Gtk-Perl

Patrice Le Borgne

24 juin 2002

Table des matières

Chapitre 1

Tour d'horizon

1.1 Introduction

1.1.1 A propos de GTK

GTK (GIMP Toolkit) est une librairie pour créer des interfaces graphiques. Elle est placée sous la licence LPGL, ainsi vous pouvez développer des softwares "ouverts", des softwares gratuits, ou même des softwares commerciaux payants sans avoir à dépenser quoi que ce soit en licences ou en royalties. Elle est appelée la boîte à outils GIMP car elle a été initialement écrite pour le GNU Image Manipulation Programm (GIMP), mais GTK a été utilisé pour un grand nombre de projets de softwares, incluant le projet GNU Network Object Model Environment (GNOME). GTK est basée sur GDK (GIMP Drawing Toolkit) qui est, à la base, un ensemble de fonctions de bas-niveau qui permet d'accéder aux fonctions de gestions de fenêtres (Xlib dans le cas du système X window). Bien qu'entièrement écrit en C, GTK est essentiellement un API orienté objet. GTK est implémenté en utilisant l'idée de classes et de rappels (pointeurs de fonctions). L'un des avantages d'avoir une librairie écrite en C est que la plupart des langages sont capables d'avoir une interface avec les librairies C, rendant GTK disponible pour plusieurs langages, comme C–, Perl, Python, Eiffel et beaucoup d'autres (voir la page principale du site GTK[1] pour avoir une liste plus complète).

1.1.2 Liens entre Perl et GTK

Perl est un langage de script qui rend les choses simples faciles et les choses difficiles possibles. Si vous êtes débutant en Perl, je recommande vivement les livres "Apprendre Perl" [2]et "Programmer en Perl"[3] par O'reilly et Associés[4]. Ce tutoriel se concentre sur les liens entre Perl et GTK c'est pourquoi une connaissance approfondie de Perl n'est pas nécessaire, mais est préférable. Je prendrais rarement le temps d'expliquer des détails sur Perl donc si vous n'êtes pas sûr de vos compétences en Perl, ayez à portée de main votre guide de référence Perl préféré.

Gtk-Perl est un module qui vous permet d'avoir accès aux libraires GTK à partir d'un programme Perl. Il est disponible sous la licence GPL, ce qui signifie qu'il possède les mêmes restrictions et les mêmes libertés que Perl. Gtk-Perl inclut plusieurs modules (GdkImlib, Gnome, GtkXmHTML, GtkGLArea et Glade) qui ne sont pas couverts par ce tutoriel mais qui pourraient bien être étudiés dans des tutoriaux séparés à l'avenir. Si vous souhaitez utiliser Gtk-Perl avec Glade, allez faire un petit tour du côté du site de Dermot Musgrove[5]. Si quelqu'un veut aider à l'élaboration de ce tutoriel, qu'il se fasse connaître. Ma seule exigence est que le style Gtk-Perl soit le même que celui que j'utilise dans ce tutoriel. Ce n'est pas que je pense avoir le Seul et Unique Style, mais parce que je pense que la cohérence est importante pour le débutant. A propos de style, je m'écarte un peu des styles Perl que l'on peut rencontrer ici où là, et j'espère que mes exemples seront lisibles et compréhensibles pour les novices en Perl, les experts et n'importe qui entre les deux.

Ce tutoriel tente de documenter aussi raisonnablement que possible Gtk-Perl, mais gardez à l'esprit, qu'il s'agit d'un tutoriel et non d'un guide de référence. Il serait bénéfique pour le lecteur qu'il ait un expérience préalable avec GTK (de préférence en C) mais ce n'est pas une condition nécessaire. Toutefois, si vous suivez sérieusement le développement de GTK, je vous recommande d'utiliser GTK en C, tout simplement parce que la documentation GTK considère que vous utilisez le C. Si vous décidez de vous lancer, je vous conseille de vous plonger dans l'excellent

[1] voir URL HTTP ://www.gtk.org
[2] voir URL http ://www.oreilly.com/catalog/lperl2/
[3] voir URL http ://www.oreilly.com/catalog/pperl2/
[4] voir URL http ://www.ora.com
[5] voir URL http ://freespace.virgin.net/dermot.musgrove/computers/perl

GTK-Tutoriel[6]le plus tôt possible. La plupart de mes exemples sont directement tirés de ceux du GTK-Tutoriel ce qui facilite la comparaison entre les deux langages. Si GTK est la première interface graphique que vous utilisez, j'apprécierai beaucoup vos commentaires sur l'utilité de ce tutoriel et les parties (s'il y en a) qui vous ont posé problèmes.

Ce document est "un travail en cours". J'aimerais beaucoup savoir si vous avez eu des difficultés à apprendre Gtk-Perl à partir de ce document et j'apprécierais également toute suggestion qui vise à améliorer le document. Regardez, s'il vous plaît, le section Contribuer pour de plus amples informations. Si vous êtes un programmeur expérimenté en Gtk-Perl, faîtes-moi savoir si j'ai commis des erreurs, s'il est des éléments que j'ai délaissés ou mal expliqués. (hey, nous sommes tous humain !).

1.1.3 Pourquoi utiliser Gtk-Perl ?

Bonne question ! Les raisons pour lesquelles un programmeur choisit Gtk-Perl sont probablement les mêmes qui l'on amené à choisir Perl. Les débutants apprécient l'apprentissage progressif (mais parfois long) de Perl. Les programmeurs systèmes aiment la faculté de Perl à "coller" les systèmes et les utilités ensemble. Les ingénieurs softwares apprécient sa capacité à servir de prototype à un programme. Les utilisateurs Unix aiment ses capacités de manipulation texte. Et beaucoup de gens préfèrent tout simplement Perl au langage C et C–. Quelques soient vos raisons d'écrire un programme en Perl, Gtk-Perl vous offre un moyen facile d'y ajouter une interface GUI.

Maintenant, ne pensez pas que Gtk soit la seule boîte à outils GUI disponible en Perl. Il y en a, à ma connaissance deux autres. Il y a une interface Tk (Perl/Tk) documentée par un livre (Apprendre Perl/Tk)[7] et une interface Qt (Perl/Qt) mais qui semble ne pas avoir été mise à jour depuis un bout de temps (peut-être abandonnée ?). Je n'ai essayé ni l'une ni l'autre donc je ne peux commenter leurs performances.

Quand on choisit un toolkit, il faut considérer les différents langages disponibles, s'il s'agit d'une plateforme croisée, sous quelle licence il est placé et sa convivialité. Pour ma part, j'ai l'impression que de toutes les boîtes à outils disponibles en Perl, GTK est le meilleur car c'est une plateforme indépendante (bien que je doute que Gtk-Perl fonctionne un jour sur des machines non-Unix), sous licence GPL et décliné pour une large variété de langages.

1.1.4 Pour bien démarrer

Si votre distribution Linux ne vous fournit pas le package GTK-Perl, vous devrez alors télécharger les sources GTK et les installer. Vous ne pouvez pas utiliser Gtk-Perl sans GTK . Vous pouvez toujours avoir la dernière version à partir du site FTP[8] de GTK. Vous pouvez aussi obtenir d'autres sources d'informations sur le site GTK[9] . GTK utilise GNU autoconf pour la configuration. Une fois que vous avez décompressé les archives tapez :

```
/configure --help
```

pour voir la liste des options.

Une fois que vous avez installé GTK, il vous faut Gtk-Perl. Vous pouvez le télécharger à partir du site FTP[10] de GTK ou à partir du CPAN[11]. Une fois téléchargé, suivez les instructions données dans le fichier INSTALL. Ces deux sites ne sont pas mis à jour régulièrement, et peuvent ne pas posséder la dernière version. A l'avenir, j'essaierai de mettre à disposition la dernière version sur mon site, mais je suis actuellement limité en place par mon serveur web. A mon avis, la première version complètement utilisable de Gtk-Perl est la 0.6123, mais vous devriez plutôt prendre la dernière version disponible (0.7008).

Nous allons bientôt travailler sur notre premier programme en Gtk-Perl. J'ai choisi un style de code qui est (je pense) clair et rigide, en opposition au style laconique et peu structuré de beaucoup de codes. J'espère que ce style sera compris par les débutants sans trop d'effort.

1.2 Goodbye World

Pour commencer notre introduction à Gtk-Perl, nous commencerons par l'un des programmes les plus simples possibles. Ce programme créera une petite fenêtre avec un bouton nommé "Goodbye World ". Cliquer sur le bouton ou fermer la fenêtre sortira du programme.

[6]voir URL http ://www.gtk.org/tutorial/
[7]URL http ://www.oreilly.com/catalog/lperltk/
[8]ftp ://ftp.gtk.org/pub/gtk/
[9]http ://www.gtk.org
[10]URL ftp ://gtk.org/pub/gtk/perl/
[11]ftp ://ftp.cpan.org/pub/perl/CPAN/modules/by-module/Gnome

1.2.1 Les sources de "Goodbye World"

```
#!/usr/bin/perl -w

use Gtk;     # charge le module Gtk-Perl
use strict;  # une bonne idée pour tout script Perl non-trivial

init Gtk;    # initialise Gtk-Perl

# variables convenables pour vrai et faux
my $false = 0;
my $true = 1;

# création d'un widget
my $window = new Gtk : :Window( "toplevel" );
my $button = new Gtk : :Button( "Goodbye World" );

# enregistrement des rappels
$window->signal_connect( "delete_event", \&CloseAppWindow );
$button->signal_connect( "clicked", \&CloseAppWindow );

# montre le bouton
$button->show();

# déclare les attributs de la fenêtre et les montre
$window->border_width( 15 );
$window->add( $button );
$window->show();

# boucle d'évènement Gtk
main Gtk;

# ne devrait jamais arriver là
exit( 0 );



### la fonction de rappel pour fermer la fenêtre
sub CloseAppWindow
  {
    Gtk->exit( 0 );
    return $false;
  }
```

Sous Unix, vous devrez rendre ce programme exécutable en utilisant la commande :

```
chmod a+x goodbyeworld
```

Si vous avez des problèmes pour l'exécuter, vérifier que le chemin de Perl mentionné à la première ligne est correct et que Gtk-Perl est correctement installé. (Essayez d'exécuter le programme de test fourni avec Gtk-Perl, si vous n'êtes pas sûr).

Quand vous lancez le programme vous devriez voir quelque chose de semblable à :

Votre programme peut ne pas ressembler exactement à l'image ci-dessus. GTK est, parmi d'autres choses, "thémable". C'est-à-dire que l'utilisateur décide du style des widgets, des couleurs et icônes à utiliser, les polices, etc... L'exécution ci-dessus utilisait le thème par défaut. Quoi qu'il en soit, la barre de titre dépend du window manager choisi ou de l'OS.

1.2.2 Cheminons à travers "Goodbye World"

Maintenant, voyons si nous ne pouvons pas cheminer à travers le programme et lui donner du sens.

`use Gtk;` doit être inclus dans tout programme Gtk-Perl. Cela permet à Perl d'utiliser les variables, fonctions structures, etc...définies par GTK. Cela peut-être facilement combiné avec d'autres directives `use` comme `use strict ;` .

`init Gtk;` doit également être inclus dans tout programme Gtk-Perl. Cela initialise le module Gtk et met en place quelques éléments comme l'aspect par défaut, la couleur, initialise la librairie à utiliser et les déclencheurs de signaux par défaut.

J'utilise les variables `$true` et `$false` pour montrer la différence entre la valeur booléenne et la valeur numérique. Je ne suis pas obligé de le faire, mais je fais ce choix par soucis de clarté pour le débutant. Sans cela, vous devriez décider si ce "1" est juste là pour représenter la vérité ou s'il a un sens différent. Que se passe-t-il si vous le changez en "2" ? J'utilise `$true` et `$false` dans tous mes programmes pour ne pas avoir à me poser ce genre de question.

`new Gtk::Window();` crée une nouvelle fenêtre. Le window manager décide comment la décorer (avec des choses comme la barre de titre) et où la placer. Tous les widgets GUI sont placés à l'intérieur et vous pouvez avoir plusieurs fenêtres par programme. L'argument indique à X le type de fenêtre dont il s'agit. Les fenêtres "Top-Level" n'ont pas de fenêtre parent et ne peuvent être contenues par aucune fenêtre. Vous pouvez également créer des fenêtres de dialogue avec `new Gtk::Dialog();` .

La fonction `border_width()` prendra un conteneur (comme une fenêtre) et placera de l'espace autour. Cela évitera à votre application d'être trop "tassée". Le nombre passé en argument est la largeur (en pixel) de la bordure créée. Essayez différentes valeurs.

`new Gtk::Button()` crée un nouveau bouton. Ici, nous en avons créé un avec un label texte. Que ce serait-il passé si nous n'avions pas spécifié de texte à mettre sur le bouton ? Essayez et voyez par vous même. Remarquez la taille du bouton et de la fenêtre quand vous le faîtes ? GTK redimensionnera automatiquement les widgets sauf si vous lui dîtes de ne pas le faire (cela inclus la fenêtre). Pour cette raison, il ne sert à rien de s'inquiéter du style ou de la taille de la police que l'utilisateur peut avoir définie par défaut. Donnez juste la disposition à GTK et laissez le faire le reste. Pour plus d'informations sur les boutons voir la section sur les boutons.

`signal_connect()` relie un évènement ou un signal à un appel de routine. Un exemple serait d'appuyer sur le bouton de la souris. En terme de profane, quand un évènement ou un signal est généré, GTK regarde si cet évènement possède une routine enregistrée pour ce widget, si c'est le cas, il exécute cette routine. Si aucune routine n'est enregistrée, il ne se passe rien. Les routines enregistrées sont appelées "rappels". Bien plus d'informations sont disponibles plus loin (incluant une description sur les différences qu'il y a entre elles).

Dans notre exemple, le premier argument de `signal_connect()` est une chaîne indiquant l'évènement que vous voulez enregistrer et le second argument est une référence à la routine à appeler si l'évènement se produit. Certains évènements peuvent nécessiter différents arguments.

La fonction `add()` ajoute un widget dans un conteneur (ici, la fenêtre). Si nous n'avions pas fait cela, nous n'aurions pas vu le bouton parce qu'il n'aurait pas été dans la fenêtre.

Le fonction `show()` rend la fenêtre visible. N'exécutez pas `show()` tant que toutes les propriétés du widget ne sont pas définies. Vous devriez toujours montrer les widgets enfants avant les widgets parents (ici, rendre visible le bouton avant la fenêtre qui le contient). La manière dont la fenêtre apparaît tout d'un coup rend le programme plus professionnel.

`main Gtk;` déclenche le processus de contrôle des évènements GTK. Il s'agit d'une ligne que vous verrez dans toutes les applications Gtk-Perl. Quand le contrôle atteint ce point, GTK se mettra en sommeil, attendant les évènements X (comme les boutons ou les touches pressés), les délais ou les notifications de fichier E/S qui peuvent se produire. Imaginez cela comme une boucle infinie dont on sort par l'appel `Gtk->exit();` . La ligne suivante ne devrait jamais être exécutée mais j'en mets une au cas ou.

La routine placée à la fin du code, est celle qui est appelée quand on clique sur le bouton ou quand on ferme la fenêtre. La seule chose intéressante ici est l'appel `Gtk->exit();` qui sortira du programme en utilisant l'argument comme code de sortie. L'état de retour est très important pour une routine exécutée quand le `delete_event` est appelé. Une valeur fausse signifie "en avant et tue la fenêtre" alors qu'une valeur vraie signifie "ne tue pas la fenêtre". C'est utile pour demander " êtes-vous sûr ?" avant que la fenêtre ne soit balayée.

1.3 Tour d'horizon

1.3.1 Créer un widget

Les différentes étapes pour créer un widget sont :

- `new Gtk::Widget()` pour créer un nouveau widget. Les widgets disponibles sont tous détaillés plus loin dans cette section.
- relier tous les signaux et évènements que vous voulez utiliser aux déclencheurs appropriés.
- Décrire les attributs du widget.
- Regrouper les widgets dans un conteneur en utilisant l'appel approprié comme `$container->add()` ou `$box->pack_`
- `show()` pour montrer le widget.

`show()` permet à Gtk-Perl de savoir que nous avons définis les attributs du widget et qu'ils sont prêts à être utilisés. Vous pouvez aussi utiliser `hide()` pour le faire disparaître. L'ordre dans lequel vous faites apparaître les widgets n'est pas important, mais je vous conseille de montrer la fenêtre en dernier ce qui lui permet d'apparaître d'un coup plutôt que d'apparaître morceaux par morceaux. Les enfants d'un widget (souvenez-vous qu'une fenêtre est également un widget) ne seront disposés que quand leurs parents seront rendus visibles à l'aide de la fonction `show()`.

1.3.2 La hiérarchie des widgets

Pour vos références, voici l'arbre de hiérarchie de classes utilisé pour implémenter les widgets.

```
Gtk : :Object
  +-Gtk : :Widget
  |   +-Gtk : :Misc
  |   |   +-Gtk : :Label
  |   |   |   +-Gtk : :AccelLabel
  |   |   |   +-Gtk : :TipsQuery
  |   |   +-Gtk : :Arrow
  |   |   +-Gtk : :Image
  |   |   +-Gtk : :Pixmap
  |   +-Gtk : :Container
  |   |   +-Gtk : :Bin
  |   |   |   +-Gtk : :Alignment
  |   |   |   +-Gtk : :Frame
  |   |   |   |   +-Gtk : :AspectFrame
  |   |   |   +-Gtk : :Button
  |   |   |   |   +-Gtk : :ToggleButton
  |   |   |   |   |   +-Gtk : :CheckButton
  |   |   |   |   |       +-Gtk : :RadioButton
  |   |   |   |   +-Gtk : :OptionMenu
  |   |   |   +-Gtk : :Item
  |   |   |   |   +-Gtk : :MenuItem
  |   |   |   |   |   +-Gtk : :CheckMenuItem
  |   |   |   |   |   |   +-Gtk : :RadioMenuItem
  |   |   |   |   |   +-Gtk : :TearoffMenuItem
  |   |   |   |   +-Gtk : :ListItem
  |   |   |   |   +-Gtk : :TreeItem
  |   |   |   +-Gtk : :Window
  |   |   |   |   +-Gtk : :ColorSelectionDialog
  |   |   |   |   +-Gtk : :Dialog
  |   |   |   |   |   +-Gtk : :InputDialog
  |   |   |   |   +-Gtk : :DrawWindow
  |   |   |   |   +-Gtk : :FileSelection
  |   |   |   |   +-Gtk : :FontSelectionDialog
  |   |   |   |   +-Gtk : :Plug
  |   |   |   +-Gtk : :EventBox
  |   |   |   +-Gtk : :HandleBox
  |   |   |   +-Gtk : :ScrolledWindow
  |   |   |   +-Gtk : :Viewport
  |   |   +-Gtk : :Box
```

```
|   |   |   +-Gtk : :ButtonBox
|   |   |   |   +-Gtk : :HButtonBox
|   |   |   |   +-Gtk : :VButtonBox
|   |   |   +-Gtk : :VBox
|   |   |   |   +-Gtk : :ColorSelection
|   |   |   |   +-Gtk : :GammaCurve
|   |   |   +-Gtk : :HBox
|   |   |   +-Gtk : :Combo
|   |   |   +-Gtk : :Statusbar
|   |   +-Gtk : :CList
|   |   |   +-Gtk : :CTree
|   |   +-Gtk : :Fixed
|   |   +-Gtk : :Notebook
|   |   |   +-Gtk : :FontSelection
|   |   +-Gtk : :Paned
|   |   |   +-Gtk : :HPaned
|   |   |   +-Gtk : :VPaned
|   |   +-Gtk : :Layout
|   |   +-Gtk : :List
|   |   +-Gtk : :MenuShell
|   |   |   +-Gtk : :MenuBar
|   |   |   +-Gtk : :Menu
|   |   +-Gtk : :Packer
|   |   +-Gtk : :Socket
|   |   +-Gtk : :Table
|   |   +-Gtk : :Toolbar
|   |   +-Gtk : :Tree
|   +-Gtk : :Calendar
|   +-Gtk : :DrawingArea
|   |   +-Gtk : :Curve
|   +-Gtk : :Editable
|   |   +-Gtk : :Entry
|   |   |   +-Gtk : :SpinButton
|   |   +-Gtk : :Text
|   +-Gtk : :Ruler
|   |   +-Gtk : :HRuler
|   |   +-Gtk : :VRuler
|   +-Gtk : :Range
|   |   +-Gtk : :Scale
|   |   |   +-Gtk : :HScale
|   |   |   +-Gtk : :VScale
|   |   +-Gtk : :Scrollbar
|   |       +-Gtk : :HScrollbar
|   |       +-Gtk : :VScrollbar
|   +-Gtk : :Separator
|   |   +-Gtk : :HSeparator
|   |   +-Gtk : :VSeparator
|   +-Gtk : :Preview
|   +-Gtk : :Progress
|   +-Gtk : :ProgressBar
+-Gtk : :Data
|   +-Gtk : :Adjustment
|   +-Gtk : :Tooltips
+-Gtk : :ItemFactory
```

La connaissance de cette hiérarchie est très utile car elle nous indique les fonctions disponibles pour chaque widget. Par exemple, le `spinbutton` hérite de toutes les fonctions disponibles pour la classe `Entry` qui hérite elle-même de la classe `Editable` et ainsi de suite.

1.3.3 Les widgets sans fenêtre

Les widgets suivants n'ont pas de fenêtre associée. Si vous voulez capturer des évènements, vous devrez utilisez la " boîte à évènements".

```
Gtk : :Alignment
Gtk : :Arrow
Gtk : :Bin
Gtk : :Box
Gtk : :Image
Gtk : :Item
Gtk : :Label
Gtk : :Pixmap
Gtk : :ScrolledWindow
Gtk : :Separator
Gtk : :Table
Gtk : :AspectFrame
Gtk : :Frame
Gtk : :VBox
Gtk : :HBox
Gtk : :VSeparator
Gtk : :HSeparator
```

Nous continuerons notre exploration de GTK en examinant chaque widget tout à tour, en créant quelques fonctions simples afin de les présenter. Une autre source d'exemples se trouve également dans les sources de Gtk-Perl.

1.3.4 Créer et détruire des widgets

Pour créer un widget, utilisez :

```
$widget = new Gtk : :Widget() ;
```

Cela crée une instance de la classe widget. Noter que certains widgets peuvent avoir plus d'une fonction `new()` (comme la fonction `new_with_label()` pour les boutons et les arbres). Heureusement, comme Gtk-Perl devient plus mature, l'usage de ces fonctions devient optionnel comme c'est déjà le cas avec `new_with_label()` pour les boutons.

La destruction d'un widget s'obtient par :

```
$widget->destroy() ;
```

Si ce widget possède une fonction de rappel enregistrée pour le signal "destroy", ce retour sera appelé, ainsi on ne peut empêcher l'objet d'être détruit. Pour des informations sur les rappels, voir la section sur 'signaux et rappels".

1.3.5 Montrer et cacher des widgets

Les formes suivants sont plutôt simples et nous en avons déjà rencontrées sous une forme ou une autre.

```
$widget->show() ;
$widget->show_now() ;
$widget->show_all() ;
$widget->hide() ;
$widget->hide_all() ;
```

Noter que si un conteneur qui contient un widget n'est pas montré, alors le widget ne sera visible que lorsque le conteneur le deviendra. Il est recommandé que les widgets soient montrés avant leurs contenants bien que l'ordre exact n'importe pas tant que l'on montre la fenêtre en dernier. Pour les fainéants, la fonction `show_all()` montre tous les widgets et ce qu'ils contiennent. Ce qui en pratique pour les conteneurs.

1.3.6 Réaliser un widget

Ces fonctions vont réaliser un widget ou non :

```
$widget->realize() ;
$widget->unrealize() ;
```

La réalisation est nécessaire quand vous voulez qu'un widget utilise une fenêtre GdkWindow pour accéder à un serveur X. Normalement, la réalisation est faite quand vous appelez la fonction `show()` mais parfois vous devrez appeler la fonction `realize()` avant `show()`. En règle générale, si vous utilisez `realize()`, c'est que vous voulez vous assurer que vous appréhender correctement le problème et que c'est la meilleure manière de s'y prendre.

1.3.7 Les accélérateurs

Les fonctions suivantes seront définies plus tard dans le chapitre "accélérateurs clavier".

```
$widget->add_accelerator( $accelerator_signal,
                          $accelerator_group,
                          $key,
                          $modifiers,
                          $accelerator_flags ) ;

$widget->remove_accelerator( $accelerator_group,
                             $key,
                             $modifiers ) ;

$widget->remove_accelerators( $accelerator_signal,
                              $visible_only ) ;

$widget->accelerator_signal( $accelerator_group,
                             $key,
                             $modifiers ) ;
```

1.3.8 Activer les widgets

La fonction suivante enverra le signal "activé" à un widget :

```
$widget->activate() ;
```

Si un widget possède un rappel enregistré pour le signal "activé", ce rappel sera appelé, bien que cela ne puisse empêcher l'objet d'être détruit. Pour des informations sur les rappels, voir la section sur 'signaux et rappels".

1.3.9 Réafilier un widget

La fonction suivante changera les parents du widget :

```
$widget->reparent($new_parent) ;
```

Je ne pense pas que cette fonction vous soit utile, toujours est-il qu'elle existe si vous en avez besoin !

1.3.10 Choix d'un widget ou widget par défaut

Gtk-Perl permet à un widget d'être celui concerné par les évènements. Par exemple, quand on a une fenêtre de dialogue avec un bouton oui et un bouton non, on peut vouloir faire en sorte que, quand on appuie sur la touche entrée, cela revient à cliquer sur le bouton oui. Pour cela il faut utiliser la fonction suivante :

```
$widget->grab_focus() ;
```

Un widget peut aussi se définir comme widget par défaut à l'aide des fonctions :

```
$widget->can_default( $default ) ;
$widget->grab_default() ;
```

Noter que vous devrez toujours appeler `can_default()` avant `grab_default()`. L'argument de `can_default()` est une valeur vraie ou fausse qui détermine si , oui ou non, le widget peut être le widget par défaut.

1.3.11 La sensibilité des widgets

Un widget insensible est un widget qui ne répond pas aux appels. On peut le considérer comme inactif. Vous pouvez déclencher la sensibilité du widget en utilisant la fonction suivante :

```
$widget->set_sensitive( $sensitive );
```

L'argument ici est une valeur vraie ou fausse qui indique si le widget doit être sensible ou non. Quoi qu'il en soit, même si vous rendez un widget sensible, il sera insensible tant que ses parents le seront. Si vous souhaitez tester la sensibilité, vous pouvez utiliser :

```
$widget->sensitive();
$widget->is_sensitive();
```

Ces fonctions diffèrent en ce que la première teste seulement la valeur de sensibilité du widget alors que la seconde teste la valeur du widget, la valeur de ses parents et ainsi de suite. Seule la seconde peut vous dire si un widget est vraiment sensible ou non.

1.3.12 Position et taille des widgets

La plupart du temps, vous voudrez utiliser des conteneurs comme des boîtes et des tables pour régler la position et la taille des widgets mais il est possible de faire ces réglages à la main.

```
$widget->set_uposition( $x, $y );
$widget->set_usize( $width, $height );
```

La fonction `set_uposition()` règle la position du widget à `$x` de la gauche et à `$y` pixels du haut. La fonction `set_usize()` la largeur du widget à `$width` pixels et la hauteur à `$height` .

Faites attention en utilisant ces fonctions parce qu'il est facile de choisir des valeurs qui pourraient être mauvaise pour certains utilisateurs. Par exemple, un utilisateur avec des problèmes de vue pourraient choisir une grosse police par défaut pour pouvoir lire et votre bouton de 60 pixels peut s'avérer trop petit pour afficher le texte. Vous devez aussi être conscient du type de caractères et de langues qui peuvent être utilisés par vos programmes. En règle générale, laisser les conteneurs comme les boites et les tables choisir les dimensions appropriées est la meilleure solution. Toutefois, certains widgets devraient avoir leurs dimensions réglées spécifiquement. Ces widgets devraient inclure les fenêtres, les arbres, les listes, les zones de dessin et autres...

1.3.13 Style du widget

Les fonctions suivantes seront détaillées dans le chapitre sur le style.

```
$widget->set_style( $style );
$widget->get_style();
$widget->set_default_style( $style );
$widget->get_default_style();
$widget->restore_default_style();
```

1.3.14 Couleur des widgets

Les fonctions suivantes pour la manipulation des couleurs seront vues plus tard dans le chapitre sur les couleurs.

```
$widget->set_colormap( $colormap );
$widget->get_colormap();
$widget->set_default_colormap( $colormap );
$widget->get_default_colormap();
```

1.4 Signaux et Évènements

1.4.1 Théorie des signaux et des rappels

Gtk est une "boîte à outils dirigée par les évènements" ce qui signifie qu'il veille dans `main Gtk` jusqu'à ce qu'un évènement se produise. Si c'est le cas, le contrôle est alors passé à la fonction appropriée.

Ce passage de contrôle se fait en utilisant l'idée de "signal". Ces signaux ne sont pas les mêmes que les signaux d'un système Unix et ne sont pas implémentés en les utilisant bien que la terminologie soit presque identique. Quand quelque chose, comme " la pression sur un bouton de souris" se produit, le signal approprié sera émis par le widget

qui fut pressé. C'est ainsi que Gtk fait la plupart de son travail utile. Il est des signaux dont tous les widgets héritent tel que "destroy", et d'autres qui sont spécifiques tel que le "toggled" signal d'un bouton "toggle".

Pour faire agir un bouton, nous plaçons un gestionnaire de signal qui appellera la fonction appropriée. On le fait avec des fonctions comme celles-ci :

```
$object->signal_connect( "signal_name", \&signal_func );
$object->signal_connect( signal_name => \&signal_func );
$object->signal_connect( "signal_name", \&signal_func,
                         $optional_data ... );
$object->signal_connect( "signal_name", \&signal_func,
                         @optional_data );
```

En Perl, les deux premières formes sont identiques ainsi que les deux suivantes car Perl envoit tous les arguments sous la forme d'une seule liste de scalaires. Bien sûr, vous pouvez envoyer autant d'éléments par liste que vous le désirez.

La variable à gauche du `signal_connect()` est le widget qui émettra le signal. Le premier argument est une chaîne de caractères représentant le signal dont vous aimeriez enregistrer le rappel de signal. Le second argument est la routine que vous voulez appeler. Cette routine est appelée rappel de signal. Si vous passez des arguments à la routine, vous les précisez en les ajoutant à la fin de la liste d'arguments. Tout ce qui est après le second argument est passé en tant que liste, c'est pourquoi vous pouvez vous permettre de passer autant d'arguments que vous le souhaitez.

Si vous ne comprenez pas les références Perl, souvenez-vous que les routines sont précédées par un `\&` et que vous ne pouvez pas placer des parenthèses après le nom de la routine.

Pour les rappels de signaux associés avec uniquement un signal et composés de quelques lignes, il est courant de voir quelque chose comme ceci :

```
$object->signal_connect( "signal_name", sub { do_something; } );
```

Une routine est habituellement définié par :

```
sub callback_func
     {
          my ( $widget , $data ) = @_;
     ...
     }
```

Le premier argument envoyé à un rappel de signal sera toujours le widget émetteur et le reste des arguments sont des données optionnelles envoyées par la fonction `signal_connect()` . Souvenez-vous que Perl ne vous oblige pas à déclarer ou à utiliser les arguments envoyés à une routine.

Noter que la forme ci-dessus pour une fonction de rappel de signal est uniquement un guide général et que certains signaux spécifiques à certains widgets génèrent différents paramètres appelants. Par exemple, le signal Clist `select row` fournit à la fois les paramètres de ligne et de colonne.

1.4.2 Évènements

En plus du mécanisme des signaux décrit précédemment, il y a un ensemble d'évènements qui reflètent les mécanismes des évènements X. Les rappels signaux peuvent également être attaché à ces évènements qui sont :

```
event
button_press_even
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
```

Afin de connecter une fonction de rappel à l'un de ces évènements, on utilise `signal_connect()` comme on la connecterait à un signal en utilisant seulement l'un des noms d'évènements à la place du nom du signal. Le dernier argument passé au rappel de signal est une structure d'évènement ainsi au début de votre fonction, vous devriez dire :

```
my ( $widget, $data, $event ) = @_;
```

ou si vous voulez passer un ensemble de données :

```
my ( $widget, @data ) = @_;
my $event = pop( @data );
```

Les domaines les plus courants dans les structures d'évènements sont les "boutons", "les touches" et "type". Le domaine des boutons contient un nombre avec le bouton pressé (typiquement 1, 2 ou 3). Le domaine " touche" contient la touche pressée (s'il y en a une). Le domaine "type" contient l'une des chaînes suivantes :

```
'nothing'
'delete'
'destroy'
'expose'
'motion_notify'
'button_press'
'2button_press'
'3button_press'
'button_release'
'key_press'
'key_release'
'enter_notify'
'leave_notify'
'focus_change'
'configure'
'map'
'unmap'
'property_notify'
'selection_clear'
'selection_request'
'selection_notify'
'proximity_in'
'proximity_out'
'drag_begin'
'drag_request'
'drop_enter'
'drop_leave'
'drop_data_avail'
'cLIent_event'
'visibiLIty_notify'
'no_expose'
```

Cela permet de déterminer facilement la cause d'un évènement. Par exemple, cela fut-il causé par un bouton pressé ? si oui, lequel ?

```
sub some_event
  {
    my ( $widget, @data ) = @_;
    my $event = pop( @data );

    if ( ( defined( $event->{'type'} ) )
         and ( $event->{'type'} eq 'button_press' ) )
      {
        if ( $event->{'button'} == 3 )
          {
            # right click
          }
        else
          {
            # non-right click
          }
      }
  }
```

Gardez à l'esprit que le code précédent ne fonctionne que si le rappel est lié à l'un des évènements mentionnés ci-dessus. Les signaux n'envoient pas une structure d'évènement ainsi, à moins que vous ne connaissiez exactement le nombre d'arguments envoyés, vous n'avez pas besoin de l'information sur la structure d'un évènement. Je milite pour ne pas connecter un signal et un évènement au même retour.

1.4.3 Plus sur les gestionnaires de signaux

La valeur de retour de la fonction `signal_connect()` est un "tag" qui identifie la fonction de rappel. Vous pouvez avoir autant de rappels par signal ou par objet que vous le souhaitez et ils seront exécutés les uns après les autres dans l'ordre ou ils ont été attaché.

Émettre un signal

Si vous voulez émettre un signal spécifique, vous pouvez le faire en appelant l'une des fonctions suivantes :

```
$widget->signal_emit( $id );
$widget->signal_emit_by_name( $signal_name );
```

L'argument de la première forme est le "id tag" qui est retourné par `signal_connect()` . L'argument de la seconde forme est une chaîne identifiant le nom du signal ? Ainsi beaucoup de widgets ont des fonctions qui émettent des signaux les plus courants. Par exemple, la fonction `destroy()` permettra au signal `'destroy'` d'être émis et la fonction `activate ()` permettra au signal `'activate'` d'être émis.

Supprimer des retours

Ce "id tag" vous permet également de supprimer un retour de la liste en utilisant `signal_disconnect()` comme ceci :

```
$widget->signal_disconnect( $id );
```

Si vous voulez ôter tous les gestionnaires d'un widget, vous pouvez appeler la fonction :

```
$widget->signal_handlers_destroy();
```

Cet appel se passe de commentaires. Il enlève simplement tous les gestionnaires de l'objet passé en tant que premier argument.

Désactiver temporairement des gestionnaires

Vous pouvez débrancher temporairement des gestionnaires avec :

```
$widget->signal_handler_block( $callback_id );
$widget->signal_handler_block_by_func( \&callback, $data );
$widget->signal_handler_block_by_data( $data );
$widget->signal_handler_unblock( $callback_id );
$widget->signal_handler_unblock_by_func( \&callback, $data );
$widget->signal_handler_unblock_by_data( $data );
```

Connecter ou déconnecter des gestionnaires

Voici un aperçu des fonctions utilisées pour connecter ou déconnecter un gestionnaire pour chacun des `signal_connect` disponibles. Vous trouverez plus de détails dans la documentation Gtk.

```
$id = $object->signal_connect( $signal_name,
                               \&function,
                               @optional_data );

$id = $object->signal_connect_after( $signal_name,
                                     \&function,
                                     @optional_data );

$id = $object->signal_connect_object( $signal_name,
                                      \&function,
                                      $slot_object );

$id = $object->signal_connect_object_after( $signal_name,
                                            \&function,
                                            $slot_object );

# Je ne suis pas sûr de celle-là
$id = $object->signal_connect_full( $name,
                                    \&function,
                                    $callback_marshal,
                                    @optional_data,
                                    \&destroy_function,
                                    $object_signal,
```

```
                                        $after ) ;

# Je ne suis pas sûr de celle-là non plus
$id = $object->signal_connect_interp( $name,
                                       \&function,
                                       @optional_data,
                                       \&destroy_function,
                                       $after ) ;

$id = $object->signal_connect_object_while_alive( $signal,
                                                   \&function,
                                                   $alive_object ) ;

$id = $object->signal_connect_while_alive( $signal,
                                           \&function,
                                           @optional_data,
                                           $alive_object ) ;

$object->signal_disconnect( $id ) ;

$object->signal_disconnect_by_func( \& ;function,
                                    @optional_data ) ;
```

Émission et propagation d'un signal

L'émission d'un signal est un processus par lequel Gtk fait fonctionner tous les gestionnaires pour un objet et un signal spécifique. Premièrement, sachez que la valeur de retour de l'émission est la valeur du dernier gestionnaire utilisé. Puisque tous les signaux d'évènements sont du type " last", ce sera le gestionnaire par défaut, à moins que vous ne les connectiez avec les signal `signal_connect_after()` .

La manière dont un évènement est pris en charge est :

- Commencer avec le widget où l'évènement a eu lieu.
- Émettre le signal générique de l'évènement. Si le gestionnaire de signal retourne la valeur "vraie", on arrête tous les processus.
- Autrement, émettre un signal spécifique "boutton_press_event". Si ça retourne la valeur "vraie", on arrête tous les processus.
- Autrement, allez jusqu'aux parents du widget et répéter les deux étapes précédentes.
- Continue jusqu'à ce que les gestionnaires retournent la valeur "vraie" ou jusqu'à ce que le " TopLevel" soit atteint.

Quelques conséquences

La valeur de retour d'un gestionnaire n'aura aucun effet s'il y a un gestionnaire par défaut, à moins que vous ne connectiez avec `signal_connect_after()` . Pour empêcher le gestionnaire par défaut de se déclencher, vous avez besoin de connecter avec `signal_connect()` et d'utiliser `signal_emit_stop_by_name()` . La valeur de retour affecte seulement si le signal est propagé et pas l'émission courante.

Chapitre 2

Les conteneurs de bases

2.1 Les conteneurs

Les conteneurs sont les widgets qui contiennent d'autres widgets. Le premier type est une sous classe de `Bin` qui est elle-même une sous-classe de `Container` . Ces conteneurs peuvent n'avoir qu'un seul enfant et dans ce cas ne sont utilisés que pour leur ajouter des fonctionnalités. Les boutons et les cadres sont de bons exemples de ce type de conteneurs.

Le second type de conteneurs peut posséder plusieurs widgets et sont utilisés pour contrôler leurs dispositions.

2.1.1 Ajouter et enlever des widgets

Les widgets sont ajoutés ou ôter d'un conteneur à l'aide des fonctions suivantes :

```
$container->add( $widget ) ;
$container->remove( $widget ) ;
```

Une fois ajoutée au contenant, le widget ne sera pas visible tant qu'on aura pas appelé la fonction `show()`.

2.1.2 Opérations sur les enfants

La fonction suivante retourne la liste de tous les widgets d'un contenant :

```
@children = $container->children() ;
```

Et les deux fonctions suivantes exécuteront une fonction (avec des données optionnelles) sur chacun des enfants du contenant. Je ne connais pas la différence (s'il y en a) qu'il y a entre elles.

```
$container>forall( \&callback_function, @data ) ;
$container->foreach( \&callback_function, @data ) ;
```

2.1.3 Largeur des bords

La largeur du bord est le nombre de pixels entre les enfants et la limite du conteneur. On peut l'imposer en utilisant l'une des fonctions suivantes :

```
$container->border_width() ;
$container->set_border_width() ;
```

La première est définie dans GTK comme macro pour la seconde, ainsi il n'y a absolument aucune différence entre les deux. Cependant, il est préférable d'utiliser la seconde. Je ne sais si une fonction `get_border_width` existe.

2.2 Les boîtes de regroupement

```
Object
  +--- Widget
       +--- Container
            +--- Box
```

En créant une application, vous voudrez mettre plus d'un widget à l'intérieur d'un fenêtre. Notre premier exemple utilisait seulement un widget ainsi nous avons pu simplement utiliser un appel `$window->add()` pour placer le widget dans la fenêtre. Mais quand on place plus d'un widget dans la fenêtre, comment contrôlez-vous l'endroit où est placé le widget ? C'est là que les boîtes de regroupement interviennent.

2.2.1 Théorie des boîtes de regroupement

Le rangement est fait en créant des boîtes puis en regroupant les widgets à l'intérieur. Ces boîtes sont des conteneurs invisibles dans lesquelles on peut regrouper les widgets. Elles sont de deux formes : une boite horizontale et une boîte verticale. En rangeant les widgets dans une boîte horizontale, les objets sont insérés de la gauche vers la droite ou de la droite vers la gauche en fonction de l'appel utilisé. Pour une boîte verticale, le rangement se fait de bas en haut ou vice-versa. Vous pouvez utiliser des combinaisons en rangeant des boîtes à l'intérieur d'autres boîtes, ou à côté afin de produire l'effet désiré.

Pour créer une nouvelle boîte horizontale ou verticale :

```
$box = new Gtk : :HBox( $homogeneous, $spacing ) ;
$box = new Gtk : :VBox( $homogeneous, $spacing ) ;
```

Si `$homogeneous` est une valeur vraie alors tous les widgets rangés à l'intérieur disposeront du même espace. `$spacing` est la taille en pixels entre chaque "marque" disponible pour les widgets.

Les fonctions suivantes sont utilisées pour placer les objets dans les boîtes :

```
$box->pack_start( $child, $expand, $fill, $padding ) ;
$box->pack_end( $child, $expand, $fill, $padding ) ;
```

La fonction `pack_start()` commencera en haut d'une `VBox` et continuera vers le bas et range de la gauche vers la droite dans une `HBox`. `pack_end()` fera le contraire. Ces fonctions nous permettent juste de justifier à droite ou à gauche la position des widgets et peuvent être mélangées pour obtenir l'effet désiré. L'objet ajouté peut-être ou bien un conteneur ou bien un widget. En fait, de nombreux widgets sont eux-même des conteneurs (les boutons par exemple, mais nous n'utilisons en général qu'un label ou un icône à l'intérieur des boutons).

Comme vous l'avez peut-être deviné, l'argument `$child` est le widget à placer dans la boîte. Si l'argument `$expand` est une valeur vraie alors les widgets seront disposés dans la boîte pour remplir l'espace qui leurs est alloué. Déclarer `$expand` comme valeur fausse vous permettra de faire des justifications à droite et à gauche de vos widgets. Noter que déclarer `$expand` vraie pour une boîte revient à déclarer `$expand` vraie pour chaque widget.

Si l'argument `$fill` est une valeur vraie alors tout espace vide est alloué aux objets eux-même. Autrement l'espace vide est un emballage dans la boîte autour des objets. Cela a de l'effet uniquement si l'argument `$expand` est vrai.

En utilisant ces appels, GTK sait ou vous voulez placer vos widgets et peut ainsi redimensionner ou faire d'autres choses sympathiques. Comme vous l'imaginez, cette méthode nous donne une certaine flexibilité quand on crée et place des widgets.

2.2.2 Détails sur les boîtes

En raison de cette flexibilité, le rangement dans les boîtes peut être déroutant au début. Il y a beaucoup d'options et il n'est pas immédiatement évident de les arranger entre elles. En fait, il y a cinq styles de base différents. Ces cinq styles sont illustrés ci-dessous.

2.2.3 Commentaire à propos de l'exemple

Chaque ligne de l'exemple contient une boîte horizontale avec plusieurs boutons. L'appel pour ranger est un raccourci pour regrouper chaque bouton dans une boîte. Chaque bouton est placé dans la boîte de la même manière (les arguments sont passées à la fonction `pack_start()`). C'est une forme raccourcie de `pack_start()` et de `pack_end()` qui déclare `$expand` vraie, `$fill` vraie et `$padding` 0.

Ces fonctions sont :

```
$box->pack_start_defaults( $widget ) ;
$box->pack_end_defaults( $widget ) ;
```

La valeur `homogeneous` de la boîte peut être allumée ou éteinte en utilisant la fonction :

```
$box->set_homogeneous( $homogeneous ) ;
```

De même pour la valeur `spacing` :

```
$box->set_spacing( $spacing ) ;
```

Si vous voulez déplacer un enfant, utilisez :

```
$box->reorder_child( $child, $position ) ;
```

`$child` est le widget à déplacer et `$position` est la position à changer en partant de 0. Si vous voulez connaître l'ordre actuel, regardez la liste fournie par l'appel à la fonction sur les conteneurs `children()`.

