# Tutorial GTK+ 2.x para iniciantes

Tutorial traduzido baseado na versão original do website do GTK+:
http://zetcode.com/tutorials/gtktutorial/

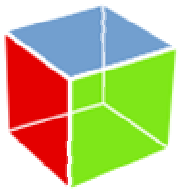
Tradução:
Bruno Sampaio Pinho da Silva
Outubro de 2009

Dúvidas e sugestões:
bpinhosilva@hotmail.com

Tutorial GTK+ 2.x para iniciantes

1. Introdução

Esta é uma introdução ao tutorial de programação GTK. O tutorial é feito para a linguagem de programação C. Foi criado e testado no GNU/Linux. O tutorial é direcionado para programadores iniciantes e intermediários.



A GTK+ é uma biblioteca para criação de interfaces gráficas de usuários. Ela foi implementada na linguagem C. Ela é também chamada de kit de ferramentas do GIMP. Originalmente, a biblioteca foi criada enquanto desenvolvia-se o programa de manipulação de imagem GIMP. Desde então, o GTK+ tornou-se um dos mais populares kits de ferramentas para GNU/Linux e BSD Unix. Hoje, muitos dos programas GUI em código aberto do mundo são criados em Qt ou em GTK+. A GTK+ é uma interface de programação de aplicativos orientada a objetos. O sistema orientado a objetos foi criado com o sistema de objeto Glib, que é a base para a biblioteca GTK+. O GObject também permite criar ligação para várias outras linguagens de programação. Ligações de linguagens existem para C++, Python, Perl, Java, C# e outras.
A GTK+ depende das seguintes bibliotecas:

- Glib
- Pango
- ATK
- GDK
- GdkPixbuf
- Cairo

A **Glib** é uma biblioteca de utilidade geral. Fornece vários tipos de dados, utilidades de strings, permite relatório de erros, registro de mensagens, trabalho com threads e outras características de programação úteis. **Pango** é uma biblioteca que habilita internacionalização. **ATK** é um conjunto de ferramentas de acessibilidade. Este conjunto de ferramentas permite uma ajuda física para pessoas desafiadoras que trabalham com computadores. A **GDK** é um envoltório de desenho e funções de janela de baixo nível que permite uma base de sistemas gráficos. No GNU/Linux, GDK varia entre o Servidor X e a biblioteca GTK+. Recentemente, muitas dessas funcionalidades foram atribuídas à biblioteca Cairo. A **GdkPixbuf** é composta por ferramentas que são responsáveis por carregar imagens e por manipular pixel. **Cairo** é uma biblioteca para criação gráfica de vetor 2D. Foi incluída em GTK+ desde a versão 2.8.
Os ambientes de desktop Gnome e o Xfce foram criados usando GTK+. SWT e wxWidgets são as estruturas de programação conhecidas. As aplicações que usam GTK+ incluem Firefox ou Inkscape.

## Compilando aplicações GTK+

Para compilar tais aplicações, nós temos que ter em mãos uma ferramenta chamada **pkg-config**. O pgk-config retorna dados sobre bibliotecas instaladas. Se nós quisermos usar uma biblioteca específica, ela permitirá bibliotecas dependentes necessárias e arquivos de cabeçalho, que precisamos. O pkg-config recupera informação sobre pacotes de arquivos especiais.

```
gcc -o simple simple.c `pkg-config --libs --cflags gtk+-2.0`
```

Aqui mostramos como podemos compilar um simples programa. O código fonte consiste em um arquivo, simple.c. Isso não são aspas simples, mas, um acento grave.

```
$ pkg-config --cflags gtk+-2.0
-I/usr/include/gtk-2.0 -I/usr/lib/gtk-2.0/include -I/usr/include/atk-1.0
-I/usr/include/cairo -I/usr/include/pango-1.0 -I/usr/include/glib-2.0
-I/usr/lib/glib-2.0/include -I/usr/include/freetype2 -I/usr/include/libpng12
```

Lista que mostra todos os arquivos de cabeçalhos necessários para programação GTK+.

```
$ pkg-config --libs gtk+-2.0
-lgtk-x11-2.0 -lgdk-x11-2.0 -latk-1.0 -lgdk_pixbuf-2.0 -lm -lpangocairo-1.0
-lfontconfig -lXext -lXrender -lXinerama -lXi -lXrandr
-lXcursor -lXfixes -lpango-1.0 -lcairo -lX11 -lgobject-2.0
-lgmodule-2.0 -ldl -lglib-2.0
```

E a lista com todas as bibliotecas necessárias.

2. Primeiro programa em GTK+
Nesta parte do tutorial, nós iremos criar nosso primeiro programa em GTK+.

Exemplo simples

Iniciaremos com um exemplo muito simples, mostraremos uma janela básica.

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

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

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_widget_show(window);

  gtk_main();

  return 0;
}
```

Este exemplo mostrará uma janela básica na tela.

```
  gcc -o simple simple.c `pkg-config --libs --cflags gtk+-2.0`
```

Aqui, como compiler o exemplo.

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

Aqui nós inicializamos a bibioteca GTK+.

```
window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

Nós criamos um widget(engenhoca, quer dizer objetos, componentes de uma janela) **GtkWindow**. O tipo da janela é **GTK_WINDOW_TOPLEVEL**. A janela Toplevel possui uma barra de títulos e uma borda.São gerenciados por um gerenciador de janela.

```
gtk_widget_show(window);
```

Depois de termos criados um widget, iremos mostrá-lo.

```
gtk_main();
```

Este código faz a GTK+ entrar em um laço de repetição. A partir daqui, a aplicação espera que algum evento aconteça.

Figura: Simples

## Centralizando a janela

Se nós não posicionarmos nossa janela, o gerenciador de janela irá posicioná-la para nós. No próximo exemplo, nós centralizamos a janela.

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

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

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "Center");
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_widget_show(window);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_main();

  return 0;
}
```

Em nosso exemplo, nós centralizamos a janela, colocamos um título e um tamanho para ela.

```
gtk_window_set_title(GTK_WINDOW(window), "Center");
```

A função **gtk_window_set_title()** irá setar um título para a janela. Se não setarmos, a GTK+ irá usar o nome do arquivo do código fonte para o título da janela. Por exemplo, se seu programa for 1.c, a janela se chamará 1.

```
gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
```

Este código seta o tamanho da janela para 230x150 pixels. Note que estamos falando sobre a área de usuário, excluindo decorações providas pelo gerenciador de janela.

```
gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
```

Este código centraliza a janela.

```
g_signal_connect_swapped(G_OBJECT(window), "destroy",
     G_CALLBACK(gtk_main_quit), NULL);
```

No exemplo anterior, a janela não era completamente fechada quando clicávamos no botão 'x'. Podemos ver isso, se executarmos o exemplo pela linha de comando. A janela não reagia ao sinal de destruição (**destroy** signal,fechamento) por padrão. Vamos explicitar a terminação da aplicação pela conexão do sinal de destruição da função **gtk_main_quit()**.

## O aplicativo ícone

No próximo exemplo, mostraremos uma aplicação de ícone. Muitos gerenciadores de janela mostram os ícones no canto esquerdo superior da barra de títulos e também na barra de tarefas.

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

GdkPixbuf *create_pixbuf(const gchar * filename)
{
   GdkPixbuf *pixbuf;
   GError *error = NULL;
   pixbuf = gdk_pixbuf_new_from_file(filename, &error);
   if(!pixbuf) {
      fprintf(stderr, "%s\n", error->message);
      g_error_free(error);
   }

   return pixbuf;
}

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

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "icon");
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_icon(GTK_WINDOW(window), create_pixbuf("web.png"));
  gtk_widget_show(window);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_main();

  return 0;
}
```

O código mostra o ícone. O arquivo de ícone (no caso web.png) tem que estar no mesmo diretório do executável e os códigos do programa.

```
 gtk_window_set_icon(GTK_WINDOW(window), create_pixbuf("web.png"));
```

A função **gtk_window_set_icon()** mostra o ícone de nossa janela. A função **create_pixbuf()** cria um **GdkPixbuf** de um arquivo png.

```
 pixbuf = gdk_pixbuf_new_from_file(filename, &error);
```

De acordo com a documentação, a função **gdk_pixbuf_new_from_file()** cria um novo pixbuf para carregar a imagem de um arquivo. O formato de arquivo é detectado automaticamente. Será retornado NULL se algum erro for detectado.

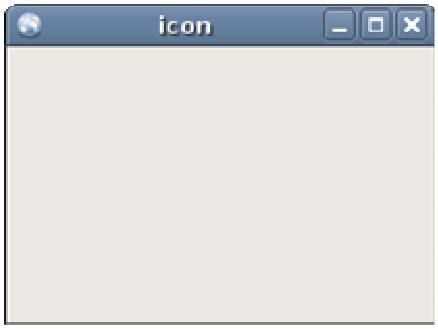




Figura: ícone

## Incremento – Decremento

Finalizamos a primeira parte do tutorial com um exemplo, onde temos 3 janelas filhas. 2 botões e uma etiqueta. A etiqueta possui um número inteiro. Os botões irão incrementar ou decrementar o número.

```
#include <gtk/gtk.h>
gint count = 0;
char buf[5];
void increase(GtkWidget *widget, gpointer label)
{
  count++;
  sprintf(buf, "%d", count);
  gtk_label_set_text(label, buf);
}
void decrease(GtkWidget *widget, gpointer label)
{
  count--;
  sprintf(buf, "%d", count);
  gtk_label_set_text(label, buf);
}
int main(int argc, char** argv) {

  GtkWidget *label;
  GtkWidget *window;
  GtkWidget *frame;
  GtkWidget *plus;
  GtkWidget *minus;
  gtk_init(&argc, &argv);
  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 180);
  gtk_window_set_title(GTK_WINDOW(window), "+-");
  frame = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), frame);
  plus = gtk_button_new_with_label("+");
  gtk_widget_set_size_request(plus, 80, 35);
  gtk_fixed_put(GTK_FIXED(frame), plus, 50, 20);
  minus = gtk_button_new_with_label("-");
  gtk_widget_set_size_request(minus, 80, 35);
  gtk_fixed_put(GTK_FIXED(frame), minus, 50, 80);
  label = gtk_label_new("0");
  gtk_fixed_put(GTK_FIXED(frame), label, 190, 58);
  gtk_widget_show_all(window);
  g_signal_connect(window, "destroy",
      G_CALLBACK (gtk_main_quit), NULL);

  g_signal_connect(plus, "clicked",
      G_CALLBACK(increase), label);

  g_signal_connect(minus, "clicked",
      G_CALLBACK(decrease), label);

  gtk_main();
  return 0;}
```

O código-exemplo incrementa ou decrementa um valor em GtkLabel.

```
g_signal_connect(plus, "clicked",
    G_CALLBACK(increase), label);
```

Nós conectamos a função de rechamada **increase()** para o botão '+'. Note que enviamos uma etiqueta como parâmetro para a rechamada (callback). Iremos trabalhar na etiqueta dentro da função de callback.

```
count++;
sprintf(buf, "%d", count);
gtk_label_set_text(label, buf);
```

Dentro da callback increase, nós criamos o contador. Fizemos dados textuais fora do valor do número e atualizamos a etiqueta.

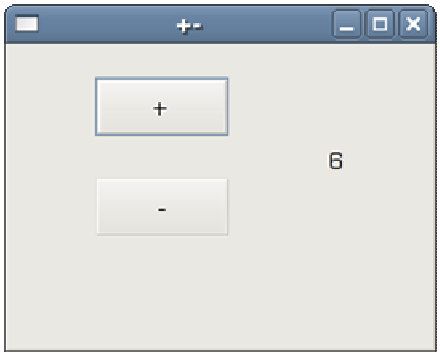


Figura: Incremento – Decremento

# Menus e barra de ferramentas em GKT+

Nesta parte do tutorial, trabalharemos com menus e barra de ferramentas.

Uma **barra de ferramentas** é uma das partes mais comuns de uma aplicação GUI. É um grupo de comandos localizado em vários menus. Enquanto que em aplicações de console você tem que lembrar todos aqueles comandos misteriosos, aqui nós temos muitos comandos agrupados em partes lógicas. Existem padrões aceitos que reduzem muito do tempo gasto para aprender uma nova aplicação.

## Exemplo simples de menu

Em nosso primeiro exemplo, criaremos uma barra de menu com um menu de arquivo. O menu terá apenas um item de menu. Ao selecionar o item, fechará a aplicação.

```
#include <gtk/gtk.h>
int main( int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *vbox;
  GtkWidget *menubar;
  GtkWidget *filemenu;
  GtkWidget *file;
  GtkWidget *quit;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 200);
  gtk_window_set_title(GTK_WINDOW(window), "menu");

  vbox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  menubar = gtk_menu_bar_new();
  filemenu = gtk_menu_new();

  file = gtk_menu_item_new_with_label("File");
  quit = gtk_menu_item_new_with_label("Quit");

  gtk_menu_item_set_submenu(GTK_MENU_ITEM(file), filemenu);
  gtk_menu_shell_append(GTK_MENU_SHELL(filemenu), quit);
  gtk_menu_shell_append(GTK_MENU_SHELL(menubar), file);
  gtk_box_pack_start(GTK_BOX(vbox), menubar, FALSE, FALSE, 3);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), NULL);

  g_signal_connect(G_OBJECT(quit), "activate",
        G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Criar uma barra de menus é um pouco confuso. Nós devemos ter em mente que ambos, barra de menus e menus são derivados de um mesmo widget, chamado **menu shell**. **Itens de menu** são apenas filhos de menus. Eles são também usado para implementar submenus.

```
menubar = gtk_menu_bar_new();
 filemenu = gtk_menu_new();
```

Neste código nós criamos uma barra de menu e um menu.

```
 gtk_menu_item_set_submenu(GTK_MENU_ITEM(file), filemenu);
```

Esta linha de código implementa um menu de arquivo. A lógica é que uma barra de menu é um menu shell. Arquivo de menu é também um menu shell. É porque olhamos o menu de arquivo como um submenu ou um subshell.

