gint column)`

Gestion de signal sur une cellule de liste

- ▷ `select_row, unselect_row` : sélection de ligne
- ▷ `click_column`: click dans une colonne

Les Listes: `GtkCList`

Exemple simple de gestion de liste de courses



Les Listes: GtkCList

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

void button_add_clicked( gpointer client_data )
{
    GtkCList* data =  (GtkCList*) client_data;
    int indx;
    gchar *drink[4][2] = { { "Milk",    "3 Oz" },
                           { "Water",   "6 l" },
                           { "Carrots", "2" },
                           { "Snakes",  "55" } };
    for ( indx=0 ; indx < 4 ; indx++ ) gtk_clist_append( data, drink[indx]);
    return;
}

void button_clear_clicked( gpointer client_data )
{
    GtkCList* data =  (GtkCList*) client_data;
    gtk_clist_clear(data);
    return;
}
```

Les Listes

```
void button_hide_show_clicked( gpointer client_data )
{
    GtkCList* data =  (GtkCList*) client_data;
    static short int flag = 0;
    if (flag == 0) {
        gtk_clist_column_titles_hide(data);
        flag++;
    }
    else {
        gtk_clist_column_titles_show(data);
        flag--;
    }
    return;
}

void selection_made( GtkWidget *clist, gint row, gint column,
                     GdkEventButton *event, gpointer client_data )
{
    gchar *text;
    gtk_clist_get_text(GTK_CLIST(clist), row, column, &text);
    g_print("Selected row: %d,\nDesired Column: %d,\nWhat is in the Cell %s\n",
             row, column, text);
    return;
}
```

Les Listes

```
int main( int    argc, gchar *argv[] )
{
    GtkWidget *window;
    GtkWidget *vbox, *hbox;
    GtkWidget *scrolled_window, *clist;
    GtkWidget *button_add, *button_clear, *button_hide_show;
    gchar *titles[2] = { "Ingredients", "Amount" };

    gtk_init(&argc, &argv);
    window=gtk_window_new(GTK_WINDOW_TOPLEVEL);                  /* fenetre principale */
    gtk_widget_set_usize(GTK_WIDGET(window), 300, 150);
    gtk_window_set_title(GTK_WINDOW(window), "GtkCList Example");
    gtk_signal_connect(GTK_OBJECT(window), "destroy",
                       GTK_SIGNAL_FUNC(gtk_main_quit), NULL);
    vbox=gtk_vbox_new(FALSE, 5);
    gtk_container_set_border_width(GTK_CONTAINER(vbox), 5);      /* GtkVBox container */
    gtk_container_add(GTK_CONTAINER(window), vbox);
    gtk_widget_show(vbox);
    scrolled_window = gtk_scrolled_window_new (NULL, NULL);      /* GtkScrolledWindow in GtkVBox */
    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),
                                    GTK_POLICY_AUTOMATIC, GTK_POLICY_ALWAYS);
    gtk_box_pack_start(GTK_BOX(vbox), scrolled_window, TRUE, TRUE, 0);
    gtk_widget_show (scrolled_window);
```

Les Listes

```
clist = gtk_clist_new_with_titles( 2, titles);              /* GtkCList in GtkScrolledWindow */
gtk_signal_connect(GTK_OBJECT(clist), "select_row",
                   GTK_SIGNAL_FUNC(selection_made),
                   NULL);
/* It isn't necessary to shadow the border, but it looks nice :) */
gtk_clist_set_shadow_type (GTK_CLIST(clist), GTK_SHADOW_OUT);
gtk_clist_set_column_width (GTK_CLIST(clist), 0, 150);
gtk_container_add(GTK_CONTAINER(scrolled_window), clist);
gtk_widget_show(clist);
hbox = gtk_hbox_new(FALSE, 0);
gtk_box_pack_start(GTK_BOX(vbox), hbox, FALSE, TRUE, 0);     /* GtkHBox in GtkVBox */
gtk_widget_show(hbox);
button_add = gtk_button_new_with_label("Add List");          /* GtkButton in GtkHBox */
button_clear = gtk_button_new_with_label("Clear List");
button_hide_show = gtk_button_new_with_label("Hide/Show titles");