Si vous voulez changer un regroupement d'enfants, vous pouvez utiliser :

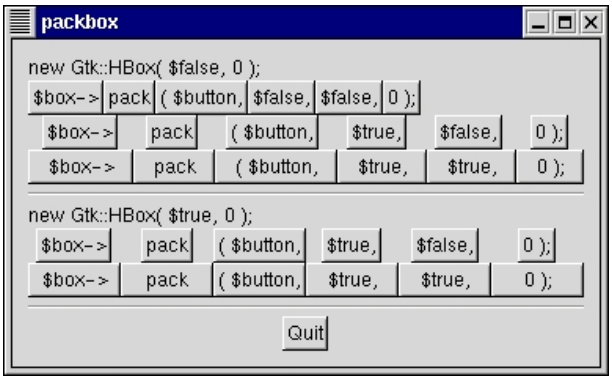
```
$box>set_child_packing( $widget, $expand, $fill, $padding, $pack_type ) ;
```

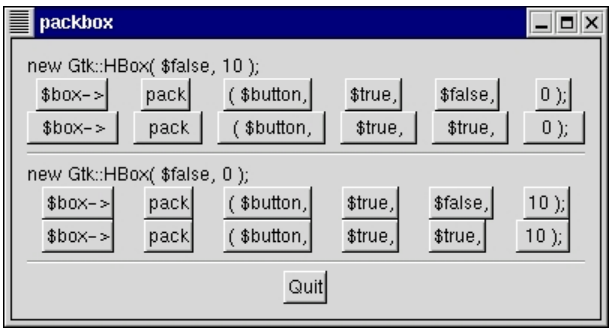
Les arguments sont les mêmes que pour les fonctions `pack_start()` et `pack_end()` à l'exception de `$pack_type` qui est soit `'start'` soit `'end'`.

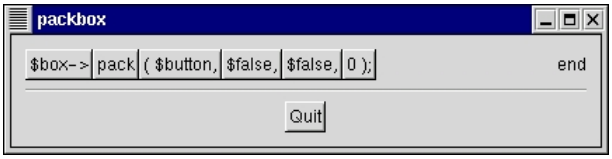
Quelle est la différence entre l'espace (spacing), déclaré quand la boîte est créée, et l'emballage (padding), déclaré quand les éléments sont rangés ? L'espace est ajouté entre les objets et l'emballage est ajouté de chaque côté de l'objet.

2.2.4 Exemple

Voici le code utilisé pour créer la fenêtre illustrée ci-dessous. Je l'ai abondamment commenté ainsi j'espère que vous n'aurez aucun problème à le comprendre, mais soyez conscient qu'il est assez long. Faîtes le tourner par vous-même et jouer avec.








```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;
```

```
unless ( $ARGV[0] )
  {
    print( "usage : packbox num, where num is 1, 2, or 3.\n" ) ;
    Gtk->exit( 1 ) ;
    exit( 1 ) ;
  }

my $false = 0 ;
my $true = 1 ;
my $which = $ARGV[0] ;

my $window ;
my $box1 ;
my $box2 ;
my $label ;
my $separator ;
my $quitbox ;
my $button ;

# Vous devriez toujours vous souvenir de connecter le signal delete_event
# à la fenêtre principale. C'est très important pour obtenir un comportement
# intuitif propre.
$window = new Gtk : :Window( "toplevel" ) ;
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ) ; } ) ;
$window->border_width( 10 ) ;

# Nous créons une boîte verticale ( vbox ) pour regrouper les boîtes
# horizontales. Cela permet d'y placer les boîtes horizontales remplies
# de boutons les unes au-dessus des autres.
$box1 = new Gtk : :VBox( $false, 0 ) ;

# quel exemple montrer. Celui-là correspond à la figure ci-dessus.
if ( $which == 1 )
  {
    # crée un nouveau label.
    $label = new Gtk : :Label( 'new Gtk : :HBox( $false, 0 ) ;' ) ;

    # Aligne le label sur le côté gauche. Nous discuterons de cette
    # fonction et des autres dans la section sur les Attributs
    # d'un widget.
        $label->set_alignment( 0, 0 ) ;

    # Place le label dans la boîte verticale (vbox box1). Souvenez vous
    # que les widgets sont ajoutés dans la vbox les uns au-dessus
    # autres.
      $box1->pack_start( $label, $false, $false, 0 ) ;

    # montre le label
    $label->show() ;

    # Appelle notre fonction fabricante de boîte- homogeneous = FALSE,
    # spacing = 0, expand = FALSE, fill = FALSE, padding = 0
    $box2 = make_box( $false, 0, $false, $false, 0 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    # Appelle notre fonction fabricante de boîte- homogeneous = FALSE,
    # spacing = 0, expand = TRUE, fill = FALSE, padding = 0
```

```
    $box2 = make_box( $false, 0, $true, $false, 0 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    # Les arguments sont : homogeneous, spacing, expand, fill, padding
    $box2 = make_box( $false, 0, $true, $true, 0 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    # Crée un séparateur, nous les verrons plus tard, mais
    # ils sont plutôt simples.
        $separator = new Gtk : :HSeparator() ;

    # Place le séparateur dans la vbox.
    $box1->pack_start( $separator, $false, $true, 5 ) ;
    $separator->show() ;

    # Crée un nouveau label et le montre.
    $label = new Gtk : :Label( 'new Gtk : :HBox( $true, 0 ) ;' ) ;
    $label->set_alignment( 0, 0 ) ;
    $box1->pack_start( $label, $false, $false, 0 ) ;
    $label->show() ;

    # Les arguments sont : homogeneous, spacing, expand, fill, padding
    $box2 = make_box( $true, 0, $true, $false, 0 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    # Les arguments sont : homogeneous, spacing, expand, fill, padding
    $box2 = make_box( $true, 0, $true, $true, 0 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    # Un autre nouveau séparateur.
    $separator = new Gtk : :HSeparator() ;

    # Les 3 derniers arguments de gtk_box_pack_start sont :
    # expand, fill, padding.
    $box1->pack_start( $separator, $false, $true, 5 ) ;
    $separator->show() ;
  }
elsif ( $which == 2 )
  {
    # Crée un nouveau label, souvenez vous que box1 est une vbox
    # créée près du début de main()
    $label = new Gtk : :Label( 'new Gtk : :HBox( $false, 10 ) ;' ) ;
    $label->set_alignment( 0, 0 ) ;
    $box1->pack_start( $label, $false, $false, 0 ) ;
    $label->show() ;

    # Les arguments sont : homogeneous, spacing, expand, fill, padding
    $box2 = make_box( $false, 10, $true, $false, 0 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    # Les arguments sont : homogeneous, spacing, expand, fill, padding
    $box2 = make_box( $false, 10, $true, $true, 0 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
```

```
    $box2->show() ;

    $separator = new Gtk : :HSeparator() ;
    #Les 3 derniers arguments de gtk_box_pack_start sont : expand, fill,
    # and padding.
    $box1->pack_start( $separator, $false, $true, 5 ) ;
    $separator->show() ;

    $label = new Gtk : :Label( 'new Gtk : :HBox( $false, 0 ) ;' ) ;
    $label->set_alignment( 0, 0 ) ;
    $box1->pack_start( $label, $false, $false, 0 ) ;
    $label->show() ;

    # Les arguments sont : homogeneous, spacing, expand, fill, padding
    $box2 = make_box( $false, 0, $true, $false, 10 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    # Les arguments sont : homogeneous, spacing, expand, fill, padding
    $box2 = make_box( $false, 0, $true, $true, 10 ) ;
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    $separator = new Gtk : :HSeparator() ;
    #Les 3 derniers arguments de pack_start sont : expand, fill,
    # and padding.
    $box1->pack_start( $separator, $false, $true, 5) ;
    $separator->show() ;
  }
elsif ( $which == 3 )
  {
    # Cela montre la possibilité d'utiliser pack_end() pour justifier
    # les widgets sur la droite. Premièrement, nous créons une
    # nouvelle boîte comme précédemment.
    $box2 = make_box( $false, 0, $false, $false, 0) ;

    # Crée un nouveau label qui sera placé à la fin.
    $label = new Gtk : :Label( "end" ) ;
    # Le place à l'aide de gtk_box_pack_end(), ainsi il est mis
    # sur la droite de la hbox créée par l'appel make_box().
    $box2->pack_end( $label, $false, $false, 0 ) ;
    # Montre le label.
    $label->show() ;

    # Place la box2 dans la box1 (la vbox, vous vous souvenez? :)
    $box1->pack_start( $box2, $false, $false, 0 ) ;
    $box2->show() ;

    # Un séparateur pour le fond.
    $separator = new Gtk : :HSeparator() ;
    # Cela déclare explicitement le widget de 400 pixels de large
    # sur 5 pixels de haut. C'est ainsi que la hbox que nous avons
    # créée sera également de 400 pixels de large, et le label de
    # ``fin'' sera séparé des autres labels dans la hbox. Autrement
    # les widgets seraient placés dans la hbox le plus près possible
    # les uns des autres.
    $separator->set_usize( 400, 5 ) ;
    # place le séparateur dans la vbox (box1) créée près du
```

```
    # début du programme.
    $box1->pack_start( $separator, $false, $true, 5 ) ;
    $separator->show() ;
  }

# Crée une autre nouvelle hbox. Souvenez vous que l'on peut
# en utiliser autant l'on en veut.
$quitbox = new Gtk : :HBox( $false, 0 ) ;

# Notre bouton quit.
$button = new Gtk : :Button( "Quit" ) ;
# Enregistre le signal qui ferme le programme quand le
# bouton est cliqué.
$button->signal_connect( "clicked", sub { Gtk->exit( 0 ) ; } ) ;
# Place le bouton dans la quitbox.
# Les 3 derniers arguments de gtk_box_pack_start sont :
# expand, fill, padding.
$quitbox->pack_start( $button, $true, $false, 0 ) ;
# place la quitbox dans la vbox (box1)
$box1->pack_start( $quitbox, $false, $false, 0 ) ;

# Place la vbox (box1) qui contient tous nos widgets,
# dans la fenêtre principale.
$window->add( $box1 ) ;

# Montre tout ce qui reste
$button->show() ;
$quitbox->show() ;

$box1->show() ;
# Montrer la fenêtre en dernier pour que tout apparaisse d'un coup.
$window->show() ;

# Et bien sûr notre fonction main.
main Gtk ;
exit( 0 ) ;



### Routines

# Crée une nouvelle boîte hbox avec des boutons labels. Les arguments
# pour les variables qui nous intéressent sont passés à la fonction.
# Nous ne montrons pas la boîte, mais montrons tout ce qui est
# à l'intérieur.
sub make_box
  {
    my ( $homogeneous, $spacing, $expand, $fill, $padding ) = @_ ;

    # Crée une nouvelle hbox avec les valeurs homogeneous
    # et spacing appropriées.
    my $box = new Gtk : :HBox( $homogeneous, $spacing ) ;

    $button = new Gtk : :Button( '$box->' ) ;
    $box->pack_start( $button, $expand, $fill, $padding ) ;
    $button->show() ;

    # Crée une série de boutons avec les valeurs appropriées.
```

```
    $button = new Gtk : :Button( "pack" ) ;
    $box->pack_start( $button, $expand, $fill, $padding ) ;
    $button->show() ;

    $button = new Gtk : :Button( '( $button,' ) ;
    $box->pack_start( $button, $expand, $fill, $padding ) ;
    $button->show() ;

    # Crée un bouton dépendant de la valeur de expand.
    if ( $expand )
      {
        $button = new Gtk : :Button( '$true,' ) ;
      }
    else
      {
        $button = new Gtk : :Button( '$false,' ) ;
      }

    $box->pack_start( $button, $expand, $fill, $padding ) ;
    $button->show() ;
    # C'est la même chose que la création du bouton pour ``expand''.
        if ( $fill )
      {
        $button = new Gtk : :Button( '$true,' ) ;
      }
    else
      {
        $button = new Gtk : :Button( '$false,' ) ;
      }

    $box->pack_start( $button, $expand, $fill, $padding ) ;
    $button->show() ;

    $button = new Gtk : :Button( "$padding ) ;" ) ;
    $box->pack_start( $button, $expand, $fill, $padding ) ;
    $button->show() ;

    return ( $box ) ;
  }
```

2.3 Les boîtes de boutons

```
Object
  +--- Widget
       +--- Container
            +--- Box
                 +--- ButtonBox
```

Les boîtes de boutons sont un moyen pratique pour disposer rapidement d'un groupe de boutons. Elles existent sous forme horizontale ou verticale. Vous créez une nouvelle boîte de boutons avec l'un des appel suivant :

```
$button_box = new Gtk : :HButtonBox() ;
$button_box = new Gtk : :VButtonBox() ;
```

Les seuls attributs ayant un rapport avec les aspects des boîtes de boutons concernent la manière dont disposer les boutons.

```
$button_box->set_spacing_default( $spacing ) ;
$button_box->get_spacing_default() ;
```

Le second attribut auquel vous avez accès agit sur la disposition des boutons à l'intérieur de la boîte.

```
$button_box->set_layout_default( $layout ) ;
```

L'argument `$layout` peut prendre les valeurs suivantes :

```
'default_style'
'spread'
'edge'
'start'
'end'
```

La disposition courante peut être retrouvée par :

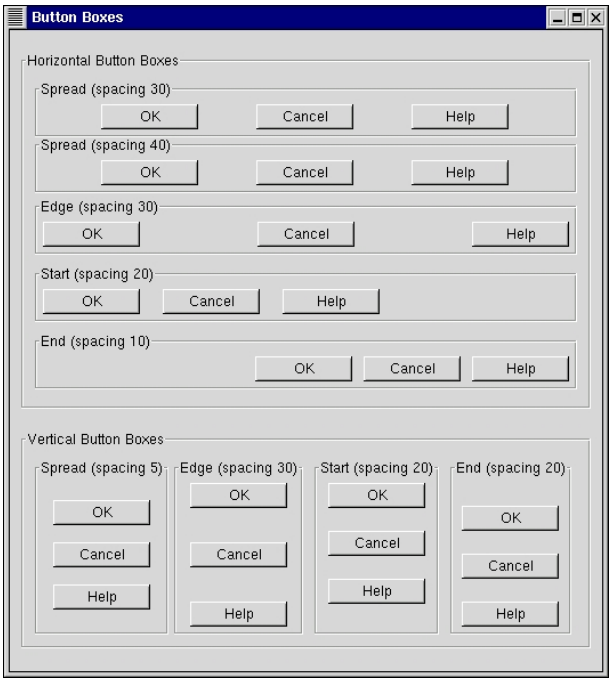
```
$button_box->get_layout_default() ;
```

Les boutons sont ajoutés par la fonction :

```
$button_box->add( $button ) ;
```

Exemple

Voici un exemple qui illustre les différentes valeurs de `$layout` dans une boîte de boutons.




```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $window ;
my $main_vbox ;
my $vbox ;
my $hbox ;
my $frame_horizontal ;
my $frame_vertical ;

$window = new Gtk : :Window( "toplevel" ) ;
```

```
$window->set_title( "Button Boxes" );
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$window->border_width( 10 );

$main_vbox = new Gtk : :VBox( $false, 0 );
$window->add( $main_vbox );

$frame_horizontal = new Gtk : :Frame( "Horizontal Button Boxes" );
$main_vbox->pack_start( $frame_horizontal, $true, $true, 10 );

$vbox = new Gtk : :VBox( $false, 0 );
$vbox->border_width( 10 );
$frame_horizontal->add( $vbox );
$vbox->pack_start( create_bbox( $true, "Spread (spacing 30)",
                                30, 85, 20, 'spread' ),
                                $true, $true, 0 );
$vbox->pack_start( create_bbox( $true, "Spread (spacing 40)",
                                40, 85, 20, 'spread' ),
                                $true, $true, 0 );
$vbox->pack_start( create_bbox( $true, "Edge (spacing 30)",
                                30, 85, 20, 'edge' ),
                                $true, $true, 5 );
$vbox->pack_start( create_bbox( $true, "Start (spacing 20)",
                                20, 85, 20, 'start' ),
                                $true, $true, 5 );
$vbox->pack_start( create_bbox( $true, "End (spacing 10)",
                                10, 85, 20, 'end' ),
                                $true, $true, 5 );

$frame_vertical = new Gtk : :Frame( "Vertical Button Boxes" );
$main_vbox->pack_start( $frame_vertical, $true, $true, 10 );

$hbox = new Gtk : :HBox( $false, 0 );
$hbox->border_width( 10 );
$frame_vertical->add( $hbox );

$hbox->pack_start( create_bbox( $false, "Spread (spacing 5)",
                                5, 85, 20, 'spread' ),
                                $true, $true, 0 );
$hbox->pack_start( create_bbox( $false, "Edge (spacing 30)",
                                30, 85, 20, 'edge' ),
                                $true, $true, 5 );
$hbox->pack_start( create_bbox( $false, "Start (spacing 20)",
                                20, 85, 20, 'start' ),
                                $true, $true, 5 );
$hbox->pack_start( create_bbox( $false, "End (spacing 20)",
                                20, 85, 20, 'end' ),
                                $true, $true, 5 );

$window->show_all();
main Gtk;
exit( 0 );



### Routines

# Crée une boîte de boutons avec des paramètres spécifiques.
```

```
sub create_bbox
  {
    my ( $horizontal, $title, $spacing, $child_w, $child_h, $layout ) = @_;

    my $frame;
    my $bbox;
    my $button;

    $frame = new Gtk : :Frame( $title );

    if ( $horizontal )
      {
        $bbox = new Gtk : :HButtonBox();
      }
    else
      {
        $bbox = new Gtk : :VButtonBox();
      }

    $bbox->border_width( 5 );
    $frame->add( $bbox );

    # Set the appearance of the Button Box
    $bbox->set_layout( $layout );
    $bbox->set_spacing( $spacing );
    $bbox->set_child_size( $child_w, $child_h );

    $button = new Gtk : :Button( "OK" );
    $bbox->add( $button );

    $button = new Gtk : :Button( "Cancel" );
    $bbox->add( $button );

    $button = new Gtk : :Button( "Help" );
    $bbox->add( $button );

    return ( $frame );
  }
```

2.4 Les tables

```
Object
  +--- Widget
       +--- Container
            +--- Table
```

Découvrons une autre manière de ranger : les tables. Elles peuvent être extrêmement utiles dans certaines situations. Utiliser des tables consiste à créer une grille dans laquelle nous plaçons les widgets. Les widgets peuvent prendre autant de cases que nous le souhaitons.

2.4.1 Créer un table

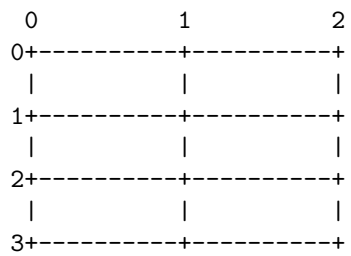
Tout d'abord, la fonction qui permet de créer un table :

```
$table = new Gtk : :Table( $num_rows, $num_columns, $homogeneous );
```

Le premier argument est le nombre de lignes et le second, évidemment le nombre de colonnes. L'argument `$homogeneous` sert à déterminer la manière dont les cases de la table sont dimensionnées. Si `$homogeneous` est une valeur vraie, les

cases sont dimensionnées en fonction de la plus grande ? Si `$homogeneous` est une valeur fausse, la taille de la case est dictée par le plus petit widget de la ligne et le plus large de la colonne.

Les colonnes et les lignes sont disposées de 0 à n ou n est le nombre spécifié à l'appel `new Gtk::Table()`. Ainsi si vous spécifiez lignes (rows)= 3 et colonnes=2, la disposition ressemblera à ceci :



Remarquez que l'origine du repère est en haut à gauche.

2.4.2 Lier les widgets à la table

Pour placer un widget dans une case, on utilise :

```
$table->attach( $child, $left_attach, $right_attach,
                      $top_attach, $bottom_attach, $xoptions,
                      $yoptions, $xpadding, $ypadding ) ;
```

Sur la gauche de l'appel de fonction, la table que vous avez créée et le premier argument est le widget que vous souhaitez placer dans la table.

Les arguments `right_attach` et `left_attach` déterminent où placer le widget et combien de cases utiliser. Si vous voulez un bouton dans la case droite du bas de notre table 2×2 et voulez remplir cette entrée uniquement, alors :

```
$left_attach = 1
$right_attach = 2
$top_attach = 1
$bottom_attach = 2
```

Maintenant si vous voulez qu'un widget occupe toute la ligne supérieure de notre table, vous devrez définir :

```
$left_attach = 0
$right_attach = 2
$top_attach = 0
$bottom_attach = 1
```

`$xoptions` et `$yoptions` servent à spécifier les options de rangements et peuvent être combinées pour permettre multiples options. Ces options sont :

`'fill'` si la case est plus grande que le widget et que `'fill'` est spécifié, le widget s'étendra pour remplir tout l'espace vide.

`'shrink'` : si on a alloué moins d'espace que requis pour le widget (habituellement quand l'utilisateur redimensionne la fenêtre) alors le widget devrait normalement dépasser du fond de la fenêtre et ne plus être visible. Si `'shrink'` est spécifié, les widgets seront réduits avec la table.

`'expand'` obligera la table à occuper tout l'espace restant dans la fenêtre.

Si vous souhaitez spécifier une de ces options, vous l'encadrez entre apostrophe comme ceci : `'option'`. Mais si vous voulez combiner ces options, vous devrez les placer dans une liste anonyme comme ceci :

```
['option1', 'option2']
```

`padding` comme pour les boîtes, crée une zone claire autour du widget en pixel.

Une variation de `attach()` est `attach_defaults()` qui permet de vous dispenser des options x et y et de padding. Les options x et y par défaut sont `['fill', 'expand']` et les `padding` x et y valent 0.

2.4.3 Espacement entre lignes et colonnes

Nous avons aussi les fonctions `set_row_spacing()` et `set_col_spacing()` qui placent des espaces entre les lignes (et les colonnes) , ou entre certaines lignes et certaines colonnes.

```
$table->set_row_spacing( $row, $spacing ) ;
$table->set_col_spacing( $column, $spacing ) ;
```

Notez que pour les colonnes, l'espace est placé à la droite de la colonne et pour les lignes, l'espace est placé dessous.

Vous pouvez aussi déclarer un espace consistant pour toutes les colonnes et les lignes avec :

```
$table-&>set_row_spacings( $spacing ) ;
$table-&>set_col_spacings( $spacing ) ;
```

Noter qu'avec ces appels, la dernière ligne et la dernière colonne ne reçoivent pas d'espace.

2.4.4 Redimensionner une table.

Si vous voulez redimensionner une table après l'avoir créée, vous pouvez utiliser :

```
$table->resize( $rows, $columns ) ;
```

2.4.5 Déclarer la propriété homogeneous

Si vous voulez changer la valeur de la propriété `homogeneous` sur une table existante, vous pouvez utiliser :

```
$table->set_homogeneous( $homogeneous ) ;
```

L'argument `$homogeneous` est une valeur vraie ou fausse.

2.4.6 Exemple

Ici, nous créons une fenêtre avec 3 boutons dans une table 2×2. Les deux premiers sont placés dans la ligne du haut. Un troisième, le bouton "quitter" est placé dans la ligne inférieure, occupant les deux colonnes. Cela doit ressembler à :


```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $window ;
my $button ;
my $table ;

$window = new Gtk : :Window( "toplevel" ) ;
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ) ; } ) ;
$window->title( "Table" ) ;
$window->border_width( 20 ) ;

$table = new Gtk : :Table( 2, 2, $true ) ;
$window->add( $table ) ;
```

```
$button = new Gtk : :Button( "button 1" ) ;
$button->signal_connect( "clicked", \&ButtonClicked, "button 1" ) ;

# Insère le bouton 1 dans la case en haut à gauche de la table
$table->attach_defaults( $button, 0, 1, 0, 1 ) ;
$button->show() ;

# Crée un second bouton
$button = new Gtk : :Button( "button 2" ) ;
$button->signal_connect( "clicked", \&ButtonClicked, "button 2" ) ;

# Insère le bouton2 dans la case en haut à droite de la table
$table->attach_defaults( $button, 1, 2, 0, 1 ) ;
$button->show() ;

# Crée le bouton "Quit"
$button = new Gtk : :Button( "Quit" ) ;
$button->signal_connect( "clicked", sub { Gtk->exit( 0 ) ; } ) ;

# Insère le bouton''quit'' dans les deux cases du bas
$table->attach_defaults( $button, 0, 2, 1, 2 ) ;
$button->show() ;

$table->show() ;
$window->show() ;

main Gtk ;
exit( 0 ) ;



### Rappels
sub ButtonClicked
  {
    my ( $button, $text ) = @_ ;
    print( "Hello again $text was pressed\n" ) ;
  }
```

2.5 Les fenêtres

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Window
```

2.5.1 Donner un nom à la fenêtre

```
$window->set_title( $title ) ;
```

`$title` contient le nom de la fenêtre.

2.5.2 Déclarer le widget par défaut et choix d'un widget

Chaque fenêtre peut avoir un widget qui attire l'attention :

```
$window->set_focus( $widget ) ;
```

Les widgets qui attirent l'attention peuvent être activés en pressant sur la barre d'espace.

Le widget par défaut pour une fenêtre peut être déclaré par :

```
$window->set_default( $widget ) ;
```

Le widget par défaut peut être activé par la touche entrée.

2.5.3 Déclarer l'attitude de la fenêtre

L'attitude de la fenêtre détermine comment elle s'accomode des requêtes de redimensionnement.

```
$window->set_policy( $allow_shrink, $allow_grow, $auto_shrink ) ;
```

L'argument `$allow_shrink` est une valeur vraie ou fausse déterminant si l'utilisateur peut réduire une fenêtre en deçà de la taille requise.

L'argument `allow_grow` est une valeur vraie ou fausse qui détermine si l'utilisateur peut agrandir le fenêtre au delà de la taille requise.

L'argument `$auto_shrink` est une valeur vraie ou fausse spécifiant si la fenêtre revient automatiquement à la taille précédent le changement si c'est une demande plus grande.

```
# la fenêtre est redimensionnable par l'utilisateur
$window->set_policy( $false, $true, $false ) ;
# la taille de la fenêtre est contrôlée par le programme
$window->set_policy( $false, $false, $true ) ;
```

2.5.4 Faire des fenêtres modales

Une fenêtre modale attire l'attention sur elle-même et reste au premier plan de telle sorte que l'utilisateur ne peut utiliser aucune autre fenêtre jusqu'à sa disparition. Les seuls évènements que l'application tolère sont ceux qui concernent la fenêtre modale. Les fenêtres sans mode ne gèlent pas le reste de l'application. Les fenêtres modales servent habituellement de boites de dialogues. Pour déclarer une fenêtre modale :

```
$window->set_modal( $modal ) ;
```

`$modal` est une valeur vraie ou fausse.

2.5.5 Taille et position de la fenêtre

Vous pouvez déclarer la taille par défaut :

```
$window->set_default_size( $width, $height ) ;
```

et la position de la fenêtre :

```
$window->set_position( $position ) ;
```

Les arguments de `$position` sont :

'`none`' aucune influence sur le placement. Le window manager du système place la fenêtre où il veut. C'est la valeur par défaut.

'`center`' la fenêtre sera placée au centre de l'écran

'`mouse`' la fenêtre sera placé à l'endroit où se trouve la souris.

2.6 Les boutons

2.6.1 Les boutons normaux

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Button
```

Créer un bouton

Il y a deux manières de créer des boutons. Vous pouvez créer un bouton vide et ajouter un enfant après, ou vous créez un bouton avec un label pour enfant. La seconde forme est fournie pour le confort d'utilisation. Les fonctions suivantes créent un bouton :

```
$button = new Gtk : :Button() ;
$button = new Gtk : :Button( $label ) ;
$button = new_with_label Gtk : :Button( $label ) ;
```

La première des fonctions ci-dessus crée un bouton vide. Les deux dernières créent un bouton avec un label. La seconde est en fait une forme raccourcie de la troisième.

Si vous créez un bouton avec un label, vous pouvez utiliser `$button->child` pour accéder au widget enfant. Par exemple, pour changer le texte d'un label, vous pourrez utiliser :

```
$button->child->set( "new label" ) ;
```

Si vous ne créez pas un bouton avec label, vous pouvez également prendre un widget et le placer dans le bouton. Par exemple, les trois sections de code suivantes sont équivalentes :

```
# crée un bouton avec label
$button = new Gtk : :Button( "text" ) ;
# même que précédemment, mais avec une fonction new() différente
$button = new_with_label Gtk : :Button( "text" ) ;
# crée un label séparé du bouton, et l'ajoute à la main
$button = new Gtk : :Button() ;
$label = new Gtk : :Label( "text" ) ;
$button->add( $label ) ;
```

Les signaux des boutons

Les boutons possèdent les signaux suivants :

`pressed` : émis quand on presse le bouton de la souris alors que le pointeur est à l'intérieur du bouton ou quand la fonction `$button->pressed()` est appelée.

`released` : émias quand on relâche le bouton de la souris alors que le pointeur est à l'intérieur du bouton ou quand la fonction `$button->released()` est appelée.

`clicked` : émis quand le bouton de la souris est pressé puis relâché à l'intérieur du bouton ou quand la fonction `$button->clicked()` est appelée.

`enter` : émis quand le pointeur de la souris entre sur le bouton ou quand la fonction `$button->enter()` est appelée.

`leave` : émis quand le pointeur de la souris sort du bouton ou quand la fonction `$button->leave()` est appelée.

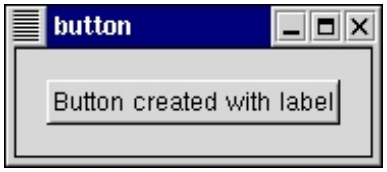
Styles de relief

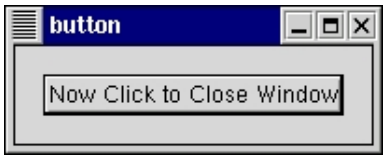
Les styles de relief des boutons peuvent être : `'normal'` , `'half'` , `'none'` . Il est évident que le style par défaut est : `'normal'` . Pour obtenir ou déclarer le style de relief, vous devrez utiliser les fonctions suivantes ;

```
$button->set_relief( $relief_style ) ;
$button->get_relief() ;
```

Exemple

Dans le programme suivant, je crée d'abord un bouton avec label auquel j'attache un gestionnaire, et plus tard, j'enlève ce bouton, le remplace par un créé sans label puis lui ajoute un label. Vous devriez être capable de comprendre à la fois ce qui concerne les boutons et ce qui concerne les signaux pour aller plus loin.






```
#!/usr/bin/perl -w


use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window = new Gtk : :Window( "toplevel" );
my $button = new Gtk : :Button( "Button created with label" );
my $label;
my $id;
my $numclicked = 0;

# enregistrement des rappels
$window->signal_connect( "delete_event", \&CloseWindowEvent );
$button->signal_connect( "clicked", \&ClickedButtonEvent );

# attribut du bouton
$button->show();

# attributs de la fenêtre
$window->border_width( 15 );
$window->add( $button );
$window->show();

# boucle d'évènement Gtk
main Gtk;

exit( 0 );



### Routines

# fonction appelée quand le bouton est clické
```

```
sub ClickedButtonEvent
  {
    if ( $numclicked == 0 )
      {
        $button->child->set( "Changed Button Label" ) ;
        $numclicked++ ;
      }
    elsif ( $numclicked == 1 )
      {
        $window->remove( $button ) ;

        $button = new Gtk : :Button() ;
        $label = new Gtk : :Label( "Label added to a button" ) ;
        $button->add( $label ) ;
        $label->show() ;
        $button->show() ;
        $window->add( $button ) ;
        $id = $button->signal_connect( "clicked", \&ClickedButtonEvent ) ;
        $numclicked++ ;
      }
    elsif ( $numclicked == 2 )
      {
        $label->set( "Now Click to Close Window" ) ;
        $numclicked++ ;
      }
    else
      {
        Gtk->exit( 0 ) ;
      }
  }

# fonction appelée quand on demande à la fenêtre de fermer
sub CloseWindowEvent
  {
    # Si vous retourner une valeur fausse au gestionnaire de signal ‘‘delete_event’’
    # GTK émettre le signal ‘‘delete_event’’.
    # Retourner une valeur vraie signifie que vous ne voulez pas que la
    # fenêtre soit détruite.
    return $true ;
  }
```

2.6.2 Les boutons Toggle ou interrupteurs

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Button
                      +--- ToggleButton
```

Les boutons toggle sont dérivés des boutons normaux et sont très similaires sauf qu'ils n'ont que deux états possibles et qui change à l'aide d'un clic. Ils peuvent être remontés et quand vous cliquez, ils redescendent. Vous recliquez et ils remontent.

Les boutons toggle servent de base aux cases à cocher et aux boutons radios, au point que ces derniers héritent de beaucoup des appels utilisés pour les boutons toggle. Nous vous indiquerons lesquels quand nous les rencontreront.

Créer un bouton toggle

```
new Gtk : :ToggleButton() ;
new Gtk : :ToggleButton( $label ) ;
```

Comme vous pouvez l'imaginer, cela fonctionne exactement comme pour les boutons normaux. La première crée un bouton toggle vierge et la seconde un bouton toggle avec label.

Pour retrouver l'état d'un widget toggle, incluant les boutons radios et les cases à cocher, nous utilisons une construction illustrée dans l'exemple ci-dessous. Cela teste l'état du toggle en accédant au domaine `actif` de la structure du widget toggle. Le signal qui nous intéresse émis par un bouton toggle (les boutons toggle, radio et les cases à cocher) est le signal `toggled`. Pour contrôler l'état de ces boutons, déclarez un gestionnaire de signal pour attraper les signal `toggled` et accéder à la structure afin de déterminer son état. Le rappel ressemblera à

```
sub toggle_button_callback
{
  $togglebutton = $_[0] ;
  if ( $togglebutton->active )
    {
      # Si le contrôle arrive là, le bouton est baissé
    }
  else
    {
      # Si le contrôle arrive là, le bouton est levé
    }
  }
```

Changer l'état du bouton

Pour forcer l'état d'un bouton toggle, et ses enfants, les boutons radios et les cases à cocher, utilisez :

```
$togglebutton->set_active( $state ) ;
```

Passer une valeur vraie ou fausse comme argument précise s'il doit être en bas (pressé) ou en haut (relâché). Le défaut est en haut ou faux.

Notez que si vous utilisez la fonction `set_active()` et que l'état est en fait changé, cela déclenchera l'émission du signal "cliqué" par le bouton (et non "toggled" comme beaucoup de personnes le pensent).

Pour obtenir l'état d'un bouton, vous pouvez utiliser :

```
$togglebutton->get_active() ;
```

Plutôt que de changer la valeur d'un bouton toggle en contrôlant son état actif et en déclarant la valeur opposée, utiliser tout simplement la fonction :

```
$togglebutton->toggled() ;
```

Cette fonction émettra aussi le signal "toggled".

2.6.3 Les cases à cocher

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Button
                      +--- ToggleButton
                           +--- CheckButton
```

Les cases à cocher héritent de beaucoup de propriétés et de fonctions des boutons toggle, mais ils ont une allure différente. Plutôt que d'être des boutons avec du texte à l'intérieur, ce sont des petites cases avec le texte à leur droite. Elles sont souvent utilisées pour choisir des options oui ou non dans une application.

Les deux fonctions de création sont les mêmes que pour un bouton normal.

```
new Gtk : :CheckButton() ;
new Gtk : :CheckButton( $label ) ;
```

Passer une chaîne en argument crée une case à cocher avec un label à son côté. Le contrôle de l'état de la case se fait de la même manière que pour le bouton toggle.

2.6.4 Les boutons radio

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Button
                      +--- ToggleButton
                           +--- CheckButton
                                +--- RadioButton
```

Créer des boutons radio

Les boutons radio sont similaires aux cases à cocher sauf qu'ils sont regroupés de sorte qu'un seul puisse être sélectionné. Ils sont très utiles dans les programmes ou l'on doit choisir dans une liste d'options.

On crée un nouveau bouton radio avec l'une des fonctions suivantes :

```
new Gtk : :RadioButton( $label ) ;
new Gtk : :RadioButton( $label, $button ) ;
```

Vous noterez la présence d'un second argument dans la seconde fonction. Les boutons radio doivent appartenir à un groupe pour accomplir leur devoir proprement. Le premier appel `new Gtk::RadioButton()` devrait avoir le label comme unique argument. Alors, vous ajoutez les autres boutons radio au groupe en utilisant l'un des boutons déjà présent dans le groupe comme second argument. Le bout de code suivant devrait vous éclaircir les idées :

```
# crée le premier bouton et l'ajoute dans une boîte.
$radio1 = new Gtk : :RadioButton( "button 1" ) ;
$box->add( $radio1 ) ;

# crée le second bouton et l'ajoute dans une boîte.
$radio2 = new Gtk : :RadioButton( "button 2", $radio1 ) ;
$box->add( $radio2 ) ;

# crée le troisième bouton et l'ajoute dans une boîte.
# notez que le premier bouton créé sert de second argument
$radio3 = new Gtk : :RadioButton( "button 3", $radio1 ) ;
$box->add( $radio3 ) ;

# crée le premier bouton et l'ajoute dans une boîte.
# notez que le troisième bouton créé sert de second argument
$radio4 = new Gtk : :RadioButton( "button 4", $radio3 ) ;
$box->add( $radio4 ) ;
```

Déclarer quel doit être le bouton pressé est souvent une bonne idée :

```
$togglebutton->set_active( $state ) ;
```

Cela fonctionne comme décrit dans la section sur les boutons toggle. Une fois que les boutons radios sont regroupés seul un peut être actif. Si l'utilisateur clique sur un bouton puis sur l'autre, le premier émettra en premier le signal "toggled" signifiant qu'il est devenu inactif puis le second émettra le signal "toggled" pour dire qu'il devient actif.

Les groupes de boutons

Si vous voulez obtenir une liste de tous les boutons appartenant à un groupe, vous pouvez utiliser la fonction :

```
@group = $radiobutton->button_group() ;
```

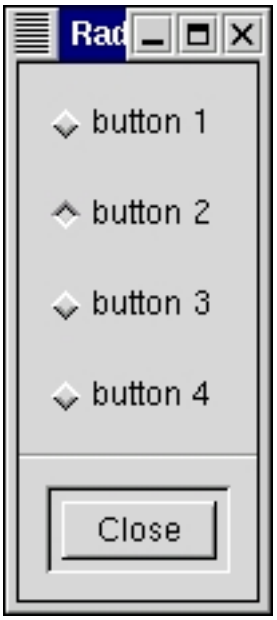
Si vous voulez ajouter un bouton radio à un groupe après qu'il ait été créé, utilisez :

```
$radiobutton->button_set_group( @group ) ;
```

L'argument `@group` est une liste des boutons radio, qui appartiennent au même groupe. On l'obtient habituellement de la fonction `button_group()`.

Exemple

L'exemple suivant crée un groupe de quatre boutons radio.




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window;
my $box1;
my $box2;
my $button;
my $separator;

$window = new Gtk::Window( "toplevel" );
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ); } );
$window->set_title( "Radio Buttons" );
$window->border_width( 0 );

$box1 = new Gtk::VBox( $false, 0 );
$box1->show();

$box2 = new Gtk::VBox( $false, 10 );
$box2->border_width( 10 );
$box1->pack_start( $box2, $false, $false, 0 );
$box2->show();
$window->add( $box1 );

$button = new Gtk::RadioButton( "button 1" );
$box2->pack_start( $button, $false, $false, 0 );
$button->show();

$button = new Gtk::RadioButton( "button 2", $button );
```

```
$button->set_active( $true ) ;
$box2->pack_start( $button, $true, $true, 0 ) ;
$button->show() ;

$button = new Gtk : :RadioButton( "button 3", $button ) ;
$box2->pack_start( $button, $true, $true, 0 ) ;
$button->show() ;

$button = new Gtk : :RadioButton( "button 4", $button ) ;
$box2->pack_start( $button, $true, $true, 0 ) ;
$button->show() ;

$separator = new Gtk : :HSeparator() ;
$box1->pack_start( $separator, $false, $false, 0 ) ;
$separator->show() ;

$box2 = new Gtk : :VBox( $false, 10 ) ;
$box2->border_width( 10 ) ;
$box1->pack_start( $box2, $false, $true, 0 ) ;
$box2->show() ;

$button = new Gtk : :Button( "Close" ) ;
$button->signal_connect( "clicked", sub { Gtk->exit( 0 ) ; } ) ;

$box2->pack_start( $button, $true, $true, 0 ) ;
$button->can_default( $true ) ;
$button->grab_default() ;
$button->show() ;
$window->show() ;

main Gtk ;
exit( 0 ) ;
```

Chapitre 3

Les widgets

3.1 Les labels

```
Object
  +--- Widget
       +--- Misc
            +--- Label
```

Les labels sont très utilisés avec GTK et sont relativement simples. Les labels n'émettent aucun signal puisqu'ils ne sont pas associés à une fenêtre X. Si vous avez besoin d'attraper des signaux, placez les dans une boîte à évènement ou dans un bouton.

Pour créer un nouveau label , utilisez :

```
new Gtk : :Label( $string ) ;
```

Le seul argument est la chaîne que vous voulez afficher.

Pour changer le texte du label après sa création, utilisez la fonction :

```
$label->set_text( $string ) ;
```

Encore une fois, l'argument est la nouvelle chaîne.

L'espace utilisé pour la nouvelle chaîne sera ajusté automatiquement si nécessaire. Vous pouvez produire des labels multi-lignes en plaçant des caractères de fin de ligne dans la chaîne.

Pour retrouver la chaîne courante, utilisez ;

```
$label->get() ;
```

Le texte du label peut être justifié avec :

```
$label->set_justify( $jtype ) ;
```

Les valeurs de `$jtype` sont :

```
'left'
'right'
'center' (default)
'fill'
```

Le widget label est également capable de faire automatiquement les césures du texte en fin de ligne. Cela peut être activé par :

```
$label->set_line_wrap( $wrap ) ;
```

L'argument `$wrap` est une valeur vraie ou fausse.

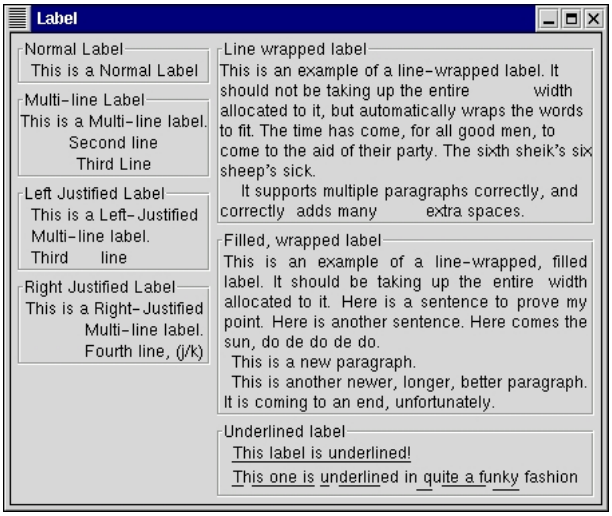
Si vous voulez un label souligné, vous pouvez déclarer un patron pour le label :

```
$label->set_pattern( $pattern ) ;
```

L'argument `pattern` indique comment souligner le texte. Cela consiste en une chaîne de caractère _ et d'espace. Le caractère _ indique les caractères du label à souligner. Par exemple , la chaîne `''_ _ _ _ _ _''` soulignera le premier, le troisième, le cinquième, le septième et le neuvième caractères.

Exemple

Voici un court exemple qui illustre ces fonctions. Il utilise le widget cadre pour mieux mettre en valeur les styles de labels. Vous pouvez l'ignorer pour le moment car le widget cadre sera expliqué plus tard.