```
 gtk_menu_shell_append(GTK_MENU_SHELL(filemenu), quit);
 gtk_menu_shell_append(GTK_MENU_SHELL(menubar), file);
```

Itens de menu são implementados chamando a função **gtk_menu_shell_append()**. Itens de menu são adicionados aos menus shell. Em nosso caso, o menu de sair é adicionado ao menu de arquivo e também o item do menu de arquivo é adicionado à barra de menu.

```
  g_signal_connect(G_OBJECT(quit), "activate",
        G_CALLBACK(gtk_main_quit), NULL);
```

Selecionando o item de menu sair, sairemos da aplicação.

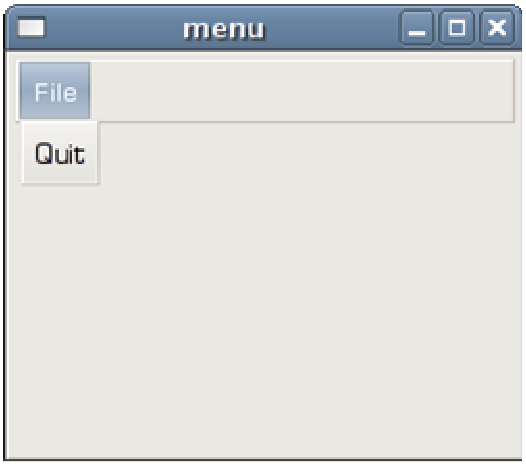



Figura: Menu simples

# Imagem de menu, mnemônicos e aceleradores

No próximo exemplo, iremos explorar mais uma funcionalidade que podemos usar em GTK+. **Aceleradores** são atalhos de teclado para ativar menus de itens. **Mnemônicos (que facilmente se grava na memória)** atalhos de teclado para elementos da GUI. Eles são representados por caracteres sublinhados.

```
#include <gtk/gtk.h>
#include <gdk/gdkkeysyms.h>
int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *vbox;
  GtkWidget *menubar;
  GtkWidget *filemenu;
  GtkWidget *file;
  GtkWidget *new;
  GtkWidget *open;
  GtkWidget *quit;

  GtkWidget *sep;
  GtkAccelGroup *accel_group = NULL;
  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 200);
  gtk_window_set_title(GTK_WINDOW(window), "menu");

  vbox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  menubar = gtk_menu_bar_new();
  filemenu = gtk_menu_new();
  accel_group = gtk_accel_group_new();
  gtk_window_add_accel_group(GTK_WINDOW(window), accel_group);
  file = gtk_menu_item_new_with_mnemonic("_File");
  new = gtk_image_menu_item_new_from_stock(GTK_STOCK_NEW, NULL);
  open = gtk_image_menu_item_new_from_stock(GTK_STOCK_OPEN, NULL);
  sep = gtk_separator_menu_item_new();
  quit = gtk_image_menu_item_new_from_stock(GTK_STOCK_QUIT, accel_group);

  gtk_widget_add_accelerator(quit, "activate", accel_group,
      GDK_q, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
  gtk_menu_item_set_submenu(GTK_MENU_ITEM(file), filemenu);
  gtk_menu_shell_append(GTK_MENU_SHELL(filemenu), new);
  gtk_menu_shell_append(GTK_MENU_SHELL(filemenu), open);
  gtk_menu_shell_append(GTK_MENU_SHELL(filemenu), sep);
  gtk_menu_shell_append(GTK_MENU_SHELL(filemenu), quit);
  gtk_menu_shell_append(GTK_MENU_SHELL(menubar), file);
  gtk_box_pack_start(GTK_BOX(vbox), menubar, FALSE, FALSE, 3);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  g_signal_connect(G_OBJECT(quit), "activate",
      G_CALLBACK(gtk_main_quit), NULL);
  gtk_widget_show_all(window);
  gtk_main();
  return 0; }
```

O exemplo mostra como adicionar uma imagem ao nosso menu de item. Como setar um acelerador e como usar mnemônicos em nossas aplicações GTK+.

```
  accel_group = gtk_accel_group_new();
  gtk_window_add_accel_group(GTK_WINDOW(window), accel_group);
  ...
  quit = gtk_image_menu_item_new_from_stock(GTK_STOCK_QUIT, accel_group);
  gtk_widget_add_accelerator(quit, "activate", accel_group,
      GDK_q, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
```

Um grupo de acelerador é um grupo de aceleradores de teclado

An accelerator group is a group of keyboard accelerators, tipicamente unido ao nível superior da janela. Aqui criamos um acelerador de teclado Ctrl + q.

```
 file = gtk_menu_item_new_with_mnemonic("_File");
```

Para crier um mnemônico temos que chamar a função **gtk_menu_item_new_with_mnemonic()**. Selecionamos o item de menu arquivo ao pressionarz Alt + F.

```
 new = gtk_image_menu_item_new_from_stock(GTK_STOCK_NEW, NULL);
 open = gtk_image_menu_item_new_from_stock(GTK_STOCK_OPEN, NULL);
```

Aqui criamos duas imagens de itens de menu. Ao setar o segundo parâmetro da função para NULL, automaticamente criamos aceleradores. Fornecemos uma imagem e texto para nosso item de menu por recursos internos do GTK+.

```
 sep = gtk_separator_menu_item_new();
```

Itens de menu podem ser separados por um separador horizontal. Isto pode colocar itens de menus em algum grupo lógico.

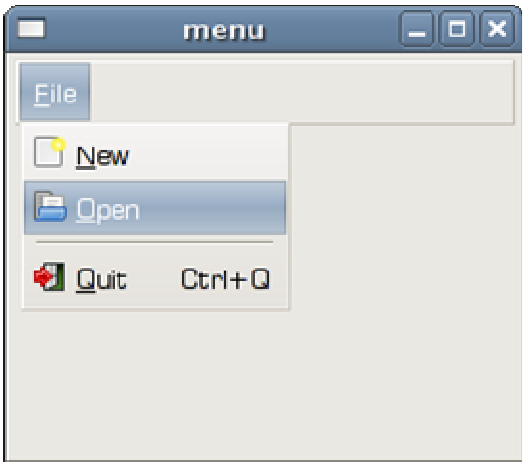


Figura: Exemplo de menu

# Item de menu de verificação

Um **GtkCheckMenuItem** é um menu com uma caixa de verificação.

```
#include <gtk/gtk.h>
void toggle_statusbar(GtkWidget *widget, gpointer statusbar)
{
  if (gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(widget))) {
    gtk_widget_show(statusbar);
  } else {
    gtk_widget_hide(statusbar);
  }
}

int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *vbox;
  GtkWidget *menubar;
  GtkWidget *viewmenu;
  GtkWidget *view;
  GtkWidget *tog_stat;
  GtkWidget *statusbar;


  gtk_init(&argc, &argv);
  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 200);
  gtk_window_set_title(GTK_WINDOW(window), "view statusbar");

  vbox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  menubar = gtk_menu_bar_new();
  viewmenu = gtk_menu_new();

  view = gtk_menu_item_new_with_label("View");
  tog_stat = gtk_check_menu_item_new_with_label("View Statusbar");
  gtk_check_menu_item_set_active(GTK_CHECK_MENU_ITEM(tog_stat), TRUE);

  gtk_menu_item_set_submenu(GTK_MENU_ITEM(view), viewmenu);
  gtk_menu_shell_append(GTK_MENU_SHELL(viewmenu), tog_stat);
  gtk_menu_shell_append(GTK_MENU_SHELL(menubar), view);
  gtk_box_pack_start(GTK_BOX(vbox), menubar, FALSE, FALSE, 3);

  statusbar = gtk_statusbar_new();
  gtk_box_pack_end(GTK_BOX(vbox), statusbar, FALSE, TRUE, 1);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), NULL);

  g_signal_connect(G_OBJECT(tog_stat), "activate",
        G_CALLBACK(toggle_statusbar), statusbar);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Em nosso código-exemplo mostramos um item de verificação. Se a caixa de verificação está ativada, o widget de barra de status é mostrado.

```
tog_stat = gtk_check_menu_item_new_with_label("View Statusbar");
```

A função de chamada **gtk_check_menu_item_new_with_label()** cria um novo item de verificação.

```
if (gtk_check_menu_item_get_active(GTK_CHECK_MENU_ITEM(widget))) {
  gtk_widget_show(statusbar);
} else {
  gtk_widget_hide(statusbar);
}
```

Se a caixa de verificação no item de menu está ativado, mostramos a barra de status. Senão, a barra de status é escondida.

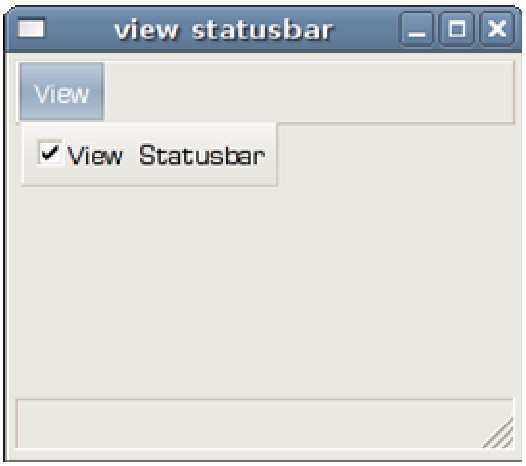


Figura: Item de verificação

# Uma barra de ferramentas

Comandos de grupos de menu que podemos usar em aplicação. Barra de ferramentas fornecem um rápido acesso aos commandos mais frequentemente usados.

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

int main( int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *vbox;
  GtkWidget *toolbar;
  GtkToolItem *new;
  GtkToolItem *open;
  GtkToolItem *save;
  GtkToolItem *sep;
  GtkToolItem *exit;


  gtk_init(&argc, &argv);
  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 200);
  gtk_window_set_title(GTK_WINDOW(window), "toolbar");
```

```
  vbox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(window), vbox);


  toolbar = gtk_toolbar_new();
  gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS);

  gtk_container_set_border_width(GTK_CONTAINER(toolbar), 2);

  new = gtk_tool_button_new_from_stock(GTK_STOCK_NEW);
  gtk_toolbar_insert(GTK_TOOLBAR(toolbar), new, -1);

  open = gtk_tool_button_new_from_stock(GTK_STOCK_OPEN);
  gtk_toolbar_insert(GTK_TOOLBAR(toolbar), open, -1);

  save = gtk_tool_button_new_from_stock(GTK_STOCK_SAVE);
  gtk_toolbar_insert(GTK_TOOLBAR(toolbar), save, -1);

  sep = gtk_separator_tool_item_new();
  gtk_toolbar_insert(GTK_TOOLBAR(toolbar), sep, -1);

  exit = gtk_tool_button_new_from_stock(GTK_STOCK_QUIT);
  gtk_toolbar_insert(GTK_TOOLBAR(toolbar), exit, -1);

  gtk_box_pack_start(GTK_BOX(vbox), toolbar, FALSE, FALSE, 5);

  g_signal_connect(G_OBJECT(exit), "clicked",
        G_CALLBACK(gtk_main_quit), NULL);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();
  return 0;
}
```

O exemplo cria uma simples barra de ferramentas.

```
  toolbar = gtk_toolbar_new();
  gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS)
```

Criamos uma nova barra de ferramentas. Especificamos que a barra mostra apenas ícones. Sem texto.

```
 new = gtk_tool_button_new_from_stock(GTK_STOCK_NEW);
 gtk_toolbar_insert(GTK_TOOLBAR(toolbar), new, -1);
```

Criamos um botão a partir de um estoque de botões. Os botões da barra de ferramentas sao inseridos na barra através da função de chamada **gtk_toolbar_insert()**.

```
 sep = gtk_separator_tool_item_new();
 gtk_toolbar_insert(GTK_TOOLBAR(toolbar), sep, -1);
```

Aqui inserimos um separados na barra de ferramentas.

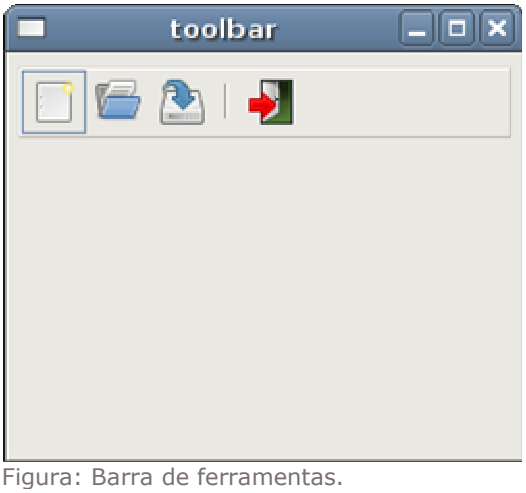


Figura: Barra de ferramentas.

## Desfaz refaz

O próximo exemplo demonstra, como podemos desativar botões da barra de ferramentas sobre ela. É uma prática comum em programação GUI. Por exemplo o botão salvar. Se salvamos todas as mudanças de nosso documento em disco, o botão de salvar é inativado na maioria dos editors de texto. Isto é uma forma da aplicação indicar ao usuário que todas as mudanças são salvas.

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


void undo_redo(GtkWidget *widget,  gpointer item)
{
  static int count = 2;
  const char *name = gtk_widget_get_name(widget);

  if ( strcmp(name, "undo") ) {
    count++;
  } else {
    count--;
  }

  if (count < 0) {
     gtk_widget_set_sensitive(widget, FALSE);
     gtk_widget_set_sensitive(item, TRUE);
  }

  if (count > 5) {
     gtk_widget_set_sensitive(widget, FALSE);
     gtk_widget_set_sensitive(item, TRUE);
  }
}


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *vbox;

  GtkWidget *toolbar;
  GtkToolItem *undo;
  GtkToolItem *redo;
  GtkToolItem *sep;
```

```
    GtkToolItem *exit;


    gtk_init(&argc, &argv);

    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
    gtk_window_set_default_size(GTK_WINDOW(window), 250, 200);
    gtk_window_set_title(GTK_WINDOW(window), "undoredo");

    vbox = gtk_vbox_new(FALSE, 0);
    gtk_container_add(GTK_CONTAINER(window), vbox);


    toolbar = gtk_toolbar_new();
    gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS);

    gtk_container_set_border_width(GTK_CONTAINER(toolbar), 2);

    undo = gtk_tool_button_new_from_stock(GTK_STOCK_UNDO);
    gtk_widget_set_name(GTK_WIDGET(undo), "undo");
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), undo, -1);

    redo = gtk_tool_button_new_from_stock(GTK_STOCK_REDO);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), redo, -1);

    sep = gtk_separator_tool_item_new();
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), sep, -1);

    exit = gtk_tool_button_new_from_stock(GTK_STOCK_QUIT);
    gtk_toolbar_insert(GTK_TOOLBAR(toolbar), exit, -1);

    gtk_box_pack_start(GTK_BOX(vbox), toolbar, FALSE, FALSE, 5);


    g_signal_connect(G_OBJECT(undo), "clicked",
          G_CALLBACK(undo_redo), redo);

    g_signal_connect(G_OBJECT(redo), "clicked",
          G_CALLBACK(undo_redo), undo);

    g_signal_connect(G_OBJECT(exit), "clicked",
          G_CALLBACK(gtk_main_quit), NULL);

    g_signal_connect_swapped(G_OBJECT(window), "destroy",
          G_CALLBACK(gtk_main_quit), NULL);

    gtk_widget_show_all(window);

    gtk_main();

    return 0;
}
```