gtk_box_pack_start(GTK_BOX(hbox), button_add, TRUE, TRUE, 0);
gtk_box_pack_start(GTK_BOX(hbox), button_clear, TRUE, TRUE, 0);
gtk_box_pack_start(GTK_BOX(hbox), button_hide_show, TRUE, TRUE, 0);
```

Les Listes

```
    /* Connect our callbacks to the three buttons */
    gtk_signal_connect_object(GTK_OBJECT(button_add), "clicked",
                              GTK_SIGNAL_FUNC(button_add_clicked),
                              (gpointer) clist);
    gtk_signal_connect_object(GTK_OBJECT(button_clear), "clicked",
                              GTK_SIGNAL_FUNC(button_clear_clicked),
                              (gpointer) clist);
    gtk_signal_connect_object(GTK_OBJECT(button_hide_show), "clicked",
                              GTK_SIGNAL_FUNC(button_hide_show_clicked),
                              (gpointer) clist);

    gtk_widget_show(button_add);
    gtk_widget_show(button_clear);
    gtk_widget_show(button_hide_show);
    gtk_widget_show(window);
    gtk_main();
    return(0);
}
```

Les Arbres: `GtkCTree`

Représentation hiérarchique de données

- ▷ fonctionnalités similaires, par héritage, à une `CList`
- ▷ imbrication de sous-arbres
- ▷ conteneur de `GtkCTreeNode`

Fonctions pour:

- ▷ création, destruction d'arbre, de noeuds
- ▷ gestion des signaux sur les noeuds et sélections
- ▷ indentation des noeuds, représentation d' "extenseur", styles des lignes

Création d'arbre et de sous-arbres

```
GtkWidget *gtk_ctree_new_with_titles( gint columns, gint tree_column,
                                      gchar *titles[] );
GtkWidget *gtk_ctree_new( gint columns, gint tree_column );
```

Les Arbres: `GtkCTree`

Création et destruction de noeuds dans un arbre

```
GtkCTreeNode *gtk_ctree_insert_node( GtkCTree     *ctree,
                                     GtkCTreeNode *parent,
                                     GtkCTreeNode *sibling,
                                     gchar        *text[],
                                     guint8        spacing,
                                     GdkPixmap    *pixmap_closed,
                                     GdkBitmap    *mask_closed,
                                     GdkPixmap    *pixmap_opened,
                                     GdkBitmap    *mask_opened,
                                     gboolean      is_leaf,
                                     gboolean      expanded );
void gtk_ctree_remove_node( GtkCTree  *ctree,
                            GtkCTreeNode *node );
```

Les Arbres: `GtkCTree`

Rôle des arguments de la fonction d'insertion

- ▷ `ctree`: l'arbre en cours de manipulation
- ▷ `parent`: le noeud parent de celui qu'on insère
- ▷ `sibling`: un noeud frère de celui qu'on insère
- ▷ `text[]`: textes de chaque colonne au niveau de ce neoud
- ▷ `spacing`: espacement entre pixmap et texte
- ▷ `pixmap_closed`: pixmap à afficher lorsque le noeud est fermé
- ▷ `mask_closed`: masque de bitmap pour la pixmap précédent
- ▷ `pixmap_opened`: pixmap à afficher lorsque le noeud est ouvert
- ▷ `mask_opened`: masque de bitmap pour la pixmap précédent
- ▷ `is_leaf`: si c'est une feuille ou non
- ▷ `expanded`: si le noeud ouvert au départ ou non

Les Arbres: `GtkCTree`

Fonctionnalités sur un `GtkCtree` pour

- ▷ indentation entre les sous-arbres (20 par défaut)
- ▷ distance entre la ligne verticale matérialisant le niveau du sous-arbre et le nom du noeud (5 par défaut)
- ▷ le styles des lignes matérialisant l'arbre
- ▷ le style des lignes matérialisant l' "extenseur" sur lequel peut agir l'utilisateur
- ▷ association de données quelconque avec les noeuds de l'arbre
- ▷ mises à jour des données lors de la destruction d'un noeud

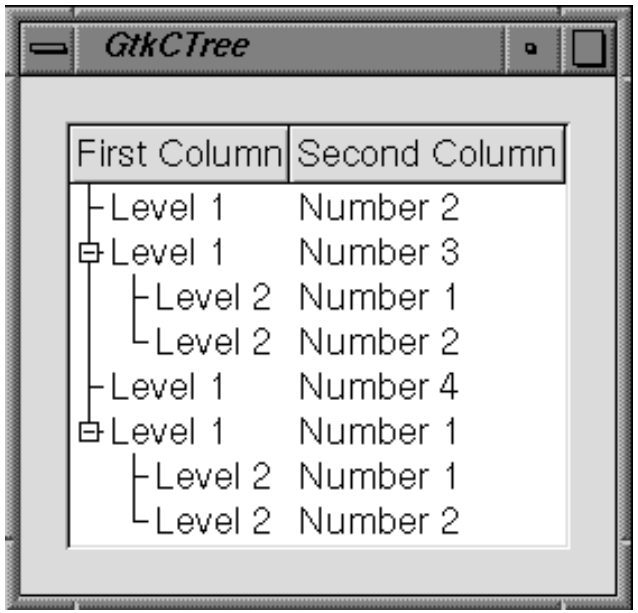
Les Arbres: GtkCTree