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window;
my $hbox;
my $vbox;
my $frame;
my $label;

$window = new Gtk::Window( "toplevel" );
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$window->set_title( "Label" );

$vbox = new Gtk::VBox( $false, 5 );
$hbox = new Gtk::HBox( $false, 5 );

$window->add( $hbox );
$hbox->pack_start( $vbox, $false, $false, 0 );
$window->border_width( 5 );

$label = new Gtk::Label( "Ceci est un label normal" );
$frame = new Gtk::Frame( "Normal Label" );
$frame->add( $label );
$vbox->pack_start( $frame, $false, $false, 0 );

$label = new Gtk::Label( "Ceci est un label multi-lignes.\\n Seconde ligne\n"
                         . "Troisième Line" );
$frame = new Gtk::Frame( " Label Multi-lignes " );
$frame->add( $label );
$vbox->pack_start( $frame, $false, $false, 0 );
```

```
$label = new Gtk : :Label( "Ceci est un label justifié à gauche\n"
                         . "Multi-lignes\n Troisième ligne") ;
$label->set_justify( 'left' ) ;
$frame = new Gtk : :Frame( "Label justifié à gauche" ) ;
$frame->add( $label ) ;
$vbox->pack_start( $frame, $false, $false, 0 ) ;

$label = new Gtk : :Label( "Ceci est un label justifié à droite  \n
                        Multi-lignes. \n"
                         . "quatrième ligne, (j/k)" ) ;
$label->set_justify( 'right' ) ;
$frame = new Gtk : :Frame( "Label justifié à droite" ) ;
$frame->add( $label ) ;
$vbox->pack_start( $frame, $false, $false, 0 ) ;

$vbox = new Gtk : :VBox( $false, 5 ) ;
$hbox->pack_start( $vbox, $false, $false, 0 ) ;
$frame = new Gtk : :Frame( "Line wrapped label" ) ;
$label = new Gtk : :Label( "Ceci est un exemple de césure de ligne. "
                         . "Cela ne devrait pas prendre la largeur totale     "
                         . "allouée mais fait passer les mots à la ligne "
                         . "automatiquement  "
                         . "    Cela supporte correctement plusieurs , "
                         . "paragraphes et ajoute également "
                         . "plusieurs         espaces " ) ;
$label->set_line_wrap( $true ) ;
$frame->add( $label ) ;
$vbox->pack_start( $frame, $false, $false, 0 ) ;

$frame = new Gtk : :Frame( "Filled, wrapped label" ) ;
$label = new Gtk : :Label( " Ceci est un exemple de césure de ligne, option fill"
                         . "toute la              largeur disponible"
                         . "Voici une phrase qui le prouve"
                         . "Et en voici une autre."
                         . "Here comes the sun, do de do de do.\n"
                         . "    Ceci est un nouveau paragraphe\n"
                         . "    Un autre nouveau , plus long,meilleur "
                         . "On arrive à la fin"
                         . "malheureusement." ) ;
$label->set_justify( 'fill' ) ;
$label->set_line_wrap( $true ) ;
$frame->add( $label ) ;
$vbox->pack_start( $frame, $false, $false, 0 ) ;

$frame = new Gtk : :Frame( "Label souligné" ) ;
$label = new Gtk : :Label( "Voici un label souligné !\n"
                         . "Celui-là est souligné à la mode funky ") ;
$label->set_justify( 'left' ) ;
$label->set_pattern( "_____ _ ________ "
                   . "_ ______      __ _______ ___") ;
$frame->add( $label ) ;
$vbox->pack_start( $frame, $false, $false, 0 ) ;

$window->show_all() ;

main Gtk ;
exit( 0 ) ;
```

3.2 Les widgets de réglage et d'échelle

```
Object
  +--- Data
       +--- Adjustment
```

3.2.1 Les réglages

GTK possède plusieurs widgets qui peuvent être ajustés visuellement par l'utilisateur à l'aide de la souris ou du clavier, comme le widget range. Il y a aussi quelques widgets qui mettent à disposition des portions ajustables pour une plus grande surface de données comme les widgets texte et viewport.

Évidemment, une application a besoin d'être capable de réagir aux changements que l'utilisateur fait aux widget range. Une manière de le faire serait que chaque widget émette son propre type de signal quand son réglage change et même passe la nouvelle valeur au gestionnaire de signal, ou lui demande de regarder à l'intérieur des données du widget afin de s'informer de la valeur. Vous pouvez aussi vouloir connecter les réglages de plusieurs widgets ensembles afin que les réglages de l'un règle les autres. L'exemple le plus simple de ce phénomène est de connecter une barre de défilement à une zone de texte ou à un "panning viewport". Si chaque widget possède son propre moyen de déclarer ou de récupérer la valeur du réglage alors le programmeur doit écrire leurs propres gestionnaires de signal pour faire la traduction entre la sortie du signal d'un widget et l'entrée d'une autre fonction déclarant les réglages.

GTK résout ce problème en utilisant l'objet Adjustment qui n'est pas vraiment un widget mais un moyen pour les widgets de stocker et de passer des informations de réglages sous une forme concise et flexible. L'usage le plus évident de l'Adjustment est de stocker les paramètres de configuration et les valeurs des widgets "échelles" comme les barres de défilement et les contrôles d'échelle. Quoi qu'il en soit, puisque les Adjustments sont dérivés de la classe Objet, ils possèdent des pouvoirs spéciaux supérieurs à ceux des structures de données normales. Plus important, ils peuvent émettre des signaux tout comme les widgets, et ces signaux peuvent être utilisés non seulement pour permettre à vos programmes de réagir aux entrées utilisateurs sur certains widgets réglables mais également pour propager les valeurs des réglages de manière transparente entre les widgets réglables. Malheureusement, les réglages peuvent être difficiles à comprendre, c'est pourquoi il est possible que vous ne voyiez pas bien le rôle des réglages avant d'avoir vu les widgets qui les utilisent comme les barres de progression, les viewport, les barres de défilement.

3.2.2 Créer des Ajustements (réglages)

Beaucoup de widgets qui utilisent les objets de réglage le font automatiquement mais dans certains cas que nous verrons plus loin, vous devrez les créer vous même avec :

```
new Gtk : :Adjustment($value,$lower,$upper,
                      $step_increment,$page_increment,$page_size) ;
```

L'argument `$value` est la valeur initiale que vos voulez donner à vos réglages, correspondant habituellement au point le plus haut et le plus à gauche du widget réglable.

L'argument `lower` spécifie la valeur la plus basse que le réglage peut prendre.

L'argument `$step_increment` spécifie le plus petit des deux incréments avec lequel l'utilisateur peut changer la valeur alors que `page_increment` est le plus grand.

L'argument `$page_size` correspond correspond habituellement à la zone visible d'un widget.

L'argument `$upper` est utilisé pour représenter les coordonnées du point le plus au fond et le plus à droite dans l'enfant du widget "paning". Par conséquent, ce n'est pas toujours la plus grande valeur que `$value` puisse prendre puisque `$page_size` est en général non nul.

3.2.3 Utiliser les réglages : la manière facile

Les widgets réglables peuvent être grossièrement divisés entre ceux qui utilisent et nécessitent des unités spécifiques pour ces valeurs et ceux qui les traitent comme des nombres arbitraires. Le second groupe inclut les widgets range (les barres de défilements, les échelles, les barres de progression et les boutons spins). Ces widgets sont tous typiquement ajustés directement par l'utilisateur à l'aide du clavier ou de la souris. Ils traiteront les valeurs `lower` et `upper` du réglage comme une intervalle à l'intérieur duquel on peut manipuler la valeur du réglage. Par défaut, ils ne modifieront que la valeur du réglage.

L'autre groupe comprend les widgets texte, viewport, listes composées et le fenêtre défilable. Ce sont des widgets typiquement ajustés "indirectement" à l'aide de barres de défilement. Alors que les widgets qui utilisent des réglages peuvent ou bien créer les leurs, ou bien utiliser ceux fournis, vous voudrez en général laisser cette catégorie de widget créer ses propres réglages. Habituellement, ils refuseront finalement toutes les valeurs, sauf la `$value` elle-même, que

vous leurs donner mais les résultats sont en général indéfinis (ce qui signifie que vous aurez à lire les codes sources et ils peuvent être différents de widget à widget).

Maintenant vous pensez probablement, puisque les widgets texte et viewport insistent sur le fait du déclarer tout sauf la valeur de leur réglage alors que les barres de défilement ne toucheront qu'à cette valeur, que si vous partagez un réglage entre une barre de défilement et un widget texte, la manipulation de la barre de défilement ajustera automatiquement le widget texte ! Et bien oui ! Juste avec :

```
# Créer son propre réglage
$text = new Gtk : :Text('''','''') ;
# utilise le réglage tout fraîchement créé pour la barre de défilement
$vscrollbar = new Gtk : :VScrollbar($text->vadj) ;
```

3.2.4 Les réglages internes

OK, vous vous dîtes, c'est bien ! Mais que se passe-t-il si vous voulez créer votre propre gestionnaire pour répondre quand l'utilisateur règle un widget range ou un bouton spin ? Et comment on obtient-on la valeur de réglage pour ces gestionnaires ? En Gtk-Perl, vous utiliserez :

```
get_adjustment->adjustment_name ;
```

ou `adjustment_name` est au choix :

```
value
lower
upper
step_increment
page_increment
page_size
```

Puisque, quand vous déclarez la `valeur` d'un réglage, vous voulez généralement que ce changement soit répété à tous les widgets qui utilisent ce réglage, GTK fournit la fonction suivante :

```
$adjustment->set_value($value) ;
```

Comme mentionné précédemment, les réglages sont capables d'émettre des signaux. C'est bien sûr la raison pour laquelle les mises à jour se produisent automatiquement quand vous partagez un objet Adjustment entre une barre de défilement et tout autre widget réglable ; tous les widgets réglables connectent des gestionnaires de signaux à leur signal d'ajustement `'value_changed'`, autant que le peut votre programme.

Les divers widgets qui utilisent l'objet Adjustment émettront ce signal sur un réglage à partir du moment ou il change sa valeur. Cela se produit à la fois quand l'entrée utilisateur oblige le curseur à glisser sur le widget range, aussi bien que quand le programme change explicitement la valeur par un `set_value()`. Ainsi, par exemple, si vous utilisez un widget échelle et que vous voulez changer l'angle de rotation d'une image, quelques soient ces changements de valeurs, vous créerez un rappel comme celui-ci :

```
sub cb_rotation_image
     {
          ($adjustment, $image)=@_ ;
          $image->rotation($adjustment->get_adjustment->value) ;
          ...
     }
```

et vous le connectez au réglage du widget échelle comme ceci :

```
$adjustment->signal_connect(''value_changed'',\&cb_rotation_image,$image) ;
```

Que se passe-t-il quand un widget reconfigure les champs `upper` et `lower` de ses réglages, comme, par exemple, quand l'utilisateur ajoute du texte à un widget texte ? Dans ce cas, il émet le signal `'changed'`.

Les widgets range connectent typiquement un gestionnaire pour ce signal ce qui change leur apparence pour refléter ce changement, par exemple la taille du curseur dans la barre de défilement grandira ou rétrécira en proportion inverse de la différence entre les valeurs `upper` et `lower` de ses réglages. Vous n'aurez probablement jamais besoin d'attacher un gestionnaire à ce signal sauf si vous écrivez un nouveau type de widget range. Quoi qu'il en soit, si vous changez les valeurs d'un réglage directement, vous devrez émettre ce signal sur lui pour reconfigurer n'importe quel widget qui l'utilise, comme ceci :

```
$adjustment->emit_by_name(''changed'') ;
```

3.2.5 Les widgets Range

La catégorie des widgets Range inclut la barre de défilement "ubiquitous" et le widget moins commun échelle (scale). Bien que ces deux widgets soient généralement utilisés dans des buts différents, ils sont presque similaires en fonction et en implémentation. Tous les widgets Range partagent un ensemble d'éléments graphiques, chacun possédant sa propre fenêtre X et recevant les évènements. Ils contiennent tous une glissière et un curseur (parfois appelé "thumbwheel" dans d'autres environnements GUI). Prendre le curseur avec le pointeur de la souris le promène le long de la glissière alors que cliquer dans la glissière avance le curseur en direction du lieu du clic, soit complètement soit d'une quantité fixée dépendant du bouton de la souris utilisé.

Comme mentionné précédemment, tous les widgets Range sont associés à un objet Adjustment à partir duquel ils calculent la longueur du curseur et sa position dans la glissière. Quand l'utilisateur manipule le curseur, le widget range changera la valeur de l'ajustement.

3.2.6 Les fonctions communes

Le widget Range a une gestion interne plutôt compliquée mais comme tous les widgets de la "classe de base", cette complexité n'est intéressante que si vous voulez le programmer. Ainsi, presque toutes les fonctions et les signaux qu'il définit, ne sont réellement utilisés qu'en écrivant des widgets dérivés.

La politique de mise à jour d'un widget Range définie à quel point, durant l'interaction de l'utilisateur, il changera le `$value` de son réglage et émettra le signal `$value_changed` de ce réglage. Les politiques de mises à jour sont :

`'continuous'` : c'est le défaut. Le signal `value_changed'` est émis continuellement même si le curseur n'est bougé que d'une valeur minuscule.

`'discontinuous'` : le signal `'value_changed'` est émis uniquement un fois que le curseur ne bouge plus et que l'utilisateur a relâché le bouton de la souris.

`'delayed'` : le signal `'value_changed'` est émis quand l'utilisateur relâche le bouton de la souris ou si le curseur s'arrête pendant une courte période de temps.

Ces politiques de mise à jour peuvent être déclarées à l'aide de la fonction :

```
$range->set_update_policy($policy) ;
```

Obtenir et déclarer l'ajustement "au vol" se fait par :

```
$range->get_adjustment() ;
$range->set_adjustment($adjustment) ;
```

`$range->get_adjustment()` retourne l'ajustement auquel `$range` est attaché.

`set_adjustment()` ne fait absolument rien si vous lui passez l'ajustement que Range utilise déjà, sans regarder si vous avez changé l'un des champs ou non. Si vous lui passez un nouvel ajustement, il ôtera la référence à l'ancien, s'il existe (peut-être en le détruisant), connectera les signaux appropriés au nouveau et appellera le fonction privée `adjustment_changed` qui recalculera (ou du moins est supposée recalculer) la taille et/ou la position du curseur et le redessinera si nécessaire. Comme mentionné dans la section sur les réglages, si vous souhaitez réutiliser le même ajustement, il vous faudra émettre le signal `'changed'` quand vous modifiez sa valeur, à l'aide de :

```
$adjustment->signal_emit ( `` changed '' ) ;
```

3.2.7 Souris et clavier

Tous les widgets Range de GTK réagissent aux clics de souris plus ou moins de la même manière. Cliquer avec le bouton 1 dans la glissière fera que le `page_increment` de l'ajustement sera ajouté ou soustrait à sa `value`. et que le curseur sera bougé en conséquence. Cliquer sur le bouton 2 dans la glissière fera sauter le curseur jusqu'à l'endroit du clic. Cliquer sur l'une des flèches de la barre de défilement provoquera un changement de la valeur du réglage égal au `step_increment`. Cela peut prendre un certain temps à maîtriser, mais par défaut, les barres de défilement, tout comme les widgets échelles, peuvent être manipulés au clavier (état "focus") en GTK. Si vous pensez que l'utilisateur sera dérouté, vous pouvez toujours rendre cette option indisponible en invalidant le flag "focus" de la barre de défilement.

```
$scrollbar->unset_flags('focus') ;
```

Les comportements claviers (qui ne sont bien sûr disponible que si le flag "focus" est actif) sont, pour des raisons évidentes, légèrement différents en fonction des widgets Range horizontaux et verticaux. Ils ne sont d'ailleurs pas tout à fait les mêmes selon qu'il s'agisse d'un widget échelle ou d'une barre de défilement, pour des raisons un peu moins évidentes (peut-être pour éviter la confusion entre les touches pour la barre horizontale et ceux pour la barre verticale dans les fenêtres défilables, qui, chacune, opèrent dans le même domaine).

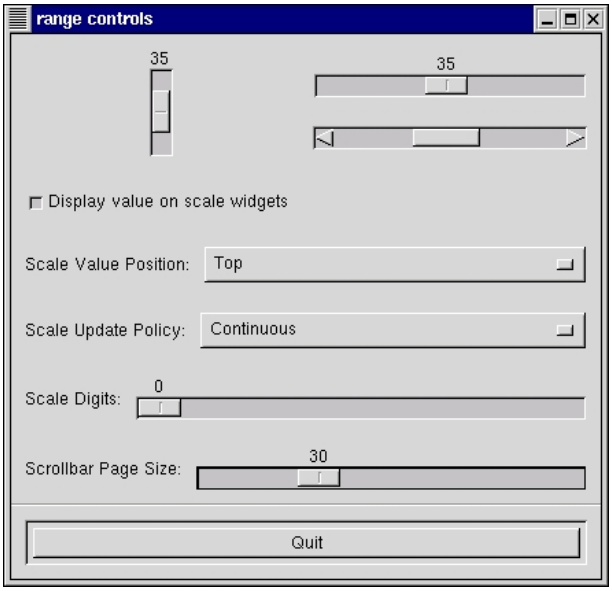
Tous les widgets Range verticaux peuvent être modifiés à l'aide des flèches vers le haut et vers le bas du clavier, aussi bien qu'avec les touches Page up et Page down. Les flèches montent et descendent le curseur selon le `step_increment`

alors que Page up et Page down le modifie selon le `page_increment`. L'utilisateur peut aussi déplacer le curseur tout en haut ou tout en bas de la glissière à l'aide du clavier. Avec le widget VScale, c'est fait avec les touches Home et End alors qu'avec VScrollbar c'est fait en tapant Control_Page up et Control_Page down.

Les flèches droite et gauche agissent comme on peut s'y attendre sur le widget Range en bougeant le curseur selon le `step_increment`. Les touches Home et End bougent le curseur au début ou à la fin de la glissière. Pour le HScale widget, bouger le curseur selon le `page_increment` se fait à l'aide des touches Control_Right et Control_Left alors que pour le HScrollbar c'est Control_Home et Control_End.

3.2.8 Exemple

Cet exemple crée une fenêtre avec trois widgets Range, tous connectés au même réglage, et un couple de contrôle pour régler quelques uns des paramètres mentionnés ci-dessus dans la section sur les réglages. Ainsi, vous pourrez voir comment ils affectent la manière dont ces widgets fonctionnent pour l'utilisateur.




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $hscale;
my $vscale;
my $window;
my $box1;
my $box2;
my $box3;
my $button;
my $scrollbar;
my $separator;
my $opt;
my $menu;
my $item;
my $label;
my $scale;
my $adj1;
```

```
my $adj2;

# Création standard de la fenêtre
$window = new Gtk : :Window( "toplevel" );
$window->set_title( "range controls" );

$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );

$box1 = new Gtk : :VBox( $false, 0 );
$window->add( $box1 );
$box1->show();

$box2 = new Gtk : :HBox( $true, 10 );
$box2->border_width( 10 );
$box1->pack_start( $box2, $true, $true, 0 );
$box2->show();

# value, lower, upper, step_increment, page_increment, page_size
# Notez que la valeur de page_size fait seulement une différence pour
# les widgets scrollbar , et la plus haute valeur que nous obtiendrons est
# (upper - page_size).
$adj1 = new Gtk : :Adjustment( 0.0, 0.0, 101.0, 0.1, 1.0, 1.0 );

$vscale = new Gtk : :VScale( $adj1 );
scale_set_default_values( $vscale );
$box2->pack_start( $vscale, $true, $true, 0 );
$vscale->show();

$box3 = new Gtk : :VBox( $false, 10 );
$box2->pack_start( $box3, $true, $true, 0 );
$box3->show();

# Réutilise le même adjustement
$hscale = new Gtk : :HScale( $adj1 );
$hscale->set_usize( 200, 30 );
scale_set_default_values( $hscale );
$box3->pack_start( $hscale, $true, $true, 0 );
$hscale->show();

# Réutilise encore le même adjustement
$scrollbar = new Gtk : :HScrollbar( $adj1 );
# Notez que cela implique que les échelles seront mises à jour
# continuellement quand la barre de défilement est bougée
$scrollbar->set_update_policy( 'continuous' );
$box3->pack_start( $scrollbar, $true, $true, 0 );
$scrollbar->show();

$box2 = new Gtk : :HBox( $false, 10 );
$box2->border_width( 10 );
$box1->pack_start( $box2, $true, $true, 0 );
$box2->show();

# Une case à cocher pour contrôler si la valeur est affichée ou non
$button = new Gtk : :CheckButton( "Display value on scale widgets" );
$button->set_active( $true );
$button->signal_connect( "toggled", \&cb_draw_value );
$box2->pack_start( $button, $true, $true, 0 );
$button->show();
```

```
$box2 = new Gtk : :HBox( $false, 10 ) ;
$box2->border_width( 10 ) ;

# Une option du menu pour changer la position de la valeur.
$label = new Gtk : :Label( "Scale Value Position :" ) ;
$box2->pack_start( $label, $false, $false, 0 ) ;
$label->show() ;

$opt = new Gtk : :OptionMenu() ;
$menu = new Gtk : :Menu() ;

$item = make_menu_item( "Top", \&cb_pos_menu_select, 'top' ) ;
$menu->append( $item ) ;

$item = make_menu_item( "Bottom", \&cb_pos_menu_select, 'bottom' ) ;
$menu->append( $item ) ;

$item = make_menu_item( "Left", \&cb_pos_menu_select, 'left' ) ;
$menu->append( $item ) ;

$item = make_menu_item( "Right", \&cb_pos_menu_select, 'right' ) ;
$menu->append( $item ) ;

$opt->set_menu( $menu ) ;
$box2->pack_start( $opt, $true, $true, 0 ) ;
$opt->show() ;

$box1->pack_start( $box2, $true, $true, 0 ) ;
$box2->show() ;

$box2 = new Gtk : :HBox( $false, 10 ) ;
$box2->border_width( 10 ) ;

# Une autre option du menu, cette fois ci pour changer la politique
# de mise à jour des widgets échelles
$label = new Gtk : :Label( "Scale Update Policy :" ) ;
$box2->pack_start( $label, $false, $false, 0 ) ;
$label->show() ;

$opt = new Gtk : :OptionMenu() ;
$menu = new Gtk : :Menu() ;

$item = make_menu_item ("Continuous", \&cb_update_menu_select, 'continuous' ) ;

$menu->append( $item ) ;

$item = make_menu_item ("Discontinuous", \&cb_update_menu_select, 'discontinuous' ) ;
$menu->append( $item ) ;

$item = make_menu_item ("Delayed", \&cb_update_menu_select, 'delayed' ) ;
$menu->append( $item ) ;

$opt->set_menu( $menu ) ;
$box2->pack_start( $opt, $true, $true, 0 ) ;
$opt->show() ;

$box1->pack_start( $box2, $true, $true, 0 ) ;
```

```
$box2->show();

$box2 = new Gtk::HBox( $false, 10 );
$box2->border_width( 10 );

# Un widget HScale pour ajuster le nombre de digits
# sur les exemple d'échelle.
$label = new Gtk::Label( "Scale Digits :" );
$box2->pack_start( $label, $false, $false, 0 );
$label->show();

$adj2 = new Gtk::Adjustment( 1.0, 0.0, 5.0, 1.0, 1.0, 0.0 );
$adj2->signal_connect( "value_changed", \&cb_digits_scale );
$scale = new Gtk::HScale( $adj2 );
$scale->set_digits( 0 );
$box2->pack_start( $scale, $true, $true, 0 );
$scale->show();

$box1->pack_start( $box2, $true, $true, 0 );
$box2->show();

$box2 = new Gtk::HBox( $false, 10 );
$box2->border_width( 10 );

# Et un dernier widget HScale pour ajuster la taille de la page de la scrollbar
$label = new Gtk::Label( "Scrollbar Page Size :" );
$box2->pack_start( $label, $false, $false, 0 );
$label->show();

$adj2 = new Gtk::Adjustment( 1.0, 1.0, 101.0, 1.0, 1.0, 0.0 );
$adj2->signal_connect( "value_changed", \&cb_page_size, $adj1 );
$scale = new Gtk::HScale( $adj2 );
$scale->set_digits( 0 );
$box2->pack_start( $scale, $true, $true, 0 );
$scale->show();

$box1->pack_start( $box2, $true, $true, 0 );
$box2->show();

$separator = new Gtk::HSeparator();
$box1->pack_start( $separator, $false, $true, 0 );
$separator->show();

$box2 = new Gtk::VBox( $false, 10 );
$box2->border_width( 10 );
$box1->pack_start( $box2, $false, $true, 0 );
$box2->show();

$button = new Gtk::Button( "Quit" );
$button->signal_connect( "clicked", sub { Gtk->exit( 0 ); } );
$box2->pack_start( $button, $true, $true, 0 );
$button->can_default( $true );
$button->grab_default();
$button->show();

$window->show();

main Gtk;
```

```
exit( 0 ) ;



### Subroutines

sub cb_pos_menu_select
  {
    my ( $item, $pos ) = @_ ;

    # Déclare le valeur de la position pour chaque widget scale
    $hscale->set_value_pos( $pos ) ;
    $vscale->set_value_pos( $pos ) ;
  }

sub cb_update_menu_select
  {
    my ( $item, $policy ) = @_ ;

    # Déclare la politique de mise à jour pour chaque widget scale
    $hscale->set_update_policy( $policy ) ;
    $vscale->set_update_policy( $policy ) ;
  }

sub cb_digits_scale
  {
    my ( $adj ) = @_ ;

    # Déclare le nombre de décimale pour adj->value
    $hscale->set_digits( $adj->value ) ;
    $vscale->set_digits( $adj->value ) ;
  }

sub cb_page_size
  {
    my ( $get, $set ) = @_ ;

    # Déclare le page size and page increment size pour l'ajustement
    # à la valeur spéficiée  par la "Page Size" de l'échelle
    $set->page_size( $get->value ) ;
    $set->page_increment( $get->value ) ;

    # Émet maintenant le signal ``changed'' pour reconfigurer tous les widgets
    # qui sont liés à ce réglage
    $set->signal_emit_by_name( "changed" ) ;
  }

sub cb_draw_value
  {
    my ( $button ) = @_ ;

    #  Branche la valeur de l'affichage, débranche les widgets échelle
    # selon l'état de la cases à cocher
    $hscale->set_draw_value( $button->active ) ;
    $vscale->set_draw_value( $button->active ) ;
  }

# Fonctions pratiques
```

```
sub make_menu_item
  {
    my ( $name, $callback, $data ) = @_;
    my $item;

    $item = new Gtk : :MenuItem( $name );
    $item->signal_connect( "activate", $callback, $data );
    $item->show();

    return $item;
  }

sub scale_set_default_values
  {
    my ( $scale ) = @_;

    $scale->set_update_policy( 'continuous' );
    $scale->set_digits( 1 );
    $scale->set_value_pos( 'top' );
    $scale->set_draw_value( $true );
  }
```

Vous remarquerez que le programme n'appelle pas `signal_connect()` pour le `'delete_event'`, mais seulement pour le signal `'destroy'`. Cela accomplira tout de même la fonction désirée car un `'delete_event'` sans gestionnaire se transformera en signal `'destroy'` passé à la fenêtre.

3.3 Les barres de défilements (scrollbar)

```
Object
  +--- Widget
       +--- Range
            +--- Scrollbar
```

Il s'agit effectivement de nos bonnes vieilles barres de défilement. Elles sont utilisées pour faire défiler d'autres widgets comme une liste, une boîte texte ou un viewport (et il est en général plus facile d'utiliser la fenêtre défilable dans la plupart des cas). Pour les autres usages, vous devriez utiliser les widgets échelles qui sont plus sympathiques et malléable.

Il existe différents types de barres de défilement, horizontale ou verticale. Il n'y a pas grand chose à dire à leurs propos. Vous pouvez les créer à l'aide des fonctions suivantes :

```
new Gtk : :HScrollbar($adjustment);
new Gtk : :VScrollbar($adjustment);
```

L'argument `adjustment` peut-être soit un pointeur sur un ajustement existant ou une chaîne vide auquel cas une sera créée pour vous. Spécifier une chaîne vide peut en fait être utile dans ce cas : si vous voulez passer l'ajustement nouvellement créé à une fonction constructrice d'autres widgets qui le construiront à votre place, comme un widget texte par exemple.

3.4 Le widget échelle (scale)

```
Object
  +--- Widget
       +--- Range
            +--- Scrollbar
```

Les widgets échelles sont utilisés pour permettre de sélectionner visuellement et de manipuler une valeur dans un intervalle spécifique. Vous pouvez vouloir utiliser, par exemple, un widget échelle pour ajuster le niveau de prévisualisation d'une image ou pour contrôler la luminosité d'une couleur, ou pour spécifier le nombre de minutes d'inactivité avant que l'économiseur d'écran s'affiche à l'écran.

Comme avec les barres de défilement, il existe des types de widgets séparés pour les widgets horizontaux et verticaux. (La plupart des programmeurs préfèrent parler des widget échelles horizontaux. Puisqu'ils fonctionnent essentiellement de la même manière, il est inutile de les traiter séparément ici. Les fonctions suivantes créent des widgets échelles horizontaux et verticaux :

```
new Gtk : :HScale($adjustment) ;
new Gtk : :VScale($adjustment) ;
```

L'argument `adjustment` peut-être soit un ajustement qui a déjà été créé, ou une chaîne "nulle" auquel cas un ajustement anonyme est créé avec toutes ses valeurs déclarées à 0.0 (ce qui n'est pas très utile dans ce cas). Afin de vous éviter toute confusion, vous voudrez probablement créé votre ajustement avec un `page_size` de 0.0 pour que sa valeur `upper` corresponde en fait à la valeur la plus élevée que l'utilisateur puisse sélectionner (si vous êtes déjà complètement perdu, relisez la section sur les ajustements pour avoir une explication sur ce que font exactement les ajustements, comment les créer et comment les manipuler).

Les widgets échelles peuvent afficher leur valeur courante sous la forme d'un nombre au côté de la glissière. Le comportement par défaut est de montrer cette valeur, mais vous pouvez le changer avec la fonction :

```
$scale->set_draw_value($draw_value) ;
```

Comme vous l'avez certainement deviné, `$draw_value` est une valeur vraie ou fausse avec les conséquences que l'on devine.

La valeur affichée est arrondie par défaut au dixième près, comme l'est l'intervalle de la valeur de l'ajustement. Vous pouvez le changer avec :

```
$scale->set_digits($digits) ;
```

`$digits` est le nombre de décimales que vous désirez. Vous pouvez donner n'importe quelle valeur à `$digits` mais sachez qu'il n'y aura pas plus de 13 décimales affichées à l'écran.

Finalement la valeur peut être affichée à différents endroits par rapport à la glissière.

```
$scale->set_value_pos($pos) ;
```

L'argument `$pos` peut être soit : `'left'`, `'right'`, `'top'`, `'bottom'`. Si vous positionnez la valeur au côté de la glissière (en haut ou en bas pour le widget horizontale), alors il suivra le curseur le long de la glissière.

3.5 Les barres de progression

```
Object
  +--- Widget
       +--- Progress
            +--- ProgressBar
```

Les barres de progression sont utilisées pour montrer l'état d'avancement d'une opération. Elles sont plutôt faciles à utiliser, comme vous le verrez dans le programme ci-dessous. Mais, tout d'abord voyons comment créer une nouvelle barre de progression.

Il existe deux manières de les créer, une simple qui ne demande aucun argument et une qui demande un objet Adjustment comme argument. Si vous utilisez la première, la barre de progression créera son propre objet Adjustment.

```
$progress = new Gtk : :ProgressBar() ;
$progress = new Gtk : :ProgressBar( $adjustment ) ;
```

La seconde méthode a l'avantage de vous permettre d'utiliser l'objet adjustment pour spécifier votre propre échelle de paramètres pour la barre de progression.

Le réglage de l'objet progress peut être changé dynamiquement par :

```
$progress->set_adjustment( $adjustment ) ;
```

Maintenant que votre barre de progression est créée vous pouvez utiliser :

```
$progress->update( $percentage ) ;
```

L'argument est la quantité "réalisée", ce qui signifie que la barre sera remplie d'une quantité variant de 0 à 100%. Cette quantité est passée à la fonction à l'aide d'un nombre réel compris entre 0.0 et 1.0.

Le sens de remplissage de la barre de progression peut être indiqué par :

```
$progress->set_orientation( $orientation ) ;
```

L'argument `$orientation` qui peut prendre les valeur suivantes :

```
'left_to_right' : de gauche à droite
'right_to_left' : de droite à gauche
'bottom_to_top' : de bas en haut
'top_to_bottom' : de haut en bas
```

Quand elle est utilisée pour montrer le degré d'avancement d'un processus, on peut choisir d'afficher la valeur de la barre de progression soit sous le mode continu soit discret. En mode continu, la barre de progression est mise à jour pour chaque valeur. En mode discret, la barre de progression est mise à jour en un certain nombre de blocs discrets. Le nombre de block est également configurable.

Le style de la barre de progression est déclaré à l'aide de la fonction :

```
$progress->set_bar_style( $style ) ;
```

Le paramètre `$style` peut correspondre à l'une des deux valeurs : `'continuous'` ou `'discrete'`.

Le nombre de blocs discrets peut être indiqué par :

```
$progress->set_discrete_blocks( $blocks ) ;
```

En plus de vous indiquer le degré d'avancement d'un processus, la barre de progression peut être déclarée juste pour montrer qu'il y a de l'activité. Cela peut être utile si la progression ne peut pas être mesurée par une valeur échelle. Le mode activité n'est pas affecté par le style de la barre de progression qui est décrit ci-dessus et ne s'en accomode pas. Le mode est soit vrai soit faux et est sélectionné par la fonction :

```
$progress->set_activity_mode( $activity_mode ) ;
```

La taille de chaque pas de l'indicateur d'activité et le nombre de blocks sont déclarés avec les fonctions suivantes :

```
$progress->set_activity_step( $step ) ;
$progress->set_activity_blocks( $blocks ) ;
```

En mode continu, la barre de progression peut aussi afficher une chaîne de texte configurable à l'intérieur de la glissière avec la fonction :

```
$progress->set_format_string( $format ) ;
```

L'argument `$format` est le même que ceci utilisé par un `printf`. Les directives suivantes peut être utilisées à l'intérieur de la chaîne format.

```
%p - pourcentage
%v - valeur
%l - valeur la plus basse de l'indicateur
%u - valeur la plus haute de l'indicateur
```

On peut afficher ou enlever ce texte à l'aide de la fonction :

```
$progress->set_show_text( $show_text ) ;
```

L'argument `$show_text` est une valeur vraie ou fausse. L'apparence du texte peut être modifiée par :

```
$progress->set_text_alignment( $x_align, $y_align ) ;
```

Les arguments `$x_align` et `$y_align` prennent des valeurs comprises entre 0.0 et 1.0. Leurs valeurs indiquent la position du texte dans la glissière. Des valeurs nulles pour les deux placent le texte en haut à gauche, les valeurs de 0.5 centrent le texte (valeurs par défaut) et les valeurs 1.0 le placent en bas à droite.

Le texte courant de l'objet progress peut être retrouvé pour l'ajustement courant ou pour un ajustement spécifique à l'aide des deux fonctions suivantes. Ces fonctions retournent la chaîne formatée qui serait affichée dans la glissière.

```
$progress->get_current_text() ;
$progress->get_text_from_value( $value ) ;
```

Il y a également un autre moyen de changer les valeurs et la longueur de l'intervalle d'une barre de progression avec :

```
$progress->configure( $value, $min, $max ) ;
```

Cette fonction fournit une simple interface à la longueur de l'intervalle et à la valeur d'une barre de progression.

Les fonctions restantes peuvent être utilisées pour obtenir ou déclarer la valeur courante de la barre de progression en fonction des différents formats :

```
$progress->set_percentage( $percentage ) ;
$progress->set_value( $value ) ;
$progress->get_value() ;
$progress->get_current_percentage() ;
$progress->get_percentage_from_value( $adjustment ) ;
```

Ces fonctions se passent de commentaires. La dernière utilise l'ajustement spécifique de la barre de progression pour déterminer le pourcentage par rapport à la valeur de la longueur de l'intervalle.

Les barres de progression sont généralement utilisées avec des temps morts et d'autres fonctions de ce genre pour donner l'impression du multitâche. Toutes emploieront la fonction `update()` de la même manière.

Exemple

Voici un exemple de barre de progression mise à jour avec des temps morts. Ce programme vous montrera également comment réinitialiser une barre de progression.

```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window;
my $pbar;
my $timer;
my $align;
my $separator;
my $table;
my $adj;
my $button;
my $check1;
my $check2;
my $vbox;

$window = new Gtk::Window( "toplevel" );
$window->set_policy( $false, $false, $true );
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$window->set_title( "Progress Bar" );
$window->border_width( 0 );

$vbox = new Gtk::VBox( $false, 5 );
$vbox->border_width( 10 );
$window->add( $vbox );
$vbox->show();

# Centre les objets
$align = new Gtk::Alignment( 0.5, 0.5, 0, 0 );
$vbox->pack_start( $align, $false, $false, 5 );
$align->show();

# Crée un objet adjustement pour l'utiliser dans la barre de progression
$adj = new Gtk::Adjustment( 0, 1, 150, 0, 0, 0 );

# Crée la barre de progression qui utilise cet ajustement
$pbar = new_with_adjustment Gtk::ProgressBar( $adj );

# Déclare le format de la chaîne qui doit être affiché dans la glissière
#   %p - pourcentage
#   %v - valeur
#   %l - borne inférieure de l'intervalle
#   %u - borne supérieur de l'intervalle
```

```
$pbar->set_format_string( "%v from [%l-%u] (=%p%%)" ) ;
$align->add( $pbar ) ;
$pbar->show() ;

# Ajoute une routine timer pour mettre à jour la barre de progression
$timer = Gtk->timeout_add( 100, \&progress_timeout ) ;

$separator = new Gtk : :HSeparator() ;
$vbox->pack_start( $separator, $false, $false, 0 ) ;
$separator->show() ;

# rows, columns, homogeneous
$table = new Gtk : :Table( 2, 3, $false ) ;
$vbox->pack_start( $table, $false, $true, 0 ) ;
$table->show() ;

# Ajoute une case à cocher pour sélectionner l'affichage ou non du texte
# dans la glissière
$check1 = new Gtk : :CheckButton( "Show text" ) ;
$table->attach( $check1, 0, 1, 0, 1, [ 'expand', 'fill' ],
                ['expand', 'fill' ], 5, 5 ) ;
$check1->signal_connect( "clicked", sub {
                          $pbar->set_show_text( $check1->active ) ; } ) ;
$check1->show() ;

# Ajoute un bouton toggle pour choisir ou non le mode activité
$check2 = new Gtk : :CheckButton( "Activity mode" ) ;
$table->attach( $check2, 0, 1, 1, 2, [ 'expand', 'fill' ],
                [ 'expand', 'fill' ], 5, 5 ) ;
$check2->signal_connect( "clicked", sub {
                          $pbar->set_activity_mode( $check2->active ) ; } ) ;
$check2->show() ;

$separator = new Gtk : :VSeparator() ;
$table->attach( $separator, 1, 2, 0, 2, [ 'expand', 'fill' ],
                [ 'expand', 'fill' ], 5, 5 ) ;
$separator->show() ;

# Ajoute un bouton radio pour le choix du mode d'affichage continu
$button = new Gtk : :RadioButton( "Continuous" ) ;
$table->attach( $button, 2, 3, 0, 1, [ 'expand', 'fill' ],
                [ 'expand', 'fill' ], 5, 5 ) ;
$button->signal_connect( "clicked", sub {
                           $pbar->set_bar_style( 'continuous' ) ; } ) ;
$button->show() ;

# Ajoute un bouton radio pour sélectionner l'affichage en mode discret
$button = new Gtk : :RadioButton( "Discrete", $button ) ;
$table->attach( $button, 2, 3, 1, 2, [ 'expand', 'fill' ],
                [ 'expand', 'fill' ], 5, 5 ) ;
$button->signal_connect( "clicked", sub {
                           $pbar->set_bar_style( 'discrete' ) ; } ) ;
$button->show() ;

$separator = new Gtk : :HSeparator() ;
$vbox->pack_start( $separator, $false, $false, 0 ) ;
$separator->show() ;
```

```
# Ajoute un bouton pour sortir du programme
$button = new Gtk : :Button( "Close" ) ;
$button->signal_connect( "clicked", sub { Gtk->exit( 0 ) ; } ) ;

$vbox->pack_start( $button, $false, $false, 0 ) ;

# Ceci en fait le bouton par défaut.
$button->can_default( $true ) ;

# Ce permet au bouton d'être le bouton  concerné par les évènemnts
# Appuyer juste sur la touche ''Entrée'' et le bouton sera activé
$button->grab_default() ;
$button->show() ;

$window->show() ;
main Gtk ;
exit( 0 ) ;



### Subroutines

sub progress_timeout
  {
    my ( $widget ) = @_ ;

    my $new_val ;
    my $adj ;

    # Calcule la valeur de la barre de progression en utilisant
    # l'intervalle de valeur déclaré par l'objet ajustement

    $new_val = $pbar->get_value() + 1 ;

    $adj = $pbar->adjustment ;
    $new_val = $adj->lower if ( $new_val > $adj->upper ) ;

    # Déclare la nouvelle valeur
    $pbar->set_value( $new_val ) ;

    # Comme c'est une fonction temps mort, retournez VRAI
    # pour continuer à être appelée
    return ( $true ) ;
  }
```

3.6 Les widgets "éditables"

```
Object
  +--- Widget
       +--- Editable
```

Le widget éditable est la classe de base de tous les widgets d'édition de texte. Cette classe ne peut être instanciée (il n'y a pas de fonction `new()`) et fournit uniquement les fonctionnalités communes aux widgets textes.

Tous les widgets qui éditent du texte vous permettent de sélectionner une région de texte avec :

```
$editable->select_region( $start, $end ) ;
```

Ou le texte sélectionné est composé des caractères de la position `$start` jusqu'à la position `$end` non inclus. Si `$end` est négatif, alors il compte à rebours à partir de la fin du texte.

Du texte peut être inséré à une certaine position en utilisant :

```
$editable->insert_text( $new_text, $position ) ;
```

`$new_text` est la chaîne à insérer et `$position` la position où il faut placer le texte.

Des caractères peuvent être effacés avec :

```
$editable->delete_text( $start, $pos ) ;
```

Comme pour `select_region`, les caractères à effacer sont compris entre `$start` et `$end`.

Vous pouvez récupérer des caractères avec :

```
$editable->get_chars( $start, $end ) ;
```

Si vous avez sélectionné du texte, vous pouvez le couper et le placer dans le bloc note avec :

```
$editable->cut_clipboard() ;
```

Cela prend les caractères sélectionnés pour les placer dans le bloc note et les efface du widget éditable.

Vous pouvez aussi copier le texte sélectionné dans le bloc note.

```
$editable->copy_clipboard() ;
```

Les caractères sont copiés dans le bloc note mais pas effacé du widget éditable.

Le texte du bloc note peut être collé dans le widget éditable à l'endroit où se trouve le curseur :

```
$editable->paste_clipboard() ;
```

Pour effacer le texte sélectionné :

```
$editable->delete_selection() ;
```

La position du curseur peut être déclarée et retrouvée avec :

```
$editable->set_position( $position ) ;
$editable->get_position() ;
```

Les widgets éditables peuvent être en lecture seule ou bien éditable :

```
$editable->set_editable( $is_editable ) ;
```

`$is_editable` est une valeur vraie ou fausse qui détermine si le widget éditable peut être édité ou non par l'utilisateur.

```
Le widget éditable possède un grand nombre de signaux disponibles qui sont :
      'changed'
      'insert-text'
      'delete-text'
      'activate'
      'set-editable'
      'move-cursor'
      'move-word'
      'move-page'
      'move-to-row'
      'move-to-column'
      'kill-char'
      'kill-word'
      'kill-line'
      'cut-clipboard'
      'copy-clipboard'
      'paste-clipboard'
```

L'utilisation de ces signaux devrait être évidente. Si vous avez une question sur leur utilisation, consulter la documentation GTK. Le seul commentaire que je ferais est que vous pouvez émettre le signal `'changed'` en appelant la fonction :

```
$editable->changed() ;
```

3.7 Le widget entrée

```
Object
  +--- Widget
       +--- Editable
            +--- Entry
```

Le widget entrée permet de taper du texte dans une boîte d'une seule ligne. Le texte peut être placé avec des appels de fonctions qui déterminent si le nouveau texte doit remplacer, se placer devant ou après le contenu actuel du widget entrée.

Il existe deux fonctions pour créer des widgets entrée :

```
$entry = new Gtk : :Entry() ;
$entry = new Gtk : :Entry( $max_length ) ;
```

La première crée juste un nouveau widget entrée alors que le second crée le widget et déclare une longueur limite pour le texte.

Il existe plusieurs fonctions qui modifient le texte contenu dans le widget Entrée :

```
$entry->set_text( $text ) ;
$entry->append_text( $text ) ;
$entry->prepend_text( $text ) ;
```

La fonction `set_text()` déclare le contenu du widget Entrée remplaçant le contenu présent. Les fonctions `append_text()` et `prepend_text()` permettent de placer du texte après ou avant le contenu du widget.

Si vous utilisez une Entrée et que vous ne voulez pas que le texte entré soit visible, par exemple quand vous entrer un mot de passe, vous pouvez utiliser la fonction suivante qui prend une valeur vraie ou fausse :

```
$entry->set_visibility( $visible ) ;
```

Si nous voulons attraper le moment ou l'utilisateur a rentré du texte, nous pouvons connecter un signal `'activate'` ou `'changed'`.

`'activate'` se manifeste quand l'utilisateur appuie sur la touche "Entrée" à l'intérieur du widget.

`'changed'` se manifeste quand le texte change, pour chaque caractère entré ou effacé.

Exemple

```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $window ;
my $vbox ;
my $hbox ;
my $entry ;
my $button ;
my $check ;

$window = new Gtk : :Window( "toplevel" ) ;
$window->set_usize( 200, 100 ) ;
$window->set_title( "GTK Entry" ) ;
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ) ; } ) ;

$vbox = new Gtk : :VBox( $false, 0 ) ;
$window->add( $vbox ) ;
$vbox->show() ;
```

```
$entry = new Gtk : :Entry( 50 ) ;
$entry->signal_connect( "activate", \&enter_callback, $entry ) ;
$entry->set_text( "Hello" ) ;
$entry->append_text( "World" ) ;
$entry->select_region( 0, length( $entry->get_text() ) ) ;
$vbox->pack_start( $entry, $true, $true, 0 ) ;
$entry->show() ;

$hbox = new Gtk : :HBox( $false, 0 ) ;
$vbox->add( $hbox ) ;
$hbox->show() ;

$check = new Gtk : :CheckButton( "Editable" ) ;
$hbox->pack_start( $check, $true, $true, 0 ) ;
$check->signal_connect( "toggled", \&entry_toggle_editable, $entry ) ;
$check->set_active( $true ) ;
$check->show() ;

$check = new Gtk : :CheckButton( "Visible" ) ;
$hbox->pack_start( $check, $true, $true, 0 ) ;
$check->signal_connect( "toggled", \&entry_toggle_visibility, $entry ) ;
$check->set_active( $true ) ;
$check->show() ;

$button = new Gtk : :Button( "Close" ) ;
$button->signal_connect( "clicked", sub { Gtk->exit( 0 ) ; } ) ;
$vbox->pack_start( $button, $true, $true, 0 ) ;
$button->can_default( $true ) ;
$button->grab_default() ;
$button->show() ;

$window->show() ;

main Gtk ;
exit( 0 ) ;



### Subroutines

sub enter_callback
  {
    my ( $widget, $entry ) = @_ ;
    my $entry_text = $entry->get_text() ;
    print( "Entry contents : $entry_text\n" ) ;
  }

sub entry_toggle_editable
  {
    my ( $checkbutton, $entry ) = @_ ;
    $entry->set_editable( $checkbutton->active ) ;
  }

sub entry_toggle_visibility
  {
    my ( $checkbutton, $entry ) = @_ ;
    $entry->set_visibility( $checkbutton->active ) ;
  }
```

3.8 Le widget Texte

```
Object
  +--- Widget
       +--- Editable
            +--- Text
```

Le widget Texte permet d'afficher et d'éditer un texte de plusieurs lignes. Il supporte à la fois les textes multicouleurs multi polices de caractères, permettant de les mélanger comme vous le souhaitez. Il existe aussi un large éventail de raccourcis claviers qui sont compatibles avec l'éditeur Emacs. Le widget Texte supporte les facilités de couper-coller incluant l'utilisation du double clic et du triple clic, pour sélectionner respectivement un mot ou une ligne complète.