Nosso exemplo cria botões desfaz e refaz através de recursos preexistente em GTK+. Após diversos cliques, cada um dos botões é inativado. Os botões tornam-se não selecionáveis.

```
 if (count < 0) {
    gtk_widget_set_sensitive(widget, FALSE);
    gtk_widget_set_sensitive(item, TRUE);
 }

 if (count > 5) {
    gtk_widget_set_sensitive(widget, FALSE);
    gtk_widget_set_sensitive(item, TRUE);
```

```
}
```

A função de chamada **gtk_widget_set_sensitive()** é usada para ativar/desativar os botões da barra de ferramentas.


Figura: Desfaz refaz

## GTK+ gerenciamento de layout

Nesta parte do tutorial mostraremos como export nossos widgets em janelas ou em diálogos.

Quando projetamos nossa aplicação GUI, decidimos quais e como usaremos e organizaremos nossos widgets na aplicação. Para organizá-los, usamos widgets especializados não visíveis chamados de **layout containers (uma espécie de recipiente)**. Neste capítulo mencionaremos **GtkAlignment**, **GtkFixed**, **GtkVBox** e **GtkTable**.

## GtkFixed

O recipiente **GtkFixed** é o lugar em que widgets filhos são fixados com posição e tamanhos próprios. Este recipiente

container places child widgets at fixed positions and with fixed sizes. Este recipiente não executa nenhum gerenciamento de layout. Em muitas aplicações, nós não usamos o GtkFixed. Existem algumas áreas especializadas onde usamos. Por exemplo jogos, aplicações especializadas que trabalham com diagramas, componentes manipuláveis que podem ser movidos (como um gráfico em um aplicativo de planilha), pequenos exemplos educacionais.

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

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

  GtkWidget *button1;
  GtkWidget *button2;
  GtkWidget *button3;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "GtkFixed");
```

```
  gtk_window_set_default_size(GTK_WINDOW(window), 290, 200);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);

  fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), fixed);

  button1 = gtk_button_new_with_label("Button");
  gtk_fixed_put(GTK_FIXED(fixed), button1, 150, 50);
  gtk_widget_set_size_request(button1, 80, 35);

  button2 = gtk_button_new_with_label("Button");
  gtk_fixed_put(GTK_FIXED(fixed), button2, 15, 15);
  gtk_widget_set_size_request(button2, 80, 35);

  button3 = gtk_button_new_with_label("Button");
  gtk_fixed_put(GTK_FIXED(fixed), button3, 100, 100);
  gtk_widget_set_size_request(button3, 80, 35);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Em nosso exemplo, criamos 3 botões e lugar onde podemos fixar suas coordenadas. Quando redimensionamos a janela da aplicação, os botões mantêm seus tamanhos e posições.

```
fixed = gtk_fixed_new();
```

Aqui um recipiente **GtkFixed** widget é criado.

```
 gtk_fixed_put(GTK_FIXED(fixed), button1, 150, 50);
```

o primeiro botão é colocado usando a função **gtk_fixed_put()** com as coordenadas x=150, y=50.

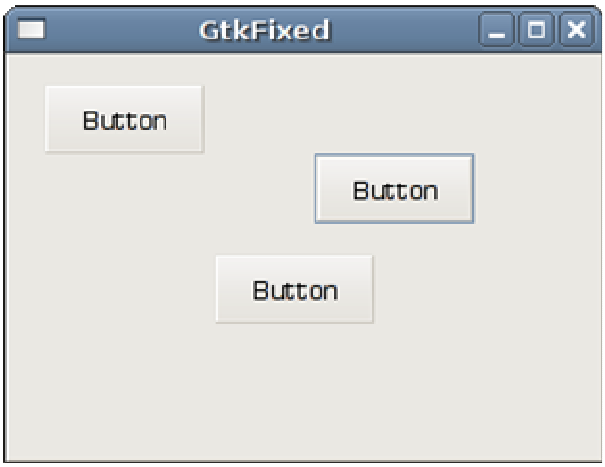


Figura: GtkFixado recipiente

# GtkVBox

**GtkVBox** é uma de recipiente vertical. Ele coloca o que é um widget filho em uma única coluna. **GtkHBox** é muito semelhante só que este coloca os widgets filhos em uma única linha.

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


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *vbox;

  GtkWidget *settings;
  GtkWidget *accounts;
  GtkWidget *loans;
  GtkWidget *cash;
  GtkWidget *debts;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 250);
  gtk_window_set_title(GTK_WINDOW(window), "GtkVBox");
  gtk_container_set_border_width(GTK_CONTAINER(window), 5);

  vbox = gtk_vbox_new(TRUE, 1);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  settings = gtk_button_new_with_label("Settings");
  accounts = gtk_button_new_with_label("Accounts");
  loans = gtk_button_new_with_label("Loans");
  cash = gtk_button_new_with_label("Cash");
  debts = gtk_button_new_with_label("Debts");

  gtk_box_pack_start(GTK_BOX(vbox), settings, TRUE, TRUE, 0);
  gtk_box_pack_start(GTK_BOX(vbox), accounts, TRUE, TRUE, 0);
  gtk_box_pack_start(GTK_BOX(vbox), loans, TRUE, TRUE, 0);
  gtk_box_pack_start(GTK_BOX(vbox), cash, TRUE, TRUE, 0);
  gtk_box_pack_start(GTK_BOX(vbox), debts, TRUE, TRUE, 0);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Este exemplo mostra um **GtkVBox** em ação. Ele empacota 5 botões em uma coluna. Se redimensionarmos a janela de nossa aplicação, os widgets filhos serão redimensionados também.

```
vbox = gtk_vbox_new(TRUE, 1);
```

O **GtkVBox** é criado. Setamos o parâmetros homogêneo para TRUE. Isto significa que todos os nossos botões terão o mesmo tamanho. O espaçamento entre widgets é setado para 1 pixel.

```
gtk_box_pack_start(GTK_BOX(vbox), settings, TRUE, TRUE, 0);
```

Empacotamos um botão de ajuestes no recipiente. Os dois parâmetros são o recipiente e o widget filho. Os próximos três parâmetros são expansão, preenchimento e estofamento. Note que o parâmetro de preenchimento não tem efeito, o de expansão é setado para FALSO. Similarmente, o parâmetro de expansão não tem efeito se criarmos um recipiente com parâmetro homogeneous TRUE.

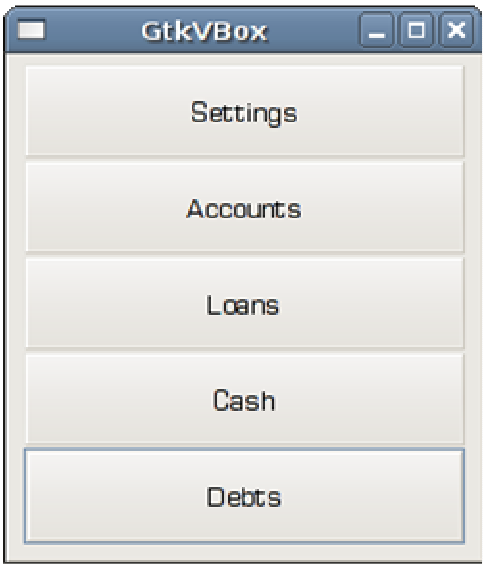



Figure: GtkVBox recipiente

# GtkTabela

O widget **GtkTable** arruma widgets em linhas e colunas.

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


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *table;
  GtkWidget *button;

  char *values[16] = { "7", "8", "9", "/",
     "4", "5", "6", "*",
     "1", "2", "3", "-",
     "0", ".", "=", "+"
  };

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 180);
  gtk_window_set_title(GTK_WINDOW(window), "GtkTable");

  gtk_container_set_border_width(GTK_CONTAINER(window), 5);

  table = gtk_table_new(4, 4, TRUE);
  gtk_table_set_row_spacings(GTK_TABLE(table), 2);
  gtk_table_set_col_spacings(GTK_TABLE(table), 2);
```

```
  int i = 0;
  int j = 0;
  int pos = 0;

  for( i=0; i < 4; i++) {
    for( j=0; j < 4; j++) {
      button = gtk_button_new_with_label(values[pos]);
      gtk_table_attach_defaults(GTK_TABLE(table), button, j, j+1, i, i+1 );
      pos++;
    }
  }

  gtk_container_add(GTK_CONTAINER(window), table);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Neste exemplo, criamos um set de botões que vemos em calculadoras.

```
table = gtk_table_new(4, 4, TRUE);
```

Criamos um novo **GtkTable** widget com 4 linhas e 4 colunas.

```
gtk_table_set_row_spacings(GTK_TABLE(table), 2);
gtk_table_set_col_spacings(GTK_TABLE(table), 2);
```

Aqui damos um pouco de espaço entre toda linha e toda coluna.

```
for( i=0; i < 4; i++) {
  for( j=0; j < 4; j++) {
    button = gtk_button_new_with_label(values[pos]);
    gtk_table_attach_defaults(GTK_TABLE(table), button, j, j+1, i, i+1 );
    pos++;
  }
}
```

Este código cria 16 botões e lugares para eles no recipiente.

Figure: GtkTable recipiente

# GtkAlinhamento

O recipiente **GtkAlignment** controla o alinhamento e o tamanho dos widget filhos.

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


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *ok;
  GtkWidget *close;

  GtkWidget *vbox;
  GtkWidget *hbox;
  GtkWidget *halign;
  GtkWidget *valign;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 350, 200);
  gtk_window_set_title(GTK_WINDOW(window), "GtkAlignment");
  gtk_container_set_border_width(GTK_CONTAINER(window), 10);

  vbox = gtk_vbox_new(FALSE, 5);

  valign = gtk_alignment_new(0, 1, 0, 0);
  gtk_container_add(GTK_CONTAINER(vbox), valign);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  hbox = gtk_hbox_new(TRUE, 3);

  ok = gtk_button_new_with_label("OK");
  gtk_widget_set_size_request(ok, 70, 30);
  gtk_container_add(GTK_CONTAINER(hbox), ok);
  close = gtk_button_new_with_label("Close");
  gtk_container_add(GTK_CONTAINER(hbox), close);

  halign = gtk_alignment_new(1, 0, 0, 0);
  gtk_container_add(GTK_CONTAINER(halign), hbox);

  gtk_box_pack_start(GTK_BOX(vbox), halign, FALSE, FALSE, 0);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Neste exemplo, colocamos dois botões no canto inferior direito da janela. Para realizar isso, usamos uma caixa horizontal e outra vertical e 2 recipientes de alinhamento.

```
valign = gtk_alignment_new(0, 1, 0, 0);
```

Isto irá colocar o widget filho no botão.

```
gtk_container_add(GTK_CONTAINER(vbox), valign);
```

Aqui colocamos o widget de alinhamento na caixa vertical.

```
hbox = gtk_hbox_new(TRUE, 3);

ok = gtk_button_new_with_label("OK");
gtk_widget_set_size_request(ok, 70, 30);
gtk_container_add(GTK_CONTAINER(hbox), ok);
close = gtk_button_new_with_label("Close");
gtk_container_add(GTK_CONTAINER(hbox), close);
```

Criamos uma caixa horizontal e colocamos 2 botoões dentro dela.

```
halign = gtk_alignment_new(1, 0, 0, 0);
gtk_container_add(GTK_CONTAINER(halign), hbox);

gtk_box_pack_start(GTK_BOX(vbox), halign, FALSE, FALSE, 0);
```

Isto cria um recipiente de alinhamento que irá colocar o widget filho à direita. Adicionamos a caixa horizontal no recipiente de alinhamento e enpacotamos o recipiente na caixa vertical. Devemos ter em mente que o reservatório de alinhamento tem apenas um Widget filho. É por isso que devemos usar caixas.