```
gtk_ctree_set_indent(  GtkCTree *ctree, gint indent );
gtk_ctree_set_spacing( GtkCTree *ctree, gint spacing );
gtk_ctree_set_line_style( GtkCTree *ctree,
                          GtkCTreeLineStyle  line_style );
gtk_ctree_set_expander_style( GtkCTree ctree,
                              GtkCTreeExpanderStyle  expander_style );
gtk_ctree_node_set_row_data( GtkCTree *ctree,
                             GtkCTreeNode *node, gpointer data );
gtk_ctree_node_set_row_data_full( GtkCTree *ctree,
                                  GtkCTreeNode *node, gpointer data,
                                  GtkDestroyNotify  destroy );
```

Les Arbres: `GtkCTree`

Gestion des signaux:

- ▷ `tree_collapse,tree_expand`: ouverture / fermeture de noeud
- ▷ `tree_select_row,tree_unselect_row`: pour une sélection de ligne
- ▷ `tree_move, change_focus_row_expansion`: ...



Les Arbres: GtkCTree

```
#include <gnome.h>

gint eventDestroy( GtkWidget *widget, GdkEvent *event, gpointer data );
static GtkWidget *makeWidget();
static GtkCTreeNode *makeNode( GtkWidget *widget, GtkCTreeNode *parent,
                               GtkCTreeNode *sibling, gint level, gint count );

int main(int argc,char *argv[])
{
    GtkWidget *app;
    GtkWidget *widget;

    gnome_init( "gtkctree", "1.0", argc, argv );
    app = gnome_app_new( "gtkctree", "GtkCTree" );
    gtk_container_set_border_width( GTK_CONTAINER(app), 20 );
    gtk_signal_connect( GTK_OBJECT(app), "destroy",
                        GTK_SIGNAL_FUNC(eventDestroy), NULL );
    widget = makeWidget();
    gnome_app_set_contents( GNOME_APP(app), widget );
    gtk_widget_show_all( app );
    gtk_main();
    exit(0);
}
```

Les Arbres: `GtkCTree`