3.8.1 Créer et configurer une boîte texte.

Une seule fonction pour créer un nouveau widget Texte :

```
$text = new Gtk : :Text( $hadj, $vadj ) ;
```

Les arguments nous permettent de donner au widget Texte des ajustements qui peuvent être utilisés pour chercher à régler le widget. Si aucun valeur n'est passée en argument, la fonction \verb+new()+ choisira les siens.

```
$text->set_adjustments( $hadj, $vadj ) ;
```

La fonction précédente permet aux ajustements horizontaux et verticaux d'être changés à n'importe quel moment. Le widget Texte ne créera pas automatiquement ses propres barres de défilements si la quantité de texte à afficher est trop grande pour la fenêtre. Nous aurons donc à les créer et à les ajouter nous même :

```
$vscrollbar = new Gtk : :VScrollbar( $text->$vadj ) ;
$hbox->pack_start( $vscrollbar, $false, $false, 0 ) ;
$vscrollbar->show() ;
```

Le code ci-dessus crée une nouvelle barre de défilement à l'attache à l'ajustement vertical du widget Texte. Il est ensuite mis dans une boîte de manière classique. Souvent les widget Texte ne supportent pas les barres de défilements horizontales.

Il y a deux manières d'utiliser le widget Texte : permettre à l'utilisateur d'éditer un texte ou nous permettre d'afficher un texte multiligne pour l'utilisateur. Afin de passer d'un mode à l'autre, il y a la fonction :

```
$text->set_editable( $editable ) ;
```

L'argument `editable` est une valeur vraie ou fausse qui précise si l'utilisateur est autorisé ou non à éditer le contenu du widget Texte. Si le widget Texte est éditable, un curseur est affiché au point d'insertion.

Vous n'êtes pas obligés de vous limiter à l'un de ces deux modes et vous pouvez changer d'état en cours d'utilisation.

Le widget Texte délimite les lignes de texte qui sont trop longues pour tenir sur une ligne d'affichage. Son comportement par défaut est de couper les mots au niveau du changement de ligne. Cela peut être changé par :

```
$text->set_word_wrap( $word_wrap ) ;
```

Cette fonction nous permet de spécifier que le widget Texte doit délimiter les lignes à la fin des mots. L'argument `$word_wrap` est une valeur vraie ou fausse.

Vous pouvez également choisir le comportement d'une ligne quand elle atteint la fin du widget, sans s'occuper de la longueurs des mots avec :

```
$text->set_line_wrap( $line_wrap ) ;
```

3.8.2 Manipulation de texte

Le point d'insertion d'un widget texte peut être déclaré par :

```
$text->set_point( $index ) ;
```

Ou `$index` correspond à la position du point d'insertion.

De manière analogue, on peut obtenir la position du point d'insertion à l'aide de :

```
$text->get_point() ;
```

Une fonction utile en combinaison avec les deux précédente :

```
$text->get_length() ;
```

qui retourne la longueur du widget texte. La longueur est le nombre de caractère contenu dans le bloc texte incluant les caractères comme `'newline'`.

Afin d'insérer du texte au point d'insertion, la fonction `insert()` est utilisée ce qui permet aussi de préciser la couleur du fond, du texte et la police.

```
$text->insert( $font, $foreground, $background, $string ) ;
```

Le passage de valeurs indéfinies donnera que les valeurs contenues dans widget Style seront utilisées.

Le widget Texte est l'un des quelques widgets GTK qui change d'aspect dynamiquement en dehors de `main Gtk`. Cela signifie que tout changement du contenu du widget prend effet immédiatement. Cela peut être indésirable quand on pratique de multiples changements sur un widget Texte. Afin de nous permettre de faire plusieurs mises à jour au widget texte sans le redessiner continuellement, on peut geler le widget ce qui l'empêche temporairement de se redessiner à chaque changement. Nous pouvons alors redessiner le widget quand le mise à jour est complète. Les fonctions qui permettent le changement d'état sont :

```
$tree->freeze()
$tree->thaw() ;
```

Le texte est effacé relativement au point d'insertion à l'aide des fonctions suivantes :

```
$text->backward_delete( $num_chars ) ;
$text->forward_delete( $num_chars ) ;
```

Si vous désirez retrouver le contenu d'un widget Texte alors utiliser `index($index)` qui vous permet de retrouver le caractère qui est la position `$index`.

3.8.3 Raccourcis clavier

Le widget Texte possède un nombre préinstallé de raccourcis pour les fonctions courantes d'édition et de sélection. On y accède avec une combinaison de touches Control et Alt.

En plus de tout cela, appuyer sur la touche Control tout en utilisant les touches de déplacement du curseur permet de faire des déplacements de mot en mot. Appuyer sur le touche Alt tout en bougeant le curseur étend la sélection.

Raccourcis de déplacements

```
Ctrl-A   Début de ligne
Ctrl-E   Fin de ligne
Ctrl-N   Ligne suivante
Ctrl-P   Ligne précédente
Ctrl-B   Recule d'un caractère
Ctrl-F   Avance d'un caractère
Alt-B    Recule d'un mot
Alt-F    Avance d'un mot
```

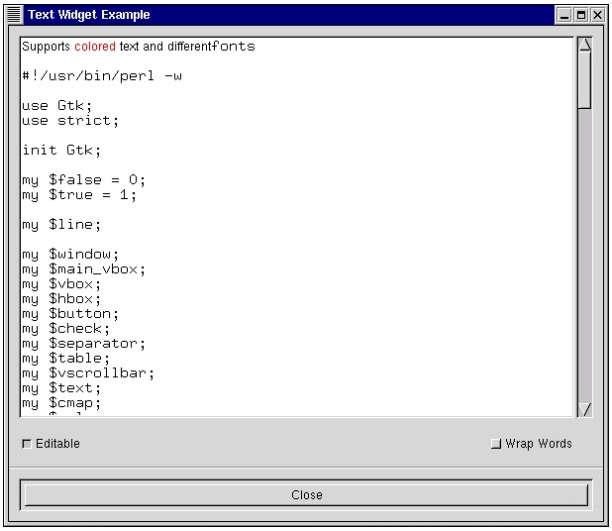
Raccourcis d'édition

```
Ctrl-H   Efface le caractère précédent
Ctrl-D   Efface le caractère suivant
Ctrl-W   Efface le mot précédent
Alt-D    Efface le mot suivant
Ctrl-K   Efface jusqu'à la fin de la ligne
Ctrl-U   Efface la ligne
```

Raccourcis de sélection

```
Ctrl-X   Couper
Ctrl-C   Copier
Ctrl-V   Coller
```

3.8.4 Exemple




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $line;

my $window;
my $main_vbox;
my $vbox;
my $hbox;
my $button;
my $check;
my $separator;
my $table;
my $vscrollbar;
my $text;
my $cmap;
my $color;
my $fixed_font;


$window = new Gtk::Window( 'toplevel' );
$window->set_usize( 600, 500 );
$window->set_policy( $true, $true, $false );
$window->signal_connect( 'destroy', sub { Gtk->exit( 0 ); } );
$window->set_title( "Text Widget Example" );
$window->border_width( 0 );

$main_vbox = new Gtk::VBox( $false, 0 );
$window->add( $main_vbox );
$main_vbox->show();

$vbox = new Gtk::VBox( $false, 10 );
```

```
$vbox->border_width( 10 );
$main_vbox->pack_start( $vbox, $true, $true, 0 );
$vbox->show();

$table = new Gtk : :Table( 2, 2, $false );
$table->set_row_spacing( 0, 2 );
$table->set_col_spacing( 0, 2 );
$vbox->pack_start( $table, $true, $true, 0 );
$table->show();

# Crée un nouveau widget GtkText
$text = new Gtk : :Text( undef, undef );
$text->set_editable( $true );
$table->attach( $text, 0, 1, 0, 1,
                [ 'expand', 'shrink', 'fill' ],
                [ 'expand', 'shrink', 'fill' ],
                0, 0 );
$text->show();

#  Ajoute un barre de défilement verticale au widget Texte
$vscrollbar = new Gtk : :VScrollbar( $text->vadj );
$table->attach( $vscrollbar, 1, 2, 0, 1, 'fill',
                [ 'expand', 'shrink', 'fill' ], 0, 0 );
$vscrollbar->show();

# Sollicite la carte couleur du système et alloue la couleur rouge
$cmap = Gtk : :Gdk : :Colormap->get_system();
$color->{ 'red' } = 0xFFFF;
$color->{ 'green' } = 0;
$color->{ 'blue' } = 0;

warn( "Couldn't allocate color\n" )
  unless ( defined( $cmap->color_alloc( $color ) ) );

# Charge une police fixed
$fixed_font = Gtk : :Gdk : :Font->load("-misc-fixed-medium-r-*-*-*-140-*-*-*-*-*-*");

# Réaliser un widget crée une fenêtre pour lui,
# nous sommes prêts à insérer du texte.
$text->realize();

# Gèle le widget texte afin que l'on puisse faire les mises à jour.
$text->freeze();

# Insère du texte coloré
$text->insert( undef, $text->style->black, undef, "Supports " );
$text->insert( undef, $color, undef, "colored " );
$text->insert( undef, $text->style->black, undef, "text and different " );
$text->insert( $fixed_font, $text->style->black, undef, "fonts\n\n" );

#Charge le fichier text.c dans la fenêtre texte

open( FILE, "text" );

foreach $line ( <FILE> )
  {
    $text->insert( $fixed_font, undef, undef, $line );
  }
```

```
close( FILE ) ;

# Dégel du widget texte ce qui rend les changements visibles
$text->thaw() ;

$hbox = new Gtk : :HButtonBox() ;
$vbox->pack_start( $hbox, $false, $false, 0 ) ;
$hbox->show() ;

$check = new Gtk : :CheckButton( "Editable" ) ;
$hbox->pack_start( $check, $false, $false, 0 ) ;
$check->signal_connect( 'toggled',
                        sub { $text->set_editable( $check->active ) ; } ) ;
$check->set_active( $true ) ;
$check->show() ;

$check = new Gtk : :CheckButton( "Wrap Words" ) ;
$hbox->pack_start( $check, $false, $true, 0 ) ;
$check->signal_connect( 'toggled',
                        sub { $text->set_word_wrap( $check->active ) ; } ) ;
$check->set_active( $false ) ;
$check->show() ;

$separator = new Gtk : :HSeparator() ;
$main_vbox->pack_start( $separator, $false, $true, 0 ) ;
$separator->show() ;

$vbox = new Gtk : :VBox( $false, 10 ) ;
$vbox->border_width( 10 ) ;
$main_vbox->pack_start( $vbox, $false, $true, 0 ) ;
$vbox->show() ;

$button = new Gtk : :Button( "Close" ) ;
$button->signal_connect( 'clicked', sub { Gtk->exit( 0 ) ; } ) ;
$vbox->pack_start( $button, $true, $true, 0 ) ;
$button->can_default( $true ) ;
$button->grab_default() ;
$button->show() ;

$window->show() ;
main Gtk ;
exit( 0 ) ;
```

3.9 Les boutons spin

```
Object
  +--- Widget
       +--- Editable
            +--- Entry
                 +--- SpinButton
```

Les boutons spins sont généralement utilisés pour permettre à l'utilisateur de sélectionner une valeur dans un intervalle de valeurs. Ils consistent en une boite d'entrée texte avec des boutons flèches attachés sur le côté. Cliquer sur l'un des boutons fait varier la valeur sur l'échelle des valeurs disponibles. La boîte entrée peut aussi être éditée directement pour entrer une valeur spécifique.

Le bouton spin permet à la valeur d'être entière ou avec un certain nombre de décimales et d'être incrémentée/décrémentée selon des valeurs configurables. L'action de maintenir un bouton enfoncé peut, optionnellement, se traduire par une accélération de la variation de la valeur en fonction de la durée de la pression.

Le bouton spin utilise les ajustements pour prendre des informations à propos de l'échelle des valeurs que le bouton peut prendre. Rappeler vous que le widget ajustement est créé par la fonction suivante et qui montre les informations qu'elle prend en compte :

```
. $adj = new Gtk : :Adjustment( $value,
                                $lower,
                                $upper,
                                $step_increment,
                                $page_increment,
                                $page_size ) ;
```

Les attributs de l'ajustement sont utilisés par les boutons spins sont :
`$value` : valeur initiale pour le bouton spin
`$lower` : valeur la plus basse
`$upper` : valeur la plus haute
`$step_increment` : valeur d'incrément/décrément en pressant le bouton 1 de la souris
`$page_increment` : valeur d'incrément/décrément en pressant le bouton 2 de la souris
`$page_size` : inutilisé
En plus, le bouton 3 de la souris peut être utilisé pour sauter directement à la valeur `$upper` ou `$lower` quand on l'utilise pour sélectionner l'un des boutons.

3.9.1 Créer un bouton spin

```
$spin = new Gtk : :SpinButton( $adjustment, $climb_rate, $digits ) ;
```

L'argument `$climb_rate` prend une valeur entre 0.0 et 1.0 et indique la quantité d'accélération du bouton spin. L'argument `$digits` spécifie le nombre de décimales que la valeur peut prendre.

3.9.2 Configuration

Un bouton spin peut être reconfiguré après sa création avec :

```
$spin->configure( $adjustment, $climb_rate, $digits ) ;
```

L'ajustement peut être réglé et retrouvé indépendamment un utilisant les deux fonctions :

```
$spin->set_adjustment( $adjustment ) ;
$spin->get_adjustment() ;
```

Le nombre de décimales peut être changé par :

```
$spin->set_digits( $digits ) ;
```

3.9.3 Valeur

La valeur que le bouton spin est en train d'afficher peut être changée par :

```
$spin->set_value( $value ) ;
```

La valeur du bouton spin peut être retrouvée sous la forme d'un nombre décimal ou d'un entier à l'aide de :

```
$spin->get_value_as_float() ;
$spin->get_value_as_int() ;
```

Si vous voulez changer la valeur du bouton spin

```
$spin->spin( $direction, $increment ) ;
```

Le paramètre `$direction` peut prendre les valeurs :

```
'forward'
'backward'
'page_forward'
'page_backward'
'home'
'end'
'user_defined'
```

'forward' et 'backward' change la valeur du bouton spin selon la valeur spécifiée par $increment sauf si $increment vaut 0 auquel cas la valeur est changée à l'aide de la valeur $step_increment de l'ajustement.

'page_forward' et 'page_backward' change simplement la valeur du bouton spin selon $increment.

' home' déclare la valeur à la valeur basse de l'ajustement.

'end' déclare la valeur à la valeur haute de l'ajustement.

'user_defined' change la valeur selon une valeur spécifiée par l'utilisateur.

3.9.4 Comportement

Laissons maintenant les fonctions de réglage et de récupération pour nous intéresser aux fonctions qui agissent sur l'apparence et sur le comportement du bouton spin.

La première de ces fonctions est utilisée pour contraindre la boîte texte du bouton à ne contenir qu'une valeur numérique. Cela empêche l'utilisateur de taper autre chose qu'une valeur numérique.

```
$spin->set_numeric( $numeric ) ;
```

Vous pouvez indiquer si le bouton est compris entre la valeur la plus basse et la plus haute.

```
$spin->set_wrap( $wrap ) ;
```

Vous pouvez obliger le bouton à arrondir la valeur au plus proche step_increment qui est déclarée à l'intérieur de l'objet ajustement utilisé par le bouton spin :

```
$spin->set_snap_to_ticks( $snap_to_ticks ) ;
```

Le comportement du bouton peut être modifié par :

```
$spin->set_update_policy( $policy ) ;
```

Les valeurs possibles de $policy sont soit 'always' soit 'if_valid'. Ces valeurs affectent le comportement du bouton quand l'utilisateur écrit la valeur. Dans le cas de 'if_valid', la valeur du bouton spin ne change que si le texte d'entrée est une valeur numérique comprise entre les valeurs spécifiées par l'ajustement. Autrement le texte est effacé et remplacé par la valeur courante. Dans le cas de 'update_always', on ignore les erreurs en convertissant le texte en valeur numérique.

L'apparence du bouton spin peut être changé avec :

```
$spin->set_shadow_type( $shadow_type ) ;
```

Comme d'habitude, le $shadow_type peut être :

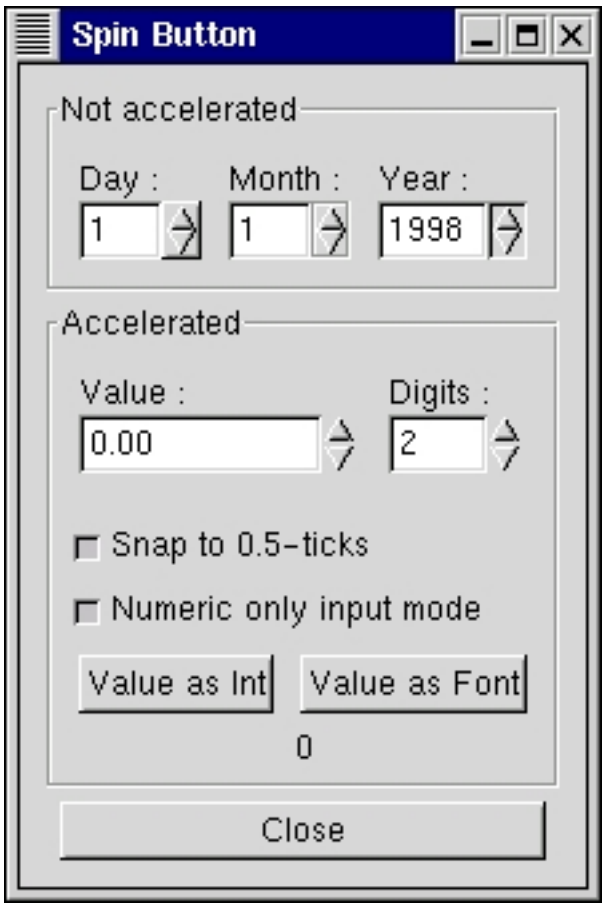
```
'in'
'out'
'etched_in'
'etched_out'
```

Vous pouvez demander que le bouton se mette à jour lui-même :

```
$spin->update() ;
```

3.9.5 Exemple

Il est l'heure de l'exemple.




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $spinner1;
my $window;
my $frame;
my $hbox;
my $main_vbox;
my $vbox;
my $vbox2;
my $spinner2;
my $spinner;
my $button;
my $label;
my $val_label;
my $adj;

$window = new Gtk : :Window( "toplevel" );
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$window->set_title( "Spin Button" );

$main_vbox = new Gtk : :VBox( $false, 5 );
```

```
$main_vbox->border_width( 10 ) ;
$window->add( $main_vbox ) ;

$frame = new Gtk : :Frame( "Not accelerated" ) ;
$main_vbox->pack_start( $frame, $true, $true, 0 ) ;

$vbox = new Gtk : :VBox( $false, 0 ) ;
$vbox->border_width( 5 ) ;
$frame->add( $vbox ) ;

# Jour, mois, année spinners
$hbox = new Gtk : :HBox( $false, 0 ) ;
$vbox->pack_start( $hbox, $true, $true, 5 ) ;

$vbox2 = new Gtk : :VBox( $false, 0 ) ;
$hbox->pack_start( $vbox2, $true, $true, 5 ) ;

$label = new Gtk : :Label( "Day :" ) ;
$label->set_alignment( 0, 0.5 ) ;
$vbox2->pack_start( $label, $false, $true, 0 ) ;

$adj = new Gtk : :Adjustment( 1.0, 1.0, 31.0, 1.0, 5.0, 0.0 ) ;
$spinner = new Gtk : :SpinButton( $adj, 0, 0 ) ;
$spinner->set_wrap( $true ) ;
$spinner->set_shadow_type( 'out' ) ;
$vbox2->pack_start( $spinner, $false, $true, 0 ) ;

$vbox2 = new Gtk : :VBox( $false, 0 ) ;
$hbox->pack_start( $vbox2, $true, $true, 5 ) ;

$label = new Gtk : :Label( "Month :" ) ;
$label->set_alignment( 0, 0.5 ) ;
$vbox2->pack_start( $label, $false, $true, 0 ) ;

$adj = new Gtk : :Adjustment( 1.0, 1.0, 12.0, 1.0, 5.0, 0.0 ) ;

$spinner = new Gtk : :SpinButton( $adj, 0, 0 ) ;
$spinner->set_wrap( $true ) ;
$spinner->set_shadow_type( 'etched_in' ) ;
$vbox2->pack_start( $spinner, $false, $true, 0 ) ;

$vbox2 = new Gtk : :VBox( $false, 0 ) ;
$hbox->pack_start( $vbox2, $true, $true, 5 ) ;

$label = new Gtk : :Label( "Year :" ) ;
$label->set_alignment( 0, 0.5 ) ;
$vbox2->pack_start( $label, $false, $true, 0 ) ;

$adj = new Gtk : :Adjustment( 1998.0, 0.0, 2100.0, 1.0, 100.0, 0.0 ) ;
$spinner = new Gtk : :SpinButton( $adj, 0, 0 ) ;
$spinner->set_wrap( $false ) ;
$spinner->set_shadow_type( 'in' ) ;
$spinner->set_usize( 55, 0 ) ;
$vbox2->pack_start( $spinner, $false, $true, 0 ) ;

$frame = new Gtk : :Frame( "Accelerated" ) ;
$main_vbox->pack_start( $frame, $true, $true, 0 ) ;
```

```
$vbox = new Gtk : :VBox( $false, 0 ) ;
$vbox->border_width( 5 ) ;
$frame->add( $vbox ) ;

$hbox = new Gtk : :HBox( $false, 0 ) ;
$vbox->pack_start( $hbox, $false, $true, 5 ) ;

$vbox2 = new Gtk : :VBox( $false, 0 ) ;
$hbox->pack_start( $vbox2, $true, $true, 5 ) ;

$label = new Gtk : :Label( "Value :" ) ;
$label->set_alignment( 0, 0.5 ) ;
$vbox2->pack_start( $label, $false, $true, 0 ) ;

$adj = new Gtk : :Adjustment( 0.0, -10000.0, 10000.0, 0.5, 100.0, 0.0 ) ;
$spinner1 = new Gtk : :SpinButton( $adj, 1.0, 2 ) ;
$spinner1->set_wrap( $true ) ;
$spinner1->set_usize( 100, 0 ) ;
$vbox2->pack_start( $spinner1, $false, $true, 0 ) ;

$vbox2 = new Gtk : :VBox( $false, 0 ) ;
$hbox->pack_start( $vbox2, $true, $true, 5 ) ;

$label = new Gtk : :Label( "Digits :" ) ;
$label->set_alignment( 0, 0.5 ) ;
$vbox2->pack_start( $label, $false, $true, 0 ) ;

$adj = new Gtk : :Adjustment( 2, 1, 5, 1, 1, 0 ) ;
$spinner2 = new Gtk : :SpinButton( $adj, 0.0, 0 ) ;
$spinner2->set_wrap( $true ) ;
$adj->signal_connect( "value_changed", \&change_digits, $spinner2 ) ;
$vbox2->pack_start( $spinner2, $false, $true, 0 ) ;

$hbox = new Gtk : :HBox( $false, 0 ) ;
$vbox->pack_start( $hbox, $false, $true, 5 ) ;

$button = new Gtk : :CheckButton( "Snap to 0.5-ticks" ) ;
$button->signal_connect( "clicked", \&toggle_snap, $spinner1 ) ;
$vbox->pack_start( $button, $true, $true, 0 ) ;
$button->set_active( $true ) ;

$button = new Gtk : :CheckButton( "Numeric only input mode" ) ;
$button->signal_connect( "clicked", \&toggle_numeric, $spinner1 ) ;
$vbox->pack_start( $button, $true, $true, 0 ) ;
$button->set_active( $true ) ;

$val_label = new Gtk : :Label( "" ) ;

$hbox = new Gtk : :HBox( $false, 0 ) ;

$vbox->pack_start( $hbox, $false, $true, 5 ) ;
$button = new Gtk : :Button( "Value as Int" ) ;
$button->set_user_data( $val_label ) ;
$button->signal_connect( "clicked", \&get_value, $spinner1, 1 ) ;
$hbox->pack_start( $button, $true, $true, 5 ) ;

$button = new Gtk : :Button( "Value as Font" ) ;
$button->set_user_data( $val_label ) ;
```

```
$button->signal_connect( "clicked", \&get_value, $spinner1, 2 ) ;
$hbox->pack_start( $button, $true, $true, 5 ) ;

$vbox->pack_start( $val_label, $true, $true, 0 ) ;
$val_label->set_text( "0" ) ;

$hbox = new Gtk : :HBox( $false, 0 ) ;
$main_vbox->pack_start( $hbox, $false, $true, 0 ) ;

$button = new Gtk : :Button( "Close" ) ;
$button->signal_connect( "clicked", sub { Gtk->exit( 0 ) ; } ) ;
$hbox->pack_start( $button, $true, $true, 5 ) ;

$window->show_all() ;

main Gtk ;
exit( 0 ) ;



### Routines

sub toggle_snap
  {
    my ( $widget, $spin ) = @_ ;

    $spin->set_snap_to_ticks( $widget->active ) ;
  }

sub toggle_numeric
  {
    my ( $widget, $spin ) = @_ ;

    $spin->set_numeric( $widget->active ) ;
  }

sub change_digits
  {
    my ( $widget, $spin ) = @_ ;

    $spinner1->set_digits( $spin->get_value_as_int() ) ;
  }

sub get_value
  {
    my ( $widget, $spin, $num ) = @_ ;

    my $buf = "" ;
    my $label ;

    $label = new Gtk : :Label( $widget->get_user_data() ) ;

    if ( $num == 1 )
      {
        $buf = $spin->get_value_as_int() ;
      }
    else
      {
```

```
            $buf = $spin->get_value_as_float() ;
          }

        $val_label->set_text( $buf ) ;
      }
```

3.10 Les menus

Ce qui est considéré comme un menu par les utilisateurs est en fait composé de trois parties. Tout d'abord, il y a la barre de menus en haut de la fenêtre, puis à l'intérieur de la barre de menus, il y a plusieurs menus, enfin, chaque menu possède en général un ou plusieurs sous-menus.

3.10.1 Le menuShell

```
Object
  +--- Widget
       +--- Container
            +--- MenuShell
```

Les widgets Barre de Menu et Menu sont des sous-classes de MenuShell. Chaque MenuShell a la capacité d'ajouter à la fin, au début ou d'insérer un widget enfant.

```
$menushell-> ;append( $child ) ;
$menushell->prepend( $child ) ;
$menushell->insert( $child, $position ) ;
```

Le MenuShell peut-être effacé de l'écran par :

```
$menushell->deactivate() ;
```

Un enfant du MenuShell peut-être sélectionné par :

```
$menushell->select_item( $child ) ;
```

Un enfant du MenuShell peut-être activé par :

```
$menushell->activate_item( $child ) ;
```

3.10.2 La Barre de Menus

```
Object
  +--- Widget
       +--- Container
            +--- MenuShell
                 +--- MenuBar
```

Une barre de menu est créée par :

```
$menubar = new Gtk : :MenuBar() ;
```

Les barres de menus sont habituellement placées en haut de la fenêtre. Normalement, vous devriez créer une VBox et l'ajouter à la fenêtre. Vous devriez ensuite placer le barre de menu en haut de la boîte verticale. La seule autre fonction concernant les barres de menus (et qui ne proviennent pas du MenuShell) déclare le type de l'ombre :

```
$menubar->set_shadow_type( $shadow_type ) ;
```

Ou `$shadow_type` est soit :

```
'none'
'in'
'out'
'etched_in'
'etched_out'
```

3.10.3 Les Menus

```
Object
  +--- Widget
       +--- Container
            +--- MenuShell
                 +--- Menu
```

Les menus sont créés par :

```
$menu = new Gtk : :Menu() ;
```

Si vous voulez bouger un enfant d'une position à l'autre, vous pouvez le faire avec :

```
$menu->reorder_child( $child, $position ) ;
```

`$child` est l'enfant à bouger et `$position` est la nouvelle position.

Les menus peuvent avoir un titre associé. Ce titre est déclaré par :

```
$menu->set_title( $title ) ;
```

L'état tear-off peut-être déclaré par ;

```
$menu->set_tearoff_state( $tear_off ) ;
```

`$tear_off` est une valeur vraie ou fausse qui détermine si le menu peut être à l'état tear-off. Un menu est normalement affiché comme un cadre qui descend et qui persiste tant que le menu est actif, alors que le menu tear-off est dans une fenêtre qui persiste jusqu'à ce que la fenêtre soit fermée.

Enfin, si vous voulez que votre menu se comporte comme un menu popup, vous devrez appeler :

```
$menu->popup( $parent_menu_shell,
              $parent_menu_item,
              $activate_time,
              $button,
              \&menu_position_function,
              @optional_data ) ;
```

Ou `$parent_menu_shell` et `$parent_menu_item` seront indéfinis si le menu n'est pas associé à une barre de menus. Pour `$activate_time`, vous pouvez habituellement vous en débarrasser à l'aide de `$event->{ 'time' }`. Si vous désirez que le menu disparaisse quand vous relâchez le bouton de la souris, vous devrez mettre `1`.

L'argument `$button` proviendra souvent de `$event->{ 'button' }`.

Enfin, `\&menu_position_function` est une référence à une fonction fournit par l'utilisateur, utilisée pour positionner le menu. La plupart des gens utilise `undef` qui définit la position en haut à gauche du menu à l'endroit du clic de la souris. `@optional_data`, s'il est inclus, est envoyé comme argument de `\&menu_position_function`.

Attendez-vous à ce que cette fonction change bientôt. On suppose qu'il y aura une interversion de `$button` et de `$activate_time` comme ceci :

```
$menu->popup( $parent_menu_shell,
              $parent_menu_item,
              $button,
              $activate_time,
              \&menu_position_function,
              @optional_data ) ;
```

3.10.4 Les Items de Menu

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Item
                      +--- MenuItem
```

Créer des items de menus est aussi simples que d'appeler l'une des fonctions suivantes :

```
$menuitem = new Gtk : :MenuItem() ;
$menuitem = new Gtk : :MenuItem( $label ) ;
```

Vous pouvez sélectionner, déselectionner et activer un item de menu à l'aide de :

```
$menuitem->select() ;
$menuitem->deselect() ;
$menuitem->activate() ;
```

Si vous voulez faire d'un item de menu un sous-menu, vous pouvez utiliser :

```
$menuitem->set_submenu( $menu ) ;
```

Vous pouvez aussi enlever un sous-menu :

```
$menuitem->remove_submenu() ;
```

Enfin, si vous voulez que le menu soit justifié à droite dans la barre de menu, comme le sont souvent les menus d'aide, vous pouvez utiliser la fonction suivante avant de l'attacher à la barre de menus.

```
$menu_item->right_justify() ;
```

3.10.5 Créer des menus

Il y a deux manières de créer des menus : la facile et la difficile. Chacune a son usage mais vous pouvez habituellement utiliser l'usine à items (ItemFactory, méthode facile). La manière "difficile" est de créer directement tous les menus en utilisant les fonctions précédentes. La manière "facile" consiste à utiliser les appels de l'usine à items. C'est plus simple mais il y a des avantages et des inconvénients à chacune des approches.

L'usine à items est bien plus facile à utiliser et on peut y ajouter de nouveaux menus bien que d'écrire quelques fonctions en utilisant la méthode manuelle puisse être d'un grand secours. Avec l'usine à items, il n'est pas possible d'ajouter des images ou des caractères "/" aux menus.

Par pure tradition pédagogique, nous vous montrerons d'abord la méthode difficile. Le widget ItemFactory étant décrit ultérieurement.

Trois choses sont nécessaires pour créer une barre de menus et des sous-menus :
- un item menu que l'utilisateur veut sélectionner, par exemple "sauver".
- un menu qui contient les items.
- une barre de menus qui contient les menus individuels.

C'est légèrement compliqué du fait que le widget item de menu est utilisé pour deux choses différentes. Il y a à la fois le widget regroupé dans le menus et le widget regroupé dans la barre de menus, qui sélectionné ouvre le menu.

Quand un menu est créé, il n'est en fait jamais montré (avec la fonction `show()`), c'est juste un conteneur pour les items de menus. J'espère que ceci deviendra plus clair en regardant les exemples ci-dessous.

Une fois que vous avez créé votre item de menu, vous devez le mettre dans un menu. On y parvient avec la fonction `append()` . Afin de capturer le moment ou l'item de menu est sélectionné par l'utilisateur, nous aurons besoin de connecter le signal `'activate'`. Ainsi, si vous voulez créer un menu Fichier standard avec les fonctions Ouvrir, Sauver et Quitter, le code ressemblera à :

```
$fichier_menu = new Gtk : :Menu() ;

# Créer les items du menu
$item_ouvrir = new Gtk : :MenuItem( "Ouvrir" ) ;
$item_sauver = new Gtk : :MenuItem( "Sauver" ) ;
$item_quitter = new Gtk : :MenuItem( "Quitter" ) ;

# Les ajouter au menu
$fichier_menu->append( $item_ouvrir ) ;
$fichier_menu->append( $item_sauver ) ;
$fichier_menu->append( $item _quitter) ;

# Attacher les fonctions de rappels au signal activate
$item_ouvrir->signal_connect( 'activate', \&ouvrir_fichier ) ;
$item_sauver->signal_connect( 'activate', \&sauver_fichier ) ;
$item_quitter->signal_connect( 'activate', \&quitter_fichier ) ;

# Nous n'avons pas besoin de montrer le menu mais nous devons montrer les items de menus
$item_ouvrir->show() ;
$item_sauver->show() ;
$item_quitter->show() ;
```

A ce moment, nous avons notre menu. Maintenant nous créons notre barre de menus et notre item menu pour l'entrée Fichier auquel nous ajoutons notre menu. On obtient :

```
$menu_bar = new Gtk : :MenuBar() ;
$window->add( $menu_bar ) ;
$menu_bar->show() ;

$item_fichier = new Gtk : :MenuItem( "Fichier" ) ;
$item_fichier->show() ;
```

Maintenant, on a besoin d'associer notre menu avec `$fichier_menu`. Cela est fait avec la fonction `set_submenu()`. Tout ce qui reste à faire est d'ajouter le menu à la barre de menus, ce qui est accompli avec le fonction `append()`.

Voici un résumé des étapes nécessaires à la création d'une barre de menus avec des menus attachés :

- créer un nouveau menu avec `new Gtk::Menu()`.
- utiliser de multiples appels `new Gtk::MenuItem()` pour chaque item que vous voulez voir apparaître au menu. Puis utiliser `append()` pour placer chacun de ces items au menu.
- créer un item menu avec `new Gtk::MenuItem()`. Il s'agira de la racine du menu, le texte apparaissant ici sera dans la barre de menus elle-même.
- utiliser `set_submenu()` pour attacher le menu à la racine de l'item menu (créé à l'étape précédente).
- créer une nouvelle barre de menus avec `new Gtk::MenuBar()`. Cette étape ne doit être faite que quand on a une série de menus à placer dans la barre de menus.
- utiliser `append()` pour placer le menu racine dans la barre de menu.

Créer une menu popup se fait de la même manière. La différence est que le menu n'est pas placé "automatiquement" par la barre de menus mais explicitement en appelant la fonction `popup()` d'un évènement bouton pressé, par exemple. Les étapes sont donc :

- créer une fonction qui contrôle un évènement pour savoir où placer le menu.
- au niveau du gestionnaire d'évènement, si l'évènement est un bouton de souris pressé, traiter `event` comme un évènement bouton (ce qu'il est !) et utiliser le comme dans l'exemple suivant pour passer l'information à la fonction `popup()`.
- lier ce gestionnaire d'évènement à un widget à l'aide de `signal_connect()`.

3.10.6 Exemple

Regardons un petit exemple pour essayer de nous clarifier les idées.


```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $i ;

my $window ;
```

```
my $menu ;
my $menubar ;
my $root_menu ;
my $menu_item ;
my $vbox ;
my $button ;

# Crée une nouvelle fenêtre
$window = new Gtk : :Window( 'toplevel' ) ;
$window->set_usize( 200, 100 ) ;
$window->set_title( "GTK Menu Test" ) ;
$window->signal_connect( 'delete_event', sub { Gtk->exit( 0 ) ; } ) ;

# Initialise le widget menu ,et rappelez-vous : ne jamais montrer
# le widget menu ! !
# C'est le menu qui contient les items du menus, celui qui apparaîtra quand
# vous cliquerez sur le "Root Menu" de l'aplication
$menu = new Gtk : :Menu() ;

# Ensuite, on fait une petite boucle qui crée trois entrées menus pour le
# ''test-menu''. Noter l'appel à append(). Là, nous ajoutons une liste d'items
# de menus à notre menu. Normalement, nous devrions aussi attraper le signal
# ''cliqué'' sur chacun des items du menu et installer un rappel pour lui,
# mais on l'omet ici pour gagner de la place.

for ( $i = 0 ; $i < 3 ; $i++ )
  {
    my $buffer ;

    # Copie les noms dans le buffer
    $buffer = "Test-undermenu - $i" ;

    # Crée un nouveau item menu avec un nom...
    $menu_item = new Gtk : :MenuItem( $buffer ) ;

    # et l'ajoute au menu
    $menu->append( $menu_item ) ;

    # Fait quelque chose d'intéressant quand le menu est sélectionné.
    $menu_item->signal_connect( 'activate', sub { print( "$buffer\n" ) ; } ) ;

    #Montre le widget
    $menu_item->show() ;
  }

# Il s'agit du menu Root, et ce sera le label affiché sur la barre de menus.
# Il n'y aura pas de gestionnaire attaché, comme il n'affiche le reste du menu
# que quand il est pressé
$root_menu = new Gtk : :MenuItem( "Root Menu" ) ;
$root_menu->show() ;

# Maintenant, nous spécifions que nous voulons que notre ''menu'' nouvellement
# créé soit le menu du "root menu"
$root_menu->set_submenu( $menu ) ;

# Une vbox pour y placer un menu et un bouton :
$vbox = new Gtk : :VBox( $false, 0 ) ;
$window->add( $vbox ) ;
```

```
$vbox->show() ;

# Crée une barre de menus qui contient nos menus et l'ajoute à notre fenêtre
$menubar = new Gtk : :MenuBar() ;
$vbox->pack_start( $menubar, $false, $false, 2 ) ;
$menubar->show() ;

# Crée un bouton auquel attacher le menu en tant que popup
$button = new Gtk : :Button( "Press Me" ) ;
$button->signal_connect( 'event', \&button_press, $menu ) ;
$vbox->pack_end( $button, $true, $true, 2 ) ;
$button->show() ;

# Enfin, nous ajoutons les items du menu à la barre de menus -- c'est
# l'item de menu ``root''  à propos duquel je m'excitais tout à l'heure :-)
$menubar->append( $root_menu ) ;

# Toujours afficher la fenêtre en dernier pour qu'elle apparaisse d'un coup
# sur l'écran
$window->show() ;

main Gtk ;
exit( 0 ) ;




### Routines

# Réponds au bouton pressé en plaçant un menu passé comme widget.
# Noter que l'argument ``widget'' est le menu placé,
# et NON le bouton qui fût pressé.
sub button_press
  {
    my ( $button, $menu, $event ) = @_ ;

    if ( defined( $event->{ 'type' } ) and
         ( $event->{ 'type' } eq 'button_press' ) )
      {
        $menu->popup( undef, undef, $event->{'time'}, $event->{'button'}, undef ) ;

        # Dit au code appelant que nous avons capté cet évènement, le lièvre
        # s'arrête là.
        return ( $true ) ;
      }

    # Dit au code appelant que nous n'avons pas capté cet évènement, continuez.
    return ( $false ) ;
  }
```

3.11 Utiliser l'usine à items

```
Object
  +--- ItemFactory
```

Pour le moment, il n'y a que cet exemple. Une explication et des commentaires suivront plus tard.

```
#!/usr/bin/perl -w
```

```
use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

#Voici la structure GtkItemFactoryEntry utilisée pour générer de nouveaux
# menus.
# Item 1 : L'arborescence du menu. La lettre après le caractère ``_'' indique
#         un raccourci clavier une fois que le menu est ouvert.
# Item 2 : Le raccourci clavier pour l'entrée.
# Item 3 : La fonction de rappel.
# Item 4 : L'action du rappel.  Cela change les paramètres avec
#         ceux de la fonction appelée. La valeur par défaut est 0.
# Item 5 : Le type d'item, utilisé pour savoir de quel type d'item l s'agit.
#         Voici les valeurs possibles.
#          NULL               -> "&lt ;Item>"
#          ""                 -> "<Item>"
#          "<Title>"          -> crée un item titre
#          "<Item>"           -> crée un simple item
#          "<CheckItem>"      -> crée un item à cocher
#          "<ToggleItem>"     -> crée un item toggle
#          "<RadioItem>"      -> crée un item radio
#          <path>             -> endroit où relier l'item radio
#          "<Separator>"      -> crée un séparateur
#          "<Branch>"         -> crée un item qui contient des
#                                   sous-items (optionel)
#          "<LastBranch>"     -> crée une branche justifiée à droite

my @menu_items = ( { path        => '/_File',
                     type        => '<Branch>' },
                   { path        => '/File/_New',
                     accelerator => '<control>N',
                     callback    => \&print_hello },
                   { path        => '/File/_Open',
                     accelerator => '<control>O',
                     callback    => \&print_hello },
                   { path        => '/File/_Save',
                     accelerator => '<control>S',
                     callback    => \&print_hello },
                   { path        => '/File/Save _As' },
                   { path        => '/File/sep1',
                     type        => '<Separator>' },
                   { path        => '/File/Quit',
                     callback    => sub { Gtk->exit( 0 ) ; } },

                   { path        => '/_Options',
                     type        => '<Branch>' },
                   { path        => '/Options/Test' },

                   { path        => '/_Help',
                     type        => '<LastBranch>' },
                   { path        => '/_Help/About' } ) ;

my $window ;
my $main_vbox ;
```

```
my $menubar;

$window = new Gtk : :Window( 'toplevel' );
$window->signal_connect( 'destroy', sub { Gtk->exit( 0 ); } );
$window->set_title( "Item Factory" );
$window->set_usize( 300, 200 );

$main_vbox = new Gtk : :VBox( $false, 1 );
$main_vbox->border_width( 1 );
$window->add( $main_vbox );
$main_vbox->show();

$menubar = get_main_menu( $window );
$main_vbox->pack_start( $menubar, $false, $true, 0 );
$menubar->show();

$window->show();
main Gtk;
exit( 0 );




### Routines

# Rappel de base obligatoire
sub print_hello
  {
    print( "Hello World!\n" );
  }

sub get_main_menu
  {
    my ( $window ) = @_;

    my $menubar;
    my $item_factory;
    my $accel_group;

    $accel_group = new Gtk : :AccelGroup();

    # Cette fonction initialise l'usine à items.
    # Param 1 : Le type de menu - peut être 'Gtk : :MenuBar', 'Gtk : :Menu',
    #          ou 'Gtk : :OptionMenu'.
    # Param 2 :L'arborescence du menu.
    # Param 3 : Le groupe de raccourcis.  L'usine à items crée la table
    #         des raccourcis en générant les menus.
    $item_factory = new Gtk : :ItemFactory( 'Gtk : :MenuBar',
                                            '<main>',
                                            $accel_group );

    # Cette fonction génère les items du menus. Passe à l'usine à items,
    # le nombre d'items dans l'alignement, l'alignement lui-même, et
    # des données pour les rappels des menus items.
    $item_factory->create_items( @menu_items );

    # Attache le groupe des raccourcis à la fenêtre.
    $window->add_accel_group( $accel_group );
```

```
  # Enfin, retourne l'actuelle barre de menu créée par l'usine à items.
  #*menubar = gtk_item_factory_get_widget (item_factory, "< ;main>") ;
  return ( $item_factory->get_widget( '<main>' ) ) ;
}
```

3.12 Le menu à options

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Button
                      +--- OptionMenu
```

Le menu à options est un widget qui permet à l'utilisateur de sélectionner à partir d'un ensemble de choix. Le choix courant est montré dans le menu à options, et si l'utilisateur clique dessus, un menu popup avec tous les choix valides s'affiche.

Ce simple widget est créé par :

```
$option = new Gtk : :OptionMenu() ;
```

Pour ajouter un menu, créer un nouveau menu popup comme décrit dans la section sur les widgets, sans tear-off, sous-menus, ou raccourcis, et l'ajouter au menu à options, on utilise :

```
$option->set_menu( $popup_menu ) ;
```

Comme vous pouvez l'imaginer, vous pouvez retrouver le menu associé au menu à options avec :

```
$popup_menu = $option->get_menu() ;
```

Si vous voulez enlever un menu d'un menu à options, vous devez utiliser :

```
$option->remove_menu() ;
```

Enfin, si vous voulez sélectionner un item dans le menu, vous devez utiliser :

```
$option->set_history( $index ) ;
```

Ou `$index` est l'index de l'item du menu à sélectionner. La valeur de l'index classe les items de 0 (pour le premier) à n-1 (pour le dernier), ou n est le nombre d'items contenus dans le menu.