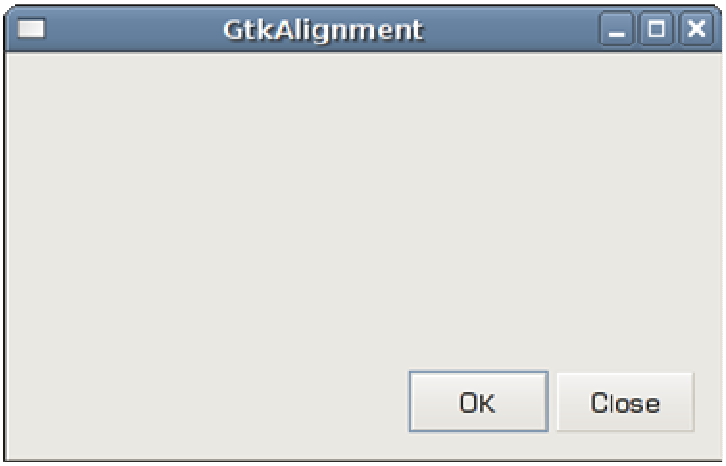


Figure: GtkAlignment container

# Janelas

Em seguida vamos criar um exemplo mais avançado. Nós mostramos uma janela, que pode ser encontrado no IDE Jdeveloper.

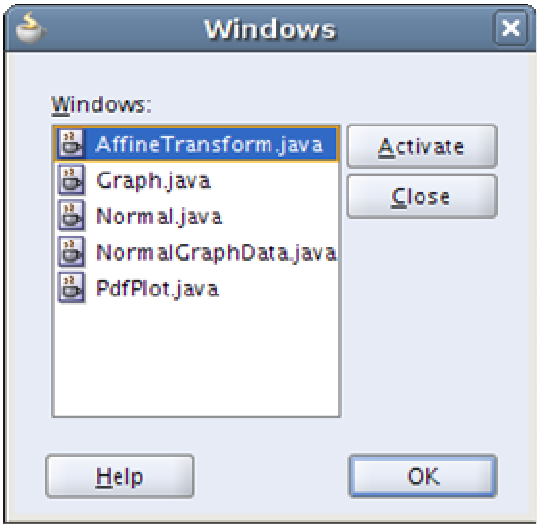


Figure: Janela de diálogo em JDeveloper

O diálogo mostra todas as janelas abertas, ou mais precisamente, guias na aplicação Jdeveloper.

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


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *table;

  GtkWidget *title;
  GtkWidget *activate;
  GtkWidget *halign;
  GtkWidget *halign2;

  GtkWidget *valign;
  GtkWidget *close;
  GtkWidget *wins;

  GtkWidget *help;
  GtkWidget *ok;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_widget_set_size_request (window, 300, 250);
  gtk_window_set_resizable(GTK_WINDOW(window), FALSE);

  gtk_window_set_title(GTK_WINDOW(window), "Windows");

  gtk_container_set_border_width(GTK_CONTAINER(window), 15);

  table = gtk_table_new(8, 4, FALSE);
  gtk_table_set_col_spacings(GTK_TABLE(table), 3);

  title = gtk_label_new("Windows");
  halign = gtk_alignment_new(0, 0, 0, 0);
  gtk_container_add(GTK_CONTAINER(halign), title);
  gtk_table_attach(GTK_TABLE(table), halign, 0, 1, 0, 1,
      GTK_FILL, GTK_FILL, 0, 0);

  wins = gtk_text_view_new();
```

```
  gtk_text_view_set_editable(GTK_TEXT_VIEW(wins), FALSE);
  gtk_text_view_set_cursor_visible(GTK_TEXT_VIEW(wins), FALSE);
  gtk_table_attach(GTK_TABLE(table), wins, 0, 2, 1, 3,
      GTK_FILL | GTK_EXPAND, GTK_FILL | GTK_EXPAND, 1, 1);

  activate = gtk_button_new_with_label("Activate");
  gtk_widget_set_size_request(activate, 50, 30);
  gtk_table_attach(GTK_TABLE(table), activate, 3, 4, 1, 2,
      GTK_FILL, GTK_SHRINK, 1, 1);

  valign = gtk_alignment_new(0, 0, 0, 0);
  close = gtk_button_new_with_label("Close");

  gtk_widget_set_size_request(close, 70, 30);
  gtk_container_add(GTK_CONTAINER(valign), close);
  gtk_table_set_row_spacing(GTK_TABLE(table), 1, 3);
  gtk_table_attach(GTK_TABLE(table), valign, 3, 4, 2, 3,
      GTK_FILL, GTK_FILL | GTK_EXPAND, 1, 1);

  halign2 = gtk_alignment_new(0, 1, 0, 0);
  help = gtk_button_new_with_label("Help");
  gtk_container_add(GTK_CONTAINER(halign2), help);
  gtk_widget_set_size_request(help, 70, 30);
  gtk_table_set_row_spacing(GTK_TABLE(table), 3, 6);
  gtk_table_attach(GTK_TABLE(table), halign2, 0, 1, 4, 5,
      GTK_FILL, GTK_FILL, 0, 0);

  ok = gtk_button_new_with_label("OK");
  gtk_widget_set_size_request(ok, 70, 30);
  gtk_table_attach(GTK_TABLE(table), ok, 3, 4, 4, 5,
      GTK_FILL, GTK_FILL, 0, 0);

  gtk_container_add(GTK_CONTAINER(window), table);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);
  gtk_main();

  return 0;
}
```

Este código mostra como que podemos criar uma janela similar em GTK+.

```
table = gtk_table_new(8, 4, FALSE);
```

Usamos um recipiente de tabela.

```
title = gtk_label_new("Windows");
halign = gtk_alignment_new(0, 0, 0, 0);
gtk_container_add(GTK_CONTAINER(halign), title);
gtk_table_attach(GTK_TABLE(table), halign, 0, 1, 0, 1,
    GTK_FILL, GTK_FILL, 0, 0);
```

Este código cria uma etiqueta que é alinhado para à esquerda. A etiqueta é colocada na primeira linha do recipiente **GtkTable.**

```
wins = gtk_text_view_new();
gtk_text_view_set_editable(GTK_TEXT_VIEW(wins), FALSE);
gtk_text_view_set_cursor_visible(GTK_TEXT_VIEW(wins), FALSE);
gtk_table_attach(GTK_TABLE(table), wins, 0, 2, 1, 3,
    GTK_FILL | GTK_EXPAND, GTK_FILL | GTK_EXPAND, 1, 1);
```

O próximo texto mostra duas linhas e duas colunas. Nós fazemos o widget não editável e escondemos o cursor.

```
valign = gtk_alignment_new(0, 0, 0, 0);
close = gtk_button_new_with_label("Close");

gtk_widget_set_size_request(close, 70, 30);
gtk_container_add(GTK_CONTAINER(valign), close);
gtk_table_set_row_spacing(GTK_TABLE(table), 1, 3);
gtk_table_attach(GTK_TABLE(table), valign, 3, 4, 2, 3,
    GTK_FILL, GTK_FILL | GTK_EXPAND, 1, 1);
```

Nós colocamos o botão de fechar próximo ao widget de texto na quarta coluna. (contamos a partir do zero) Adicionamos o botão ao widget de alinhamento, para que possamos alinhar para o topo.

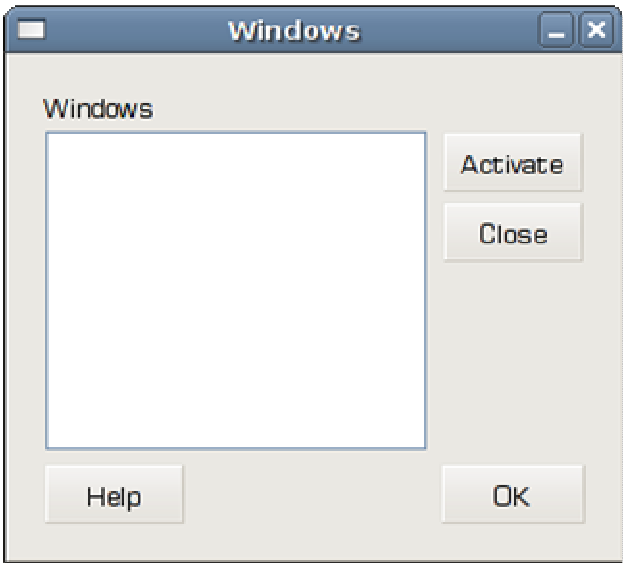


Figure: Janelas

# GTK+ eventos e sinais

Nesta parte do tutorial vamos falar sobre o sistema de eventos na biblioteca GTK+.

A biblioteca GTK+ é um sistema orientado a eventos. Todas as aplicações GUI são orientadas a eventos. As aplicações iniciam um loop principal, que verifica continuamente por eventos recém-gerados. Se não existe evento, a aplicação espera e não faz nada. Um GTK+ **evento** é uma mensagem do servidor X. Quando o evento atinge um widget, ele pode reagir a este evento através da emissão de um **sinal**. O programador GTK+ pode ligar uma **callback (chamada)** específica para um sinal. O callback é uma função de manipulador que reage a um sinal.

```
#include <gtk/gtk.h>
void button_clicked(GtkWidget *widget, gpointer data)
{
  g_print("clicked\n");
}

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

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

```
  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "GtkButton");
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);

  fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), fixed);

  button = gtk_button_new_with_label("Click");
  gtk_fixed_put(GTK_FIXED(fixed), button, 50, 50);
  gtk_widget_set_size_request(button, 80, 35);

  g_signal_connect(G_OBJECT(button), "clicked",
      G_CALLBACK(button_clicked), NULL);

  g_signal_connect(G_OBJECT(window), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();
  return 0;
}
```

Em nosso aplicação, temos 2 sinais. O sinal **clicked (clicado)** e o **destroy (destruir)**.

```
g_signal_connect(G_OBJECT(button), "clicked",
    G_CALLBACK(button_clicked), NULL);
```

Nós usamos a função **g_signal_connect()** para ligar o sinal clicado para a chamada **button_clicked()**.

```
void button_clicked(GtkWidget *widget, gpointer data)
{
  g_print("clicked\n");
}
```

A chamada irá mostrar o texto "clicked" no console. O primeiro parâmetro da função de chamada é o objeto que emitiu o sinal. Em nosso caso é o botão Click. O Segundo parâmetro é opcional. Nós podemos enviar alguns dados para a chamada. Em nosso caso, nós não enviamos nenhum dado. Enviamos um parâmetro NULL para a função g_signal_connect().

```
g_signal_connect(G_OBJECT(window), "destroy",
    G_CALLBACK(gtk_main_quit), NULL);
```

Se nós pressionarmos o botão x localizado no canto superior direito da barra de título, ou pressionarmos Atl + F4, um sinal **destroy** é emtido. Chamamos a função**gtk_main_quit()**, que irá terminar a aplicação.

# Movendo janela

O próximo exemplo mostra como nós reagimos ao mover os eventos de uma janela.

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

void frame_callback(GtkWindow *window,
      GdkEvent *event, gpointer data)
{
   int x, y;
   char buf[10];
   x = event->configure.x;
   y = event->configure.y;
   sprintf(buf, "%d, %d", x, y);
   gtk_window_set_title(window, buf);
}


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

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_title(GTK_WINDOW(window), "Simple");
  gtk_widget_add_events(GTK_WIDGET(window), GDK_CONFIGURE);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  g_signal_connect(G_OBJECT(window), "configure-event",
        G_CALLBACK(frame_callback), NULL);

  gtk_widget_show(window);
  gtk_main();

  return 0;
}
```

No exemplo, vamos mostrar a atual posição do canto superior esquerdo da nossa janela na barra de título.

```
gtk_widget_add_events(GTK_WIDGET(window), GDK_CONFIGURE);
```

A máscara de eventos do widget determina que tipo de evento vai receber um elemento específico. Alguns eventos são preconfigurados, outros eventos precisam ser adicionados à máscara de eventos. A **gtk_widget_add_events()** adiciona um tipo de evento **GDK_CONFIGURE** para a mascara. O tipo de evento **GDK_CONFIGURE** conta por todos os tamanhos, posições e ordem dos eventos empilhados.

```
g_signal_connect(G_OBJECT(window), "configure-event",
    G_CALLBACK(frame_callback), NULL);
```

A **configure-event** é emitido para tamanho, posição e pilha de ordem dos eventos.

```
void frame_callback(GtkWindow *window,
    GdkEvent *event, gpointer data)
{
  int x, y;
  char buf[10];
  x = event->configure.x;
  y = event->configure.y;
  sprintf(buf, "%d, %d", x, y);
  gtk_window_set_title(window, buf);
}
```

A função de callback tem três parâmetros. O objeto que emitiu o sinal, **GdkEvent** e os dados opcionais. Nós determinamos a x, y posições e ajustá-lo ao título.

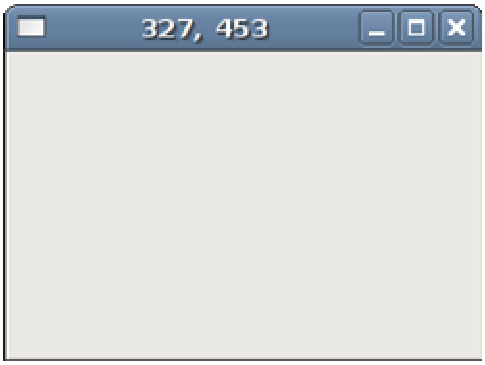



Figure: Mover evento

# A entrada de sinal

O próximo exemplo mostra como que reagimos a uma entrada de sinal **(enter)**. A entrada de sinal é emitida quando entramos na área de um widget com o ponteiro do mouse.

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


void enter_button(GtkWidget *widget, gpointer data)
{
  GdkColor color;
  color.red = 27000;
  color.green = 30325;
  color.blue = 34181;
  gtk_widget_modify_bg(widget, GTK_STATE_PRELIGHT, &color);
}


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *fixed;
  GtkWidget *button;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_title(GTK_WINDOW(window), "enter signal");

  fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), fixed);
```

```
  button = gtk_button_new_with_label("Button");
  gtk_widget_set_size_request(button, 80, 35);
  gtk_fixed_put(GTK_FIXED(fixed), button, 50, 50);

  g_signal_connect(G_OBJECT(button), "enter",
      G_CALLBACK(enter_button), NULL);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Nós vamos mudar a cor de fundo do elemento de botão, uma vez que passe um ponteiro do mouse sobre ele.