```
gint eventDestroy( GtkWidget *widget, GdkEvent *event, gpointer data)
{
    gtk_main_quit();
    return(0);
}
static GtkWidget *makeWidget()
{
    GtkCTreeNode *node1_1;
    GtkCTreeNode *node1_3;
    GtkWidget *widget;
    static gchar *titles[] = {"First Column","Second Column"};
    widget = gtk_ctree_new_with_titles( 2, 0, titles );
    node1_1 = makeNode( widget, NULL, NULL, 1, 1 );
    makeNode( widget, NULL, node1_1, 1, 2 );
    node1_3 = makeNode( widget, NULL, node1_1, 1, 3);
    makeNode( widget, NULL, node1_1, 1, 4 );
    makeNode( widget, node1_1,NULL, 2, 1 );
    makeNode( widget, node1_1,NULL, 2, 2 );
    makeNode( widget, node1_3,NULL, 2, 1 );
    makeNode( widget, node1_3,NULL, 2, 2 );
    return( widget );
}
```

Les Arbres: GtkCTree

```
static GtkCTreeNode *makeNode( GtkWidget *widget, GtkCTreeNode *parent,
       GtkCTreeNode *sibling, gint level,gint count)
{
    GtkCTreeNode *node;
    gchar *text[2];
    gchar line1[20];
    gchar line2[20];
    text[0] = line1;
    text[1] = line2;

    sprintf(line1,"Level %d",level);
    sprintf(line2,"Number %d",count);

    node = gtk_ctree_insert_node(GTK_CTREE(widget),
            parent,
            sibling,
            text,
            0,
            NULL,NULL,NULL,NULL,
            FALSE,TRUE);
    return(node);
}
```

Les Arbres: `GtkTree`

Représentation hiérarchique de données

- ▷ imbrication de sous-arbres
- ▷ conteneur de `GtkTreeItem`

Fonctions pour:

- ▷ Création, destruction d'arbre
- ▷ sélection d'item dans un arbre
- ▷ gestion des signaux sur les sélections
- ▷ ...

Les Arbres: `GtkTree`

Création d'arbre et de sous-arbres

- `GtkWidget *gtk_tree_new(void)`
- `GtkWidget *gtk_tree_item_new_with_label(gchar *label)`
- `gtk_tree_append(GtkTree *tree, GtkWidget *tree_item)`
- `gtk_tree_prepend(GtkTree *tree, GtkWidget *tree_item)`
- `gtk_tree_insert(GtkTree *tree, GtkWidget* tree_item, gint position)`

Destruction dans un arbre

- `gtk_tree_remove_items(GtkTree *tree, GList *items)`
- `gtk_tree_clear_items(GtkTree *tree, gint start, gint end)`
- `gtk_tree_item_set_subtree(GtkTreeItem *tree_item, GtkWidget *subtree)`

Les Arbres: `GtkTree`

sélection dans un arbre

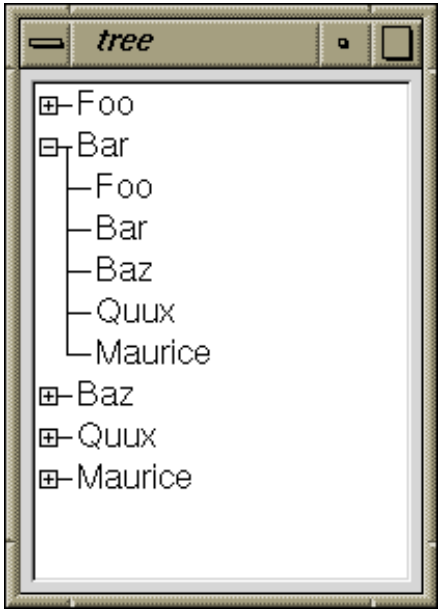
- `gtk_tree_select_item(GtkTree *tree, gint item)`
- `gtk_tree_unselect_item(GtkTree *tree, gint item)`
- `gtk_tree_select_child(GtkTree *tree, GtkWidget *tree_item)`
- `gtk_tree_unselect_child(GtkTree *tree, GtkWidget *tree_item)`
- `int gtk_tree_child_position(GtkTree *tree,, GtkWidget *tree_item)`

gestion de signaux

- `selection_changed( GtkTree *tree )`
- `select_child( GtkTree *tree, GtkWidget *child )`
- `unselect_child (GtkTree *tree, GtkWidget *child )`

Les Arbres: `GtkTree`

Exemple simple de gestion d'arborescence



Les Arbres: GtkTree

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

static void
cb_itemsignal( GtkWidget *item, gchar *signame )
{
    gchar *name;
    GtkLabel *label;
/* It's a Bin, so it has one child, which we know to be a label, so get that */
    label = GTK_LABEL ( GTK_BIN(item)->child );
    gtk_label_get ( label, &name );
/* Get the level of the tree which the item is in */
    g_print ( "%s called for item %s->%p, level %d\n", signame, name, item, GTK_TREE (item->parent)->level );
}

/* Note that this is never called */
static void
cb_unselect_child( GtkWidget *root_tree, GtkWidget *child, GtkWidget *subtree )
{
    g_print ( "unselect_child called for root tree %p, subtree %p, child %p\n", root_tree, subtree, child );
}
```