3.13 Les listes de choix (Combo widgets)

```
Object
  +--- Widget
       +--- Container
            +--- Box
                 +--- HBox
                      +--- Combo
```

Une liste de choix est un widget plutôt simple qui n'est qu'en fait qu'une collection d'autres widgets. Du point de vue de l'utilisateur, le widget consiste en une entrée texte avec un menu déroulant dans lequel l'utilisateur peut choisir une entrée prédéfinie. Par ailleurs, l'utilisateur peut taper un texte différent dans la boîte.

L'extrait suivant de la structure qui définie une liste de choix identifie plusieurs de ses composants :

```
Gtk : :Combo
          -> hbox
          -> entry
          -> button
          -> popup
          -> popwin
          -> list
          .
          .
          .
```

Comme vous pouvez le voir, une liste de choix est composée de deux parties : une entrée texte et une liste.

Tout d'abord, pour créer une liste de choix :

```
$combo = new Gtk : :Combo() ;
```

Maintenant, si vous voulez déclarer une chaîne de caractères dans l'entrée texte de la liste de choix, cela se fait en manipulant directement le widget Entrée :

```
$combo->entry->set_text( "Ma chaîne." ) ;
```

Pour déclarer les valeurs de la liste déroulante, utiliser la fonction :

```
$combo->set_popdown_strings( @strings ) ;
```

Avant de faire ça, vous devez créer la liste des chaînes de caractères que vous voulez. N'importe quelle expression Perl, qui peut être considérée comme une liste peut être utilisée en argument. Les liste encadrées par une apostrophe et celles encadrées par des guillemets sont les plus communes. Pour en ajouter à la liste, utiliser :

```
$combo->list = ( $combo->list, "ajout1", "ajout2", ... ) ;
```

Comme il est suggéré précédemment, tout ce qu'on peut faire à une liste Perl (ce qui est beaucoup), on peut le faire à une liste de choix. Voici un bout de code typique pour créer un ensemble d'options :

```
@list = ( "Chaîne1", "Chaîne2", "Chaîne3" ) ;
$combo->set_popdown_strings( @list ) ;
```

Ou, si vous préférez compresser le code :

```
$combo->set_popdown_strings( "Chaîne1", "Chaîne2", "Chaîne3" ) ;
```

A ce moment, vous avez créé une liste de choix qui fonctionne. Il y a quelques aspects de son comportement que vous pouvez changer. Ils sont accomplis avec les fonctions :

```
$combo->set_use_arrows( $use_arrows ) ;
```

Ou `$use_arrows` est une valeur vraie ou fausse. Cela ne mentionne pas la liste mais plutôt remplace le contenu actuel de l'entrée texte avec l'élément suivant de la liste (précèdent ou suivant en fonction des flèches qui vous avez utilisées au clavier). On fait cela en cherchant dans la liste l'item correspondant à l'entrée texte et en sélectionnant l'item précédent/suivant. Habituellement, les flèches sont utilisées pour changer l'élément actif (vous pouvez toujours le faire avec TAB). Noter que quand l'item courant est le dernier de la liste, quand vous pressez la flèche vers le bas, vous changez l'élément actif (de même s'il s'agit du premier élément).

Si la valeur courante de l'entrée texte n'est pas dans la liste, la fonction `set_use_arrows()` est indisponible.

Si vous voulez que les flèches haut/bas tournent en rond dans l'ensemble des choix, si ce n'est que cela limite les valeurs de la liste, utilisez :

```
$combo->set_use_arrows_always( $use_arrows ) ;
```

Cela rendra complètement indisponible l'utilisation des flèches pour changer l'élément actif.

Pour déterminer si GTK cherche les entrées en prenant en compte les différentes formes d'un même mot ou non, utiliser :

```
$combo->set_case_sensitive( $case ) ;
```

On l'utilise quand la liste de choix doit trouver une valeur dans la liste à partir de l'entrée texte. Ce complément peut se faire de manière sensible ou insensible, en fonction de l'usage de cette fonction. La liste de choix peut aussi compléter simplement l'entrée courant si l'utilisateur utilise la combinaison de touche MOD-1 et "TAB". MOD-1 est souvent la touche "Alt" selon `xmodmap`. Noter toutefois que certains windows managers utilise également cette combinaison ce qui supprime son usage pour GTK.

Si vous voulez spécifier si la valeur entrée doit ou non être l'un des éléments de la liste, vous pouvez le faire avec :

```
$combo->set_value_in_list( $must_match, $ok_if_empty ) ;
```

Si `$must_match` est une valeur vraie alors la valeur entrée doit être dans la liste. `$ok_if_empty` est une valeur vraie ou fausse qui détermine si une valeur vide est acceptable.

Si l'un des éléments de la liste n'est pas un simple label, vous devez dire à GTK de quel label il s'agit. Vous utilisez alors :

```
$combo->set_item_string( $item, $Chaîne ) ;
```

`$item` est l'élément de la liste que vous voulez déclarer et `$Cha\^{\i}ne` est le label. Sans cela, GTK ne saura pas quoi mettre dans l'entrée texte.

Vous pouvez empêcher la liste de choix de montrer la liste déroulante quand l'entrée texte émet un signal "activé" en utilisant :

```
$combo->disable_activate() ;
```

Cela peut être utile si vous voulez que la touche “Entrée” termine le dialogue. Dans la plupart des cas cependant, vous devriez le laisser se débrouiller tout seul.

Maintenant que nous avons une liste de choix, qui ressemble à quelque chose et qui se comporte comme on le veut, tout ce qui reste à savoir est comment obtenir les informations de la liste de choix. C'est relativement direct. La plupart du temps, on ne se préoccupe que de l'entrée texte. On y accède simplement avec `combo->entry`. Les deux principales choses que vous voudrez en faire seront d'y attacher un signal “activé” qui indiquera que le touche “Entrée” a été pressée, et de lire le texte. La première étape est réalisée par :

```
$combo->entry->signal_connect( "activate", \&callback ) ;
```

Lire le texte à n'importe quel moment est tout simplement fait par le fonction :

```
$combo->entry->get_text() ;
```

3.14 Le widget Liste

```
Object
  +--- Widget
       +--- Container
            +--- List
```

3.14.1 Le widget Liste

Le widget Liste est conçu pour agir comme un conteneur vertical pour des widgets qui devraient être du type élément de liste (ListItem). Un widget Liste possède sa propre fenêtre pour recevoir les évènements et sa propre couleur de fond qui est généralement blanche.

Afin de ne pas vous perturber, une Liste (majuscule) fait référence à un widget Gtk : :List alors qu'une liste (minuscule) fait référence à une liste Perl (et ce qui peut aussi y faire référence, quoique de manière grossière, comme les rangées).

Il y a deux champs dans la définition d'une structure d'un widget Liste qui seront d'un grand intérêt pour nous : Il y a `selection` et `selection_mode`.

La champs `selection` d'une Liste est une liste de tous les éléments qui sont actuellement sélectionnés ou une liste vide si la sélection est vide. Ainsi pour en apprendre plus sur la sélection courante, nous lisons le champs `selection` mais nous ne le modifions pas car les champs internes sont maintenus par les fonctions de Liste.

Le champs `selection_mode` d'une Liste détermine les facilités de sélection d'une Liste et par conséquent les contenus du champs de sélection. Le `selection_mode` peut être :

`'single'` - La sélection est soit vide soit sa liste ne contient qu'un seul élément sélectionné.

`'browse'` - La sélection est vide si la liste ne contient aucun widget ou seulement des insensibles, autrement il contient une liste avec un élément.

`'multiple'` - La sélection est vide si aucun des éléments n'est sélectionné ou bien une liste de tous les items sélectionnés.

`'extended'` - La sélection est toujours vide.

Le défaut est `'multiple'`.

3.14.2 Les signaux

Le signal `'selection changed'` sera invoqué si le champs de `selection` de la Liste a changé. Cela arrive quand un enfant de la Liste est sélectionné ou désélectionné.

Le signal `'select_child'` est invoqué un enfant de la Liste est sur le point d'être sélectionné. Cela arrive principalement quand on appelle `select_item()`, `select_child()`, quand on appuie sur les boutons et parfois c'est déclenché indirectement en d'autres occasions quand des enfants sont ajoutés ou enlevés de la Liste.

Le signal `'unselect_child'` est invoqué quand un enfant de la Liste est sur le point d'être désélectionné. Cela arrive principalement quand on appelle `unselect_item()`, `unselect_child()`, quand on appuie sur les boutons et parfois c'est déclenché indirectement en d'autres occasions quand des enfants sont ajoutés ou enlevés de la Liste.

3.14.3 Les fonctions

Pour créer une Liste :

```
$list = new Gtk : :List() ;
```

Les éléments sont ajoutés à la Liste en utilisant :

```
$list->insert_items( @items, $position ) ;
```

Ou `@items` est la liste des éléments à ajouter et `$position` est la position de départ des éléments.

On peut ajouter des éléments à la fin de la Liste ou au début avec les fonctions :

```
$list->append_items( @items ) ;
$list->prepend_items( @items ) ;
```

Pour enlever des éléments de la Liste :

```
$list->remove_items( @items ) ;
```

Ou `@items` est la liste des éléments à enlever.

Vous pouvez également enlever un ensemble d'éléments avec :

```
$list->clear_items( $start, $end ) ;
```

Les éléments peuvent être sélectionnés par :

```
$list->select_item( $index ) ;
$list->select_child( $child ) ;
```

Cela invoquera également soit le signal `'select_item'` soit le signal `'select_child'` sur l'élément de la liste.

On désélectionne de la même manière :

```
$list->unselect_item( $index ) ;
$list->unselect_child( $child ) ;
```

Si vous voulez l'index pour un enfant de la Liste, utilisez :

```
$index = $list->child_position( $child ) ;
```

Si une erreur se produit, une valeur -1 est retournée.

On peut choisir le mode de sélection à l'aide de :

```
$list->set_selection_mode( $mode ) ;
```

Ou `mode` peut être `'single'`, `'browse'`, `'multiple'`, `'extended'`.

3.14.4 Le widget ListItem

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Item
                      +--- ListItem
```

Les widgets ListItem sont conçus pour agir comme des conteneurs ayant un enfant et fournissant des fonctions de sélection/désélection tout comme le widget List l'exige d'eux pour ses enfants.

Un ListItem possède sa propre fenêtre pour recevoir les évènements et sa propre couleur de fond qui est habituellement blanche.

Comme il est directement dérivé d'un Item, il peut être traité comme tel. Pour plus de données la dessus, voir la section sur les Items. Habituellement un ListItem possède juste un label pour identifier un nom de fichier dans un Liste, c'est pourquoi la fonction bien pratique `new_with_label()` est fourni. Le même effet peut être obtenu en créant un Label à part entière, déclarant ses alignements à xalign=0 et yalign=0.5 avec une addition sous-jacente de conteneur au ListItem.

Comme personne n'est obligé d'ajouter un GtkLabel à un GtkListItem, vous pouvez également ajouter une GtkVBox ou une GtkArrow, etc, au GtkItem.

3.14.5 Les fonctions ListItem

Pour créer un ListItem, appeler l'une des fonctions suivantes :

```
$item = new Gtk : :ListItem() ;
$item = new Gtk : :ListItem( $label ) ;
```

Si vous passez une chaîne en argument, cela crée une ListItem avec un Label comme seul enfant du ListItem.

Pour sélectionner un ListItem :

```
$item->select() ;
```

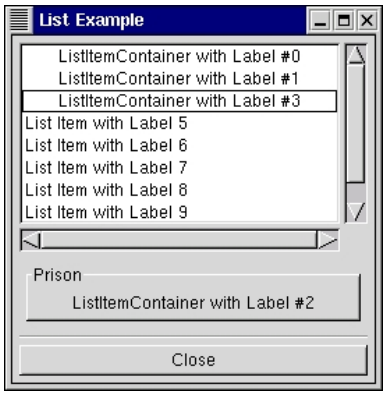
Cette méthode émettra le signal `'select'` sur le ListItem.

La désélection fonctionne de la même manière :

```
$item->deselect() ;
```

3.14.6 Exemple

L'exemple suivant montrera les changements de sélection d'une liste et vous laissera "arrêter" une liste d'items et les placer en prison en les sélectionnant avec le bouton droit de la souris.




```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $separator ;
my $window ;
my $vbox ;
my $scrolled_window ;
my $frame ;
my $gtklist ;
my $button ;
my $list_item ;

my @dlist = () ;
my $i ;
my $buffer ;

$window = new Gtk : :Window( 'toplevel' ) ;
$window->set_title( "List Example" ) ;
$window->signal_connect( 'destroy', sub { Gtk->exit( 0 ) ; } ) ;
```

```
# A l'intérieur de la fenêtre on a besoin d'une boîte pour ranger les
# widgets verticalement
$vbox = new Gtk : :VBox( $false, 5 ) ;
$vbox->set_border_width( 5 ) ;
$window->add( $vbox ) ;
$vbox->show() ;

# Ceci est une fenêtre défilable pour y placer le widget List
$scrolled_window = new Gtk : :ScrolledWindow( undef, undef ) ;
$scrolled_window->set_usize( 250, 150 ) ;
$vbox->add( $scrolled_window ) ;
$scrolled_window->show() ;

# Crée un widget List. Connecte la fonction sigh_print_selection()
# au signal "selection_changed" de la List pour imprimer les items
# sélectionnés chaque fois que la sélection a changé.
$gtklist = new Gtk : :List() ;
$scrolled_window->add_with_viewport( $gtklist ) ;
$gtklist->show() ;
$gtklist->signal_connect( 'selection_changed', \&sigh_print_selection ) ;

#Nous créons une prison pour y placer une liste d'items ;)
$frame = new Gtk : :Frame( "Prison" ) ;
$frame->set_usize( 200, 50 ) ;
$frame->set_border_width( 5 ) ;
$frame->set_shadow_type( 'out' ) ;
$vbox->add( $frame ) ;
$frame->show() ;

# Connecte le sigh_button_event() à la List, qui déclenchera
# l'arrestation des items de la liste.
$gtklist->signal_connect( 'button_release_event',
                          \&sigh_button_event,
                          $frame ) ;

# Crée un séparateur
$separator = new Gtk : :HSeparator() ;
$vbox->add( $separator ) ;
$separator->show() ;

# Enfin crée un bouton et connecte son signal ``cliqué'' à la destruction
# de la fenêtre.
$button = new Gtk : :Button( "Close" ) ;
$vbox->add( $button ) ;
$button->show() ;
$button->signal_connect( 'clicked', sub { $window->destroy() ; } ) ;

# Maintenant, nous créons 5 éléments de liste, chacun ayant son propre label
# et nous les ajoutons à la List à l'aide de la fonction add().
for $i ( 0..4 )
  {
    my $label ;
    my $string ;

    $buffer = "ListItemContainer with Label #" . $i ;
    $label = new Gtk : :Label( $buffer ) ;

    $list_item = new Gtk : :ListItem() ;
```

```
    $list_item->add( $label );
    $label->show();
    $gtklist->add( $list_item );
    $list_item->show();
    $string = $label->get();
  }

# Ici, nous créons 5 autres Labels, cette fois on envoie le Label à la fonction
# new().
for $i ( 5..9 )
  {
    $buffer = "List Item with Label $i";
    $list_item = new Gtk : :ListItem( $buffer );
    @dlist = ( $list_item, @dlist );
    $list_item->show();
  }

$gtklist->append_items( @dlist );

# Enfin, on veut voir la fenêtre, n'est-ce-pas ? ;)
$window->show();

# Déclenche la boucle Gtk
main Gtk;
exit( 0 );




### Routines

# Ceci est le gestionnaire de signal connecté au bouton pressé/relâché
# de la List.
sub sigh_button_event
  {
    my ( $gtklist, $frame, $event ) = @_;

    # Nous faisons seulement quelque chose cela si le troisième
    # (le plus à droite) est relâché
    if ( ( defined( $event->{ 'type' } ) ) and
         ( defined( $event->{ 'button' } ) ) and
         ( $event->{ 'type' } eq 'button_release' ) and
         ( $event->{ 'button' } == 3 ) )
      {
        my @dlist = ();
        my $new_prisoner;
        my $list_item;

        # Rapporte la liste actuellement sélectionnée
        # qui sera notre prochain prisonnier.
        @dlist = $gtklist->selection;
        $new_prisoner = $dlist[ 0 ] if ( @dlist );

        # Cherche les items de la liste déjà prisonniers, nous les
        # replacerons dans la liste.
        foreach $list_item ( $frame->children() )
          {
            $list_item->reparent( $gtklist );
```

```
        }

      # Si nous avons un nouveau prisonnier, on l'enlève de la liste et on
      # le place dans le cadre ``Prison''. Nous devons d'abord le
      # désélectionner. Je ne sais pourquoi, j'ai reçu une erreur de
      # valeur non initialisée durant cette section de code, bien que
      # cela fonctionne.
      if ( defined( $new_prisoner ) )
        {
          $gtklist->unselect_child( $new_prisoner ) ;
          $new_prisoner->reparent( $frame ) ;
        }
    }
  }

# C'est le gestionnaire qui est appelé si le widget List émet le signal
# "selection_changed"
sub sigh_print_selection
  {
    my ( $gtklist ) = @_ ;

    my @dlist ;
    my $list_item ;

    # Rapporte l'item de la liste actuellement sélectionné qui sera
    # notre prochain prisonnier
    @dlist = $gtklist->selection ;

    # Si aucun des items n'est sélectionné, il n'y a rien d'autre à
    # faire que de le dire à l'utilisateur.

    unless ( defined( @dlist ) )
      {
        print "Selection cleared\n" ;
        return ;
      }
  }
```

3.15 Le widget CList

```
Object
  +--- Widget
       +--- Container
            +--- CList
```

Le widget CList a remplacé le widget List (qui est toujours disponible).

Le widget CList est un widget liste Multi-colonnes qui est capable de manipuler littéralement des milliers de colonnes d'information. Chaque colonne peut optionnellement avoir un titre, qui lui-même est optionnellement actif, ce qui nous permet de relier une fonction à sa sélection.

3.15.1 Créer un widget CList

Créer un widget CList se fait plutôt facilement quand on a compris le fonctionnement des widgets en général. Il fournit les presque standards deux méthodes : la difficile et la facile. Mais, avant de le créer, il y a une chose qu'il nous faut connaître : combien devrait-il y avoir de colonnes ?

Toues les colonnes n'ont pas besoin d'être visibles mais peuvent stocker des données qui font référence à certains cellules de la liste.

```
$clist = new Gtk : :CList( $columns ) ;
$clist = new_with_titles Gtk : :CList( @titles ) ;
```

La première méthode est très directe en revanche la seconde nécessite quelques explications. Chaque colonne peut posséder un titre associé et ce titre peut être un label ou un bouton qui réagit quand on clique dessus. Si nous utilisons la seconde forme, nous fournissons une liste de titres et le nombre d'éléments de cette liste correspond au nombre de colonnes de notre CList. Bien sûr, nous pouvons toujours utiliser la première méthode et ajouter manuellement les titres plus tard.

Note : Le widget CList ne fournit pas sa propre barre de défilement et devrait être placé dans une fenêtre défilable si vous avez besoin de cette fonctionnalité. C'est un changement par rapport à l'implémentation 1.0 de GTK.

3.15.2 Modes d'opération

Il y a plusieurs attributs qui peuvent être utiliser pour changer le comportement d'une CList. Tout d'abord :

```
$clist->set_selection_mode( $mode ) ;
```

qui, comme son nom l'indique, déclare le mode de sélection de la CList. L'argument spécifie le mode de sélection d'un cellule (ils sont définis dans le fichier header de Gtk en C gtkenum.h). Au moment ou j'écris, les modes suivants sont disponibles :

`'single'` - La sélection est soit vide soit sa liste ne contient qu'un seul élément sélectionné.

`'browse'` - La sélection est vide si la liste ne contient aucun widget ou seulement des insensibles, autrement il contient une liste avec un élément.

`'multiple'` - La sélection est vide si aucun des éléments n'est sélectionné ou bien une liste de tous les items sélectionnés. Valeur par `d\'efaut` pour le widget CList.

`'extended'` - La sélection est toujours vide.

D'autres peuvent avoir été ajoutées aux dernières versions de GTK.

Vous pouvez aussi définir à quoi va ressembler le bord du widget CList, grâce à la fonction :

```
$clist->set_shadow_type( $border ) ;
```

Les valeurs possibles de `$border` sont :

```
'none'
'in'
'out'
'etched_in'
'etched_out'
```

3.15.3 Travailler avec les titres

Quand vous créez un widget CList, vous récupérerez également un ensemble de boutons automatiquement. Ils sont situés en haut de la fenêtre CList, et peuvent soit se comporter comme des boutons normaux qui répondent quand on appuie dessus soit être inactifs auquel cas ils ne sont que de simples titres. Il existe quatre fonctions pour nous aider à déclarer le statut des boutons de titres :

```
$clist->column_title_active( $column ) ;
$clist->column_title_passive( $column ) ;
$clist->column_titles_active() ;
$clist->column_titles_passive() ;
```

Un titre actif est un titre qui réagit comme un bouton normal, un titre inactif n'est qu'un label. Les deux premiers appels ci-dessus activent/désactivent le bouton de titre au dessus de colonnes spécifiques, alors que les deux derniers activent.désactivent tous les boutons de titres du widget CList.

Mais, bien sûr, il a des cas ou vous ne les voudrez pas du tout, c'est pourquoi ils peuvent être cachés ou montrés à volonté à l'aide des fonctions :

```
$clist->column_titles_show() ;
$clist->column_titles_hide() ;
```

Pour que les titres soient vraiment utiles, nous avons besoin d'un mécanisme pour les déclarer et les changer. Cela se fait avec :

```
$clist->set_column_title( $column, $title ) ;
```

Notez que seul le titre d'un colonne peut être déclaré à la fois, c'est pourquoi si tous les titres sont connus dès le début, je suggère d'utiliser `new_with_titles Gtk::CList()`. Cela vous économise du temps et vos programmes sont plus courts. Il est des fois ou le travail manuel est plus efficace, entre autres, quand tous les titres ne sont pas des textes. CList nous fournit des boutons de titres qui peuvent contenir des widgets entiers, comme par exemple un pixmap. Cela se fait avec :

```
$clist->set_column_widget( $column, $widget ) ;
```

qui ne nécessite aucune explication particulière.

3.15.4 Manipuler la liste elle-même

Il est possible de modifier la justification d'une colonne avec :

```
$clist->set_column_justification( $column, $justification ) ;
```

`$justification` peut prendre les valeurs suivantes :

`'left'` - Le texte commencera à partir du bord gauche de la colonne.

`'right'` - Le texte commencera à partir du bord droit de la colonne.

`'center'` - Le texte sera placé au centre de la colonne.

`'fill'` - Le texte utilisera toute la place disponible dans la colonne. Cela se fait en insérant des espaces blancs entre les mots (ou entre chaque lettre s'il s'agit d'un seul mot). Comme dans n'importe quel éditeur WYSIWYG.

La prochaine fonctionne est très importante et devrait être standard pour l'installation de tous les widgets CList. Quand la liste est créée, la largeur des diverses colonnes est choisie par rapport au titre, et comme c'est rarement la largeur qui convient, nous aurons donc à la déclarer avec :

```
$clist->set_column_width( $column, $width ) ;
```

Notez que la largeur est donnée en pixels et non en lettres. On fait de même pour la hauteur des cellules de la colonne mais comme la valeur par défaut est fonction de la taille de la police de caractère, ce n'est pas aussi critique pour l'application. Toutefois, on peut le faire avec :

```
$clist->set_row_height( $height ) ;
```

Encore, notez que la hauteur est donnée en pixels.

Nous pouvons également déplacer un liste sans que l'utilisateur intervienne, toutefois cela nécessite de savoir ce que nous voulons. En d'autres termes, nous avons besoin de la ligne et de la colonne de l'élément que nous voulons bouger.

```
$clist->moveto( $row, $column, $row_align, $col_align ) ;
```

`$row_align` est très important à comprendre. C'est une valeur entre 0.0 et 1.0 ou 0.0 signifie que nous bougeons la ligne en faisant en sorte que la ligne apparaisse en haut alors que la valeur de 1.0 implique la ligne apparaîtra en bas. Toutes les autres valeurs entre 0.0 et 1.0 sont également valables et placerons la ligne entre le haut et le bas. Le dernier argument `$col_align` fonctionne de la même manière sauf que 0.0 correspond à gauche et 1.0 à droite.

En fonction des besoins de l'application, nous n'avons pas toujours à déplacer un élément qui nous est déjà visible. Oui mais comment savoir si il est visible ? Comme d'habitude, il y a une fonction qui nous permet de répondre :

```
$clist->row_is_visible( $row ) ;
```

La valeur retournée est l'une des suivante :

```
'none'
'partial'
'full'
```

Notez que cela ne nous renseigne que sur la colonne. Pour l'instant, Il n'y a aucun moyen de le déterminer pour la colonne. Nous pouvons cependant avoir une information partielle parce que si la valeur de retour est `partial`, alors un bout de la ligne est caché mais nous ne savons pas si la ligne est coupé dans sa partie inférieure ou si elle possède des colonnes à l'extérieur.

Nous pouvons également changer la couleur de la police et la couleur du fond d'une ligne particulière. Cela est utile pour indiquer la ligne sélectionneée par l'utilisateur. Les deux fonctions qui le permettent sont :

```
$clist->set_foreground( $row, $color ) ;
$clist->set_background( $row, $color ) ;
```

Notez toutefois que les couleurs doivent avoir été préalablement allouées.

3.15.5 Ajouter des lignes à la liste

Vous pouvez ajouter des lignes de trois manières. Elles peuvent être placées avant la liste ou après à l'aide de :

```
$clist->prepend( @text );
$clist->append( @text );
```

La valeur de retour de ces deux fonctions indique la position à laquelle a été ajoutée la colonne. Vous pouvez insérer une colonne à une position donnée :

```
$clist->insert( $row, @text );
```

Avec ces appels, nous devons fournir une liste des textes que nous voulons mettre dans les colonnes. Le nombre de textes devraient être égal au nombre de colonnes de la liste. Si la liste de texte est vide alors il n'y aura pas de texte dans les colonnes de la ligne. C'est utile si vous voulez placer des pixmaps à la place (chose qui doit être faite manuellement).

Faites attention toutefois au fait que l'on commence à numéroter les lignes et les colonnes par 0.

Pour supprimer une ligne, on utilise :

```
$clist->remove( $row );
```

Il existe également un appel qui enlève toutes les lignes de la listes. C'est un peu plus rapide que d'appeler `$clist->remove( $row )` pour chaque colonne, ce qui représente une alternative. Il suffit de faire :

```
$clist->clear();
```

Il existe également deux fonctions bien utile qui doivent être utilisées quand beaucoup de changements doivent être faits sur une liste. Cela empêche la liste de clignoter quand on la met à jour de manière répétitive ce qui peut être très ennuyeux pour l'utilisateur. C'est pourquoi c'est une bonne idée de geler la liste, de faire les mises à jour et enfin de la dégeler ainsi la liste mise à jour apparaîtra à l'écran.

```
$clist->freeze();
$clist->thaw();
```

3.15.6 Placer du texte et des pixmaps dans les cellules

Une cellule peut contenir un pixmap, du texte ou les deux. Pour les placer, les fonctions suivantes peuvent être utilisées :

```
$clist->set_text( $row, $column, $text );
$clist->set_pixmap( $row, $column, $pixmap, $mask );
$clist->set_pixtext( $row, $column, $text, $spacing, $pixmap, $mask );
```

C'est plutôt clair. Tous les appels possèdent la ligne et la colonne suivis de la donnée à placer. L'argument `$spacing` dans `set_pixtext()` est le nombre de pixels entre le pixmap et le début du texte. Dans tous les cas la donnée est copiée dans le widget.

Pour lire les données, on utilise en revanche :

```
$text = $clist->get_text( $row, $column );
( $pixmap, $mask ) = $clist->get_pixmap( $row, $column );
( $text, $spacing, $pixmap, $mask = $clist->get_pixtext( $row, $column );
```

Les valeurs de retour sont toutes des références aux données stockées donc les données référencées ne devraient pas être modifiées.

Il y a un appel supplémentaire qui relate ce qui est dans une cellule de la liste :

```
$clist->get_cell_type( $row, $column );
```

qui retourne le type de donnée contenue dans la cellule. La valeur de retour peut être :

```
'empty'
'text'
'pixmap'
'pixtext'
'widget'
```

Il existe également une fonction qui permet de déclarer l'indentation à la fois verticale et horizontale d'une cellule. La valeur de l'indentation est donnée en pixels et peut être soit positive soit négative.

```
$clist->set_shift( $row, $column, $vertical, $horizontal );
```

3.15.7 Stocker des données

Avec une CList il est possible de déclarer une donnée. Cette donnée ne sera pas visible de l'utilisateur mais simplement une possibilité pour le programmeur d'associer une colonne avec des données supplémentaires.

Ces fonctions sont suffisamment explicites :

```
$clist->set_row_data( $row, $data );
$clist->set_row_data_full( $row, $data, $destroy );
$clist->get_row_data( $row );
$clist->get_row_from_data( $data );
```

3.15.8 Travailler avec les sélections

Il existe des fonctions qui nous permettent de forcer une sélection ou une désélection d'une ligne :

```
$clist->select_row( $row, $column );
$clist->unselect_row( $row, $column );
```

Il y a également une fonction qui lit les coordonnées X et Y (par exemple, du pointeur de souris) et les transcrit par rapport à la liste, retournant la ligne et la colonne correspondantes.

```
( $row, $column ) = $clist->get_selection_info( $x, $y );
```

Quand nous détectons quelque chose digne d'intérêt (cela peut être un mouvement de la souris ou un clic quelque part dans la liste) nous pouvons lire les coordonnées du pointeur et trouver où il se trouve dans la liste. Trop lourd ? Heureusement, il y a une méthode plus simple ...

3.15.9 Les signaux qui regroupent tout ça

Comme pour tous les autres widgets, quelques signaux peuvent être utilisés. Le widget CList est dérivé du widget Container donc il possède les mêmes signaux mais vous pouvez y ajouter :

`'select_row'` - Le signal enverra les informations suivantes dans l'ordre : la CList, la ligne, la colonne et l'évènement.

`'unselect_row'` - Quand l'utilisateur désélectionne une ligne, ce signal est activé. Il envoie les mêmes informations que `'select_row'`.

`'click_column'` - Envoie une CList et une colonne.

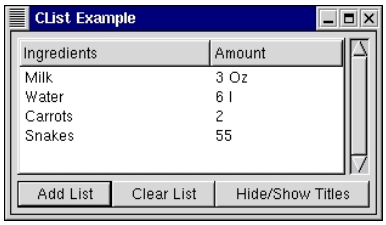
Si vous voulez connecter un rappel à `'select_row'`, la fonction de rappel sera déclarée comme ceci :

```
sub select_row_callback
  {
    my ( $widget, $row, $column, $event, @data ) = @_;
  }
```

Le rappel est connecté comme d'habitude par :

```
$clist->signal_connect( "select_row", \&select_row_callback, @data );
```

3.15.10 Exemple




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;
```

```
my $false = 0 ;
my $true = 1 ;

my $window ;
my $vbox ;
my $hbox ;
my $scrolled_window ;
my $clist ;
my $button_add ;
my $button_clear ;
my $button_hide_show ;

my $titles_visible = $false ;
my @titles = ( "Ingredients", "Amount" ) ;

$window = new Gtk : :Window( "toplevel" ) ;
$window->set_usize( 300, 150 ) ;
$window->set_title( "CList Example" ) ;
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ) ; } ) ;

$vbox = new Gtk : :VBox( $false, 5 ) ;
$window->add( $vbox ) ;
$vbox->border_width( 5 ) ;
$vbox->show() ;

# Crée une fenêtre défilable pour y placer la CLIst
$scrolled_window = new Gtk : :ScrolledWindow( undef, undef ) ;
$vbox->pack_start( $scrolled_window, $true, $true, 0 ) ;
$scrolled_window->set_policy( 'automatic', 'always' ) ;
$scrolled_window->show() ;

# Crée la CList. Pour cet exemple, nous utilisons 2 colonnes
$clist = new_with_titles Gtk : :CList( @titles ) ;

# Quand une sélection est faite, nous voulons le savoir. Le rappel
# utilisé est selection_made et son code peut être trouvé plus loin
$clist->signal_connect( "select_row", \&selection_made ) ;

# Il n'est pas nécessaire qu'il y ait une ombre au bord
# mais ça a une bonne tête :)
$clist->set_shadow_type( 'out' ) ;

# Autrement plus important, déclarer la largeur des colonnes sinon
# ce ne sera jamais bon. Noter que les colonnes sont numérotées de 0 à plus
# ( 1 dans notre cas )
$clist->set_column_width( 0, 150 ) ;

# Ajoute la CList dans la boîte verticale et la montre
$scrolled_window->add( $clist ) ;
$clist->show() ;

# Crée les boutons et les ajoute à la fenêtre
$hbox = new Gtk : :HBox( $false, 0 ) ;
$vbox->pack_start( $hbox, $false, $true, 0 ) ;
$hbox->show() ;

$button_add = new Gtk : :Button( "Add List" ) ;
```

```
$button_clear = new Gtk : :Button( "Clear List" ) ;
$button_hide_show = new Gtk : :Button( "Hide/Show Titles" ) ;

$hbox->pack_start( $button_add, $true, $true, 0 ) ;
$hbox->pack_start( $button_clear, $true, $true, 0 ) ;
$hbox->pack_start( $button_hide_show, $true, $true, 0 ) ;

# Connecte nos retours à nos trois boutons
$button_add->signal_connect( "clicked", \&button_add_clicked, $clist ) ;
$button_clear->signal_connect( "clicked", sub { $clist->clear() ; } ) ;
$button_hide_show->signal_connect( "clicked", \&button_hide_show_clicked, $clist ) ;

$button_add->show() ;
$button_clear->show() ;
$button_hide_show->show() ;

$window->show() ;
main Gtk ;
exit( 0 ) ;



### Routines

# L'utilisateur a cliqué sur le bouton ``add list'' ( ajoute une liste )
sub button_add_clicked
  {
    my ( $widget, $clist ) = @_ ;

    my $i ;

    # Quelque chose de bête à ajouter à la liste.
    # 4 lignes de 2 colonnes chacune.
    my @drink = ( [ Milk    => "3 Oz" ],
                  [ Water   => "6 l"  ],
                  [ Carrots => "2"    ],
                  [ Snakes  => "55"   ] ) ;

    # Ici nous ajoutons effectivement les textes.
    # C'est fait une fois pour chaque ligne.
    for ( $i = 0 ; $i < 4 ; $i++ )
      {
        $clist->append( @{$drink[ $i ]} ) ;
      }

    # Pour ceux qui n'aurait pas compris ce qui précède, voici une brève
    # explication. @drink est une liste de référence de liste ( les références
    # sont des scalaires contenant une adresse mémoire, semblables aux
    # pointeurs). Ils sont utilisés car Perl ne supporte pas directement
    # les tableaux à deux dimensions. A cause de cela, $drink[i] n'est
    # pas une liste mais une référence à une liste, ainsi nous devons
    # la déréférencer en l'enfermant ainsi : @{référence} ( nous devons
    # utiliser @ parce ce que c'est une référence à une liste. Pour plus
    # d'informations sur les références, voir la page man perlref(1)
    # ou le chapitre 4 de Programmer en Perl.

    return ;
  }
```

```
# L'utilisateur  a cliqué sur le bouton ``Hide/Show titles''
# Cacher ou montrer les titres
sub button_hide_show_clicked
  {
    my ( $widget, $clist ) = @_;

    if ( $titles_visible )
      {
        # Cache les titres et déclare le flag égal à 1
        $clist->column_titles_hide();
        $titles_visible++;
      }
    else
      {
        # Montre les titres et déclare le flag égal à 0
        $clist->column_titles_show();
        $titles_visible--;
      }

    return;
  }

# Si on arrive là, l'utilisateur a sélectionné une ligne de la liste
sub selection_made
  {
    my ( $clist, $row, $column, $event, @data ) = @_;

    my $text;

    # Récupère le texte qui est stocké à la ligne et la colonne cliquées.
    # Nous le recevrons comme un pointeur du texte de l'argument
    $text=$clist->get_text( $row, $column );

    # Imprime juste quelques informations à propos de la ligne sélectionnée
    print( "You selected row $row.  More specifically, you clicked in\n" );
    print( "column $column, and the text in this cell is :\n" );
    print( "    $text\n\n" );

    return;
  }
```

3.16 Le widget Arbre

Le but du widget Arbre est d'afficher des données hiérarchiquement organisées. Le widget Arbre lui-même est une conteneur vertical pour les widget de type TreeItem. Arbre n'est pas très différent d'une CList - les deux sont dérivés de Container, et les méthodes de Container fonctionnent aussi bien pour les widgets Arbre que pour les widgets CList. La différence est que les widgets Arbre peuvent être nichés dans d'autres widgets Arbre. Nous verrons brièvement comment faire cela.

Le widget Arbre possède sa propre fenêtre et possède par défaut un fond blanc comme les CList. Quoi qu'il en soit, Arbre ne dérive pas de CList donc nous ne pouvons utiliser l'un ou l'autre indifféremment.

3.16.1 Créer un Arbre

Un arbre est créé selon la méthode habituelle :

```
$tree = new Gtk : :Tree();
```

Comme les widgets CList, un widget Arbre continuera de grandir tant qu'on lui ajoutera des éléments, ou quand les subtrees (sous-arbres) prendront de l'ampleur. Pour cette raison, ils sont presque toujours placés dans une fenêtre

défilable. Vous voudrez certainement utiliser `set_usize()` pour la fenêtre défilable afin de vous assurer qu'elle est assez grosse pour afficher les items de l'Arbre, car la taille par défaut d'une fenêtre défilable est plutôt petite.

Maintenant que nous avez un arbre vous voudrez certainement y ajouter quelques éléments. La section sur les TreeItems donne des détails "crus" sur les TreeItem. Pour l'instant, il suffit d'en créer un :

```
$tree = new_with_label Gtk : :TreeItem( $label ) ;
```

Vous pouvez l'ajouter à l'arbre en utilisant une des fonctions suivantes :

```
$tree->append( $tree_item ) ;
$tree->prepend( $tree_item ) ;
```

Notez que vous devez ajouter les éléments un par un - Il n'y a pas l'équivalent des fonctions sur les items d'un Clist.

3.16.2 Ajouter un sous-arbre

Un sous-arbre est créé comme un autre widget Arbre. Un sous-arbre est ajouté à un arbre sous un item arbre à l'aide de :

```
$tree_item->set_subtree( $subtree ) ;
```

Vous n'avez pas besoin d'appeler la fonction `show()` pour un sous-arbre avant ou après l'avoir ajouté à un TreeItem. Quoiqu'il en soit, vous devez ajouter le TreeItem en question à un arbre parent avant d'appeler `set_subtree()`. C'est parce que, techniquement, le parent du sous-arbre n'est pas le TreeItem qui le possède mais l'arbre qui possède le TreeItem.

Quand vous ajoutez un sous-arbre à un TreeItem, un signe plus ou moins apparaît à son côté sur lequel l'utilisateur peut cliquer pour le "développer" ou le "réduire ", ce qui signifie montrer ou cacher le sous-arbre. Les TreeItems sont par défaut réduits. Notez que quand vous réduisez un sous-arbre, n'importe quel item sélectionné dans son sous-arbre reste sélectionné ce que l'utilisateur n'attend pas spécialement.

3.16.3 Manipuler une liste de sélection

Comme avec une CList, le type arbre possède un champs de sélection et il est possible de contrôler le comportement de l'arbre (à peu près) en déclarant le type de sélection avec :

```
$tree->set_selection_mode( $mode ) ;
```

Le sens des mots associés aux différents modes de sélection sont décrits dans la section sur les widgets Clist. Comme avec les widgets CList, les signaux `select_child`, `unselect_child` (pas vraiment - voir la section sur les signaux ci-dessous)et `selection_changed` sont émis quand des items de la liste sont sélectionnés ou désélectionnés. Toutefois, pour tirer parti de ces signaux il faut savoir par quel widget arbre ils sont émis et où trouver la liste des items sélectionnés.

C'est une source de confusion potentielle. Le meilleur moyen d'expliquer ceci est de penser que tous les widgets Arbres sont créés égaux, et certains sont plus égaux que d'autres. Tous les widgets Arbres possèdent leurs propres fenêtres X, et peuvent par conséquent recevoir des évènements comme les clics de souris (si leurs TreeItems ou leurs enfants peuvent les capter en premier). Toutefois pour faire que des sélections du type `'single'` ou `'browse'` se comportent d'une manière saine, la liste des items sélectionnés est spécifique au widget Arbre le plus élevé dans la hiérarchie, connu comme "l'arbre racine".

Ainsi, accéder à une champs de `s\'election` directement dans un widget arbre n'est pas une bonne idée à moins que vous ne sachiez s'il s'agit de l'arbre racine.

Enfin, les signaux `select_child` (et en théorie, `unselect_child`) sont émis par tous les arbres mais le signal `selection_changed` est uniquement émis par l'arbre racine. Par conséquent, si vous voulez manipuler le signal `select_child` pour un arbre et tous ses sous-arbres, vous devrez appeler `signal_connect()` pour tous les sous-arbres.

3.16.4 Composition d'un widget Arbre

Le champs d'une arbre (à partir de sa structure C) ressemble à :

```
container
children
root_tree
tree_owner
selection
level
```

```
indent_value
current_index
selection_mode
view_mode
view_line
```

En plus, `is_root_tree()` retourne une valeur booléenne qui indique si un arbre est un arbre racine dans une hiérarchie arbre alors que `root_tree()` retourne l'arbre racine.

Plutôt que d'accéder directement au champs enfant d'un widget arbre, il est probablement meilleur d'y accéder en utilisant la fonction `children()`, héritée du widget Container.

Le champs `tree_owner` est défini uniquement dans les sous-arbres, où il pointe vers le widget TreeItem qui possède l'arbre en question. Le champs `level` indique à quel point est "niché" un arbre particulier ; les arbres racines ont le niveau 0 et chaque niveau successif de sous-arbres a une niveau supérieur à celui du niveau parent augmenté de 1. En fait, ce champs est déclaré un fois que le widget Arbre est représenté (tracé à l'écran).

3.16.5 Les signaux

Le signal `selection_changed` sera émis quelque soit le champs `selection` de l'arbre modifié. Cela arrive quand un enfant de l'arbre est sélectionné ou désélectionné.

Le signal `select_child` est émis quand un enfant de l'arbre est sur le point d'être sélectionné. Cela arrive sur les appels `select_item()`, `select_child()`, sur tous les boutons pressés et les appels `item_toggle()` et `toggle()`. Cela peut parfois être déclenché en d'autres occasions, quand des enfants sont ajoutés ou enlevés de l'arbre.

Le signal `unselect_child` est émis quand un enfant de l'arbre est sur le point d'être désélectionné. Comme avec GTK 1.0.4, cela semble n'arriver que lors des appels `unselect_item()` ou `unselect_child()` et peut être en d'autres occasions, mais pas quand un bouton pressé désélectionne un enfant (le signal `select_child` est émis à la place), ni lors de l'émission du signal `toggle` par gtk_item_toggle().

3.16.6 Les fonctions

```
$tree->insert( $tree_item, $position ) ;
```

Insère un item arbre dans un Arbre à la position spécifiée

```
$tree->remove_items( @items ) ;
```

Enlève une liste d'éléments de l'Arbre. Notez qu'enlever un élément d'un arbre le déréférence et (habituellement) le détruit lui et son sous-arbre, s'il en a, et tous les sous-arbres dans ces sous-arbres. Si vous voulez enlever uniquement un élément, vous pouvez utiliser `remove()`.

```
$tree->clear_items( $start, $end ) ;
```

Enlève les éléments compris entre la position `$start` et `$end`. Le même avertissement ici à propos du déférencement qui s'applique car `clear_items()` construit simplement la liste et la passe ensuite à `remove_items()`.

```
$tree->select_item( $item ) ;
```

Émet le signal `select_item` pour un enfant à la position `$item`, le sélectionnant ainsi (sauf si vous le désélectionnez avec un gestionnaire de signal).

```
$tree->unselect_item( $item ) ;
```

Émet le signal `unselect_item` pour un enfant à la position `$item`, le désélectionnant ainsi.

```
$tree->select_child( $tree_item ) ;
```

Émet le signal `select_item` pour un enfant `$tree_item`, le sélectionnant ainsi.

```
$tree->unselect_child( $tree_item ) ;
```

Émet le signal `unselect_item` pour un enfant `$tree_item`, le désélectionnant ainsi.

```
$tree->child_position( $child ) ;
```

Retourne la position dans un un arbre de `$child` à moins que `$child` ne soit pas dans l'arbre, dans ce cas retourne -1.

```
$tree->set_selection_mode( $mode ) ;
```

Déclare le mode de sélection, qui peut être `single` (le défaut), `browse`, `multiple` ou `extended`. Cela est seulement défini pour les arbres racines, ce qui tombe sous le sens puisque les arbres racines "possèdent" la sélection. Faire cette déclaration sur un sous-arbre n'a aucun effet, la valeur est simplement ignorée.

```
$tree->set_view_mode( $mode ) ;
```

Déclare le mode "view" qui peut être soit `line` (le défaut) soit `item`. Le mode "view" se propage d'un arbre à ses sous-arbres quand ils sont représentés, et ne concerne pas exclusivement un sous-arbre. (le déclarer sur un sous-arbre après qu'il ait été représenté pourrait avoir des effet quelque peu imprévisibles).