```
g_signal_connect(G_OBJECT(button), "enter",
   G_CALLBACK(enter_button), NULL);
```

Nós chamamos a função de usuário **enter_button()** quando a entrada de sinal (**enter** signal) ocorre.

```
GdkColor color;
color.red = 27000;
color.green = 30325;
color.blue = 34181;
gtk_widget_modify_bg(widget, GTK_STATE_PRELIGHT, &color);
```

Dentro da chamada, mudamos o fundo do botão, chamando a fuunção **gtk_widget_modify_bg()**.

# Desligar uma chamada

Nós podemos desligar uma chamada de um sinal. Próximo código demonstra um caso.

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


int handler_id;

void button_clicked(GtkWidget *widget, gpointer data)
{
  g_print("clicked\n");
}

void toogle_signal(GtkWidget *widget, gpointer window)
{
  if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(widget))) {
     handler_id = g_signal_connect(G_OBJECT(window), "clicked",
           G_CALLBACK(button_clicked), NULL);
  } else {
     g_signal_handler_disconnect(window, handler_id);
  }
}
```

```
int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *fixed;
  GtkWidget *button;
  GtkWidget *check;


  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 150);
  gtk_window_set_title(GTK_WINDOW(window), "Disconnect");

  fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), fixed);

  button = gtk_button_new_with_label("Click");
  gtk_widget_set_size_request(button, 80, 30);
  gtk_fixed_put(GTK_FIXED(fixed), button, 30, 50);

  check = gtk_check_button_new_with_label("Connect");
  gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
  gtk_fixed_put(GTK_FIXED(fixed), check, 130, 50);

  handler_id = g_signal_connect(G_OBJECT(button), "clicked",
        G_CALLBACK(button_clicked), NULL);

  g_signal_connect(G_OBJECT(check), "clicked",
        G_CALLBACK(toogle_signal), (gpointer) button);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

No código exemplo, temos um botão e uma caixa de seleção. A caixa de seleção liga ou desliga uma chamada (callback) de um sinal de clique de botão.

```
handler_id = g_signal_connect(G_OBJECT(button), "clicked",
    G_CALLBACK(button_clicked), NULL);
```

**g_signal_connect()** retorna um ID do manipulador que identifica a chamada.

```
if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(widget))) {
   handler_id = g_signal_connect(G_OBJECT(window), "clicked",
         G_CALLBACK(button_clicked), NULL);
} else {
   g_signal_handler_disconnect(window, handler_id);
}
```

Este código determina o estado da caixa de seleção. Ele liga a chamada se é clicada ou, caso contrário, desliga.

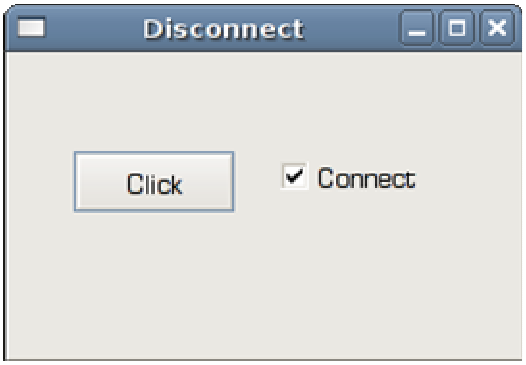


Figure: Desligar

# Exemplo de arrasta e solta

No próximo exemplo, mostramos uma interessante característica. Mostraremos uma janela sem bordas e, como podemos arrastar e mover-se como uma janela.

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

gboolean on_button_press (GtkWidget* widget,
  GdkEventButton * event, GdkWindowEdge edge)
{
  if (event->type == GDK_BUTTON_PRESS)
  {
    if (event->button == 1) {
      gtk_window_begin_move_drag(GTK_WINDOW(gtk_widget_get_toplevel(widget)),
          event->button,
          event->x_root,
          event->y_root,
          event->time);
    }
  }

  return FALSE;
}


int main( int argc, char *argv[])
{

  GtkWidget *window;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_title(GTK_WINDOW(window), "Drag & drop");
  gtk_window_set_decorated(GTK_WINDOW (window), FALSE);
  gtk_widget_add_events(window, GDK_BUTTON_PRESS_MASK);

  g_signal_connect(G_OBJECT(window), "button-press-event",
      G_CALLBACK(on_button_press), NULL);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show(window);
```

```
  gtk_main();

  return 0;
}
```

O exemplo demonstra um arrasta e solta de uma janela sem borda.

```
 gtk_window_set_decorated(GTK_WINDOW (window), FALSE);
```

Nós removemos a decoração da janela. Isto significa que a janela não terá bordas e barra de títulos.

```
  g_signal_connect(G_OBJECT(window), "button-press-event",
      G_CALLBACK(on_button_press), NULL);
```

Ligamos a janela ao sinal **button-press-event**.

```
gboolean on_button_press (GtkWidget* widget,
  GdkEventButton * event, GdkWindowEdge edge)
{
  if (event->type == GDK_BUTTON_PRESS)
  {
    if (event->button == 1) {
      gtk_window_begin_move_drag(GTK_WINDOW(gtk_widget_get_toplevel(widget)),
          event->button,
          event->x_root,
          event->y_root,
          event->time);
    }
  }

  return FALSE;
}
```

Dentro de **on_button_press()**, fazemos executar a operação de arrastar e soltar. Checamos se o botão esquerdo do mouse foi pressionado. Então chamamos a função **gtk_window_begin_move_drag().**

# Exemplo de temporizador

O próximo código demonstra um exemplo de temporizador. Eles são usados quando temos algumas tarefas de repetição. Poderia ser um relógio, uma contagem regressiva, efeitos visuais e animações.

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


static char buffer[256];


static gboolean
on_expose_event(GtkWidget *widget,
    GdkEventExpose *event,
    gpointer data)
{
  cairo_t *cr;
```

```
  cr = gdk_cairo_create(widget->window);

  cairo_move_to(cr, 30, 30);
  cairo_show_text(cr, buffer);

  cairo_destroy(cr);

  return FALSE;
}

static gboolean
time_handler(GtkWidget *widget)
{
  if (widget->window == NULL) return FALSE;

  time_t curtime;
  struct tm *loctime;

  curtime = time(NULL);
  loctime = localtime(&curtime);
  strftime(buffer, 256, "%T", loctime);

  gtk_widget_queue_draw(widget);
  return TRUE;
}

int
main (int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *darea;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

  darea = gtk_drawing_area_new();
  gtk_container_add(GTK_CONTAINER (window), darea);

  g_signal_connect(darea, "expose-event",
      G_CALLBACK(on_expose_event), NULL);
  g_signal_connect(window, "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 170, 100);

  gtk_window_set_title(GTK_WINDOW(window), "timer");
  g_timeout_add(1000, (GSourceFunc) time_handler, (gpointer) window);
  gtk_widget_show_all(window);
  time_handler(window);

  gtk_main();

  return 0;
}
```

Vamos mostrar uma hora local atual na janela. Usamos a biblioteca Cairo 2D.

```
g_signal_connect(darea, "expose-event",

     G_CALLBACK(on_expose_event), NULL);
```

Vamos desenhar o tempo dentro da chamada **on_expose_event()**. A chamada é conectada ao sinal **expose-event**. O sinal é emitido ,quando a janela vai ser redesenhada.

```
g_timeout_add(1000, (GSourceFunc) time_handler, (gpointer) window);
```

Esta função registra o tempo. A função **time_handler()** é chamada repetidamente em intervalos regulares. Em nosso caso, a cada segundo. A função de temporizador é chamada até que ela retorne FALSE.

```
time_handler(window);
```

Isto chama a função de temporizador imediatamente. Caso contrário haveria um atraso de segundo.

```
cairo_t *cr;

cr = gdk_cairo_create(widget->window);

cairo_move_to(cr, 30, 30);
cairo_show_text(cr, buffer);

cairo_destroy(cr);
```

Este código desenha o tempo atual na janela. Para mais informações sobre a biblioteca Cairo 2D, veja: ZetCode's Cairo graphics tutorial.

```
if (widget->window == NULL) return FALSE;
```

Quando a janela é destruída, pode acontecer que a função do temporizador seja chamada. Esta linha impedirá trabalho sobre widgets já destruídos.

```
time_t curtime;
struct tm *loctime;

curtime = time(NULL);
loctime = localtime(&curtime);
strftime(buffer, 256, "%T", loctime);
```

Estas linhas determinam a hora local.

```
gtk_widget_queue_draw(widget);
```

Isto invalidará a área da janela que emitirá o sinal **expose-event.**

## GTK+ diálogos

Nesta parte do tutorial de programação em GTK+, introduziremos diálogos.

Janelas de diálogo ou diálogos são uma parte indispensável da maioria dos aplicativos de interface gráfica moderna. Um diálogo é definido como uma conversa entre duas ou mais pessoas. Em uma aplicação de computador uma caixa de diálogo é uma janela que é usada para "conversar" com o aplicativo. Uma caixa de diálogo é usada para entrada de dados, alterar dados, altere as configurações do aplicativo etc. Diálogos são um importante meio de comunicação entre um usuário e um programa de computador.

## Diálogos de mensagem

Diálogos de mensagens são diálogos conveniente que oferecem mensagens para o usuário da aplicação. A mensagem é constituída por textos e dados de imagem.

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

void show_info(GtkWidget *widget, gpointer window)
{
  GtkWidget *dialog;
  dialog = gtk_message_dialog_new(window,
            GTK_DIALOG_DESTROY_WITH_PARENT,
            GTK_MESSAGE_INFO,
            GTK_BUTTONS_OK,
            "Download Completed", "title");
  gtk_window_set_title(GTK_WINDOW(dialog), "Information");
  gtk_dialog_run(GTK_DIALOG(dialog));
  gtk_widget_destroy(dialog);
}

void show_error(GtkWidget *widget, gpointer window)
{
  GtkWidget *dialog;
  dialog = gtk_message_dialog_new(window,
            GTK_DIALOG_DESTROY_WITH_PARENT,
            GTK_MESSAGE_ERROR,
            GTK_BUTTONS_OK,
            "Error loading file");
  gtk_window_set_title(GTK_WINDOW(dialog), "Error");
  gtk_dialog_run(GTK_DIALOG(dialog));
  gtk_widget_destroy(dialog);
}

void show_question(GtkWidget *widget, gpointer window)
{
  GtkWidget *dialog;
  dialog = gtk_message_dialog_new(window,
            GTK_DIALOG_DESTROY_WITH_PARENT,
            GTK_MESSAGE_QUESTION,
            GTK_BUTTONS_YES_NO,
            "Are you sure to quit?");
  gtk_window_set_title(GTK_WINDOW(dialog), "Question");
  gtk_dialog_run(GTK_DIALOG(dialog));
  gtk_widget_destroy(dialog);
}

void show_warning(GtkWidget *widget, gpointer window)
{
  GtkWidget *dialog;
  dialog = gtk_message_dialog_new(window,
            GTK_DIALOG_DESTROY_WITH_PARENT,
            GTK_MESSAGE_WARNING,
            GTK_BUTTONS_OK,
            "Unallowed operation");
  gtk_window_set_title(GTK_WINDOW(dialog), "Warning");
  gtk_dialog_run(GTK_DIALOG(dialog));
  gtk_widget_destroy(dialog);
}

int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *table;

  GtkWidget *info;
  GtkWidget *warn;
  GtkWidget *que;
  GtkWidget *err;
```

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

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 220, 150);
  gtk_window_set_title(GTK_WINDOW(window), "Message dialogs");

  table = gtk_table_new(2, 2, TRUE);
  gtk_table_set_row_spacings(GTK_TABLE(table), 2);
  gtk_table_set_col_spacings(GTK_TABLE(table), 2);

  info = gtk_button_new_with_label("Info");
  warn = gtk_button_new_with_label("Warning");
  que = gtk_button_new_with_label("Question");
  err = gtk_button_new_with_label("Error");

  gtk_table_attach(GTK_TABLE(table), info, 0, 1, 0, 1,
      GTK_FILL, GTK_FILL, 3, 3);
  gtk_table_attach(GTK_TABLE(table), warn, 1, 2, 0, 1,
      GTK_FILL, GTK_FILL, 3, 3);
  gtk_table_attach(GTK_TABLE(table), que, 0, 1, 1, 2,
      GTK_FILL, GTK_FILL, 3, 3);
  gtk_table_attach(GTK_TABLE(table), err, 1, 2, 1, 2,
      GTK_FILL, GTK_FILL, 3, 3);

  gtk_container_add(GTK_CONTAINER(window), table);
  gtk_container_set_border_width(GTK_CONTAINER(window), 15);

  g_signal_connect(G_OBJECT(info), "clicked",
        G_CALLBACK(show_info), (gpointer) window);

  g_signal_connect(G_OBJECT(warn), "clicked",
        G_CALLBACK(show_warning), (gpointer) window);

  g_signal_connect(G_OBJECT(que), "clicked",
        G_CALLBACK(show_question), (gpointer) window);

  g_signal_connect(G_OBJECT(err), "clicked",
        G_CALLBACK(show_error), (gpointer) window);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

No nosso exemplo, vamos mostrar quatro tipos de janelas de mensagem. Informações, Aviso, perguntas e diálogos de mensagem de erro.

```
GtkWidget *dialog;
dialog = gtk_message_dialog_new(window,
          GTK_DIALOG_DESTROY_WITH_PARENT,
          GTK_MESSAGE_QUESTION,
          GTK_BUTTONS_YES_NO,
          "Are you sure to quit?");
```

Na função show_question(), mostramos uma janela pop-up de mensagem de diálogo. A mensagem de diálogo é mostrada usando a chamada **gtk_message_dialog_new()**. Os parâmetros da função específica que tipo de mensagem de diálogo criamos. A constante **GTK_MESSAGE_QUESTION** cria tipo de diálogo de pergunta. A constante

**GTK_BUTTONS_YES_NO** vai botões 'No' e 'Yes' no diálogo. O último parâmetro é o texto que apresentamos no diálogo.

```
gtk_window_set_title(GTK_WINDOW(dialog), "Warning");
gtk_dialog_run(GTK_DIALOG(dialog));
gtk_widget_destroy(dialog);
```

Aqui vamos definir um título para a mensagem de diálogo. E, finalmente, executá-los. Eles devem ser destruídos manualmente.