Les Arbres: `GtkTree`

```
static void
cb_select_child ( GtkWidget *root_tree, GtkWidget *child, GtkWidget *subtree)
{
    g_print ( "select_child called for root tree %p, subtree %p, child %p\n", root_tree, subtree, child );
}

static void
cb_selection_changed( GtkWidget *tree )
{
    GList *i;
    g_print ( "selection_change called for tree %p\n", tree );
    g_print ( "selected objects are:\n" );
    i = GTK_TREE_SELECTION(tree);
    while (i) {
        gchar *name;
        GtkLabel *label;
        GtkWidget *item;
/* Get a GtkWidget pointer from the list node */
        item = GTK_WIDGET ( i->data );
        label = GTK_LABEL ( GTK_BIN (item)->child );
        gtk_label_get ( label, &name );
        g_print ( "\t%s on level %d\n", name, GTK_TREE(item->parent)->level );
        i = i->next;
     }
}
```

Les Arbres: `GtkTree`

```
int main( int argc, char *argv[] )
{
    GtkWidget *window, *scrolled_win, *tree;
    static gchar *itemnames[] = {"Foo", "Bar", "Baz", "Quux", "Maurice"};
    gint i;
    gtk_init (  &argc, &argv );
    window = gtk_window_new ( GTK_WINDOW_TOPLEVEL );
    gtk_signal_connect ( GTK_OBJECT(window), "delete_event", GTK_SIGNAL_FUNC (gtk_main_quit), NULL );
    gtk_container_set_border_width ( GTK_CONTAINER(window), 5 );
    scrolled_win = gtk_scrolled_window_new ( NULL, NULL );
    gtk_scrolled_window_set_policy ( GTK_SCROLLED_WINDOW(scrolled_win),
                                     GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC );
    gtk_widget_set_usize ( scrolled_win, 150, 200 );
    gtk_container_add ( GTK_CONTAINER(window), scrolled_win );
    gtk_widget_show (scrolled_win);
/* Create the root tree, connect all GtkTree:: signals, set selection mode */
    tree = gtk_tree_new();
    g_print ("root tree is %p\n", tree);
    gtk_signal_connect ( GTK_OBJECT(tree), "select_child", GTK_SIGNAL_FUNC(cb_select_child), tree );
    gtk_signal_connect ( GTK_OBJECT(tree), "unselect_child", GTK_SIGNAL_FUNC(cb_unselect_child), tree );
    gtk_signal_connect ( GTK_OBJECT(tree), "selection_changed", GTK_SIGNAL_FUNC(cb_selection_changed),tree);
    gtk_scrolled_window_add_with_viewport ( GTK_SCROLLED_WINDOW(scrolled_win), tree );
    gtk_tree_set_selection_mode ( GTK_TREE(tree), GTK_SELECTION_MULTIPLE );
    gtk_widget_show (tree);
```