Le terme mode "view" est plutôt ambigu - à la base, cela contrôle la manière dont la coloration est représentée quand l'un des enfant de l'arbre est sélectionné. Si c'est `line`, l'arbre entier est coloré alors que si c'est `item`, seul le widget enfant (habituellement le label) est coloré.

```
$tree->set_view_lines( $show_lines ) ;
```

Contrôle si les lignes qui relient les éléments de l'arbre sont dessinées. `$show_lines` est une valeur vraie ou fausse.

```
$tree->root_tree() ;
```

Retourne l'arbre racine d'un objet arbre.

```
$tree->selection() ;
```

Retourne la liste de sélection d'un arbre racine d'un objet "GtkTree".

3.16.7 Le widget TreeItem

Le widget TreeItem, comme le CListItem, est dérivé de Item, qui à son tour est dérivé de Bin. Par conséquent, l'item lui-même est un conteneur générique contenant exactement un seul widget enfant qui peut être de n'importe quel type. Le widget TreeItem possède un nombre supplémentaire de champs mais le seul qui nous intéresse vraiment est le champs sous-arbre.

Les champs disponibles pour la structure TreeItem ressemblent à :

```
item
subtree
pixmaps_box
plus_pix_widget
minus_pix_widget
pixmaps
expanded
```

Le champs `pixmaps_box` est une Boîte d'évènements qui attrapent les clics sur le signe plus/moins qui contrôle le développement ou la réduction. Le champs `pixmaps` pointe vers une structure de données internes. Puisque vous pouvez toujours obtenir le sous-arbre d'un TreeItem avec la fonction `subtree()`, il est vivement conseillé de ne pas toucher à l'intérieur d'un TreeItem sauf si vous savez exactement ce que vous faîtes.

Un TreeItem possède habituellement un label, donc la fonction `new_with_label Gtk::TreeItem()` est fournie. Le même effet peut être obtenu en utilisant le code suivant :

```
$tree_item = new Gtk : :TreeItem() ;
$label = new Gtk : :Label( $text ) ;
$label->set_alignment( 0.0, 0.5 ) ;

$tree_item->add( $label ) ;
$show( $label ) ;
```

Comme personne ne nous oblige à ajouter un label à un TreeItem, nous pouvons également ajouter une HBox ou une flèche, ou même un Notebook (bien que dans ce cas, votre application risque d'être plutôt impopulaire).

Si vous enlevez tous les éléments d'un sous-arbre, celui-ci sera détruit et ne sera plus rattaché à aucun parent, à moins que vous ne le référenciez plus tôt, et le TreeItem auquel il appartient sera réduit. Donc, si vous voulez qu'il reste en place, faîtes comme ceci :

```
$tree->ref() ;
$owner = $tree->tree_owner ;
$tree->remove( $item ) ;
if ( $tree->parent )
{
```

```
    $tree->unref() ;
}
    else
{
    $owner->expand() ;
    $owner->set_subtree( $tree ) ;
}
```

Enfin, le drag and drop fonctionne avec les TreeItems. Vous devez juste vous assurer que le TreeItem dans lequel vous prenez un élément ou vous en placer un, a non seulement été ajouté à l'arbre mais que chaque widget parent successif possède lui-même un parent, tous jusqu'à une fenêtre toplevel ou dialogue, quand vous utilisez `dnd_drag_set()` ou `dnd_drop_set()`. Autrement de drôles de choses peuvent se produire.

TreeItem héritent des signaux `select`, `unselect` et `toggle` des Items. En plus, il possède deux autres signaux qui lui sont propres :`expand` et `expand()`.

Le signal `expand` est émis quand le sous-arbre d'un arbre est sur le point d'être développé, c'est-à-dire quand l'utilisateur clique sur le signe plus placé à côté de l'élément, ou quand le programme appelle la fonction :`expand()`.

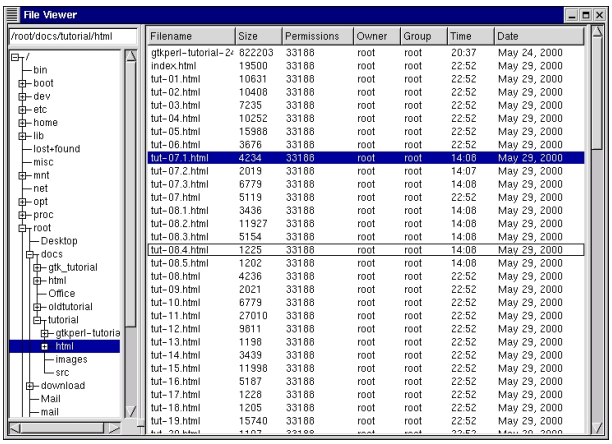
Le signal `collapse` est émis quand le sous-arbre d'un arbre est sur le point d'être réduit, c'est-à-dire quand l'utilisateur clique sur le signe moins placé à côté de l'élément, ou quand le programme appelle la fonction :`collapse()`.

```
$tree_item->remove_subtree() ;
```

Cela enlève tous les enfants de sous-arbres de tree_item (donc les déréférence, les détruit, tous les sous-arbres des enfants et ainsi de suite...), puis le sous-arbre lui-même et cache le signe plus/moins.

3.16.8 Exemple

C'est un exemple un peu plus compliqué que les autres mais je pense que le moment est approprié. Bien qu'il illustre les widgets arbres, cet exemple utilise quelque peu les widget CList. A partir de là, nous pourrions facilement ajouter un bouton ou un menu et ajouter des fonctionnalités comme effacer, copier ou bouger un fichier ce qui nous ferait un file manager "light".




```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $root_dir = "/" ;
my @titles ;

my $window ;
my $pane ;
```

```
my $vbox ;
my $tree_scrolled_win ;
my $list_scrolled_win ;
my $entry ;
my $tree ;
my $leaf ;
my $subtree ;
my $item ;
my $list ;


$window = new Gtk : :Window( 'toplevel' ) ;
$window->set_usize( 725, 500 ) ;
$window->set_title( "File Viewer" ) ;
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ) ; } ) ;

$pane = new Gtk : :HPaned() ;
$window->add( $pane ) ;
$pane->set_handle_size( 10 ) ;
$pane->set_gutter_size( 8 ) ;
$pane->show() ;

# Crée une VBox pour l'entrée texte et la fenêtre défilable de l'arbre
$vbox = new Gtk : :VBox( $false, 0 ) ;
$pane->add1( $vbox ) ;
$vbox->show() ;

# Crée l'entrée texte
$entry = new Gtk : :Entry() ;
$vbox->pack_start( $entry, $false, $false, 4 ) ;
$entry->signal_connect( 'activate', \&entry_activate ) ;
$entry->show() ;

# Crée une fenêtre défilable pour l'arbre
$tree_scrolled_win = new Gtk : :ScrolledWindow( undef, undef ) ;
$tree_scrolled_win->set_usize( 150, 400 ) ;
$vbox->pack_start( $tree_scrolled_win, $true, $true, 0 ) ;
$tree_scrolled_win->set_policy( 'automatic', 'automatic' ) ;
$tree_scrolled_win->show() ;

# Crée une fenêtre défilable pour la liste
$list_scrolled_win = new Gtk : :ScrolledWindow( undef, undef ) ;
$pane->add2( $list_scrolled_win ) ;
$list_scrolled_win->set_policy( 'automatic', 'automatic' ) ;
$list_scrolled_win->show() ;

# Crée l'arbre racine
$tree = new Gtk : :Tree() ;
$tree_scrolled_win->add_with_viewport( $tree ) ;
$tree->set_selection_mode( 'single' ) ;
$tree->set_view_mode( 'item' ) ;
$tree->show() ;

# Crée le widget tree item racine
$leaf = new_with_label Gtk : :TreeItem( $root_dir ) ;
$tree->append( $leaf ) ;
$leaf->signal_connect( 'select', \&select_item, "/" ) ;
$leaf->set_user_data( "/" ) ;
```

```
$leaf->show();

# Crée le sous-arbre
if ( has_sub_trees( $root_dir ) )
  {
    $subtree = new Gtk : :Tree();
    $leaf->set_subtree( $subtree );
    $leaf->signal_connect( 'expand', \&expand_tree, $subtree );
    $leaf->signal_connect( 'collapse', \&collapse_tree );
    $leaf->expand();
  }

# Crée une boîte pour la liste
@titles = qw( Filename Size Permissions Owner Group Time Date );
$list = new_with_titles Gtk : :CList( @titles );
$list_scrolled_win->add_with_viewport( $list );
$list->set_column_width( 0, 100 );
$list->set_column_width( 1, 50 );
$list->set_column_width( 2, 80 );
$list->set_column_width( 3, 50 );
$list->set_column_width( 4, 50 );
$list->set_column_width( 5, 50 );
$list->set_column_width( 6, 100 );
$list->set_selection_mode( 'multiple' );
$list->set_shadow_type( 'none' );
$list->show();

$window->show();
main Gtk;
exit( 0 );



### Routines

# Rappel pour développer un arbre - trouve les sous répertoires et les
# ajoute à l'arbre
sub expand_tree
  {
    my ( $item, $subtree ) = @_;

    my $dir;
    my $dir_entry;
    my $path;
    my $item_new;
    my $new_subtree;

    $dir = $item->get_user_data();

    chdir( $dir );

    foreach $dir_entry ( <*> )
      {
        if ( -d $dir_entry )
          {
            $path = $dir . "/" . $dir_entry;
            $path =~ s|//|/|g;
            $item_new = new_with_label Gtk : :TreeItem( $dir_entry );
```

```
            $item_new->set_user_data( $path ) ;
            $item_new->signal_connect( 'select', \&select_item, $path ) ;
            $subtree->append( $item_new ) ;
            $item_new->show() ;

            if ( has_sub_trees( $path ) )
              {
                $new_subtree = new Gtk : :Tree() ;
                $item_new->set_subtree( $new_subtree ) ;
                $item_new->signal_connect( 'expand',
                                           \&expand_tree,
                                           $new_subtree ) ;
                $item_new->signal_connect( 'collapse', \&collapse_tree ) ;
              }
          }
      }
    chdir( ".." ) ;
  }


# Rappel pour réduire un arbre-- enlève le sous-arbre
sub collapse_tree
  {
    my ( $item ) = @_ ;

    my $subtree = new Gtk : :Tree() ;

    $item->remove_subtree() ;
    $item->set_subtree( $subtree ) ;
    $item->signal_connect( 'expand', \&expand_tree, $subtree ) ;
  }


# Teste si le répertoire possède des sous-répertoires
sub has_sub_trees
  {
    my ( $dir ) = @_ ;
    my $file ;
    my $has_dirs = $false ;

    foreach $file ( <$dir/*> )
      {
        $has_dirs = $true if ( -d $file ) ;
      }
    return ( $has_dirs ) ;
  }


# Appelé si on clique sur un élément de l'arbre
sub select_item
  {
    my ( $widget, $path ) = @_ ;

    my $file ;

    $entry->set_text( $path ) ;

    show_files( $path ) ;
```

```
  }


# Appelé si Entrée est pressée dans l'entrée texte
sub entry_activate
  {
    my ( $entry ) = @_;

    my $file;
    my $path = $entry->get_text();

    if ( -d $path )
      {
        show_files( $path );
      }
    else
      {
        $entry->set_text( "/" );
      }
  }


sub show_files
  {
    my ( $path ) = @_;

    my $file;

    $list->clear();

    for $file ( <$path/*> )
      {
        unless ( -d $file )
          {
            my ( $mode, $uid, $gid, $size, $mtime) =
              ( stat( $file ) )[ 2, 4, 5, 7, 9 ];

            my ( $mon, $day, $year, $hour, $min ) =
              ( localtime( $mtime ) )[ 4, 3, 5, 2, 1 ];

            $min = "0" . $min if ( $min < 10 );

            my $time = $hour . " :" . $min;
            my $month = ( "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                          "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" )[$mon];
            my $date = $month . " " . $day . ", " . ( 1900 + $year );

            my $user = getpwuid( $uid );
            my $group = getgrgid( $gid );

            $file =~ s|/.*/||g;
            $list->append( $file, $size, $mode, $user, $group, $time, $date );
          }
      }
  }
```

3.17 Les barres d'état

Les barres d'état sont de simples widgets utilisés pour afficher un message texte. Ils gardent une pile des messages qu'on leur donne ainsi, quand on enlève le message courant, le message précédent réapparaît.

Afin de permettre aux différentes parties du programme d'utiliser la même barre d'état, le widget barre d'état possède un identificateur de contexte qui, comme son nom l'indique, identifie les utilisateurs. Le message sur le haut de la pile est celui qui est affiché, sans s'occuper de son contexte. Les messages sont gérés dans la pile selon le principe du dernier entré - premier sorti et non en fonction de l'ordre des identificateurs de contexte.

Une barre d'état est créée grâce à l'appel :

```
$statusbar = new Gtk : :Statusbar() ;
```

Un nouvel identificateur de contexte est requis pour utiliser la fonction suivante ainsi qu'une brève description du contexte.

```
$statusbar->get_context_id( $context_description ) ;
```

Les trois fonctions suivantes peuvent opérer sur les barres d'état :

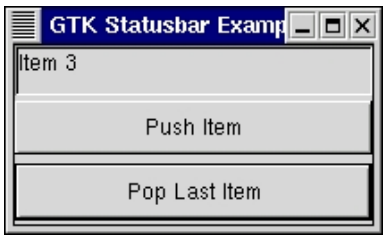
```
$statusbar->push( $context_id, $text ) ;
$statusbar->pop( $context_id ) ;
$statusbar->remove( $context_id, $message_id ) ;
```

La première, `push()`, est utilisée pour ajouter un nouveau message dans la barre d'état. Elle retourne une identificateur de Message, qui peut être passé plus tard à la fonction `$status->remove()` pour enlever le message avec le Message donné et l'identificateur de contexte de la pile de la barre d'état.

La fonction `pop()` enlève le plus haut message de la pile avec l'identificateur de contexte donné.

3.17.1 Exemple

L'exemple suivant crée une barre d'état avec deux boutons, un pour pousser les éléments dans la barre d'état, l'autre pour enlever le dernier élément placé dans la pile.




```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $window ;
my $vbox ;
my $button ;
my $statusbar ;
my $context_id ;
my $count = 1 ;

$window = new Gtk : :Window( "toplevel" ) ;
$window->set_usize( 200, 100 ) ;
$window->set_title( "GTK Statusbar Example" ) ;
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ) ; } ) ;
```

```
$vbox = new Gtk : :VBox( $false, 1 ) ;
$window->add( $vbox ) ;
$vbox->show() ;

$statusbar = new Gtk : :Statusbar() ;
$vbox->pack_start( $statusbar, $true, $true, 0 ) ;
$statusbar->show() ;

$context_id = $statusbar->get_context_id( "Statusbar Example" ) ;

$button = new Gtk : :Button( "Push Item" ) ;
$button->signal_connect( "clicked", \&push_item, $context_id ) ;
$vbox->pack_start( $button, $true, $true, 2 ) ;
$button->show() ;

$button = new Gtk : :Button( "Pop Last Item" ) ;
$button->signal_connect( "clicked", \&pop_item, $context_id ) ;
$vbox->pack_start( $button, $true, $true, 2 ) ;
$button->show() ;

$window->show() ;

main Gtk ;
exit( 0 ) ;



### Routines

sub push_item
  {
    my ( $widget, $context_id ) = @_ ;

    my $buff = ( "Item " . $count++ ) ;
    $statusbar->push( $context_id, $buff ) ;
  }

sub pop_item
  {
    my ( $widget, $context_id ) = @_ ;

    $statusbar->pop( $context_id ) ;
  }
```

3.18 La sélection des couleurs

Le widget de sélection de couleurs est, ce n'est pas surprenant, un widget pour sélectionner les couleurs de manière interactive. Ce widget composé laisse l'utilisateur choisir une couleur en manipulant les triplés RGB (Rouge, Vert, Bleu) ou HVS (Teinte, Saturation, Valeur) On le fait soit en ajustant chacune des valeurs avec un curseur ou avec une entrée texte soit en pointant sur la couleur désirée dans le cercle de teinte-saturation et dans la barre de valeur.

Le widget de sélection de couleurs n'émet qu'un seul signal `color_changed` qui est émis si la couleur courante du widget change ou quand l'utilisateur la change ou si elle est choisie explicitement à l'aide de la fonction `selection_set_color()`.

Regardons ce que le widget de sélection de couleurs nous propose. Le widget existe sous deux moutures : `Gtk::ColorSelection` et `Gtk::ColorSelectionDialog`.

```
$color = new Gtk : :ColorSelection() ;
```

Vous n'utiliserez probablement jamais ce constructeur directement. Il crée juste un widget de sélection de couleurs orphelin que vous aurez à parenter vous-même. Le widget de sélection de couleurs hérite du widget VBox.

```
$color = new Gtk : :ColorSelectionDialog ( $title) ;
```

C'est le constructeur le plus commun. Il crée une fenêtre de dialogue de sélection de couleur. Elle est composée d'un cadre contenant un widget de sélection de couleurs, un séparateur horizontal, une HBox avec trois boutons, "Ok", "Cancel" et "Help". Vous pouvez atteindre ces boutons en accédant aux widgets `ok_button`, `cancel_button` et `help_button` dans la structure du widget de sélection de couleurs. (i.e., `$colordialog->ok_button`).

```
$colordialog->set_update_policy ( $policy ) ;
```

Cette fonction déclare la politique de mise à jour. La politique par défaut est `'continuous'` qui signifie que la couleur courante est mise à jour continuellement quand l'utilisateur prend le curseur, clique sur la souris, se déplace dans le cercle de teinte-saturation ou la barre de valeur. Si vous rencontrez des problèmes de performance, n'hésitez pas à choisir `'discontinuous'` ou `'delayed'`.

```
$colordialog->set_opacity ( $use_opacity) ;
```

Le widget de sélection de couleurs supporte le réglage de l'opacité d'une couleur (également connu comme courbe alpha). Ce n'est pas disponible par défaut. Appeler cette fonction avec une valeur de `$use_opacity` permet l'opacité. De même, une valeur fausse rend l'opacité indisponible.

```
$colordialog->set_color ( $color ) ;
```

Vous pouvez déclarer explicitement la couleur courante en appelant cette fonction avec une liste de couleur. La longueur de l'array, dépend si l'opacité est disponible ou non. La position 0 contient la composante de rouge, 1 celle de vert, 2 celle de bleu (seulement si l'opacité est disponible). Toutes les valeurs sont comprises entre 0.0 et 1.0.

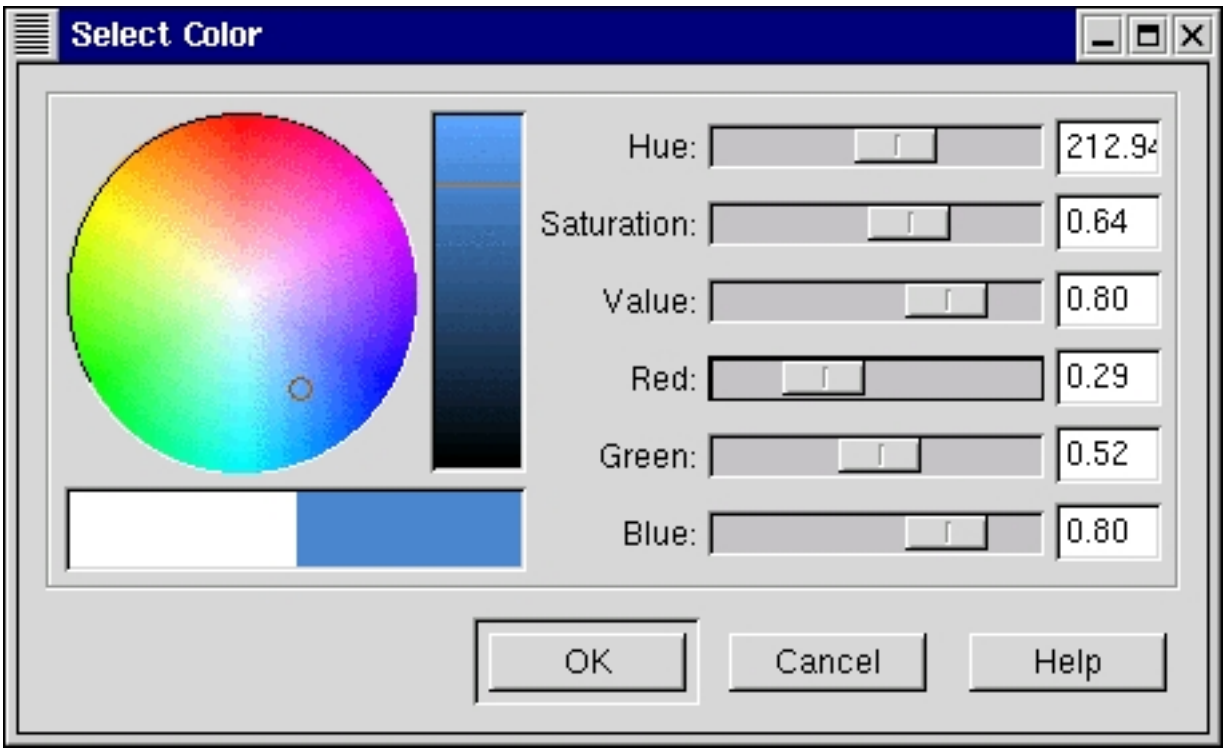
```
$colordialog->get_color() ;
```

Quand vous voulez vous renseigner sur la couleur actuelle, typiquement quand vous avez reçu le signal `color_changed`, vous pouvez utiliser cette fonction. Une liste de couleurs à remplir est retournée. Voir `set_color()` pour avoir une description de cette liste.

3.18.1 Exemple

Voici un exemple simple montrant l'usage d'une fenêtre de dialogue de sélection de couleurs. Ce programme affiche une fenêtre contenant une zone de dessin. Si vous cliquez dessus ouvre une fenêtre de sélection de couleurs et changer la couleur dans cette fenêtre de dialogue change la couleur de fond de la zone de dessin. En bonus, il affiche les couleurs au format hexadécimal utilisé en HTML. Avec ce format, les deux premiers caractères sont la valeur de la composante rouge, les deux suivants celle de vert et les deux derniers celle de bleu.

Un bon exercice serait de changer le label en une entrée texte afin de vous permettre de saisir une couleur HTML et de changer la couleur de la zone de dessin en accord. Vous pourriez aussi vérifier que la couleur HTML entrée est valide.




```
#!/usr/bin/perl -w


use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $dialog_shown = $false;

my $window;
my $mainbox;
my $drawingarea;
my $buttonbox;
my $button;
my $label;
my $colorsel;
my $colormap;
my $gdk_color;

$window = new Gtk::Window( "toplevel" );
$window->set_title( "Color Selection Test" );
$window->border_width( 10 );
$window->realize();
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ); } );

$mainbox = new Gtk::VBox( $false, 10 );
$window->add( $mainbox );
$mainbox->show();

# Crée un bouton avec une aire de dessin à l'intérieur
```

```
$button = new Gtk : :Button() ;
$button->signal_connect( "clicked", \&select_color ) ;
$button->show() ;
$mainbox->pack_start( $button, $false, $false, 0 ) ;

$buttonbox = new Gtk : :VBox( $false, 0 ) ;
$buttonbox->show() ;
$button->add( $buttonbox ) ;

$drawingarea = new Gtk : :DrawingArea() ;
$drawingarea->size( 32, 32 ) ;
$drawingarea->show() ;
$buttonbox->pack_start( $drawingarea, $false, $false, 0 ) ;

$colormap = $drawingarea->window->get_colormap() ;
$gdk_color = Gtk : :Gdk : :Color->parse_color( "black" ) ;
$gdk_color = $colormap->color_alloc( $gdk_color ) ;
$drawingarea->window->set_background( $gdk_color ) ;

$label = new Gtk : :Label( "HTML color :  #FFFFFF" ) ;
$label->show() ;
$mainbox->pack_start( $label, $false, $false, 0 ) ;

$window->show() ;

main Gtk ;
exit( 0 ) ;



### Routines

# gestionnaire d'évènement de l'aire de dessin
sub select_color
  {
    my ( $drawingarea ) = @_ ;

    my $colordialog = "" ;

    unless ( $dialog_shown )
      {
        # Crée une fenêtre de dialogue de sélection de couleur
        $colordialog = new Gtk : :ColorSelectionDialog( "Select Color") ;

        $dialog_shown = $true ;

        # Récupère le widget de sélection de couleurs
        $colorsel = $colordialog->colorsel ;

        $colordialog->ok_button->signal_connect( "clicked",
                                                 \&close_dialog,
                                                 $colordialog ) ;
        $colordialog->cancel_button->signal_connect( "clicked",
                                                     \&close_dialog,
                                                     $colordialog ) ;

        # Connecte le signal "color_changed" et déclare la donnée cliente
        # au widget colorsel
```

```
        $colorsel->signal_connect( "color_changed", \&color_changed_cb ) ;

        # Montre la fenêtre de dialogue
        $colordialog->show() ;
      }
  }


sub color_changed_cb
  {
    my ( $colorsel ) = @_ ;
    my @color ;
    my $red ;
    my $green ;
    my $blue ;

    @color = $colorsel->get_color() ;

    $gdk_color->{ 'red' } = $color[0] * 65535.0 ;
    $gdk_color->{ 'green' } = $color[1] * 65535.0 ;
    $gdk_color->{ 'blue' } = $color[2] * 65535.0 ;

    # Convertit au format HTML
    $red = $gdk_color->{ 'red' } / 256 ;
    $green = $gdk_color->{ 'green' } / 256 ;
    $blue = $gdk_color->{ 'blue' } / 256 ;

    $red = uc( sprintf( "%lx", $red ) ) ;
    $green = uc( sprintf( "%lx", $green ) ) ;
    $blue = uc( sprintf( "%lx", $blue ) ) ;

    $red = "0" . $red if ( $red =~ /^\d$/ ) ;
    $green = "0" . $green if ( $green =~ /^\d$/ ) ;
    $blue = "0" . $blue if ( $blue =~ /^\d$/ ) ;

    $label->set_text( "HTML color :  #" . $red . $green . $blue . "\n" ) ;

    $gdk_color = $colormap->color_alloc( $gdk_color ) ;
    $drawingarea->window->set_background( $gdk_color ) ;
    $drawingarea->window->clear() ;
  }

sub close_dialog
  {
    my ( $button, $colordialog ) = @_ ;

    $colordialog->hide() ;
    $dialog_shown = $false ;
  }
```

3.19 Les règles

Les règles sont utilisées pour indiquer la position du pointeur de souris dans une fenêtre donnée. Une fenêtre peut avoir une règle horizontale qui s'étire sur toute la largeur et une règle verticale qui s'étire sur toute la hauteur de la fenêtre. Un petit indicateur triangulaire sur la règle montre la position exacte du pointeur.

Une règle doit d'abord être créée. Les règles horizontales et verticales sont créées par :

```
$hruler = new Gtk : :HRuler() ;
```

```
$vruler = new Gtk : :VRuler() ;
```

Une fois que la règle est créée, nous pouvons définir l'unité de mesure. Les unités possibles sont `'pixels'`, `'inches'`, `'centimeters'`. Ce choix se fait avec :

```
$ruler->set_metric($metric) ;
```

L'unité de mesure par défaut est `'pixels'`.

Une autre caractéristique important des règles est la possibilité d'indiquer les graduations et de placer l'indicateur initialement. Ces réglages se font en utilisant :

```
$ruler->set_range( $lower , $upper , $position , $max_size ) ;
```

Les arguments `$lower` et `$upper` définit la longueur de la règle et `$max_size` est le plus grand nombre possible qui sera affiché. `$position` définit la position initiale de l'indicateur de pointeur sur la règle.

Une règle verticale peut mesurer une fenêtre de 800 pixels de haut ainsi :

```
$vruler->set_range( 0 , 800 , 0 , 800 ) ;
```

Les marques affichées sur la règle iront de 0 à 800 et un nombre sera affiché tous les 100 pixels. Si, à la place, vous désirez une règle qui mesure entre 7 et 16, vous coderez :

```
$vruler->set_range( 7 , 16 , 0 , 20 ) ;
```

L'indicateur est une petite marque triangulaire qui indique la position du pointeur relatif à la règle. Si cette règle est utilisée pour suivre le pointeur de la souris, le signal `''motion_notify_event''` doit être connecté à la méthode `''motion_notify_event''` de la règle. Pour suivre tous les mouvements de la souris à l'intérieur d'une surface, vous pouvez utiliser :

```
$surface->signal_connect(''motion_notify_event'',
                         sub{ $ruler->motion_notify_event ( $_[1] ) ; } ) ;
```

3.19.1 Exemple

L'exemple suivant crée une aire de dessin avec une règle horizontale en haut et une verticale à gauche. La taille de l'aire de dessin est de 600 pixels de large sur 400 pixels de haut. La règle horizontale mesure de 7 à 13 avec une marque tous les 100 pixels alors que la règle verticale mesure de 0 à 400 avec une marque tous les 100 pixels. Le placement de l'aire de dessin se fait en utilisant une table.

Si vous regardez bien, vous remarquerez que le pointeur de la souris n'apparaît pas sur l'image ci-dessus, mais vous pouvez dire où il est grâce aux indicateurs sur les règles. Je chercherais plus tard comment faire apparaître le pointeur dans un screenshot.

```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
```

```
my $true = 1 ;

my $xsize = 600 ;
my $ysize = 400 ;

my $window ;
my $table ;
my $area ;
my $hrule ;
my $vrule ;

$window = new Gtk : :Window( "toplevel" ) ;
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ) ; } ) ;
$window->border_width( 10 ) ;

# Crée une table pour y placer les règles et l'aire de dessin
$table = new Gtk : :Table( 3, 2, $false ) ;
$window->add( $table ) ;

$area = new Gtk : :DrawingArea() ;
$area->size( $xsize, $ysize ) ;
$table->attach( $area, 1, 2, 1, 2, [ 'expand', 'fill' ], 'fill', 0, 0 ) ;
$area->set_events( [ 'pointer_motion_mask', 'pointer_motion_hint_mask' ] ) ;

# La règle horizontale va en haut. Quand la souris traverse l'aire de dessin,
# un motion_notify_event est passé au gestionnaire de signal approprié
# de la règle
$hrule = new Gtk : :HRuler() ;
$hrule->set_metric( 'pixels' ) ;
$hrule->set_range( 7, 13, 0, 20 ) ;
$area->signal_connect( "motion_notify_event",
                       sub { $hrule->motion_notify_event( $_[1] ) ; } ) ;
$table->attach( $hrule, 1, 2, 0, 1,
                [ 'expand', 'shrink', 'fill' ],
                'fill', 0, 0 ) ;
# La règle  verticale va à gauche. Quand la souris traverse l'aire de dessin,
# un motion_notify_event est passé au gestionnaire de signal approprié
# de la règle
$vrule = new Gtk : :VRuler() ;
$vrule->set_metric( 'pixels' ) ;
$vrule->set_range( 0, $ysize, 10, $ysize ) ;
$area->signal_connect( "motion_notify_event",
                       sub { $vrule->motion_notify_event( $_[1] ) ; } ) ;
$table->attach( $vrule, 0, 1, 1, 2,
                [ 'fill', 'expand', 'shrink' ],
                'fill', 0, 0 ) ;

# On montre tout
$area->show() ;
$hrule->show() ;
$vrule->show() ;
$table->show() ;
$window->show() ;

main Gtk ;
exit( 0 ) ;
```

3.20 Les flèches

Le widget flèche dessine une pointe de flèche. On a le choix entre un certain nombre de directions possibles et une certain nombre de styles. Placée dans un bouton, une flèche peut être très utile. Comme le widget Label, elle n'émet aucun signal.

Il n'y a que deux fonction pour manipuler les flèches.

```
$arrow = new Gtk : :Arrow ( $arrow_type , $shadow_type ) ;
$arrow ->set ( $arrow_type , $shadow_type ) ;
```

La première crée une nouvelle flèche en précisant le type et l'apparence. La seconde permet à ces valeurs d'être modifiées rétrospectivement.

L'argument `$arrow_type` peut prendre l'une des valeurs suivantes :

```
'up'
'down'
'left'
'right'
```

Ces valeurs indiquent évidemment la direction dans laquelle pointe la flèche.

L'argument `$shadow_type` peut prendre les valeurs :

```
'in'
'out' ( valeur par défaut )
'etched in'
'etched out'
```

3.20.1 Exemple

Voici un bref exemple pour illustrer l'usage des flèches.


```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window;
my $button;
my $box;

$window = new Gtk : :Window( "toplevel" ) ;
$window->set_title( "Arrow Buttons" ) ;
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ) ; } ) ;
$window->border_width( 10 ) ;

# Crée une boîte qui contiendra les boutons-flèches
$box = new Gtk : :HBox( $false, 0 ) ;
$box->border_width( 2 ) ;
$window->add( $box ) ;
```

```
# Regroupe et montre tous les widgets
$box->show() ;

$button = create_arrow_button( 'up', 'in' ) ;
$box->pack_start( $button, $false, $false, 3 ) ;

$button = create_arrow_button( 'down', 'out' ) ;
$box->pack_start( $button, $false, $false, 3 ) ;

$button = create_arrow_button( 'left', 'in' ) ;
$box->pack_start( $button, $false, $false, 3 ) ;

$button = create_arrow_button( 'right', 'out' ) ;
$box->pack_start( $button, $false, $false, 3 ) ;

$window->show() ;
main Gtk ;
exit( 0 ) ;



### Routines

# Crée un widget flèche avec les paramètres spécifiés et la
# place dans un bouton
sub create_arrow_button
  {
    my ( $arrow_type, $shadow_type ) = @_ ;

    my $button ;
    my $arrow ;

    $button = new Gtk : :Button() ;
    $arrow = new Gtk : :Arrow( $arrow_type, $shadow_type ) ;

    $button->add( $arrow ) ;
    $button->show() ;
    $arrow->show() ;

    return ( $button ) ;
  }
```

3.21 Les bulles d'aide

Ce sont des petits textes qui apparaissent quand vous laissez votre pointeur immobile pendant quelques secondes sur un bouton ou un autre widget.

Les widgets qui ne reçoivent pas d'évènements (ceux qui n'ont pas leur propre fenêtre) ne fonctionneront pas avec les bulles d'aide.

Le premier appel que vous utilisez crée une bulle d'aide. Vous avez juste besoin de le faire une fois pour un ensemble de bulles d'aide. Cette fonction peut donc être utilisée pour créer plusieurs bulles d'aide.

```
new Gtk : :Tooltips() ;
```

Une fois que vous avez créé une nouvelle bulle d'aide ainsi que le widget sur lequel vous voulez l'utiliser, il vous suffit d'appeler :

```
$tooltips->set_tip ( $widget , $tip_text , $tip_private ) ;
```

Le premier argument est le widget sur lequel vous voulez que la bulle d'aide apparaisse et le suivant le texte de la bulle d'aide. Le dernier argument est une chaîne de caractères qui peut être utilisée comme un identificateur quand

on utilise le widget GTK TipsQuery pour implémenter l'aide sensitive du contexte. Pour le moment, vous pouvez le laisser blanc.

Voici un court exemple :

```
$button = new Gtk : :Button(‘‘Bouton’’) ;
$tooltips = new Gtk : :Tooltips() ;
$tooltips->set_tip ( $button , ‘‘C’est un bouton’’ , ‘‘’’ ) ;
```

Vous pouvez utiliser d'autres appels avec les bulles d'aide. Je vais juste les lister avec une brève description de ce qu'ils peuvent faire.

Pour rendre disponible un ensemble de bulles d'aide indisponibles :

```
$tooltips->enable() ;
```

Pour rendre indisponible un ensemble de bulles d'aide disponibles :

```
$tooltips->disable() ;
```

Pour déclarer le délai en `millisecondes`, avant que n'apparaisse la bulle d'aide.

```
$tooltips->set_delay ( $delay ) ;
```

La valeur par défaut est 500 millisecondes (une demie seconde).

On a ainsi vu toutes les fonctions associées aux bulles d'aide. Bien plus que ce vous voudrez savoir :-)

3.22 Le calendrier

Le widget calendrier est une moyen efficace d'afficher et de retrouver des informations liées aux dates. C'est un widget très simple à créer et à utiliser.

Créer un widget calendrier est aussi simple que :

```
$calendar = new Gtk : :Calendar() ;
```

Il arrive parfois que vous ayez beaucoup d'informations à changer à l'intérieur de ce widget. Les fonctions suivantes vous permettent de faire de multiples changements sans que l'utilisateur voient les multiples mises à jour à l'écran :

```
$calendar->freeze() ;
$calendar->thaw() ;
```

Elles fonctionnent comme les fonctions `freeze` et `thaw` des autres widgets.

Le widget calendrier possède quelques options qui vous permettent de changer le look du widget ainsi que la manière dont il opère :

```
$calendar->display_options( $flags ) ;
```

L'argument `$flag` peut être formé en combinant les cinq options suivantes :

`'heading'` - cette option spécifie que le mois et l'année doivent être montrés quand on dessine le calendrier.

`'show_day_names'` - cette option spécifie que les trois premières lettres de chaque jour doivent être affichées (e.g. MON, TUE,...)

`'no_month_change'` - cette option stipule que l'utilisateur ne devrait pas et ne peut pas changer le mois affiché. Cela peut être bon si vous avez seulement besoin d'un mois particulier, par exemple quand vous afficher 12 widgets calendrier pour chaque mois d'une année particulière.

`'show_week_numbers'` - cette option spécifie que le numéro de chaque semaine doit être affiché sous le côté gauche sur calendrier (e.g. Jan 1 = Week 1, Dec 31 = Week 52).

`week_start_monday'` - cette option stipule que le premier jour de la semaine est lundi à la place de dimanche qui est la valeur par défaut. Cela affecte uniquement l'ordre dans lequel sont affichés les jours de la gauche vers la droite.

Les fonctions suivantes sont utilisées pour déclarer la date courante affichée :

```
$calendar->selected_month ( $ mois , $année ) ;
$calendar->selected_day ( $jour ) ;
```

La valeur de retour de `selected_month` est une valeur booléenne qui indique si la sélection est réussie.

Avec `selected_day`, le nombre spécifié est sélectionné à l'intérieur du mois courant, si c'est possible. Une valeur `$jour` de 0 déselectionnera la sélection courante.

En plus d'avoir un jour sélectionné, n'importe quel nombre de jour dans le mois peuvent être “marqués”. Un jour marqué est surligné dans l'affichage du calendrier. Les fonctions suivantes sont fournies pour manipuler les jours marqués :

```
$calendar->mark_day ( $jour ) ;
$calendar->unmark_day ( $jour ) ;
$calendar->clear_marks() ;
```

Les jours marqués actuellement sont stockés dans un tableau. Ce tableau est composé de 31 éléments ainsi si vous voulez savoir si un jour est marqué, vous devez accéder à l'élément correspondant du tableau (n'oubliez pas que les éléments d'un tableau sont numérotés de 0 à n-1). Par exemple :

```
if ( $calendar->marked_date [ $jour - 1 ] )
   {
    print ( ‘‘Le jour $jour est marqué.\n ) ;
   }
```

Notez que les marques sont persistantes à travers les changements de mois et d'années.

La dernière fonction concernant le calendrier est utilisée pour retrouver la date courante sélectionnée.

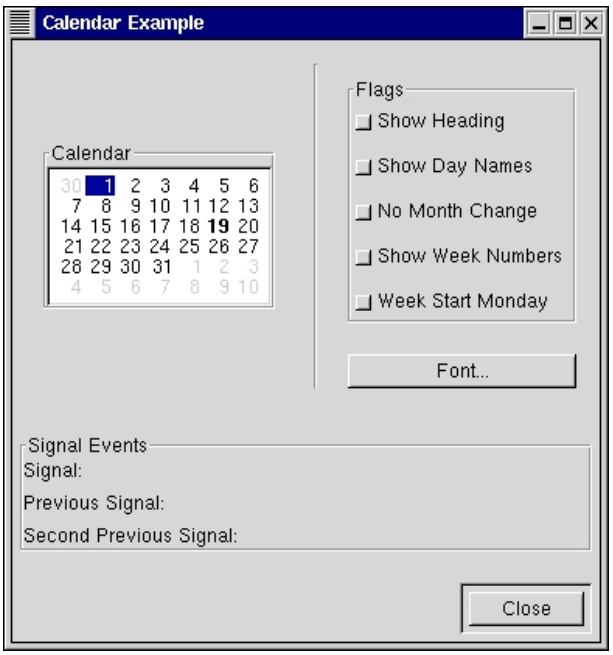
```
( $ année , $mois , $jour ) = $calendar->get_date() ;
```

Le widget calendrier peut générer un nombre de signaux indiquant les sélections de dates et les changements. Les noms des signaux sont très explicites :

`'month_changed'` - mois changé
`'day_selected'` - jour sélectionné
`'day_selected_double_click'` - jour sélectionné par un double clic
`'prev_month'` - mois précédent
`'next_month'` - mois suivant
`'prev_year'` - année précédente
`'next_year'` - année suivante

3.22.1 Exemple

Il ne nous reste plus qu'à mettre ensemble toutes ces fonctions. On obtient :




```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

Gtk->set_locale() ;
init Gtk ;
```

```
my $false = 0 ;
my $true = 1 ;
my $year_base = 1900 ;
my $i ;

my @flags = ( "Show Heading",     # Montre les titres
              "Show Day Names",   # Montre les noms des mois
              "No Month Change",   # Pas de changement de mois
              "Show Week Numbers",  # Montre les numéros de semaine
              "Week Start Monday" ) ;   # La semaine commence Lundi

my %calendar_data = ( "", "" ) ;

my $window ;
my $vbox ;
my $vbox2 ;
my $vbox3 ;
my $hbox ;
my $hbbox ;
my $calendar ;
my $toggle ;
my $button ;
my $frame ;
my $separator ;
my $label ;
my $bbox ;


$calendar_data{ "window" } = undef ;
$calendar_data{ "font" } = undef ;
$calendar_data{ "font_dialog" } = undef ;
$calendar_data{ "settings" } = [ 0, 0, 0, 0, 0 ] ;

$window = new Gtk : :Window( 'toplevel' ) ;
$window->set_title( "Calendar Example" ) ;
$window->border_width( 5 ) ;
$window->signal_connect( 'destroy', sub { Gtk->exit( 0 ) ; } ) ;
$window->signal_connect( 'delete-event', \&gtk_false ) ;
$window->set_policy( $false, $false, $true ) ;

$vbox = new Gtk : :VBox( $false, 10 ) ;
$window->add( $vbox ) ;

# La partie haute de la fenêtre, Calendrier, flags et fontsel.

$hbox = new Gtk : :HBox( $false, 10 ) ;
$vbox->pack_start( $hbox, $true, $true, 10 ) ;

$hbbox = new Gtk : :HButtonBox() ;
$hbox->pack_start( $hbbox, $false, $false, 10 ) ;
$hbbox->set_layout( 'spread' ) ;
$hbbox->set_spacing( 5 ) ;

# Le widget calendrier
$frame = new Gtk : :Frame( "Calendar" ) ;
$hbbox->pack_start( $frame, $false, $true, 10 ) ;

$calendar = new Gtk : :Calendar() ;
```

```
$calendar_data{ "window" } = $calendar;
calendar_set_flags();
$calendar->mark_day( 19 );
$frame->add( $calendar );
$calendar->signal_connect( 'month_changed',
                           \&calendar_month_changed );
$calendar->signal_connect( 'day_selected',
                           \&calendar_day_selected );
$calendar->signal_connect( 'day_selected_double_click',
                           \&calendar_day_selected_double_click );
$calendar->signal_connect( 'prev_month',
                           \&calendar_prev_month );
$calendar->signal_connect( 'next_month',
                           \&calendar_next_month );
$calendar->signal_connect( 'prev_year',
                           \&calendar_prev_year );
$calendar->signal_connect( 'next_year',
                           \&calendar_next_year );

$separator = new Gtk : :VSeparator();
$hbox->pack_start( $separator, $false, $true, 0 );

$vbox2 = new Gtk : :VBox( $false, 10 );
$hbox->pack_start( $vbox2, $false, $false, 10 );

# Construit le bon cadre avec le flag à l'intérieur
$frame = new Gtk : :Frame( "Flags" );
$vbox2->pack_start( $frame, $true, $true, 10 );

$vbox3 = new Gtk : :VBox( $true, 5 );
$frame->add( $vbox3 );

for ( $i = 0; $i < 5; $i++ )
  {
    $toggle = new Gtk : :CheckButton( $flags[ $i ] );
    $toggle->signal_connect( 'toggled', \&calendar_toggle_flag );
    $vbox3->pack_start( $toggle, $true, $true, 0 );
    $calendar_data{ 'flag_checkboxes' }[ $i ] = $toggle;
  }

# Contruit le bon bouton de police
$button = new Gtk : :Button( "Font..." );
$button->signal_connect( 'clicked', \&calendar_select_font );
$vbox2->pack_start( $button, $false, $false, 0 );

#  Construit la partie signaux et évènements
$frame = new Gtk : :Frame( "Signal Events" );
$vbox->pack_start( $frame, $true, $true, 10 );

$vbox2 = new Gtk : :VBox( $true, 5 );
$frame->add( $vbox2 );

$hbox = new Gtk : :HBox( $false, 3 );
$vbox2->pack_start( $hbox, $false, $true, 0 );
$label = new Gtk : :Label( "Signal :" );
$hbox->pack_start( $label, $false, $true, 0 );
$calendar_data{ "last_sig" } = new Gtk : :Label( "" );
$hbox->pack_start( $calendar_data{ "last_sig" }, $false, $true, 0 );
```

```
$hbox = new Gtk : :HBox( $false, 3 ) ;
$vbox2->pack_start( $hbox, $false, $true, 0 ) ;
$label = new Gtk : :Label( "Previous Signal :" ) ;
$hbox->pack_start( $label, $false, $true, 0 ) ;
$calendar_data{ "prev_sig" } = new Gtk : :Label( "" ) ;
$hbox->pack_start( $calendar_data{ "prev_sig" }, $false, $true, 0 ) ;

$hbox = new Gtk : :HBox( $false, 3 ) ;
$vbox2->pack_start( $hbox, $false, $true, 0 ) ;
$label = new Gtk : :Label( "Second Previous Signal :" ) ;
$hbox->pack_start( $label, $false, $true, 0 ) ;
$calendar_data{ "prev2_sig" } = new Gtk : :Label( "" ) ;
$hbox->pack_start( $calendar_data{ "prev2_sig" }, $false, $true, 0 ) ;

$bbox = new Gtk : :HButtonBox() ;
$vbox->pack_start( $bbox, $false, $false, 0 ) ;
$bbox->set_layout( 'end' ) ;

$button = new Gtk : :Button( "Close" ) ;
$button->signal_connect( 'clicked', sub { Gtk->exit( 0 ) ; } ) ;
$bbox->add( $button ) ;
$button->can_default( $true ) ;
$button->grab_default() ;

$window->show_all() ;

main Gtk ;
exit( 0 ) ;




### Routines

sub calendar_date_to_string
  {
    my $string ;
    my $year ;
    my $month ;
    my $day ;

    my @months = qw( January February March     April   May      June
                     July    August   September October November December ) ;

    ( $year, $month, $day ) = $calendar_data{ 'window' }->get_date() ;

    $string = $months[ $month ] . " " . $day . ", " . $year ;

    return ( $string ) ;
  }

sub calendar_set_signal_strings
  {
    my ( $sig_str ) = @_ ;

    my $prev_sig ;
```

```
    $prev_sig = $calendar_data{ 'prev_sig' }->get();
    $calendar_data{ 'prev2_sig' }->set( $prev_sig );

    $prev_sig = $calendar_data{ 'last_sig' }->get();
    $calendar_data{ 'prev_sig' }->set( $prev_sig );

    $calendar_data{ 'last_sig' }->set( $sig_str );
  }

sub calendar_month_changed
  {
    my $buffer = "month_changed : " . calendar_date_to_string();

    calendar_set_signal_strings( $buffer );
  }

sub calendar_day_selected
  {
    my $buffer = "day_selected : " . calendar_date_to_string();

    calendar_set_signal_strings( $buffer );
  }

sub calendar_day_selected_double_click
  {
    my $year;
    my $month;
    my $day;
    my $buffer;

    $buffer = "day_selected_double_click : " . calendar_date_to_string();

    calendar_set_signal_strings( $buffer );

    ( $year, $month, $day ) = $calendar_data{ 'window' }->get_date();

    if ( $calendar_data{ 'window' }->marked_date( $day - 1 ) == 0 )
      {
        $calendar_data{ 'window' }->mark_day( $day );
      }
    else
      {
        $calendar_data{ 'window' }->unmark_day( $day );
      }
  }

sub calendar_prev_month
  {
    my $buffer = "prev_month : " . calendar_date_to_string();

    calendar_set_signal_strings( $buffer );
  }

sub calendar_next_month
  {
    my $buffer = "next_month : " . calendar_date_to_string();

    calendar_set_signal_strings( $buffer );
```

```
  }

sub calendar_prev_year
  {
    my $buffer = "prev_year : " . calendar_date_to_string();

    calendar_set_signal_strings( $buffer );
  }

sub calendar_next_year
  {
    my $buffer = "next_year : " . calendar_date_to_string();

    calendar_set_signal_strings( $buffer );
  }

sub calendar_set_flags
  {
    my $i;
    my $options = 0;

    for ( $i = 0; $i < 5; $i++ )
      {
        if ( $calendar_data{ 'settings' }[ $i ] )
          {
            $options = $options + ( 1 << $i );
          }
      }
    if ( $calendar_data{ 'window' } )
      {
        $calendar_data{ 'window' }->display_options( $options );
      }
  }

sub calendar_toggle_flag
  {
    my ( $toggle ) = @_;

    my $i;
    my $j = 0;

    for ( $i = 0; $i < 5; $i++ )
      {
        if ( $calendar_data{ 'flag_checkboxes' }[ $i ] == $toggle )
          {
            $j = $i;
          }
      }

    $calendar_data{ 'settings' }[$j] = not $calendar_data{ 'settings' }[$j];
    calendar_set_flags();
  }

sub calendar_font_selection_ok
  {
    ### Il y a une erreur dans cette fonction. Aucune erreur n'est reportée
    ### Mais la police ne change pas. Envoyez moi la solution si vous l'avez
    ### à swilhelm@computer.org.
```

```
    ### Désolé pour les inconvénients.

    my ( $button ) = @_ ;

    my $style ;
    my $font ;

    $calendar_data{'font'} = $calendar_data{ 'font_dialog' }->get_font_name() ;

    if ( $calendar_data{ 'window' } )
      {
        $font = $calendar_data{ 'font_dialog' }->get_font() ;

        if ( $font )
          {
            $style = $calendar_data{ 'window' }->get_style()->copy() ;
            $style->{ 'font' } = $font ;
            $calendar_data{ 'window' }->set_style( $style ) ;
          }
      }
  }

sub calendar_select_font
  {
    my ( $button ) = @_ ;

    my $window ;

    if ( $calendar_data{ 'font_dialog' } )
      {
        $window = $calendar_data{ 'font_dialog' } ;
      }
    else
      {
        $window = new Gtk : :FontSelectionDialog( "Font Selection Dialog" ) ;

        $window->position( 'mouse' ) ;

        $window->signal_connect( 'destroy', sub { $window->destroyed() ; } ) ;

        $window->ok_button->signal_connect( 'clicked',
                                            \&calendar_font_selection_ok ) ;
        $window->cancel_button->signal_connect( 'clicked',
                                                sub { $window->hide() ; } ) ;

        $calendar_data{ 'font_dialog' } = $window ;
      }

    if ( $window->visible )
      {
        $window->destroy() ;
      }
    else
      {
        $window->show() ;
      }
}
```

3.23 Pixmaps

Les pixmaps sont des structures de données qui contiennent des images. Ces images peuvent être utilisées en un nombre varié d'endroits, mais le plus souvent on les utilise comme icône ou curseur.