## GtkAboutDialog

O GtkAboutDialog mostra informação sobre a aplicação. GtkAboutDialog pode mostrar uma logo, o nome da aplicação, versão, copyright, website ou informação de licença. É também possível dar os créditos dos autores, documentação, tradutores e artistas.

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


void show_about(GtkWidget *widget, gpointer data)
{

  GdkPixbuf *pixbuf = gdk_pixbuf_new_from_file("battery.png", NULL);

  GtkWidget *dialog = gtk_about_dialog_new();
  gtk_about_dialog_set_name(GTK_ABOUT_DIALOG(dialog), "Battery");
  gtk_about_dialog_set_version(GTK_ABOUT_DIALOG(dialog), "0.9");
  gtk_about_dialog_set_copyright(GTK_ABOUT_DIALOG(dialog),
      "(c) Jan Bodnar");
  gtk_about_dialog_set_comments(GTK_ABOUT_DIALOG(dialog),
     "Battery is a simple tool for battery checking.");
  gtk_about_dialog_set_website(GTK_ABOUT_DIALOG(dialog),
      "http://www.batteryhq.net");
  gtk_about_dialog_set_logo(GTK_ABOUT_DIALOG(dialog), pixbuf);
  g_object_unref(pixbuf), pixbuf = NULL;
  gtk_dialog_run(GTK_DIALOG (dialog));
  gtk_widget_destroy(dialog);
```

```
}


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *about;
  GdkPixbuf *battery;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 220, 150);
  gtk_window_set_title(GTK_WINDOW(window), "Battery");

  gtk_container_set_border_width(GTK_CONTAINER(window), 15);
  gtk_widget_add_events(window, GDK_BUTTON_PRESS_MASK);

  battery = gtk_image_get_pixbuf(GTK_IMAGE(
      gtk_image_new_from_file("battery.png")));

  g_signal_connect(G_OBJECT(window), "button-press-event",
        G_CALLBACK(show_about), (gpointer) window);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

O exemplo de código usa um GtkAboutDialog com algumas das suas características. Para mostrar o diálogo, clicamos na área cliente da janela.

```
GtkWidget *dialog = gtk_about_dialog_new();
```

Criamos um novo GtkAboutDialog.

```
gtk_about_dialog_set_name(GTK_ABOUT_DIALOG(dialog), "Battery");
gtk_about_dialog_set_version(GTK_ABOUT_DIALOG(dialog), "0.9");
gtk_about_dialog_set_copyright(GTK_ABOUT_DIALOG(dialog),
     "(c) Jan Bodnar");
```

Estas funções de chamada definem um nome, versão e copyright.

```
GdkPixbuf *pixbuf = gdk_pixbuf_new_from_file("battery.png", NULL);
...
gtk_about_dialog_set_logo(GTK_ABOUT_DIALOG(dialog), pixbuf);
g_object_unref(pixbuf), pixbuf = NULL;
```

Este código cria uma logo.

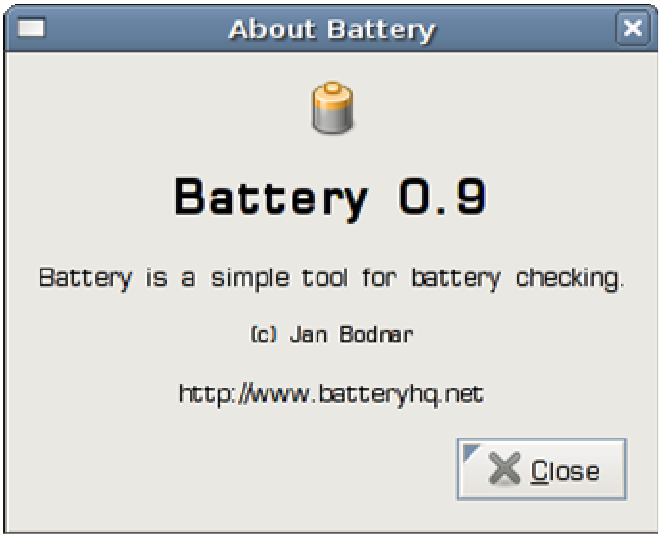


Figure: GtkAboutDialog

## GtkFontSelectionDialog

O GtkFontSelectionDialog é um diálogo para selecionar fontes. Tipicamente usado em aplicativos que fazem edição de texto ou formatação.

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


void select_font(GtkWidget *widget, gpointer label)
{

  GtkResponseType result;

  GtkWidget *dialog = gtk_font_selection_dialog_new("Select Font");
  result = gtk_dialog_run(GTK_DIALOG(dialog));

  if (result == GTK_RESPONSE_OK || result == GTK_RESPONSE_APPLY)
  {

    PangoFontDescription *font_desc;
    gchar *fontname = gtk_font_selection_dialog_get_font_name(
                            GTK_FONT_SELECTION_DIALOG(dialog));

    font_desc = pango_font_description_from_string(fontname);

    gtk_widget_modify_font(GTK_WIDGET(label), font_desc);

    g_free(fontname);
   }


  gtk_widget_destroy(dialog);
}


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *label;
  GtkWidget *vbox;
```

```
  GtkWidget *toolbar;
  GtkToolItem *font;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 280, 200);
  gtk_window_set_title(GTK_WINDOW(window), "Font Selection Dialog");

  vbox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(window), vbox);


  toolbar = gtk_toolbar_new();
  gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS);

  gtk_container_set_border_width(GTK_CONTAINER(toolbar), 2);

  font = gtk_tool_button_new_from_stock(GTK_STOCK_SELECT_FONT);
  gtk_toolbar_insert(GTK_TOOLBAR(toolbar), font, -1);

  gtk_box_pack_start(GTK_BOX(vbox), toolbar, FALSE, FALSE, 5);

  label = gtk_label_new("ZetCode");
  gtk_label_set_justify(GTK_LABEL(label), GTK_JUSTIFY_CENTER);
  gtk_box_pack_start(GTK_BOX(vbox), label, TRUE, FALSE, 5);


  g_signal_connect(G_OBJECT(font), "clicked",
        G_CALLBACK(select_font), label);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Neste exemplo nós definimos uma simples etiqueta no centro da janela. Mostramos um diálogo de seleção de fonte clicando no botão da barra de ferramentas.

```
  GtkWidget *dialog = gtk_font_selection_dialog_new("Select Font");
  result = gtk_dialog_run(GTK_DIALOG(dialog));
```

We create and show the **GtkFontSelectionDialog.**

```
 if (result == GTK_RESPONSE_OK || result == GTK_RESPONSE_APPLY)
 {
   PangoFontDescription *font_desc;
   gchar *fontname = gtk_font_selection_dialog_get_font_name(
                            GTK_FONT_SELECTION_DIALOG(dialog));

   font_desc = pango_font_description_from_string(fontname);

   gtk_widget_modify_font(GTK_WIDGET(label), font_desc);
   g_free(fontname);
  }
```

Se o usuário clica no botão OK ou APPLY, prosseguimos. Ficamos com o nome da fonte selecionado. Então mudamos a fonte da etiqueta pela fonte selecionada.

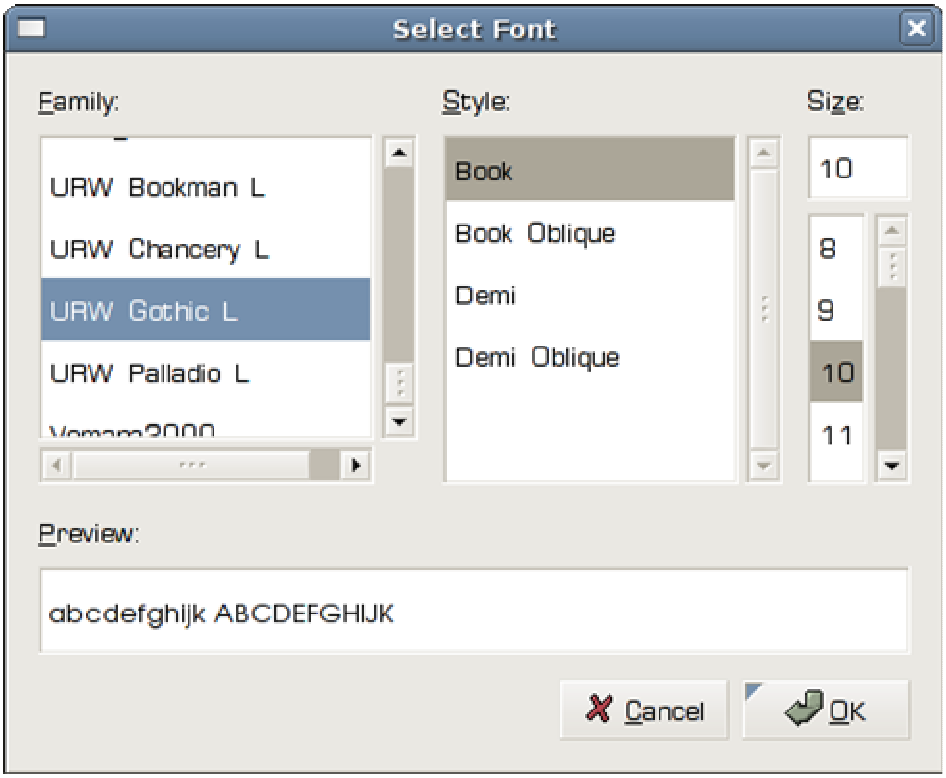


Figura: GtkFontSelectionDialog

## GtkColorSelectionDialog

GtkColorSelectionDialog é um diálogo para selecionar uma cor.

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


void select_font(GtkWidget *widget, gpointer label)
{

  GtkResponseType result;
  GtkColorSelection *colorsel;

  GtkWidget *dialog = gtk_color_selection_dialog_new("Font Color");
  result = gtk_dialog_run(GTK_DIALOG(dialog));

  if (result == GTK_RESPONSE_OK)
  {

    GdkColor color;

    colorsel = GTK_COLOR_SELECTION(
                  GTK_COLOR_SELECTION_DIALOG(dialog)->colorsel);
    gtk_color_selection_get_current_color(colorsel,
                  &color);

    gtk_widget_modify_fg(GTK_WIDGET(label),
                  GTK_STATE_NORMAL,
```

```
                    &color);
  }

  gtk_widget_destroy(dialog);
}


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *widget;
  GtkWidget *label;
  GtkWidget *vbox;

  GtkWidget *toolbar;
  GtkToolItem *font;


  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 280, 200);
  gtk_window_set_title(GTK_WINDOW(window), "Color Selection Dialog");

  vbox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(window), vbox);


  toolbar = gtk_toolbar_new();
  gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS);

  gtk_container_set_border_width(GTK_CONTAINER(toolbar), 2);

  font = gtk_tool_button_new_from_stock(GTK_STOCK_SELECT_COLOR);
  gtk_toolbar_insert(GTK_TOOLBAR(toolbar), font, -1);

  gtk_box_pack_start(GTK_BOX(vbox), toolbar, FALSE, FALSE, 5);

  label = gtk_label_new("ZetCode");
  gtk_label_set_justify(GTK_LABEL(label), GTK_JUSTIFY_CENTER);
  gtk_box_pack_start(GTK_BOX(vbox), label, TRUE, FALSE, 5);


  g_signal_connect(G_OBJECT(font), "clicked",
        G_CALLBACK(select_font), label);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

O exemplo é muito parecido com o anterior. Neste momento nós mudamos a cor da etiqueta.

```
GtkWidget *dialog = gtk_color_selection_dialog_new("Font Color");


result = gtk_dialog_run(GTK_DIALOG(dialog));
```

Nós criamos e mostramos o **GtkColorSelectionDialog**.

```
if (result == GTK_RESPONSE_OK)
{
  GdkColor color;

  colorsel = GTK_COLOR_SELECTION(
                GTK_COLOR_SELECTION_DIALOG(dialog)->colorsel);
  gtk_color_selection_get_current_color(colorsel,
                &color);

  gtk_widget_modify_fg(GTK_WIDGET(label),
                GTK_STATE_NORMAL,
                &color);
}
```

Se o usuário pressionar OK, nós definimos e modificamos a cor da etiqueta.

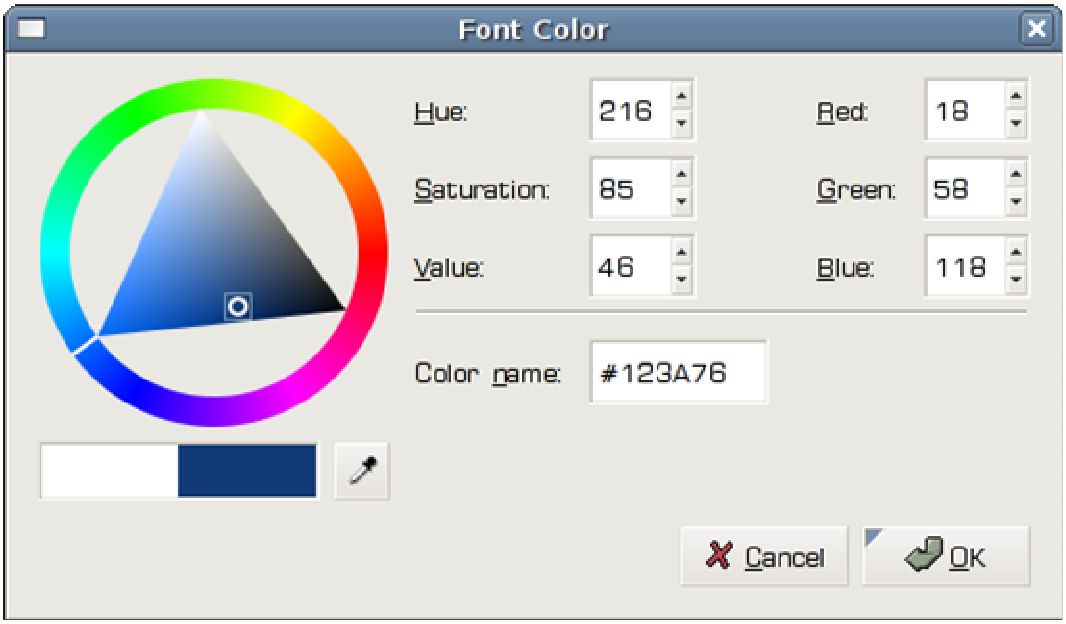


Figura: GtkColorSelectionDialog

# GTK+ Widgets

Nesta parte do tutorial de programação GTK+ nós introduziremos alguns GTK+ widgets.