Les Arbres: `GtkTree`

```
    for (i = 0; i < 5; i++
    {
        GtkWidget *subtree, *item;
        gint j;

        item = gtk_tree_item_new_with_label (itemnames[i]);
        gtk_signal_connect ( GTK_OBJECT(item), "select", GTK_SIGNAL_FUNC(cb_itemsignal), "select" );
        gtk_signal_connect ( GTK_OBJECT(item), "deselect", GTK_SIGNAL_FUNC(cb_itemsignal), "deselect" );
        gtk_signal_connect ( GTK_OBJECT(item), "toggle", GTK_SIGNAL_FUNC(cb_itemsignal), "toggle ");
        gtk_signal_connect ( GTK_OBJECT(item), "expand", GTK_SIGNAL_FUNC(cb_itemsignal), "expand" );
        gtk_signal_connect ( GTK_OBJECT(item), "collapse", GTK_SIGNAL_FUNC(cb_itemsignal), "collapse" );
        gtk_tree_append ( GTK_TREE(tree), item );
        gtk_widget_show ( item );
        subtree = gtk_tree_new();
        g_print ( "-> item %s->%p, subtree %p\n", itemnames[i], item, subtree );
/* This is still necessary if you want these signals to be called
   for the subtree's children.  Note that selection_change will be
   signalled for the root tree regardless. */
        gtk_signal_connect ( GTK_OBJECT(subtree), "select_child",
                             GTK_SIGNAL_FUNC(cb_select_child), subtree);
        gtk_signal_connect ( GTK_OBJECT(subtree), "unselect_child",
                             GTK_SIGNAL_FUNC(cb_unselect_child), subtree);
/* This has absolutely no effect, because it is completely ignored in subtrees */
        gtk_tree_set_selection_mode (GTK_TREE(subtree),
```

Les Arbres: `GtkTree`

```
        for (j = 0; j < 5; j++)
        {
            GtkWidget *subitem;
            subitem = gtk_tree_item_new_with_label (itemnames[j]);
            gtk_signal_connect ( GTK_OBJECT(subitem), "select",
                                 GTK_SIGNAL_FUNC(cb_itemsignal), "select" );
            gtk_signal_connect ( GTK_OBJECT(subitem), "deselect",
                                 GTK_SIGNAL_FUNC(cb_itemsignal), "deselect" );
            gtk_signal_connect ( GTK_OBJECT(subitem), "toggle",
                                 GTK_SIGNAL_FUNC(cb_itemsignal), "toggle" );
            gtk_signal_connect ( GTK_OBJECT(subitem), "expand",
                                 GTK_SIGNAL_FUNC(cb_itemsignal), "expand" );
            gtk_signal_connect ( GTK_OBJECT(subitem), "collapse",
                                 GTK_SIGNAL_FUNC(cb_itemsignal), "collapse" );
            g_print ("-> -> item %s->%p\n", itemnames[j], subitem );
            /* Add it to its parent tree */
            gtk_tree_append (GTK_TREE(subtree), subitem);
            /* Show it */
            gtk_widget_show (subitem);
        }
    }
    gtk_widget_show (window);
    gtk_main();
    return 0;
}
```

Bibliographie

... à suivre.

Pour plus d'informations:

- ▷ GNOME: `http://www.gnome.org`
- ▷ GTK: `http://www.gtk.org/tutorial`
- ▷ David ODIN: "Programmation Linux avec GTK+"
  Edition Eyrolles 2000, `www.editions-eyrolles.com/livres/odin`
- ▷ Havoc Pennington: "Programmation GTK+/GNOME"
  Campus Press 2000, `www.campuspress.fr`
- ▷ Arthur Griffith: "GNOME/GTK+ Le guide du développeur"
  Osman Eyrolles Multimédia 2000, `www.oemweb.com`