Un pixmap à deux couleurs s'appelle un bitmap, il existe des routines supplémentaires pour manipuler ce cas particulier.

Pour comprendre les pixmaps, il peut être utile de comprendre comment fonctionne le système X window. Sous X, les applications n'ont pas besoin de tourner sur l'ordinateur sur lequel vous utilisez celle-ci. En revanche, de nombreuses applications, appelées "clientes" , communiquent toutes avec un programme qui gèrent l'affichage, la souris et le clavier. Ce programme qui interagit directement avec l'utilisateur est appelé un "serveur d'affichage " ou "serveur X". Puisque les communications peuvent avoir lieu sur un réseau, il est important de conserver des informations à l'aide du serveur X. Les pixmaps, par exemple ; sont stockés dans la mémoire du serveur X. Cela signifie, qu'une fois que les valeurs du pixmaps sont définies, vous n'avez pas à passer votre temps à les transmettre sur le réseau ; on envoie en revanche la commande " afficher le pixmap numéro XYZ ici ". Même si pour le moment vous n'utilisez pas X avec GTK, utiliser des constructions comme les pixmaps feront que vos programmes fonctionneront acceptablement sous X.

Pour utiliser des pixmaps en GTK, vous devrez d'abord construire une structure de Pixmap en utilisant des routines du niveau GDK. Les Pixmaps peuvent être créés à partir de données stockées en mémoire ou bien lus à partir d'un fichier. Nous parcourrons les deux méthodes pour créer nos pixmaps.

```
$gdkpixmap = Gtk : :Gdk : :Pixmap->create_from_data( $window,
                                                     $data,
                                                     $width,
                                                     $height ) ;
```

Cette routine sert à créer une pixmap de simple niveau (2 couleurs) à partir de données stockées en mémoire. Chaque bit de la donnée indique si ce pixel est allumé ou non. La largeur et la hauteur sont en pixels. Le pointeur GdkWindow est sur la fenêtre courante puisque les ressources d'un pixmap n'ont de signification que dans le contexte de l'écran où il est affiché.

```
$gdkpixmap = Gtk : :Gdk : :Pixmap->create_from_data( $window,
                                                     $data,
                                                     $width,
                                                     $height,
                                                     $depth,
                                                     $foreground,
                                                     $background ) ;
```

Celle-ci permet de créer un pixmap de profondeur donnée (nombre de couleurs) à partir de données bitmap spécifiées. `$foreground` et `$background` sont les couleurs de premier plan et d'arrière plan à utiliser.

```
$gdkpixmap = Gtk : :Gdk : :Pixmap->create_from_xpm( $window,
                                                    $mask,
                                                    $transparent_color,
                                                    $filename ) ;
```

Le format XPM est une représentation lisible d'un pixmap pour le système X Window. Il est largement utilisé et beaucoup d'utilitaires différents sont disponibles pour créer des fichiers images dans ce format. Le fichier spécifié par `$filename` doit contenir une image dans ce format et il sera chargé dans la structure du pixmap. Le `$mask` précise quels bits du pixmaps doivent être opaques. Tous les autres bits sont colorés en utilisant la couleur spécifiée par `$transparent_color`. Un exemple utilisant cette méthode est présent plus loin.

```
$gdkpixmap = Gtk : :Gdk : :Pixmap->create_from_xpm_d( $window,
                                                      $mask,
                                                      $transparent_color,
                                                      $data ) ;
```

De petites images peuvent être incorporées à un programme en tant que donnée au format XPM. Un pixmap est créé en utilisant cette donnée, plutôt que de la lire dans un fichier. Voici un exemple :

```
my @xpm_data = ( "16 16 3 1",
                 "       c None",
                 ".      c #000000000000",
                 "X      c #FFFFFFFFFFFF",
                 "                ",
```

```
                "   ......       ",
                "   .XXX.X.      ",
                "   .XXX.XX.     ",
                "   .XXX.XXX.    ",
                "   .XXX.....    ",
                "   .XXXXXXX.    ",
                "   .XXXXXXX.    ",
                "   .XXXXXXX.    ",
                "   .XXXXXXX.    ",
                "   .XXXXXXX.    ",
                "   .XXXXXXX.    ",
                "   .XXXXXXX.    ",
                "   .........    ",
                "                ",
                "                " );
```

Une fois que vous avez créé un pixmap, nous pouvons l'afficher dans un widget GTK. Nous devons créer un widget pixmap GTK pour contenir le pixmap GDK. On y arrive avec :

```
$pixmap = new Gtk : :Pixmap( $gdkpixmap, $mask ) ;
```

Les autres appels de widget pixmap sont :

```
$pixmap->get_type() ;
$pixmap->set( $val, $mask ) ;
$pixmap->get( $val, $mask ) ;
```

`set()` est utilisé pour changer le pixmap dont le widget s'occupe actuellement. `$val` est le pixmap créé en utilisant GDK.

3.23.1 Un exemple

L'exemple suivant utilise un pixmap dans un bouton (voir également l'exemple sur un bouton pixmap plus loin).

```
#!/usr/bin/perl -w


use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;


# Donnée XPM d'un icône Ouvrir-Fichier
my @xpm_data = ( "16 16 4 1",
                 "       c None s None",
                 ".      c black",
                 "X      c #808080",
                 "o      c white",
                 "                ",
                 "  ..            ",
                 " .Xo.    ...    ",
```

```
                    " .Xoo. ..oo.    ",
                    " .Xooo.Xooo...  ",
                    " .Xooo.oooo.X.  ",
                    " .Xooo.Xooo.X.  ",
                    " .Xooo.oooo.X.  ",
                    " .Xooo.Xooo.X.  ",
                    " .Xooo.oooo.X.  ",
                    "  .Xoo.Xoo..X.  ",
                    "   .Xo.o..ooX.  ",
                    "    .X..XXXXX.  ",
                    "    ..X.......  ",
                    "     ..         ",
                    "                ");

my $window;
my $pixmapwid;
my $button;
my $pixmap;
my $mask;
my $style;

$window = new Gtk : :Window( "toplevel" );
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ); } );
$window->border_width( 10 );
$window->show();

# maintenant le pixmap de gdk
$style = $window->get_style()->bg( 'normal' );
( $pixmap, $mask ) = Gtk : :Gdk : :Pixmap->create_from_xpm_d( $window->window,
                                                              $style,
                                                              @xpm_data );

# un widget pixmap pour contenir le pixmap
$pixmapwid = new Gtk : :Pixmap( $pixmap, $mask );
$pixmapwid->show();

# un bouton pour contenir le widget pixmap
$button = new Gtk : :Button();
$button->add( $pixmapwid );
$button->show();
$button->signal_connect( "clicked", sub { print( "button clicked\n" ); } );
$window->add( $button );

main Gtk;
exit( 0 );
```

Pour charger un fichier à partir d'un fichier appelé icon0.xpm placé dans le répertoire courant, on créerait le pixmap ainsi :

```
( $pixmap, $mask ) = Gtk : :Gdk : :Pixmap->create_from_xpm( $window->window,
                                                            $style;
                                                            "./icon0.xpm" );
$pixmapwid = new Gtk : :Pixmap( $pixmap, $mask );
$pixmapwid->show();
```

3.23.2 Les fenêtres taillées : la brouette

Le désavantage d'utiliser les pixmaps est que l'objet affiché est toujours rectangulaire, quelque soit l'image. Nous aimerions créer des bureaux et des applications avec des icônes ayant des formes plus naturelles. Par exemple, pour

l'interface d'un jeu, nous aimerions avoir des boutons ronds à pousser. La manière de faire cela est d'utiliser les fenêtres taillées.

Une fenêtre taillée est simplement un pixmap dont les pixels du fond sont transparents. De cette manière, quand l'image du fond est une image multicolore, on ne l'efface pas avec des coins rectangulaires autour de notre icône. L'exemple suivant affiche une brouette sur le bureau.

```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my @wheelbarrowFull = ( '48 48 64 1',
                        '       c None',
                        '.      c #DF7DCF3CC71B',
                        'X      c #965875D669A6',
                        'o      c #71C671C671C6',
                        'O      c #A699A289A699',
                        '+      c #965892489658',
                        '@      c #8E38410330C2',
                        '#      c #D75C7DF769A6',
                        '$      c #F7DECF3CC71B',
                        '%      c #96588A288E38',
                        '&      c #A69992489E79',
                        '*      c #8E3886178E38',
                        '=      c #104008200820',
                        '-      c #596510401040',
                        ';      c #C71B30C230C2',
                        ' :      c #C71B9A699658',
                        '>      c #618561856185',
                        ',      c #20811C712081',
                        '<      c #104000000000',
                        '1      c #861720812081',
                        '2      c #DF7D4D344103',
                        '3      c #79E769A671C6',
                        '4      c #861782078617',
                        '5      c #41033CF34103',
```

```
'6      c #000000000000',
'7      c #49241C711040',
'8      c #492445144924',
'9      c #082008200820',
'0      c #69A618611861',
'q      c #B6DA71C65144',
'w      c #410330C238E3',
'e      c #CF3CBAEAB6DA',
'r      c #71C6451430C2',
't      c #EFBEDB6CD75C',
'y      c #28A208200820',
'u      c #186110401040',
'i      c #596528A21861',
'p      c #71C661855965',
'a      c #A69996589658',
's      c #30C228A230C2',
'd      c #BEFBA289AEBA',
'f      c #596545145144',
'g      c #30C230C230C2',
'h      c #8E3882078617',
'j      c #208118612081',
'k      c #38E30C300820',
'l      c #30C2208128A2',
'z      c #38E328A238E3',
'x      c #514438E34924',
'c      c #618555555965',
'v      c #30C2208130C2',
'b      c #38E328A230C2',
'n      c #28A228A228A2',
'm      c #41032CB228A2',
'M      c #104010401040',
'N      c #492438E34103',
'B      c #28A2208128A2',
'V      c #A699596538E3',
'C      c #30C21C711040',
'Z      c #30C218611040',
'A      c #965865955965',
'S      c #618534D32081',
'D      c #38E31C711040',
'F      c #082000000820',
'                                                ',
'          .XoO                                  ',
'         +@#$%o&                                ',
'         *=- ;# : :o+                             ',
'           >,<12# :34                           ',
'             45671# :X3                         ',
'               +89<02qwo                        ',
'e*                >,67 ;ro                      ',
'ty>                 459@>+&&                    ',
'$2u+                  ><ipas8*                  ',
'%$ ;=*                *3 :.Xa.dfg>                ',
'Oh$ ;ya             *3d.a8j,Xe.d3g8+            ',
' Oh$ ;ka          *3d$a8lz,,xxc :.e3g54           ',
'  Oh$ ;kO       *pd$%svbzz,sxxxxfX..&wn>         ',
'   Oh$@mO    *3dthwlsslszjzxxxxxxx3 :td8M4       ',
'    Oh$@g& *3d$XNlvvvlllm,mNwxxxxxxxfa. :,B*     ',
'     Oh$@,Od.czlllllzlmmqV@V#V@fxxxxxxxf :%j5&   ',
'      Oh$1hd5lllslllCCZrV#r# :#2AxxxxxxxxxcdwM*  ',
```

```
                        '       OXq6c.%8vvvllZZiqqApA :mq :XxcpcxxxxxfdcM* ',
                        '        2r<6gde3bllZZrVi7S@SV77A : :qApxxxxxxfdcM ',
                        '        :,q-6MN.dfmZZrrSS :#riirDSAX@Af5xxxxxfevo',
                        '         +A26jguXtAZZZC7iDiCCrVVii7Cmmmxxxxxx%3g',
                        '          *#16jszN..3DZZZZrCVSA2rZrV7Dmmwxxxx&en',
                        '           p2yFvzssXe :fCZZCiiD7iiZDiDSSZwwxx8e*>',
                        '           OA1<jzxwwc :$d%NDZZZZCCCZCCZZCmxxfd.B ',
                        '            3206Bwxxszx%et.eaAp77m77mmmf3&eeeg* ',
                        '             @26MvzxNzvlbwfpdettttttttttt.c,n&  ',
                        '             *;16=lsNwwNwgsvslbwwvccc3pcfu<o    ',
                        '              p;<69BvwwsszslllbBlllllllu<5+     ',
                        '              OS0y6FBlvvvzvzss,u=Blllj=54       ',
                        '               c1-699Blvllllllu7k96MMMg4         ',
                        '               *10y8n6FjvllllB<166668            ',
                        '                S-kg+>666<M<996-y6n<8*           ',
                        '                p71=4 m69996kD8Z-66698&&         ',
                        '                &i0ycm6n4 ogk17,0<6666g          ',
                        '                 N-k-<>     >=01-kuu666>         ',
                        '                 ,6ky&      &46-10ul,66,         ',
                        '                 Ou0<>       o66y<ulw<66&        ',
                        '                  *kk5       >66By7=xu664        ',
                        '                   <<M4      466lj<Mxu66o        ',
                        '                   *>>       +66uv,zN666*        ',
                        '                             566,xxj669          ',
                        '                             4666FF666>          ',
                        '                              >966666M           ',
                        '                               oM6668+           ',
                        '                                 *4              ',
                        '                                                 ',
                        '                                                 ' );
```

```
my $window;
my $pixmap;
my $fixed;
my $gdkpixmap;
my $mask;
my $style;
my $gc;

# Crée la fenêtre principale et attache le signal delete_event pour
# terminer l'application. Notez que la fenêtre principale n'aura pas
# de barre de titre puisque nous l'avons fait ''popup''
$window = new Gtk::Window( "popup" );
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ); } );
$window->show();

# maintenant le pixmap et le widget pixmap
$style = Gtk::Widget->get_default_style();
$gc = $style->black_gc;
( $gdkpixmap, $mask ) = Gtk::Gdk::Pixmap->create_from_xpm_d( $window->window,
                                                             $style->bg( 'normal' ),
                                                             @wheelbarrowFull );
$pixmap = new Gtk::Pixmap( $gdkpixmap, $mask );
$pixmap->show();

# nous utilisons un widget fixed pour afficher le pixmap
$fixed = new Gtk::Fixed();
$fixed->set_usize( 200, 200 );
```

```
$fixed->put( $pixmap, 0, 0 ) ;
$window->add( $fixed ) ;
$fixed->show() ;

# Cela masque tout sauf l'image elle-même
$window->shape_combine_mask( $mask, 0, 0 ) ;

# montre la fenêtre
$window->set_uposition( 400, 400 ) ;
$window->show() ;
main Gtk ;
exit( 0 ) ;
```

Pour rendre la brouette sensible, nous pourrions lui attacher le signal button press event. Les quelques lignes suivantes rende l'image sensible au bouton de la souris, quand on clique dessus, l'application se termine.

```
$window->set_events( [ $window->get_events(), 'button_press_mask' ] ) ;
$window->signal_connect( "button_press_event", sub { Gtk->exit( 0 ) ; } ) ;
```

3.23.3 Exemple de bouton pixmap

```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $window ;
my $box ;
my $button ;

$window = new Gtk : :Window( "toplevel" ) ;
$window->set_title( "Pixmap'd Buttons !" ) ;
$window->border_width( 10 ) ;
$window->realize() ;
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ) ; } ) ;
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ) ; } ) ;

# Ceci appelle notre fonction créatrice de boîte.
$box = xpm_label_box( $window, "info.xpm", "cool button" ) ;

$button = new Gtk : :Button() ;
$button->signal_connect( "clicked", sub {
                          print( "The cool button was pressed\n" ) ; } ) ;

# Regroupe et montre tous nos widgets
$box->show() ;
$button->add( $box ) ;
$button->show() ;
$window->add( $button ) ;
$window->show() ;

# Reste dans main et attend que le jeu commence !
main Gtk ;
exit( 0 ) ;
```

```
# Crée une nouvelle hbox avec une image et un label à l'intérieur
# et retourne la boîte
sub xpm_label_box
  {
    my ( $parent, $xpm_filename, $label_text ) = @_ ;

    my $box ;
    my $style ;
    my $pixmap ;
    my $mask ;
    my $pixmapwid ;
    my $label ;

    # Crée une boîte pour xpm et le label
    $box = new Gtk : :HBox( $false, 0 ) ;

    # Récupère le style du bouton pour l'appliquer à la couleur du fond
    $style = $parent->get_style()->bg( 'normal' ) ;

    # Maintenant le XPM
    ( $pixmap, $mask ) = Gtk : :Gdk : :Pixmap->create_from_xpm( $parent->window,
                                                                $style,
                                                                $xpm_filename ) ;
    $pixmapwid = new Gtk : :Pixmap( $pixmap, $mask ) ;

    # Crée une label pour le bouton
    $label = new Gtk : :Label( $label_text ) ;

    # Place le label et le pixmap dans la boîte
    $box->pack_start( $pixmapwid, $false, $false, 3 ) ;
    $box->pack_start( $label, $false, $false, 3 ) ;

    $box->border_width( 2 ) ;
    $pixmapwid->show() ;
    $label->show() ;

    return ( $box ) ;
  }
```

Avant de faire fonctionner le programme précédent, vous devrez charger cette image et la placer dans le répertoire contenant le code ci-dessus. Si vous ne faîtes pas cela, `$pixmap` ne contiendra pas de pixmap et le programme ne fonctionnera pas normalement. Une fois que vous avez fait cela, le programme ressemble à ça :

La fonction créée `xpm_label_box()` peut être utilisée pour placer des xpm et des labels dans n'importe quel widget qui peut être un conteneur.

Notez dans `xpm_label_box()`, l'appel `get_style()`. Tous les widgets possèdent un "style", consistant en des couleurs de premier plan, de fond pour une variété de situations, une sélection de police, et d'autres données graphiques en rapport avec le widget. Ces valeurs de style sont définiées par défaut dans chacun des widgets, et sont requises par certains appels de fonction GDK comme `$pixmap->create_from_xpm()`, à qui ici on donne la couleur du fond "normale". Utilisant les fichiers de ressources, les données de style de widget peuvent être personnalisées.

Remarquez également l'appel de la fonction `realize()` après avoir déclaré la largeur des bords de la fenêtre. Cette fonction utilise GDK pour créer les fenêtres X reliées au widget. Cette fonction est appelée automatiquement quand on invoque la fonction `show()` pour un widget, et d'ailleurs on ne l'a pas utilisé dans les exemples précédents. Mais l'appel de la fonction `create_from_xpm()` requiert que son argument `window` fasse référence à une réelle fenêtre X, c'est pourquoi il est nécessaire de réaliser ce widget avant cet appel GDK.

Chapitre 4

Les conteneurs avancés

4.1 Les dialogues

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Window
                      +--- Dialog
```

Le widget dialogue est très simple et n'est en fait qu'une fenêtre avec quelques éléments préemballés pour vous. Sont inclus dans le widget une `fen\^etre`, une `vbox` et une `action_area`.

Une widget dialogue crée une fenêtre et regroupe une vbox dans le haut qui contient un séparateur et dessous une HBox appelée `action_area`.

Il peut être utilisé pour les messages popup ou les autres tâches similaires. C'est vraiment très simple et il n'y a qu'une fonction pour les boîtes de dialogues :

```
$dialog = new Gtk : :Dialog() ;
```

Cela crée la boîte de dialogue et c'est maitenant à vous de l'utiliser. Nous pourrions placer une bouton dans l'`action_area` en faisant :

```
$button = new Gtk : :Button( "Close" ) ;
$dialog->action_area->pack_start( $button, $true, $true, 0 ) ;
$button->show() ;
```

Nous pourrions l'ajouter à la vbox en plaçant un label à l'intérieur. Essayons donc :

```
$label = new Gtk : :Label( "Dialogs are groovy !" ) ;
$dialog->vbox->pack_start( $label, $true, $true, 0 ) ;
$label->show() ;
```

Par exemple, vous pourriez placer 2 boutons dans l'`action_area`, un bouton “Annuler” et “Ok”, et un label dans la partie supérieure posant une question à l'utilisateur, ou indiquant une erreur, etc...

Alors vous pourriez attacher à chaque bouton un signal qui déclenche l'action que l'utilisateur désire. Si la simple fonctionnalité fournit par défaut, la boîte horizontale et la verticale, ne vous donne pas satisfaction, vous pouvez toujours placer d'autres widgets dans les boîtes fournies. Par exemple, vous pourriez placer une table dans la boîte verticale.

4.2 Dialogues de sélection de fichiers

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Window
                      +--- FileSelectionDialog
```

4.2.1 Créer une boîte de dialogue de sélection de fichiers

Cette boîte de dialogue est un moyen simple et rapide pour que l'utilisateur choisisse un nom de fichier. Elle est complète avec les boutons Ok, Cancel et Help. C'est donc un moyen simple d'économiser du temps de programmation. Pour créer une nouvelle boîte de dialogue :

```
$file_dialog = new Gtk : :FileSelection( $title ) ;
```

4.2.2 Utiliser un nom de fichier

Pour déclarer le nom de fichier, par exemple pour accéder à un répertoire spécifique ou donner un nom de fichier par défaut, on utilise :

```
$file_dialog->set_filename( $filename ) ;
```

Pour prendre le texte que l'utilisateur a entré ou sur lequel il a cliqué, utilisez ;

```
$file_dialog->get_filename() ;
```

Vous pouvez utiliser un filtre de fichier dans le répertoire courant à l'aide de :

```
$file_dialog->complete( $pattern ) ;
```

Si un fichier correspond, il apparaîtra dans l'entrée texte de la boîte de dialogue. Si un ensemble de fichiers correspond au filtre alors la liste des fichiers ne contiendra que ceux-là. Un exemple de filtre : `*.txt` ou `gtk*`.

4.2.3 Opérations sur les fichiers

La boîte de dialogue de sélection de fichiers peut montrer des boutons d'opérations sur les fichiers :

```
$file_dialog->show_fileop_buttons() ;
```

ou bien les cacher avec :

```
$file_dialog->hide_fileop_buttons() ;
```

4.2.4 Les widgets de dialogues

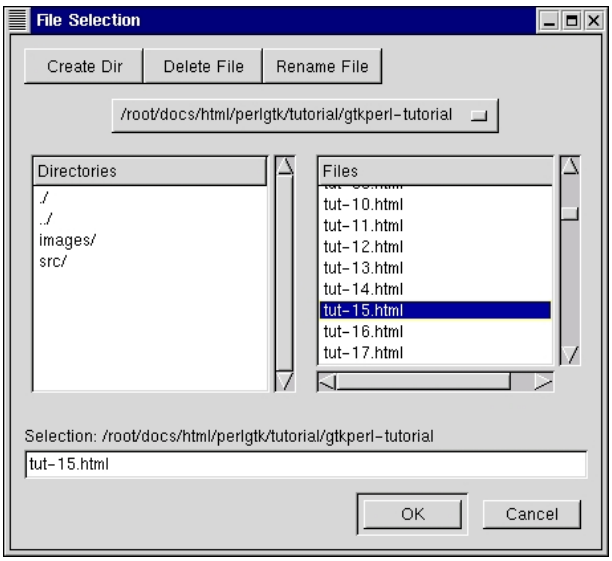
Vous pouvez aussi accéder aux widgets contenus dans le widget de sélection de fichiers. Il s'agit de

```
dir_list
file_list
selection_entry
selection_text
main_vbox
ok_button
cancel_button
help_button
```

Vous utiliserez certainement les boutons, Ok, Cancel et Help en signalant leurs utilisations.

4.2.5 Exemple

Voila un exemple volé au programme testgtk.c et modifié pour qu'il tourne tout seul. Ce n'est rien d'autre qu'un widget de sélection de fichiers. Vous remarquerez que le bouton Help ne fait rien car on ne lui a attaché aucun signal.




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $file_dialog;

# Crée un nouveau widget de sélection de fichiers
$file_dialog = new Gtk : :FileSelection( "File Selection" );
$file_dialog->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );

# Connecte le bouton ok à la fonction file_ok_sel
$file_dialog->ok_button->signal_connect( "clicked",
                                         \&file_ok_sel,
                                         $file_dialog );

# Connecte le bouton cancel à la fonction qui détruit le widget
$file_dialog->cancel_button->signal_connect( "clicked",
                                             sub { Gtk->exit( 0 ); } );

#  Déclare une nom de fichier, comme quand on veut sauver un fichier
$file_dialog->set_filename( "penguin.png" );

$file_dialog->show();
main Gtk;
exit( 0 );



# Récupère le nom du fichier et l'imprime sur la console
sub file_ok_sel
  {
    my ( $widget, $file_selection ) = @_;
    my $file = $file_selection->get_filename();
```

```
  print( "$file\n" ) ;
}
```

4.3 Dialogue de sélection de police

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Window
                      +--- FontSelectionDialog
```

Il s'agit d'une boîte de dialogue qui permet à l'utilisateur de choisir une police et un style de police à l'aide du widget préconçu FontSelection.

La fonction suivante crée un widget de sélection de police :

```
$font_dialog = new Gtk : :FontSelectionDialog( $title ) ;
```

Pour obtenir la police sélectionnée dans la boîte de dialogue :

```
$font_dialog->get_font() ;
```

Pour obtenir la police courante :

```
$font_dialog->get_font_name() ;
```

Pour déclarer la police sélectionnée dans la boîte de dialogue :

```
font_dialog->set_font_name( $font_name ) ;
```

Ou `$font_name` est un chaîne de caractères avec le nom de la police à charger. Cette fonction retourne une valeur vraie si la police a été trouvée ou fausse si ce n'est pas la cas. Pour limiter les polices montrées vous pouvez installer un filtre avec :

```
font_dialog->set_filter( $filter_type,
                         $font_type,
                         $foundries,
                         $weights,
                         $slants,
                         $setwidths,
                         $spacings,
                         $charsets ) ;
```

Ou les options sont :

`$filter_type` - le filtre peut être soit de `'base'` soit `'user'`. Un filtre de base est permanent alors qu'un filtre utilisateur peut être changer par celui-ci.

`$font_type` - le type de police à montrer. Cela peut être une combinaison de `'bitmap'`, `'scalable'`, `'scalable_bitmap'` ou `'all'`.

`$foundries` - une liste de chaîne de caractères contenant le nom des "fondries" qui seront montrées ou une chaîne vide les montrant toutes.

Même principe que `$foundries` pour les arguments suivants : `$weights, $slants, $setwidths, $spacings, $charsets.`

4.4 Poignée

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- HandleBox
```

Une poignée est un conteneur qui permet à son enfant de se détacher d'une fenêtre. On l'utilise habituellement pour des barres d'outils détachables, comme dans l'exemple sur les barres outils.

On crée une poignée avec :

```
$handlebox = new Gtk : :HandleBox() ;
```

Vous pouvez déclarer le type d'ombre de la poignée avec :

```
$handlebox->set_shadow_type( $shadow_type ) ;
```

Ou `$shadow_type` peut être `'none'`, `'in'`, `'out'`, `'etched_in'`, `'etched_out'`.

Vous pouvez choisir le côté où sera dessiné la poignée :

```
$handlebox->set_handle_position( $handle_position ) ;
```

La position `$handle_position` peut être `'left'`, `'right'`, `'top'`, `'bottom'`

Vous pouvez également choisir le côté de rattachement de la poignée. Il s'agit du côté de l'enfant détaché qui doit être aligné avec le côté correspondant du "fantôme" laissé derrière quand l'enfant fut détaché pour se réattacher à la fenêtre éteinte. On utilise :

```
$handlebox->set_snap_edge( $snap_edge ) ;
```

Les valeurs de `$snap_edge` sont les mêmes que pour `$handle_position` ci-dessus. Habituellement, le côté de rattachement est choisi de sorte qu'il reste à la même place à l'écran quand la poignée est éteinte. Si le côté de rattachement n'est pas choisi, alors une valeur appropriée serait déduite à partir de la position de la poignée. Si la position de la poignée est `'right'` ou `'left'` alors le côté de rattachement sera `'top'` autrement ce sera `'left'`.

4.5 Les barres d'outils

Les barres d'outils sont habituellement utilisées pour regrouper un certain nombre de widgets dans le but de simplifier la prise en main de leur look et de leur disposition. Typiquement, les barres d'outils sont constitués de boutons avec des icônes, des labels et des bulles d'aides, mais n'importe quel autre widget peut être placé dans une barre d'outils. Enfin, les éléments peuvent être arrangés soit horizontalement soit verticalement et les boutons peuvent être affichés avec des icônes, des labels ou les deux.

4.5.1 Créer un barre d'outils

Pour créer un barre d'outils (on s'en serait douté !), on utilise la fonction :

```
$toolbar = new Gtk : :Toolbar( $orientation, $style ) ;
```

ou `$orientation` peut être soit `'horizontal'` soit `'vertical'` et le `$style` soit `'text'`, soit `'icons'`, soit `'both'`. Le style s'applique à tous les boutons créés avec la fonction `'item'` (et non aux boutons insérés dans la barre en tant que widgets séparés).

4.5.2 Ajouter des items

Après avoir créer une barre d'outils, on peut placer des items (ce qui signifie de simples chaînes de caractères) ou des éléments (ce qui signifie n'importe quel type de widget) après ou avant les éléments existants. Pour décrire un item, on a besoin d'un label texte, d'un texte à placé dans la bulle d'aide, un icône pour le bouton et une fonction de rappel pour lui. Par exemple, pour placer un item au début ou à la fin de la barre, vous pouvez utiliser :

```
$toolbar->append_item( $text,
                       $tooltip_text,
                       $tooltip_private_text,
                       $pixmap ) ;
$toolbar->prepend_item( $text,
                        $tooltip_text,
                        $tooltip_private_text,
                        $pixmap ) ;
```

Ces fonctions retournent un bouton que vous pourriez alors utiliser pour ajouter des rappels pour les gestionnaires de signaux et d'évènements.

Si vous voulez utiliser , `insert_item()`, le seul paramètre supplémentaire qui doit être précisé est la position où l'on veut insérer l'item. On obtient :

```
$toolbar->insert_item( $text,
                       $tooltip_text,
                       $tooltip_private_text,
                       $pixmap,
                       $position ) ;
```

4.5.3 Ajouter des widgets

Les barres d'outils peuvent être utilisées pour contenir n'importe quel type de widget. Les fonctions suivantes sont similaires à celles déjà mentionnées, mais elles utilisent un widget à la place du texte ou d'un pixmap.

```
$toolbar->append_widget( $widget,
                         $tooltip_text,
                         $tooltip_private_text );
$toolbar->prepend_widget( $widget,
                          $tooltip_text,
                          $tooltip_private_text );
$toolbar->insert_widget( $widget,
                         $tooltip_text,
                         $tooltip_private_text,
                         $position );
```

4.5.4 Ajouter de l'espace

Pour simplifier l'insertion d'espace entre les items de la barre outils, il y a les fonctions :

```
$toolbar->append_space();
$toolbar->prepend_space();
$toolbar->insert_space( $position );
```

La taille de l'espace ajouté peut être déclaré globalement pour toute la barre avec la fonction ;

```
$toolbar->set_space_size( $space_size );
```

Vous pouvez également choisir entre deux styles d'espace. L'espace peut être soit une espace vide soit une ligne. Ce choix est aussi simple que la fonction qui le permet :

```
$toolbar->set_space_style( $space_style );
```

ou `$space_style` peut être soit `'empty'` ou `'line'`.

4.5.5 Ajouter des éléments

Les items, les widgets et les espaces sont des exemples d'éléments. Habituellement, Vous utiliserez les fonctions précédentes pour leur facilité d'emploi mais les fonctions suivantes sont aussi disponibles :

```
$toolbar->append_element( $child_type,
                          $widget,
                          $text,
                          $tooltip_text,
                          $tooltip_private_text,
                          $icon,
                          \&callback,
                          @callback_data );
$toolbar->prepend_element( $child_type,
                           $widget,
                           $text,
                           $tooltip_text,
                           $tooltip_private_text,
                           $icon,
                           \&callback,
                           @callback_data );
$toolbar->insert_element( $child_type,
                          $widget,
                          $text,
                          $tooltip_text,
                          $tooltip_private_text,
                          $icon,
                          \&callback,
                          @callback_data,
                          $position );
```

Les arguments sont :

`$child_type` - le type d'enfant peut être soit `'space'`, soit `'button'`, soit `'togglebutton'`, `'radiobutton'` ou `'widget'`.

`$widget` - le widget à ajouter à la barre d'outils.

`$text` - le label de l'élément.

`$tooltip_text` - le texte pour la bulle d'aide du bouton.

`$tooltip_private_text` - utilisé pour l'aide contextuelle sensible à propos de l'élément de la barre d'outils.

`$icon` - la représentation picturale de la fonction de l'élément.

`\&callback` - la routine à exécuter quand le bouton est pressé.

`@callback_data` - les données que vous souhaitez passer à votre rappel.

`$position` - la position où insérer l'élément. (seulement pour insert)

4.5.6 Orientation et style

Si c'est nécessaire, l'orientation et le style de la barre peuvent être changés " au vol " en utilisant les fonctions :

```
$toolbar->set_orientation( $orientation );
$toolbar->set_style( $style );
$toolbar->set_tooltips( $enable );
```

Ou `$orientation` est soit `'horizontal'` soit `'vertical'` et le `$style` est soit `'icons'`, soit `'text'` soit `'both'`.

4.5.7 Styles de relief des boutons

On déclare le style de relief par :

```
$toolbar->set_button_relief( $relief_style );
```

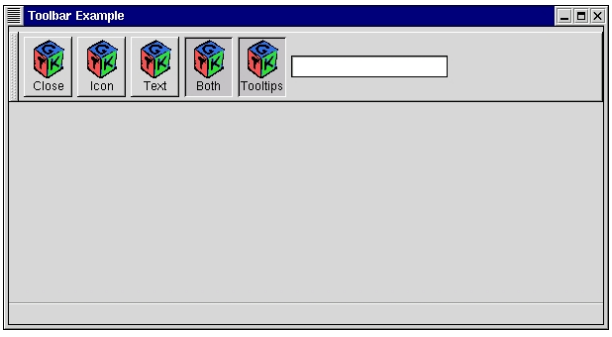
sachant que `$relief_style` peut être soit `'normal'`, soit `'halt'`, soit `'none'`.

On peut retrouver le style de relief avec :

```
$toolbar->get_button_relief();
```

4.5.8 Exemple

Pour voir d'autres choses faisables avec une barre d'outils, regardons le programme ci-dessous. On m'a rapporté que selon la version de Gtk-Perl utilisée, vous devez remplacer `'radiobutton'` par `'radio_button'` et `'togglebutton'` par `'toggle_button'`.




```
#!/usr/bin/perl -w
use Gtk;
use strict;
init Gtk;
my $false = 0;
my $true = 1;
my @gtk_xpm = ( '32 39 5 1',
                '.      c none',
                '+      c black',
                '@      c #3070E0',
                '#      c #F05050',
                '$      c #35E035',
```

```
                '.................+.............',
                '..............+++++.............',
                '............+++++@@++...........',
                '..........+++++@@@@@@++.........',
                '........++++@@@@@@@@@@++........',
                '......++++@@++++++++@@@++.......',
                '.....+++@@@++++++++++@@@++.....',
                '...+++@@@@+++@@@@@@++++@@@@+....',
                '..+++@@@@+++@@@@@@@@+++@@@@@++..',
                '.++@@@@@@@+++@@@@@@@@@@@@@@@@@++',
                '.+#+@@@@@@@++@@@@+++@@@@@@@@@@@@+',
                '.+##++@@@@+++@@@+++++@@@@@@@@$@.',
                '.+###++@@@@+++@@@+++@@@@@++$$$@.',
                '.+####+++@@@++++++@@@@@+@$$$$@.',
                '.+#########+++@@@@+++@@@@++@$$$$$$+.',
                '.+##########++++@@@@@@@++@$$$$$$$$+.',
                '.+########+##+@@@@+++$$$$$$@@$$+.',
                '.+########+##+@@++@$$$$$$++$$$+.',
                '.+########+##+##+@@$$$$$$$$@+@$$@+.',
                '.+###########++++@$$@+@$$@++$$$@+.',
                '.+############+++$$@+@$$++$$$$+..',
                '.++############+++$$@+@$++@$$$$+..',
                '.+############+##++$$++@+++$$$$$+..',
                '.++##########+##+#++$$++++++@$$$$+..',
                '.++##########+######++$$++++++@$$$@+..',
                '.+######++######++$$+++@++++@$@+..',
                '.+######++######++$$++@$$@+++$@@..',
                '.++#####++######++$$++$$$$$+@$@++.',
                '.++#####++######++$$++$$$$$$$$+++.',
                '.+++####+######++$$++$$$$$$$@+++.',
                '..+++##########+@$$+@$$$$$$+++...',
                '...+++#########+@$$$$$$$$@+++....',
                '.....+++#######+@$$$$$$$+++......',
                '......+++######+@$$$$$@++........',
                '.......+++#####+@$$$$+++.........',
                '.........++###+$$$@++............',
                '...........++##+$@+++............',
                '.............+++++...............',
                '..............++++..............' );
my $dialog;
my $handlebox;
my $toolbar;
my $icon;
my $mask;
my $iconw;
my $close_button;
my $icon_button;
my $text_button;
my $both_button;
my $tooltips_button;
my $entry;
$dialog = new Gtk : :Dialog();
$dialog->set_title( "Toolbar Example" );
$dialog->set_usize( 600, 300 );
$dialog->signal_connect( "delete_event", sub { Gtk->exit( 0 ); } );
# Nous devons réaliser la fenêtre parce qu'on utilise des pixmaps
# en tant qu'items
$dialog->realize();
```

```
# Pour faire les chose bien, nous plaçons la barre d'outils dans une
# poignée, ainsi elle peut être détachée de la fenêtre principale. Une poignée
# est juste une boîte dans laquelle on peut placer des choses. La seule
# différence est que l'on peut la détacher de la fenêtre parent. On
# l'utilise souvent pour les barre d'outils.
$handlebox = new Gtk : :HandleBox() ;
$dialog->vbox->pack_start( $handlebox, $false, $false, 5 ) ;

# La barre d'outils sera horizontale, avec à la fois des icônes et du texte
# et des espaces larges de 5 pixels entre les items. Enfin, nous la
# mettons dans notre poignée.
$toolbar = new Gtk : :Toolbar( 'horizontal', 'both' ) ;
$toolbar->border_width( 5 ) ;
$toolbar->set_space_size( 5 ) ;
$handlebox->add( $toolbar ) ;

# maintenant nous créons des icônes avec masque : nous le réutiliserons
# pour créer des widgets icônes pour les items de la barre d'outils.
( $icon, $mask ) = Gtk : :Gdk : :Pixmap->create_from_xpm_d( $dialog->window,
                                                            $dialog->style->white,
                                                            @gtk_xpm ) ;


# notre premier item est le bouton ``Close''
$iconw = new Gtk : :Pixmap( $icon, $mask ) ;
$close_button = $toolbar->append_item( "Close",
                                       "Close this app",
                                       "Private",
                                       $iconw ) ;
$close_button->signal_connect( "clicked", sub { Gtk->exit( 0 ) ; } ) ;
$toolbar->append_space() ;

# maintenant, on crée notre groupe de boutons radio.
$iconw = new Gtk : :Pixmap( $icon, $mask ) ;
$icon_button = $toolbar->append_element( 'radio_button',
                                         new Gtk : :RadioButton(),
                                         "Icon",
                                         "Only icons in toolbar",
                                         "Private",
                                         $iconw ) ;
$icon_button->signal_connect( "clicked", \&radio_event, $toolbar ) ;
$toolbar->append_space() ;

# les boutons radios suivants font référence au premier
$iconw = new Gtk : :Pixmap( $icon, $mask ) ;
$text_button = $toolbar->append_element( 'radio_button',
                                         $icon_button,
                                         "Text",
                                         "Only texts in toolbar",
                                         "Private",
                                         $iconw ) ;
$text_button->signal_connect( "toggled", \&radio_event, $toolbar ) ;
$toolbar->append_space() ;

$iconw = new Gtk : :Pixmap( $icon, $mask ) ;
$both_button = $toolbar->append_element( 'radio_button',
                                         $text_button,
```

```
                                        "Both",
                                        "Icons and text is toolbar",
                                        "Private",
                                        $iconw ) ;
$both_button->signal_connect( "toggled", \&radio_event, $toolbar ) ;
$toolbar->append_space() ;
$both_button->set_active( $true ) ;

# les boutons radios suivants font référence au premier
$iconw = new Gtk : :Pixmap( $icon, $mask ) ;
$tooltips_button = $toolbar->append_element( 'toggle_button',
                                             "",
                                             "Tooltips",
                                             "Toolbar with or without tips",
                                             "Private",
                                             $iconw ) ;
$tooltips_button->signal_connect( "toggled", \&toggle_event, $toolbar ) ;
$toolbar->append_space() ;
$tooltips_button->set_active( $true ) ;

# pour placer un widget dans la barre d'outils, nous devons le créer et
# le placer après un autre avec la bulle d'aide appropriée.
$entry = new Gtk : :Entry() ;
$toolbar->append_widget( $entry, "This is just an entry", "Private" ) ;

# bien, ce n'est pas créé dans la barre d'outils, donc nous devons
# le montrer.
$entry->show() ;
$toolbar->show() ;
$handlebox->show() ;
$dialog->show() ;
main Gtk ;
exit( 0 ) ;

### Routines

# quand l'un des bouton est enclenché, nous devons vérifier lequel est actif
# et déclarer le style de la barre d'outils en accord.
sub radio_event
  {
    my ( $widget, $toolbar ) = @_ ;
    if ( $text_button->active )
      {
        $toolbar->set_style( 'text' ) ;
      }
    elsif ( $icon_button->active )
      {
        $toolbar->set_style( 'icons' ) ;
      }
    elsif ( $both_button->active )
      {
        $toolbar->set_style( 'both' ) ;
      }
  }

# vérifie le bouton toggle donné et rend disponible ou indisponible les bulles
# d'aide.
sub toggle_event
```

```
{
  my ( $widget, $toolbar ) = @_ ;
  $toolbar->set_tooltips( $widget->active ) ;
}
```

4.6 Les notebooks

```
Object
  +--- Widget
       +--- Container
            +--- Notebook
```

Le widget Notebook est une collection de "pages" qui se recouvrent, chaque page contient des informations différentes et seule l'une des pages est visible. Ce widget est devenu très commun dernièrement dans la programmation GUI, et c'est un bon moyen de montrer des blocks d'informations similaires tout en garantissant une séparation de leurs affichages.

La première fonction, bien évidemment, sert à créer un nouveau notebook :

```
$notebook = new Gtk : :Notebook() ;
```

Une fois que le notebook est créé, il existe un certain nombre de fonctions qui opèrent sur le widget Notebook. Regardons les individuellement .

Le premier que nous rencontrerons sert à positionner les indicateurs de pages. Ces indicateurs de pages ou "tabs" c'est ainsi qu'ils sont référencés, peuvent être positionnés de quatre manières : en haut, en bas, à gauche ou à droite.