Widgets são basicamente blocos básicos de construção de uma aplicação gráfica. Ao longo dos anos, vários widgets tornaram-se padrão em todos os kits de ferramentas em todas as plataformas de Sistemas Operacionais (SO). Por exemplo, um botão, uma caixa de verificação ou barra de rolagem. A filosofia dos kits de ferramentas do GTK+ é manter um número mínimo de elementos. Mais widgets especializados são criados como GTK+ widgets personalizados.

## GtkButton

GtkButton é um simples widget que é usado para desencadear uma ação.

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


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

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "GtkButton");
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);

  fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), fixed);

  button = gtk_button_new_with_label("Quit");

  gtk_fixed_put(GTK_FIXED(fixed), button, 50, 50);
  gtk_widget_set_size_request(button, 80, 35);

  g_signal_connect(G_OBJECT(button), "clicked",
      G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

O exemplo mostra um botão que é posicionado é um recipiente fixo. A aplicação fecha quando clicamos no botão.

```
 button = gtk_button_new_with_label("Quit");
```

Esta linha de código cria um **GtkButton** com um etiqueta.

```
  g_signal_connect(G_OBJECT(button), "clicked",
      G_CALLBACK(gtk_main_quit), G_OBJECT(window));
```

Aqui conectamos um sinal ao clicar (**clicked**) no botão. O sinal irá disparar a função **gtk_main_quit()**, que finaliza a aplicação.

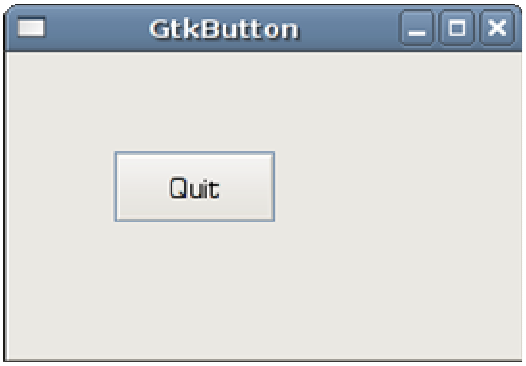


Figura: GtkButton

## GtkCheckButton

GtkCheckButton é um widget que possui 2 estados: ligado e desligado (ON e OFF). O estado ligado é visualizado por uma marca de seleção.

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


void toggle_title(GtkWidget *widget, gpointer window)
{
  if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(widget))) {
      gtk_window_set_title(window, "GtkCheckButton");
  } else {
      gtk_window_set_title(window, "");
  }
}

int main(int argc, char** argv) {

  GtkWidget *window;
  GtkWidget *frame;
  GtkWidget *check;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_title(GTK_WINDOW(window), "GtkCheckButton");


  frame = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), frame);


  check = gtk_check_button_new_with_label("Show title");
  gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
  GTK_WIDGET_UNSET_FLAGS(check, GTK_CAN_FOCUS);
  gtk_fixed_put(GTK_FIXED(frame), check, 50, 50);

  g_signal_connect_swapped(window, "destroy",
          G_CALLBACK(gtk_main_quit), NULL);

  g_signal_connect(check, "clicked",
          G_CALLBACK(toggle_title), (gpointer) window);

  gtk_widget_show_all(window);
```

```
  gtk_main();

  return 0;
}
```

Nós vamos mostrar um título dependendo do estado de **GtkCheckButton**.

```
check = gtk_check_button_new_with_label("Show title");
gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(check), TRUE);
```

O **GtkCheckButton** é criado e é marcado por padrão. Porque mostramos um título por padrão.

```
GTK_WIDGET_UNSET_FLAGS(check, GTK_CAN_FOCUS);
```

Esta linha de código desativa o foco. Eu simplesmente não gostava do retângulo sobre o botão de seleção. Não parece legal.

```
if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(widget))) {
   gtk_window_set_title(window, "GtkCheckButton");
} else {
   gtk_window_set_title(window, "");
}
```

Mostramos o título da janela dependendo do estado de **GtkCheckButton**.


Figura: GtkCheckButton

# GtkFrame

GtkFrame é uma caixa com moldura decorativa e etiqueta opcional.

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


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *table;

  GtkWidget *frame1;
  GtkWidget *frame2;
  GtkWidget *frame3;
  GtkWidget *frame4;
```

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

    window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
    gtk_window_set_default_size(GTK_WINDOW(window), 250, 250);
    gtk_window_set_title(GTK_WINDOW(window), "GtkFrame");

    gtk_container_set_border_width(GTK_CONTAINER(window), 10);

    table = gtk_table_new(2, 2, TRUE);
    gtk_table_set_row_spacings(GTK_TABLE(table), 10);
    gtk_table_set_col_spacings(GTK_TABLE(table), 10);
    gtk_container_add(GTK_CONTAINER(window), table);


    frame1 = gtk_frame_new("Shadow In");
    gtk_frame_set_shadow_type(GTK_FRAME(frame1), GTK_SHADOW_IN);
    frame2 = gtk_frame_new("Shadow Out");
    gtk_frame_set_shadow_type(GTK_FRAME(frame2), GTK_SHADOW_OUT);
    frame3 = gtk_frame_new("Shadow Etched In");
    gtk_frame_set_shadow_type(GTK_FRAME(frame3), GTK_SHADOW_ETCHED_IN);
    frame4 = gtk_frame_new("Shadow Etched Out");
    gtk_frame_set_shadow_type(GTK_FRAME(frame4), GTK_SHADOW_ETCHED_OUT);


    gtk_table_attach_defaults(GTK_TABLE(table), frame1, 0, 1, 0, 1);
    gtk_table_attach_defaults(GTK_TABLE(table), frame2, 0, 1, 1, 2);
    gtk_table_attach_defaults(GTK_TABLE(table), frame3, 1, 2, 0, 1);
    gtk_table_attach_defaults(GTK_TABLE(table), frame4, 1, 2, 1, 2);

    g_signal_connect_swapped(G_OBJECT(window), "destroy",
          G_CALLBACK(gtk_main_quit), G_OBJECT(window));

    gtk_widget_show_all(window);

    gtk_main();

    return 0;
}
```

Este exemplo mostra quarto tipos diferentes de molduras. As molduras são unidas em um recipiente de tabela.

```
 frame1 = gtk_frame_new("Shadow In");
 gtk_frame_set_shadow_type(GTK_FRAME(frame1), GTK_SHADOW_IN);
```

Criamos um **GtkFrame** e definimos o seu tipo de sombra.

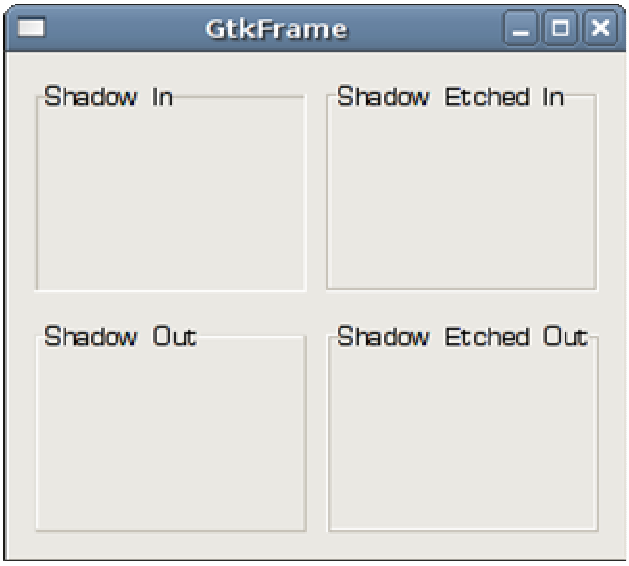


Figura: GtkFrame

## GtkLabel

O widget GtkLabel exibe texto.

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


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

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_title(GTK_WINDOW(window), "Nymphetamine");
  gtk_window_set_default_size(GTK_WINDOW(window), 350, 400);

  label = gtk_label_new("Cold was my soul\n\
Untold was the pain\n\
I faced when you left me\n\
A rose in the rain....\n\
So I swore to the razor\n\
That never, enchained\n\
Would your dark nails of faith\n\
Be pushed through my veins again\n\n\
\n\
Bared on your tomb\n\
I'm a prayer for your loneliness\n\
And would you ever soon\n\
Come above onto me?\n\
For once upon a time\n\n\
On the binds of your lowliness\n\
I could always find the slot for your sacred key ");

  gtk_label_set_justify(GTK_LABEL(label), GTK_JUSTIFY_CENTER);
  gtk_container_add(GTK_CONTAINER(window), label);

  g_signal_connect_swapped(window, "destroy",
      G_CALLBACK (gtk_main_quit), NULL);
```

```
  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

O exemplo mostra a letra de uma música.

```
  label = gtk_label_new("Cold was my soul\n\
Untold was the pain\n\
...
```

Criamos um widget **GtkLabel** widget. Nós podemos criar etiquetas de texto com várias linhas usando um caractere de nova linha. Note o character de escape. Nós usamos uma sequencia bastante longa e não queremos colocar todo o texto é uma linha. Em tais casos, podemos usar um caractere de escape.

```
gtk_label_set_justify(GTK_LABEL(label), GTK_JUSTIFY_CENTER);
```

Nós centralizamos nossa etiqueta.

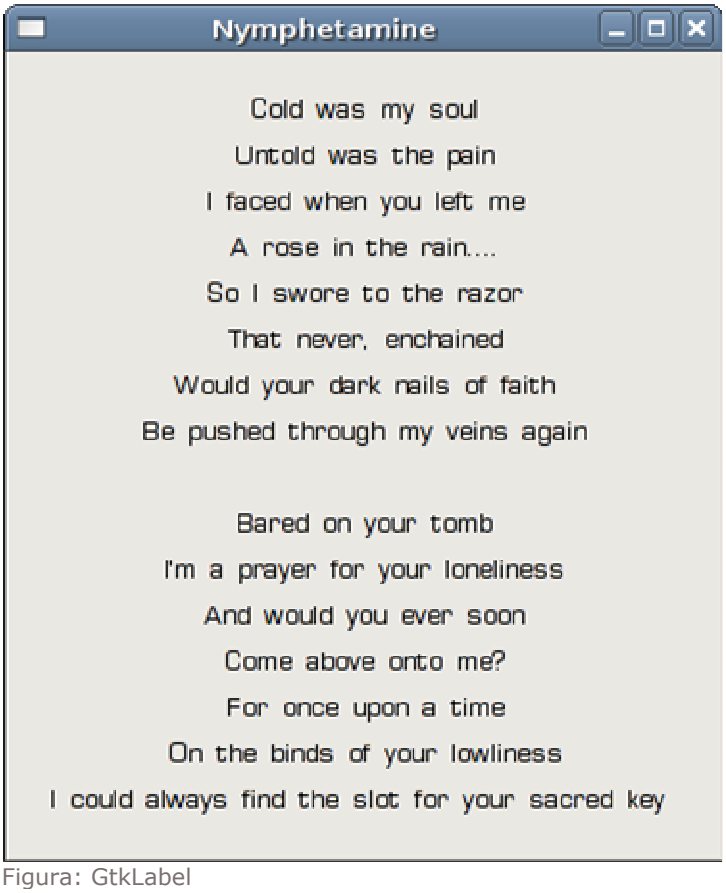


Figura: GtkLabel

Em GtkLabel nós podemos também usar linguagem de marcação. O próximo exemplo mostra como que podemos fazer isso.

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

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

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_title(GTK_WINDOW(window), "markup label");

  char *str = "<b>ZetCode</b>, Knowledge only matters";

  label = gtk_label_new(NULL);
  gtk_label_set_markup(GTK_LABEL(label), str);

  gtk_label_set_justify(GTK_LABEL(label), GTK_JUSTIFY_CENTER);
  gtk_container_add(GTK_CONTAINER(window), label);
  gtk_widget_show(label);

  gtk_window_set_default_size(GTK_WINDOW(window), 300, 100);

  g_signal_connect(window, "destroy",
      G_CALLBACK (gtk_main_quit), NULL);

  gtk_widget_show(window);

  gtk_main();

  return 0;
}
```

O exemplo mostra uma parte do texto em negrito.

```
 char *str = "<b>ZetCode</b>, Knowledge only matters";
```

Isto é a string (cadeia de caracteres) que vamos exibir.

```
  label = gtk_label_new(NULL);
  gtk_label_set_markup(GTK_LABEL(label), str);
```

Nós criamos uma etiqueta vazia e definimos um texto de marcação para ela.

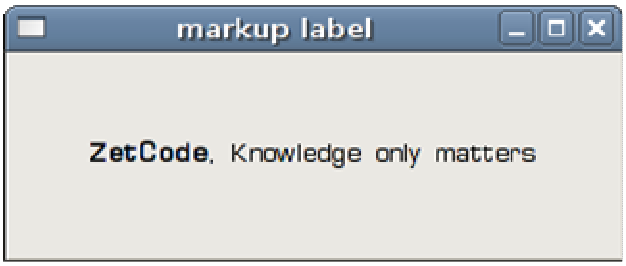



Figura: markup label

# GTK+ Widgets II

Nesta parte do tutorial de programação GTK+, nós continuaremos introduzindo diversos GTK+ widgets.