```
$notebook->set_tab_pos( $position ) ;
```

Ou `$position` peut être :
`'left'` - gauche
`'right'` - droite
`'top'` - en haut - le défaut
`'bottom'` - en bas

Ensuite, voyons comment ajouter des pages au notebook. Il existe trois moyens d'ajouter des pages à un Notebook. Les deux premiers sont presque similaires :

```
$notebook->append_page( $child, $tab_label ) ;
$notebook->prepend_page( $child, $tab_label ) ;
```

Les fonctions ajoutent des pages les pages à la fin ou au début du notebook. `$child` est le widget qui est placé dans la page du notebook, et `$tab_label` est la label pour la page ajoutée. Le widget `$child` doit être créé séparément, et est typiquement un ensemble de déclarations d'options placés dans un autre widget conteneur, comme une table par exemple.

La dernière fonction pour ajouter des pages à un notebook contient toutes les propriétés des deux précédentes mais permet de spécifier à quelle position vous voulez placer la page.

```
$notebook->insert_page( $child, $tab_label, $position ) ;
```

Les paramètres sont les mêmes que pour `append` et `prepend` avec un paramètre supplémentaire, `$ position`. Ce paramètre indique la position de la page à insérer sachant que la première page possède la position zéro.

Maintenant que nous savons comment ajouter des pages, voyons comment les enlever.

```
$notebook->remove_page( $page_num ) ;
```

Cette fonction enlève la page `$page_num` du notebook.

Nous retrouvons la page courante à l'aide de :

```
$notebook->get_current_page() ;
```

Nous retrouvons le numéro de la page courante à l'aide de :

```
$notebook->page_num( $child ) ;
```

Cette fonction retourne -1 si `$child` n'est pas une page du `$notebook`.

Si vous voulez changer le numéro de page d'un enfant, vous pouvez utiliser :

```
$notebook->reorder_child( $child, $position ) ;
```

Les deux fonctions suivantes sont de simples appels qui permettent de bouger vers l'avant ou l'arrière les pages du notebook. Il suffit de fournir à chaque appel de fonction le widget notebook sur lequel on veut agir. Notez que quand le notebook est sur la dernière page et que `next_page()` est appelé, le notebook se retrouvera sur la première page. De même, quand on est sur la première page et que l'on appelle `prev_page()`, on se retrouve sur la dernière page.

```
$notebook->next_page() ;
$notebook->prev_page() ;
```

La fonction suivante déclare une page active. Si vous voulez que le notebook soit ouvert en page 5 par exemple, vous utiliserez cette fonction. Sans cette fonction , la page par défaut est la première.

```
$notebook->set_page( $page_num ) ;
```

Les deux fonctions suivantes ajoutent ou enlèvent respectivement les indicateurs des pages et les bords du notebook.

```
$notebook->set_show_tabs( $show_tabs ) ;
$notebook->set_show_border( $show_border ) ;
```

La fonction suivante est utile quand vous avez un grand nombre de pages, et que les indicateurs de pages ne conviennent pas à la page. Cela permet de la faire défiler en utilisant deux boutons flèches.

```
$notebook->set_scrollable( $scrollable ) ;
```

La largeur des bords autour du bookmark peut être déclarée avec :

```
$notebook->set_tab_border( $border_width ) ;
```

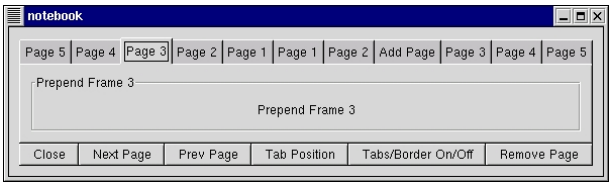
Vous pouvez également décider si tous les indicateurs de pages sont de la même taille ou non :

```
$notebook->set_homogeneous_tabs( $homogeneous ) ;
```

`$homogeneous` est une valeur vraie ou fausse.

4.6.1 Exemple

Maintenant voyons un exemple. Il est inspiré du code de textgtk.c qui est fourni avec la distribution GTK. Ce petit programme crée une fenêtre avec un notebook et six boutons. Le notebook contient 11 pages, ajoutées selon les trois méthodes, au début, à la fin ou insérées,. Les boutons vous permettent de décaler la position des indicateurs de pages, ajouter ou enlever les indicateurs ou les bords, enlever une page, changer les pages de deux manières (en avant , en arrière) et de sortir du programme.




```
#!/usr/bin/perl -w

use Gtk ;
use strict ;

init Gtk ;

my $false = 0 ;
my $true = 1 ;

my $window ;
my $button ;
my $table ;
my $notebook ;
my $frame ;
my $label ;
my $checkbutton ;
```

```
my $i ;
my $bufferf ;
my $bufferl ;

$window = new Gtk : :Window( "toplevel" ) ;
$window->signal_connect( "delete_event", sub { Gtk->exit( 0 ) ; } ) ;
$window->border_width( 10 ) ;

$table = new Gtk : :Table( 3, 6, $false ) ;
$window->add( $table ) ;

# Crée un nouveau notebook,détermine la position des tabs
$notebook = new Gtk : :Notebook() ;
$notebook->set_tab_pos( 'top' ) ;
$table->attach_defaults( $notebook, 0, 6, 0, 1 ) ;
$notebook->show() ;

# Ajoutons quelques pages à la fin du notebook
for ( $i = 0 ; $i < 5 ; $i++ )
  {
    $bufferf = "Append Frame " . ( $i + 1 ) ;
    $bufferl = "Page " . ( $i + 1 ) ;

    $frame = new Gtk : :Frame( $bufferf ) ;
    $frame->border_width( 10 ) ;
    $frame->set_usize( 100, 75 ) ;
    $frame->show() ;

    $label = new Gtk : :Label( $bufferf ) ;
    $frame->add( $label ) ;
    $label->show() ;

    $label = new Gtk : :Label( $bufferl ) ;
    $notebook->append_page( $frame, $label ) ;
  }

# Maintenant ajoutons une page à une marque spécifique
$checkbutton = new Gtk : :CheckButton( "Check me please !" ) ;
$checkbutton->set_usize( 100, 75 ) ;
$checkbutton->show() ;

$label = new Gtk : :Label( "Add Page" ) ;
$notebook->insert_page( $checkbutton, $label, 2 ) ;

# Maintenant, enfin, ajoutons des pages au début du notebook
for ( $i = 0 ; $i < 5 ; $i++ )
  {
    $bufferf = "Prepend Frame " . ( $i + 1 ) ;
    $bufferl = "Page " . ( $i + 1 ) ;

    $frame = new Gtk : :Frame( $bufferf ) ;
    $frame->border_width( 10 ) ;
    $frame->set_usize( 100, 75 ) ;
    $frame->show() ;

    $label = new Gtk : :Label( $bufferf ) ;
    $frame->add( $label ) ;
    $label->show() ;
```

```
    $label = new Gtk : :Label( $bufferl ) ;
    $notebook->prepend_page( $frame, $label ) ;
  }

# Indiquons à quel page ouvrir le notebook (page 4)
$notebook->set_page( 3 ) ;

# Crée une brochette de boutons
$button = new Gtk : :Button( "Close" ) ;
$button->signal_connect( "clicked", sub { Gtk->exit( 0 ) ; } ) ;
$table->attach_defaults( $button, 0, 1, 1, 2 ) ;
$button->show() ;

$button = new Gtk : :Button( "Next Page" ) ;
$button->signal_connect( "clicked", sub { $notebook->next_page() ; } ) ;
$table->attach_defaults( $button, 1, 2, 1, 2 ) ;
$button->show() ;

$button = new Gtk : :Button( "Prev Page" ) ;
$button->signal_connect( "clicked", sub { $notebook->prev_page() ; } ) ;
$table->attach_defaults( $button, 2, 3, 1, 2 ) ;
$button->show() ;

$button = new Gtk : :Button( "Tab Position" ) ;
$button->signal_connect( "clicked", \&rotate_book, $notebook ) ;
$table->attach_defaults( $button, 3, 4, 1, 2 ) ;
$button->show() ;

$button = new Gtk : :Button( "Tabs/Border On/Off" ) ;
$button->signal_connect( "clicked", \&tabsborder_book, $notebook ) ;
$table->attach_defaults( $button, 4, 5, 1, 2 ) ;
$button->show() ;

$button = new Gtk : :Button( "Remove Page" ) ;
$button->signal_connect( "clicked", \&remove_book, $notebook ) ;
$table->attach_defaults( $button, 5, 6, 1, 2 ) ;
$button->show() ;

$table->show() ;
$window->show() ;
main Gtk ;
exit( 0 ) ;



### Routines

# Cette fonction décale la position des tabs
sub rotate_book
  {
    my ( $button, $notebook ) = @_ ;

    my %rotate = ( top    => 'right',
                   right  => 'bottom',
                   bottom => 'left',
                   left   => 'top' ) ;
```

```
    $notebook->set_tab_pos( $rotate{ $notebook->tab_pos } ) ;
  }

# Ajoute/Enlève les tabs des pages et les bords
sub tabsborder_book
  {
    my ( $button, $notebook ) = @_ ;

    my $tval = $false ;
    my $bval = $false ;

    if ( $notebook->show_tabs == 0 )
      {
        $tval = $true ;
      }
    if ( $notebook->show_border == 0 )
      {
        $bval = $true ;
      }

    $notebook->set_show_tabs( $tval ) ;
    $notebook->set_show_border( $bval ) ;
  }

# Enlève une page du notebook
sub remove_book
  {
    my ( $button, $notebook ) = @_ ;

    my $page ;

    $page = $notebook->get_current_page() ;
    $notebook->remove_page( $page ) ;

    # On a besoin de rafraîchir le widget --Cela force
    # le widget à se redessiner
    $notebook->draw( undef ) ;
  }
```

4.7 Les cadres

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Frame
```

Les cadres sont utilisés pour enfermer un ou un groupe de widgets dans une boîte qui peut optionnellement avoir un titre. La position du titre et le style de la boîte peuvent être changés pour convenir à vos souhaits.

Un cadre est créé à l'aide de :

```
$frame = new Gtk : :Frame( $label ) ;
```

Le label est par défaut placé en haut à gauche du cadre. Une chaîne vide en tant qu'argument `$label` aura pour conséquence qu'aucun label ne sera affiché. Le texte du label peut être changé avec :

```
$frame->set_label( $label ) ;
```

La position du label peut être modifiée :

```
$frame->set_label_align( $xalign, $yalign ) ;
```

`$xalign` et `$yalign` prennent des valeurs comprises entre 0.0 et 1.0. `$xalign` indique la position du label le long du bord horizontal supérieur du cadre. `$yalign` n'est pour l'instant pas utilisé. La valeur par défaut de `$xalign` est 0.0 ce qui place le label sur la gauche du cadre.

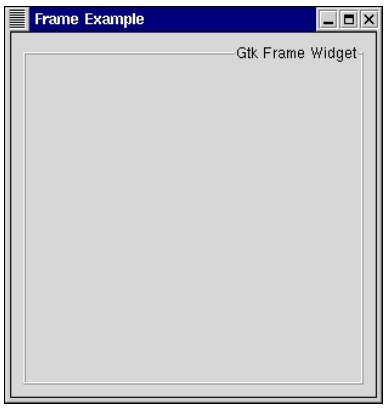
La fonction suivante change le style de la boîte qui est utilisé pour matérialiser le cadre :

```
$frame->set_shadow_type( $type ) ;
```

L'argument `$type` peut prendre l'une des valeurs suivantes :

```
'none'
'in'
'out'
'etched_in' - le défaut
'etched_out'
```

4.7.1 Exemple




```
 #!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $i;

my $window;
my $frame;
my $button;

$window = new Gtk : :Window( "toplevel" ) ;
$window->set_title( "Frame Example" ) ;
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ) ; } ) ;
$window->set_usize( 300, 300 ) ;
$window->border_width( 10 ) ;

# Crée un cadre
$frame = new Gtk : :Frame() ;
$window->add( $frame ) ;

# Déclare le label du cadre
$frame->set_label( "Gtk Frame Widget" ) ;
```

```
# Aligne le label sur la droite du cadre
$frame->set_label_align( 1.0, 0.0 );

# Déclare le style du cadre
$frame->set_shadow_type( 'etched_out' );

$frame->show();
$window->show();

main Gtk;
exit( 0 );
```

4.8 Les cadres d'aspects

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Frame
                      +--- AspectFrame
```

Les cadres d'aspects sont les mêmes que les widgets cadres, sauf qu'ils ont la possibilité d'obliger le ratio d'aspect (le rapport entre la largeur et la hauteur) d'un enfant à conserver une certaine valeur, en ajoutant des espaces si nécessaire. C'est utile, par exemple, si vous voulez prévisualiser une image plus grande. La taille de la prévisualisation devrait varier quand l'utilisateur redessine la fenêtre, mais le ratio a besoin de correspondre toujours à celui de l'image originale.

Pour créer un cadre d'aspect :

```
$aspect = new Gtk : :AspectFrame( $label,
                                  $xalign,
                                  $yalign,
                                  $ratio,
                                  $obey_child );
```

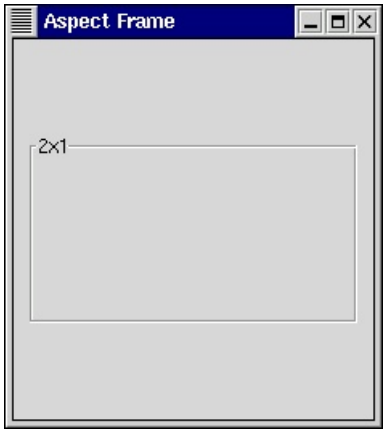
Les arguments `$xalign` et `$yalign` spécifient l'alignement horizontal et vertical et varient de 0.0 (aligné en haut ou à gauche) à 1.0 (aligné en bas ou à droite). Si `$obey_child` est vrai, la ratio du widget enfant sera égal au ratio de la taille idéale qu'il recherche. Autrement, il est donné par `$ratio`.

Pour changer les options d'un cadre aspect existant, vous pouvez utiliser :

```
$aspect->frame_set( $xalign, $yalign, $ratio, $obey_child );
```

4.8.1 Exemple

Comme exemple, le programme suivant utilise un cadre d'aspect pour présenter une aire de dessin dont le rapport d'aspect doit toujours être 2 :1, quelque soit la manière dont l'utilisateur redimensionne la fenêtre.



```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window;
my $aspect;
my $drawing_area;

$window = new Gtk : :Window( "toplevel" );
$window->set_title( "Aspect Frame" );
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$window->border_width( 10 );

# Crée un cadre d'aspect et l'ajoute à notre fenêtre top level

$aspect = new Gtk : :AspectFrame( "2x1", 0.5, 0.5, 2, $false );

$window->add( $aspect );
$aspect->show();

# Maintenant, on ajoute un widget enfant à notre cadre d'aspect
$drawing_area = new Gtk : :DrawingArea();

# Demande un fenêtre 200x200, mais le cadre d'aspect nous donnera une
# fenêtre de 200x100 puisque nous imposons le rapport 2x1.
$drawing_area->set_usize( 200, 200 );
$aspect->add( $drawing_area );
$drawing_area->show();

$window->show();
main Gtk;
exit( 0 );
```

4.9 Les grilles de placement

```
Object
  +--- Widget
       +--- Container
            +--- Fixed
```

Les grilles de placement vous permettent de placer les widgets à des positions fixes à l'intérieur d'une fenêtre, relatives au coin en haut à gauche. La position des widgets peut être changées dynamiquement.

Il existe trois fonctions associées aux grilles de placement :

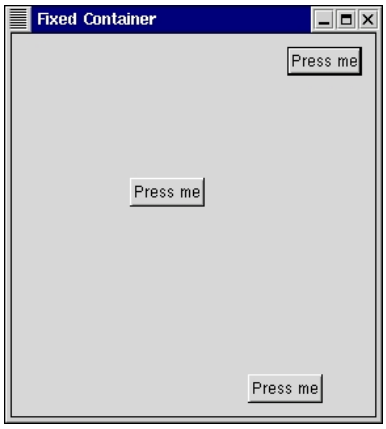
```
$fixed = new Gtk : :Fixed();
$fixed->put( $widget, $x, $y );
$fixed->move( $widget, $x, $y );
```

La fonction `new()` permet de créer une grille de placement.

`put()` place un `$widget` dans la grille à la position spécifiée par `$x` et `$y`.

`move()` permet de déplacer un widget à une nouvelle position.

4.9.1 Exemple




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $x = 50;
my $y = 50;
my $i;

my $window;
my $fixed;
my $button;

# Crée une nouvelle fenêtre
$window = new Gtk : :Window( "toplevel" );
$window->set_title( "Fixed Container" );
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );

# Déclare la largeur des bords de la fenêtre
$window->border_width( 10 );

# Crée une grille de placement
$fixed = new Gtk : :Fixed();
$window->add( $fixed );
$fixed->show();

for ( $i = 1; $i <= 3; $i++ )
  {
    # Crée un  bouton avec le label "Press me"
    $button = new Gtk : :Button( "Press me" );

    # Quand le bouton reçoit le signal ‘‘cliqué’’, il appellera la fonction
    # move_button() lui passant la grille de placement comme argument.
    $button->signal_connect( "clicked", \&move_button, $fixed );

    # Ceci place le bouton dans la grille de placement.
    $fixed->put( $button, $i * 50, $i * 50 );
```

```
    # L'étape finale est d'afficher ce widget nouvellement créé.
    $button->show() ;
  }

# Affiche la fenêtre
$window->show() ;

main Gtk ;
exit( 0 ) ;



### Routines

# Cette fonction déplace le bouton vers une nouvelle position
sub move_button
  {
    my ( $widget, $fixed ) = @_ ;

    $x = ( $x + 30 ) % 300 ;
    $y = ( $y + 50 ) % 300 ;
    $fixed->move( $widget, $x, $y ) ;
  }
```

4.10 Le conteneur Layout

```
Object
  +--- Widget
       +--- Container
            +--- Layout
```

Le conteneur Layout est similaire à la grille de placement sauf qu'il implémente une zone défilable infinie (ou l'infini est inférieur à 2^32). Le système X Window est limité à des fenêtres de 32767 pixels de large ou de hauteur. Le conteneur Layout contourne cette limitation en faisant quelques rembourrages exotiques et en utilisant les "gravités" de fenêtre et de bit, de sorte que l'on obtient un défilement linéaire même s'il y a plusieurs widgets enfants dans votre zone de défilement.

Un conteneur Layout est créé par :

```
$layout = new Gtk : :Layout( $hadjustment, $vadjustment ) ;
```

Comme vous pouvez le voir, vous pouvez optionnellement spécifier les objets Ajustement que le widget Layout utilisera pour son défilement.

Vous pouvez ajouter ou bouger des widgets dans un conteneur Layout en utilisant les deux fonctions suivantes :

```
$layout->put( $widget, $x, $y ) ;
$layout->move( $widget, $x, $y ) ;
```

On peut déclarer la taille du conteneur Layout à l'aide de :

```
$layout->set_size( $width, $height ) ;
```

Les conteneurs Layout font partie des ces quelques widgets de GTK qui se redessinent eux-même à l'écran quand on les change à l'aide des fonctions précédentes (la grande majorité des widgets fait la queue en attendant des requêtes qui seront lancées quand le contrôle retourne à `main Gtk`.

A cause de cela, quand vous voulez faire une grand nombre de changement dans un conteneur Layout, vous devrez utiliser les deux fonctions suivantes qui empêchent ou permettent ce rafraîchissement.

```
$layout->freeze() ;
$layout->thaw() ;
```

Les quatre dernières fonctions à utiliser avec les widgets Layout servent à manipuler les widgets d'ajustements horizontaux et verticaux.

```
$layout->get_hadjustment() ;
$layout->get_vadjustment() ;

$layout->set_hadjustment( $adjustment ) ;
$layout->set_vadjustment( $adjustment ) ;
```

4.11 Les fenêtres défilables

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- ScrolledWindow
```

Le fenêtre défilable est utilisée pour créer une aire défilable avec un autre widget à l'intérieur. Vous pouvez insérer n'importe quel type de widget dans une fenêtre défilable et il sera accessible quelque soit sa taille grâce aux barres de défilement.

La fonction suivante crée une nouvelle fenêtre défilable.

```
$scrolled_window = new Gtk : :ScrolledWindow( $hadjustment,
                                              $vadjustment ) ;
```

Ou le premier argument est l'ajustement dans la direction horizontale et le second celui dans le direction verticale. Ils sont presque toujours déclarés en valeur nulle.

```
$scrolled_window->set_policy( $hscrollbar_policy,
                              $vscrollbar_policy ) ;
```

Ceci déclare la politique à utiliser avec respect pour les barres de défilement. Le premier argument déclare la politique pour les barres horizontales et le second pour les barres verticales.

Cette politique peut être soit `'automatic'`, soit `'always'`. `'automatic'` décidera automatiquement si vous avez besoin de barres de défilement alors que `'always'` laissera les barres de défilement à leurs places.

Vous pouvez maintenant placer votre objet dans la fenêtre défilable en utilisant :

```
$scrolled_window->add_with_viewport( $child ) ;
```

Le positionnement du widget enfant en respect avec les barres de défilement peut être déclaré par :

```
$scrolled_window->set_placement( $placement ) ;
```

Ou `$placement` peut être `'top_left` (le défaut), `'top_right', 'bottom_left', 'bottom_right'`.

Une valeur `'top_left'` signifie que le widget est en haut à gauche avec les barres de défilement à droite et en bas du widget.

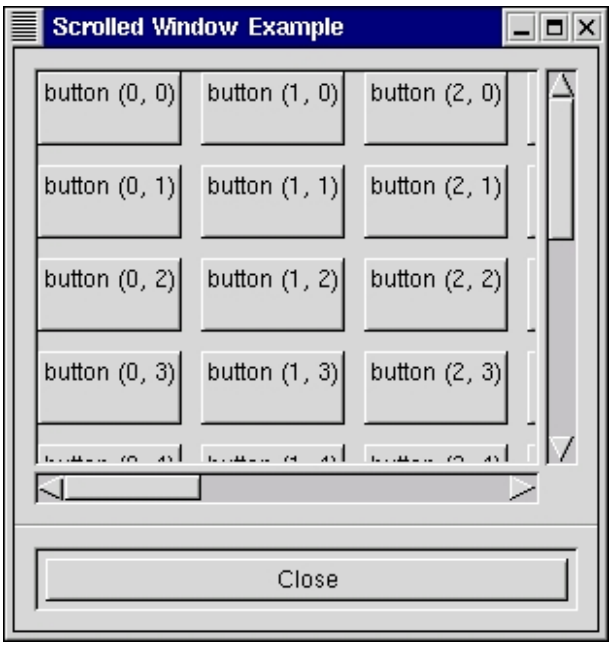
Vous pouvez récupérer ou déclarer les ajustements des barres horizontales et verticales à l'aide de :

```
$scrolled_window->get_hadjustment() ;
$scrolled_window->get_vadjustment() ;

$scrolled_window->set_hadjustment( $adjustment ) ;
$scrolled_window->set_vadjustment( $adjustment ) ;
```

4.11.1 Exemple

Voici un exemple simple qui regroupe une table avec 100 boutons toggle à l'intérieur de la fenêtre défilable.




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window;
my $scrolled_window;
my $table;
my $button;

my $buffer;
my $i;
my $j;

# Crée une nouvelle fenêtre de dialogue pour y placer la fenêtre défilable.
$window = new Gtk : :Dialog();
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$window->set_title( "Scrolled Window Example" );
$window->border_width( 0 );
$window->set_usize( 300, 300 );

# Crée une nouvelle fenêtre défilable.
$scrolled_window = new Gtk : :ScrolledWindow( "", "" );
$scrolled_window->border_width( 10 );

# La politique est soit ‘‘automatic’’ soit ‘‘always’’.’automatic’ décidera
#automatiquement si vous avez besoin de barres de défilement alors que
# ’always’ laissera les barres de défilement là. Le premier est la barre
# de défilement horizontale, le second la verticale.
$scrolled_window->set_policy( "automatic", "always" );

# La fenêtre de dialogue est créée avec une vbox à l’intérieur.
```

```
$window->vbox->pack_start( $scrolled_window, $true, $true, 0 ) ;
$scrolled_window->show() ;

# crée une table de 10 cases sur 10.
$table = new Gtk : :Table( 10, 10, $false ) ;

# Déclare l'espacement de 10 et x et en y
$table->set_row_spacings( 10 ) ;
$table->set_col_spacings( 10 ) ;

# Place la table dans la fenêtre défilable
$scrolled_window->add_with_viewport( $table ) ;
$table->show() ;

# Cela crée simplement une grille de bouton toggle sur la table
# pour faire une démonstration de la fenêtre défilable
for ( $i = 0 ; $i < 10 ; $i++ )
  {
    for ( $j = 0 ; $j < 10 ; $j++ )
      {
        $buffer = "button (" . $i . ", " . $j . ")\n" ;
        $button = new Gtk : :ToggleButton( $buffer ) ;
        $table->attach_defaults( $button, $i, $i + 1, $j, $j + 1 ) ;
        $button->show() ;
      }
  }

#  Ajoute une bouton ‘‘Close’’ à la fenêtre de dialogue. Notez que plutôt
# que de sortir directement, nous indiquons plutôt la fenêtre à fermer.
# De cette manière, si nous voulons différents comportements quand on
# ferme une fenêtre, nous n'avons qu'à changer le signal connect. Un moyen
# encore meilleur serait d'appeler une fonction pour clore la fenêtre.
$button = new Gtk : :Button( "Close" ) ;
$button->signal_connect_object( "clicked", \&Gtk : :widget_destory, $window ) ;

# Cela fait que le bouton est le bouton par défaut
$button->can_default( $true ) ;
$window->action_area->pack_start( $button, $true, $true, 0 ) ;

# Cela rend permet de faire du bouton le bouton par défaut en faisant
# que les évènements du clavier le concerne. Ainsi, taper simplement sur la
# touche ‘‘Enter’’ et cela activera le bouton.
$button->grab_default() ;

$button->show() ;
$window->show() ;
main Gtk ;
exit( 0 ) ;
```

Amuser vous à redimensionner la fenêtre. Vous verrez comment les barres de défilements réagissent. Vous pouvez aussi souhaiter l'appel `set_usize()` pour déclarer la taille par défaut de la fenêtre ou des autres widgets.

4.12 Les vues

```
Object
  +--- Widget
       +--- Container
            +--- Bin
                 +--- Viewport
```

Il y a peu de chance que que vous ayez un jour le besoin d'utiliser un widget vue directement. Il est plus probable que vous utilisiez une fenêtre défilable qui utilise une vue.

Un widget vue vous permet de placer un plus grand widget à l'intérieur de sorte que vous puissiez en voir un bout. Cela utilise les ajustements pour définir l'aire visible. Voir la section sur les ajustements pour des informations complémentaires.

Pour créer une vue :

```
$viewport = new Gtk : :Viewport( $hadjustment, $vadjustment ) ;
```

Comme vous pouvez le constater, vous pouvez spécifier les ajustements horizontaux et verticaux que le widget doit utiliser quand vous le créez. Il créera les siens si vous passez des valeurs nulles en arguments.

Vous pouvez récupérer ou déclarer les ajustements des barres horizontales et verticales à l'aide de :

```
$viewport->get_hadjustment() ;
$viewport->get_vadjustment() ;

$viewport->set_hadjustment( $adjustment ) ;
$viewport->set_vadjustment( $adjustment ) ;
```

La seule fonction qui puisse modifier l'apparence d'une vue :

```
$viewport->set_shadow_type( $type ) ;
```

Les valeurs possibles du paramètre `$type` sont :

```
'none'
'in'
'out'
'etched_in'
'etched_out'
```

4.13 La boîte à évènements

```
Object
 +--- Widget
      +--- Container
           +--- Bin
                +--- EventBox
```

Certains widgets GTK n'ont pas de fenêtre X associée donc ils dessinent juste sur leurs parents. En raison de cela, ils ne peuvent recevoir d'évènements et s'ils sont incorrectement dimensionnés, ils ne se retaillent pas de sorte que vous pouvez rencontrer des recouvrements désastreux, etc... Si vous voulez tirer plus de ces widgets, la boîte à évènements est faîte pour vous.

Au premier regard, le widget Boîte à évènements peut apparaître totalement inutile. Il ne dessine rien à l'écran et ne répond à aucun évènement. En revanche, il sert une fonction - il fournit une fenêtre X pour ses widgets enfants. C'est important en raison du nombre de widgets qui n'ont pas de fenêtre X associée. Ne pas avoir de fenêtre X sauve de la mémoire et améliore les performances, mais il y a tout de même quelques bémols. Un widget sans fenêtre ne peut pas recevoir d'évènements, et ne peut pas retailler ses contenus. Bien que le nom boîte à évènements insiste sur le côté manipulation des évènements, on peut également l'utiliser pour retailler (et plus, voir l'exemple ci-dessous).

Pour créer une nouvelle boîte à évènements :

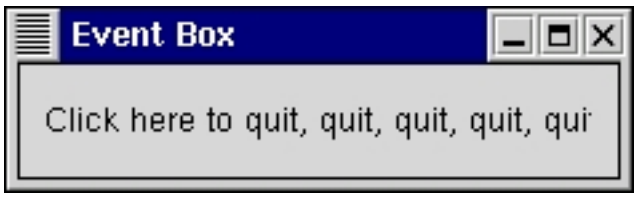
```
new Gtk : :EventBox() ;
```

Un enfant peut être ajouté à cette boîte à évènements grâce à :

```
$event_box->add($child_widget) ;
```

4.13.1 Exemple

L'exemple suivant illustre les deux usages d'une boîte à évènements - un label est créé et est retaillé pour entrer dans une petite boîte, et on déclare qu'un clic de souris sur le label sort du programme. Redimensionner la fenêtre révèle les quantités variables du label.




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window;
my $event_box;
my $label;

$window = new Gtk : :Window( "toplevel" );
$window->set_title( "Event Box" );
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$window->border_width( 10 );

# Crée une boîte à évènements et l'ajoute à notre fenêtre Top Level

$event_box = new Gtk : :EventBox();
$window->add( $event_box );
$event_box->show();

# Crée un long label

$label = new Gtk : :Label( "Click here to quit, quit, quit, quit, quit" );
$event_box->add( $label );
$label->show();

# Le raccourcit
$label->set_usize( 110, 20 );

# On le relie à une action
$event_box->set_events( 'button_press_mask' );
$event_box->signal_connect( "button_press_event", sub { Gtk->exit( 0 ); } );

# Encore une chose supplémentaire pour laquelle vous avez besoin d'une fenêtre
# X window
$event_box->realize();

# 58 est le nombre pour 'hand1' ce qui ne marche pas pour plusieurs raisons
$event_box->window->set_cursor( new Gtk : :Gdk : :Cursor( 58 ) );

$window->show();

main Gtk;
exit( 0 );
```

4.14 Le widget Alignement

Le widget alignement vous permet de placer un widget dans une fenêtre et dont la position et la taille est relative à la taille du widget alignement lui-même. Par exemple, cela peut être très utile pour centrer un widget dans une fenêtre. Il y a deux fonctions associées au widget Alignement :

```
$alignment = new Gtk : :Alignment ( $xalign, $yalign, $xscale, $yscale ) ;
$alignment->set($xalign, $yalign, $xscale, $yscale ) ;
```

La première crée un nouveau widget Alignement avec les paramètres spécifiés. La seconde fonction permet de changer les paramètres d'un widget Alignement existant.

Les quatre paramètre d'alignements sont des nombres décimaux compris entre 0.0 et 1.0. Les arguments `$xalign` et `$yalign` affecte la position du widget placé dans le widget Alignement. Les arguments `$xscale` et `$yscale` déclare la quantité d'espace alloué au widget.

Un widget enfant peut être ajouté à un widget Alignement par :

```
$alignment->add($widget_enfant) ;
```

Pour voir un exemple de widget Alignement, se référer à l'exemple sur les barres de progression.

4.15 Les widgets Paned Window

```
Object
  +--- Widget
       +--- Container
            +--- Paned
```

Les widgets Paned Window sont très utiles quand vous voulez diviser une surface en deux parties et dont la taille relative de ces parties est contrôlée par l'utilisateur. Une barre est tracée entre les deux parties avec une poignée que l'utilisateur peut saisir pour changer le ratio. La division peut être soit horizontale (HPaned) soit verticale (VPaned).

Pour créer une nouvelle fenêtre Paned :

```
$paned = new Gtk : :HPaned() ;
$paned = new Gtk : :VPaned() ;
```

Après avoir créer une fenêtre Paned, il faut ajouter les widgets enfants aux deux moitiés. Pour faire cela, utilisez l'une des fonctions suivantes :

```
$paned->pack1( $child, $resize, $shrink ) ;
$paned->add1( $child ) ;

$paned->pack2( $child, $resize, $shrink ) ;
$paned->add2( $child ) ;
```

`add1()` et `pack1()` ajoute le widget enfant dans la moitié gauche ou haute de la fenêtre Paned. `add2()` et `pack2()` ajoute le widget enfant dans la moitié droite ou basse de la fenêtre Paned.

L'argument `$resize` spécifie si l'enfant doit s'étendre quand la fenêtre Paned est redimensionnée et l'argument `$shrink` précise si l'enfant peut devenir plus petit que la taille requise.

La fonction `add1()` est équivalente à :

```
$paned->pack1( $child, $false, $true ) ;
```

La fonction `add2()` est équivalente à :

```
$paned->pack2( $child, $false, $false ) ;
```

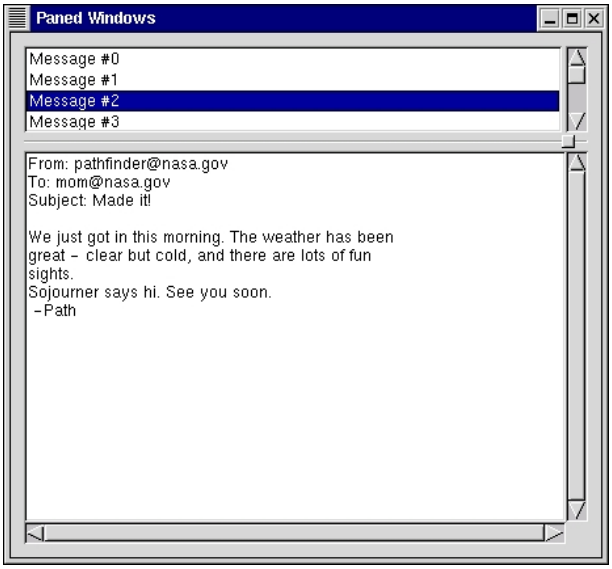
Une fenêtre Paned peut être changée visuellement à l'aide des deux fonctions suivantes :

```
$paned->set_handle_size( $size ) ;
$paned->set_gutter_size( $size ) ;
```

La première déclare la taille de la poignée et la seconde la taille de la gouttière qui sépare les deux parties de la fenêtre.

4.15.1 Exemple

Comme exemple, nous créerons un bout d'une interface utilisateur d'un programme imaginaire d'émails. Une fenêtre est divisée en deux verticalement, avec en partie haute la liste des messages et en partie basse le texte des messages. La majeure partie du programme est simple. Deux points à noter : on ne peut ajouter du texte à un widget Texte qui n'est pas réalisé. Cela peut être fait en appelant la fonction `realize()`, mais pour illustrer une technique alternative, nous avons connecté un gestionnaire au signal "realize" pour ajouter le texte. D'autre part, nous devons ajouter une option `'shrink'` à certains items de la table contenant la fenêtre texte et ses barres de défilement, de sorte que quand la partie basse est réduite, la part correcte diminue plutôt que d'être poussée en dehors du bas de la fenêtre.




```
#!/usr/bin/perl -w

use Gtk;
use strict;

init Gtk;

my $false = 0;
my $true = 1;

my $window;
my $paned;
my $list;
my $text;

$window = new Gtk : :Window( "toplevel" );
$window->set_title( "Paned Windows" );
$window->signal_connect( "destroy", sub { Gtk->exit( 0 ); } );
$window->border_width( 10 );
$window->set_usize( 450, 400 );

# crée un widget vpaned et l'ajoute à notre fenêtre Top Level
$paned = new Gtk : :VPaned();
$window->add( $paned );
$paned->set_handle_size( 10 );
$paned->set_gutter_size( 15 );
$paned->show();

# Maintenant, crée le contenu de chacune des moitiés.
$list = create_list();
```

```
$paned->add1( $list );
$list->show();

$text = create_text();
$paned->add2( $text );
$text->show();
$window->show();

main Gtk;
exit( 0 );



### Routines

sub create_list
  {
    my $scrolled_window;
    my $list;
    my $list_item;

    my $i;
    my @buffer;

    # Crée une fenêtre défilable avec des barres de défilement uniquement
    # en cas de besoin
    $scrolled_window = new Gtk : :ScrolledWindow( "", "" );
    $scrolled_window->set_policy( 'automatic', 'automatic' );

    # Crée une nouvelle liste et la place dans la fenêtre défilable
    $list = new Gtk : :List();
    $scrolled_window->add_with_viewport( $list );
    $list->show();

    # Ajoute quelques messages dans la fenêtre
    for ( $i = 0; $i < 10; $i++ )
      {
        $buffer[ $i ] = "Message \#" . $i;
        $list_item = new Gtk : :ListItem( $buffer[ $i ] );
        $list->add( $list_item );
        $list_item->show();
      }

    return ( $scrolled_window );
}

# Ajoute du texte à notre widget texte - c'est un rappel qui est invoqué
# quand notre fenêtre est réalisée. Nous pourrions également forcer notre
# fenêtre à être réalisée avec realize() mais cela doit d'abord être une
# partie de la hiérarchie.
sub realize_text
  {
    my ( $text ) = @_;

    $text->freeze();
    $text->insert( "", $text->style->black, $text->style->white,
                   "From : pathfinder\@nasa.gov\n" .
                   "To : mom\@nasa.gov\n" .
```

```
                    "Subject : Made it !\n" .
                    "\n" .
                    "We just got in this morning. The weather has been\n" .
                    "great - clear but cold, and there are lots of fun\n" .
                    "sights.\n" .
                    "Sojourner says hi. See you soon.\n" .
                    " -Path\n" ) ;

    $text->thaw() ;
  }

# Crée une zone de texte défilable qui affiche les messages.
sub create_text
  {
    my $table ;
    my $text ;
    my $hscrollbar ;
    my $vscrollbar ;

    # Crée une table pour contenir le widget texte et les barres de défilement
    $table = new Gtk : :Table( 2, 2, $false ) ;

    # Place un widget texte dans le coin en haut à gauche. Notez l'usage de
    # 'shrink' dans la direction y
    $text = new Gtk : :Text( "", "" ) ;
    $table->attach( $text, 0, 1, 0, 1,
                    [ 'fill', 'expand' ],
                    [ 'fill', 'expand', 'shrink' ],
                    0, 0 ) ;
    $text->show() ;

    # Place une HScrollbar dans le coin inférieur gauche
    $hscrollbar = new Gtk : :HScrollbar( $text->hadj ) ;
    $table->attach( $hscrollbar, 0, 1, 1, 2,
                    [ 'expand', 'fill' ],
                    'fill', 0, 0 ) ;
    $hscrollbar->show() ;

    #Et une VScrollbar en haut à droite
    $vscrollbar = new Gtk : :VScrollbar( $text->vadj ) ;
    $table->attach( $vscrollbar, 1, 2, 0, 1, 'fill',
                    [ 'expand', 'fill', 'shrink' ], 0, 0 ) ;
    $vscrollbar->show() ;

    # Ajoute un gestionnaire pour mettre le message dans le widget texte quand
    # il est réalisé
    $text->signal_connect( "realize", \&realize_text ) ;

    return ( $table ) ;
}
```

Chapitre 5

Général

5.1 Les fonctions Timeouts, IO et Idle

5.1.1 Timeouts

Peut être vous êtes vous déjà demandé comment faire pour que GTK fournisse un travail utile durant `main Gtk`. Et bien, vous avez plusieurs possibilités. En utilisant la fonction suivante vous pouvez créer une fonction timeout qui sera appelée à intervalle régulier (donné en millisecondes).

```
$timer = Gtk->timeout_add( $interval, \&function, @function_data ) ;
```

Le premier argument est le nombre de millisecondes entre deux appels de votre fonction. Le second argument est la fonction que vous voulez appeler et le troisième les données passées à votre fonction. La valeur de retour est un entier "tag" qui peut être utiliser pour stopper le timeout à l'aide de :

```
Gtk->timeout_remove( $timer ) ;
```

Vous pouvez également stopper une fonction timeout en retournant une valeur fausse à partir de votre fonction de rappel. Évidemment, cela signifie que si vous voulez que votre fonction timeout continue il faut retourner une valeur vraie.

La déclaration de votre fonction de rappel doit ressembler à :

```
  sub timeout_callback
{
  my ( @data ) = @_ ;
  ...
}
```

5.1.2 Surveiller les IO

Une facette de GDK (la librairie sur laquelle repose GTK) est sa capacité à contrôler pour vous des données d'un fichier descripteur (comme en retourne open(2) ou socket(2)). C'est particulièrement utile pour les applications réseaux. La fonction :

```
$id = Gtk : :Gdk->input_add( $source, $condition, \&function, @data ) ;
```

ou le premier argument est le fichier descripteur que vous voulez regarder, le second précise ce que GDK doit chercher. Ce peut être soit :

`'read'` - Appelle votre fonction quand il y a une donnée prête à être lue dans le fichier descripteur.

`'write'` - Appelle votre fonction quand le fichier descripteur est prêt pour l'écriture.

Et je suis sûr que vous avez déjà deviné que le troisième argument est la fonction que vous voulez appeler quand l'une des conditions précédentes est satisfaite, et le quatrième les données à passer à votre fonction.

La valeur de retour est un tag qui peut être utilisé pour que GDK arrête sa surveillance du fichier descripteur en utilisant la fonction suivante :

```
Gtk : :Gdk->input_remove( $id ) ;
```

La déclaration de votre fonction de rappel doit ressembler à :

```
  sub input_callback
{
  my ( $source, $condition, @data ) = @_;
  ...
}
```

5.1.3 Les fonctions Idle

Et si vous voulez appeler une fonction quand il ne se passe rien ?

```
$idle = Gtk->idle_add( \&function, @data );
```

Cela fait que GTK appelle la fonction spécifiée dès que rien ne se passe.

```
Gtk->idle_remove( $idle );
```

La signification des arguments est identique à la précédente. La fonction appelée par le premier argument de `idle_add()` sera appelée dès que l'opportunité se présentera. Comme les autres, si on retourne une valeur fausse la fonction idle ne sera plus appelée.

Chapitre 6

Administration

6.1 Copyright du tutoriel et Permissions

6.1.1 Version française

6.1.2 Version originale