## GtkComboBox

**GtkComboBox** é um widget que permite ao usuário escolher entre uma lista de opções.

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

void combo_selected(GtkWidget *widget, gpointer window)
{
  gchar *text =  gtk_combo_box_get_active_text(GTK_COMBO_BOX(widget));
  gtk_label_set_text(GTK_LABEL(window), text);
  g_free(text);
}


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *fixed;
  GtkWidget *combo;
  GtkWidget *label;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "GtkCombo");
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);

  fixed = gtk_fixed_new();

  combo = gtk_combo_box_new_text();
  gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Ubuntu");
  gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Mandriva");
  gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Fedora");
  gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Mint");
  gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Gentoo");
  gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Debian");

  gtk_fixed_put(GTK_FIXED(fixed), combo, 50, 50);
  gtk_container_add(GTK_CONTAINER(window), fixed);

  label = gtk_label_new("-");
  gtk_fixed_put(GTK_FIXED(fixed), label, 50, 110);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  g_signal_connect(G_OBJECT(combo), "changed",
        G_CALLBACK(combo_selected), (gpointer) label);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

O exemplo mostra uma caixa de combinação e uma etiqueta. A caixa de combinação tem uma lista de seis opções. Estes são os nomes de Distribuições Linux. O Widget de etiqueta mostra a opção selecionada na caixa de combinação.

```
combo = gtk_combo_box_new_text();
gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Ubuntu");
gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Mandriva");
gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Fedora");
gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Mint");
gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Gentoo");
gtk_combo_box_append_text(GTK_COMBO_BOX(combo), "Debian");
```

Criamos um **GtkComboBox** e enchemos com uma lista de Distribuições Linux.

```
label = gtk_label_new("-");
```

Também criamos um widget etiqueta.

```
gchar *text =  gtk_combo_box_get_active_text(GTK_COMBO_BOX(widget));
gtk_label_set_text(GTK_LABEL(window), text);
g_free(text);
```

Nós definimos o texto selecionado e setamos a etiqueta de texto para ele.  A documentação para a caixa de combinação (combo box) diz que **gtk_combo_box_get_active_text()** retorna uma string recém-alocada que contém o texto ativo atualmente. Isto significa que temos a memória livre.

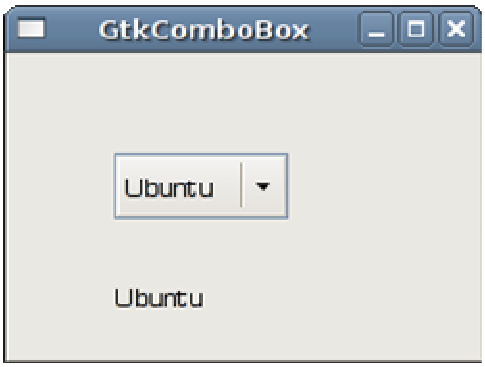


Figura: GtkComboBox

# GtkHSeparator

O **GtkHSeparator** é um separador horizontal. É uma espécie de widget enfeitado. Estes widgets servem para algum propósito do projeto. Há também uma widget irmã **GtkVSeparator**.

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


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *label1;
  GtkWidget *label2;
  GtkWidget *hseparator;
```

```
  GtkWidget *vbox;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_title(GTK_WINDOW(window), "GtkHSeparator");
  gtk_window_set_resizable(GTK_WINDOW(window), FALSE);

  gtk_container_set_border_width(GTK_CONTAINER(window), 20);

  label1 = gtk_label_new("Zinc is a moderately reactive, blue gray metal \
that tarnishes in moist air and burns in air with a bright bluish-green
flame,\
giving off fumes of zinc oxide. It reacts with acids, alkalis and other non-
metals.\
If not completely pure, zinc reacts with dilute acids to release hydrogen.");

  gtk_label_set_line_wrap(GTK_LABEL(label1), TRUE);

  label2 = gtk_label_new("Copper is an essential trace nutrient to all high \
plants and animals. In animals, including humans, it is found primarily in \
the bloodstream, as a co-factor in various enzymes, and in copper-based
pigments. \
However, in sufficient amounts, copper can be poisonous and even fatal to
organisms.");

  gtk_label_set_line_wrap(GTK_LABEL(label2), TRUE);

  vbox = gtk_vbox_new(FALSE, 10);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  hseparator = gtk_hseparator_new();

  gtk_box_pack_start(GTK_BOX(vbox), label1, FALSE, TRUE, 0);
  gtk_box_pack_start(GTK_BOX(vbox), hseparator, FALSE, TRUE, 10);
  gtk_box_pack_start(GTK_BOX(vbox), label2, FALSE, TRUE, 0);


  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

O código-exemplo mostra definições dois elemento químicos. Eles são separados por separadores horizontais. Isto faz com que o exemplo seja mais atraente visualmente.

```
   label1 = gtk_label_new("Zinc is a moderately reactive, blue gray metal \
that tarnishes in moist air and burns in air with a bright bluish-green
flame,\
giving off fumes of zinc oxide. It reacts with acids, alkalis and other non-
metals.\
If not completely pure, zinc reacts with dilute acids to release hydrogen.");
```

Criamos a primeira etiqueta com para a definição do elemento Zinco.

```
 gtk_label_set_line_wrap(GTK_LABEL(label2), TRUE);
```

Envolvemos o texto.

```
hseparator = gtk_hseparator_new();
```

Criamos um separados horizontal.

```
 gtk_box_pack_start(GTK_BOX(vbox), label1, FALSE, TRUE, 0);
 gtk_box_pack_start(GTK_BOX(vbox), hseparator, FALSE, TRUE, 10);
 gtk_box_pack_start(GTK_BOX(vbox), label2, FALSE, TRUE, 0);
```

Aqui colocamos o separador entre duas etiquetas..

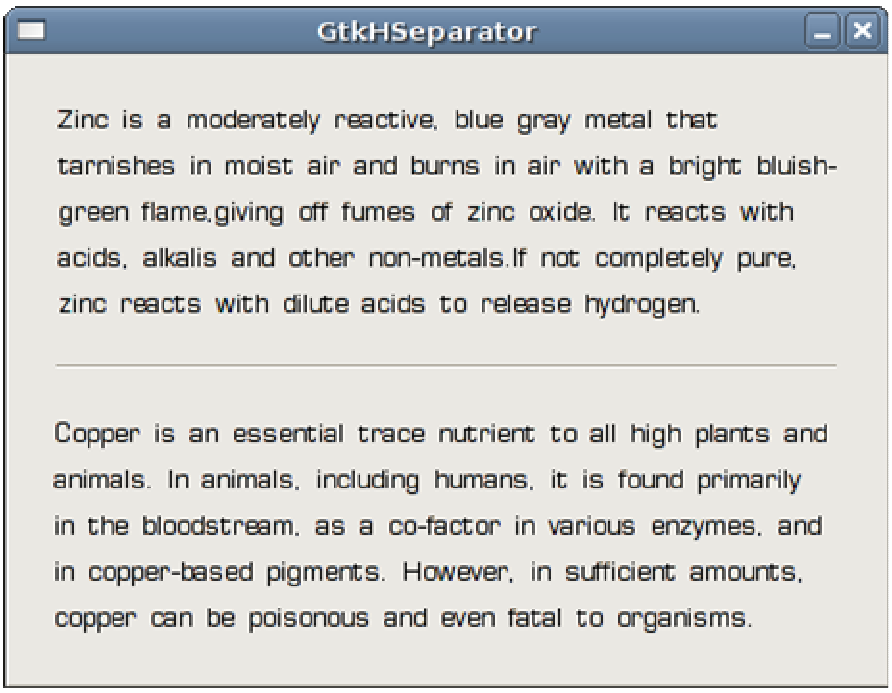


Figura: GtkHSeparator

# GtkEntry

**GtkEntry** é uma única linha de campo de texto. Este elemento é usado para inserir dados textuais.

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


int main(int argc, char *argv[]) {

  GtkWidget *window;
  GtkWidget *table;

  GtkWidget *label1;
  GtkWidget *label2;
  GtkWidget *label3;

  GtkWidget *entry1;
  GtkWidget *entry2;
  GtkWidget *entry3;

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

```
  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_title(GTK_WINDOW(window), "GtkEntry");
  gtk_container_set_border_width(GTK_CONTAINER(window), 10);

  table = gtk_table_new(3, 2, FALSE);
  gtk_container_add(GTK_CONTAINER(window), table);

  label1 = gtk_label_new("Name");
  label2 = gtk_label_new("Age");
  label3 = gtk_label_new("Occupation");


  gtk_table_attach(GTK_TABLE(table), label1, 0, 1, 0, 1,
      GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
  gtk_table_attach(GTK_TABLE(table), label2, 0, 1, 1, 2,
      GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
  gtk_table_attach(GTK_TABLE(table), label3, 0, 1, 2, 3,
      GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);

  entry1 = gtk_entry_new();
  entry2 = gtk_entry_new();
  entry3 = gtk_entry_new();

  gtk_table_attach(GTK_TABLE(table), entry1, 1, 2, 0, 1,
      GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
  gtk_table_attach(GTK_TABLE(table), entry2, 1, 2, 1, 2,
      GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
  gtk_table_attach(GTK_TABLE(table), entry3, 1, 2, 2, 3,
      GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);

  gtk_widget_show(table);

  gtk_widget_show(label1);
  gtk_widget_show(label2);
  gtk_widget_show(label3);

  gtk_widget_show(entry1);
  gtk_widget_show(entry2);
  gtk_widget_show(entry3);

  gtk_widget_show(window);

  g_signal_connect(window, "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_main();

  return 0;
}
```

No nosso exemplo vamos mostrar três entradas de texto e três etiquetas.

```
 table = gtk_table_new(3, 2, FALSE);
 gtk_container_add(GTK_CONTAINER(window), table);
```

Para organizar os nossos widgets, usamos um widget de tabela de recipiente.

```
 entry1 = gtk_entry_new();
 entry2 = gtk_entry_new();
 entry3 = gtk_entry_new();
```

Aqui criamos três entradas de texto.

```
 gtk_table_attach(GTK_TABLE(table), entry1, 1, 2, 0, 1,
```

```
    GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
gtk_table_attach(GTK_TABLE(table), entry2, 1, 2, 1, 2,
    GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
gtk_table_attach(GTK_TABLE(table), entry3, 1, 2, 2, 3,
    GTK_FILL | GTK_SHRINK, GTK_FILL | GTK_SHRINK, 5, 5);
```

Nós conferimos os elementos para o widget de tabela.

Figura: GtkEntry

# GtkImage

**GtkImage** é um widget usado para exibir uma imagem.

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


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *image;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 230, 150);
  gtk_window_set_title(GTK_WINDOW(window), "Red Rock");
  gtk_window_set_resizable(GTK_WINDOW(window), FALSE);

  gtk_container_set_border_width(GTK_CONTAINER(window), 2);

  image = gtk_image_new_from_file("redrock.png");
  gtk_container_add(GTK_CONTAINER(window), image);


  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Em nosso exemplo mostramos a imagem de um castelo. O castelo é chamado de Red Rock (Rocha Vermelha) é está situado na parte ocidental da Eslováquia. Você pode baixar a imagem aqui.

```
gtk_container_set_border_width(GTK_CONTAINER(window), 2);
```

Nós colocamos uma pequena borda de 2px ao redor da imagem.

```
image = gtk_image_new_from_file("redrock.png");
gtk_container_add(GTK_CONTAINER(window), image);
```

Carregamos a imagem de um arquivo e a adicionamos ao recipiente.

Figura: GtkImage

# GtkStatusbar

**GtkStatusbar** exibe a informação de status. É colocado na parte inferior da janela do aplicativo.

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


void button_pressed(GtkWidget *widget, gpointer window)
{
   gchar *str;
   str = g_strdup_printf("Button %s clicked",
         gtk_button_get_label(GTK_BUTTON(widget)));

   gtk_statusbar_push(GTK_STATUSBAR(window),
         gtk_statusbar_get_context_id(GTK_STATUSBAR(window), str), str);
   g_free(str);
}

int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *fixed;
  GtkWidget *button1;
```

```
  GtkWidget *button2;
  GtkWidget *statusbar;
  GtkWidget *vbox;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 280, 150);
  gtk_window_set_title(GTK_WINDOW(window), "GtkStatusbar");

  vbox = gtk_vbox_new(FALSE, 2);

  fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), vbox);

  gtk_box_pack_start(GTK_BOX(vbox), fixed, TRUE, TRUE, 1);

  button1 = gtk_button_new_with_label("OK");
  gtk_widget_set_size_request(button1, 80, 30 );
  button2 = gtk_button_new_with_label("Apply");
  gtk_widget_set_size_request(button2, 80, 30 );

  gtk_fixed_put(GTK_FIXED(fixed), button1, 50, 50);
  gtk_fixed_put(GTK_FIXED(fixed), button2, 150, 50);


  statusbar = gtk_statusbar_new();
  gtk_box_pack_start(GTK_BOX(vbox), statusbar, FALSE, TRUE, 1);

  g_signal_connect(G_OBJECT(button1), "clicked",
           G_CALLBACK(button_pressed), G_OBJECT(statusbar));

  g_signal_connect(G_OBJECT(button2), "clicked",
           G_CALLBACK(button_pressed), G_OBJECT(statusbar));

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

Em nosso código-exemplo, mostramos 2 botões e uma barra de status. Se clicarmos no botão, uma mensagem é exibida na barra de status. Isto significa que o botão foi pressionado.

```
 gchar *str;
 str = g_strdup_printf("Button %s clicked",
       gtk_button_get_label(GTK_BUTTON(widget)));
```

Criamos uma mensagem.

```
 gtk_statusbar_push(GTK_STATUSBAR(window),
      gtk_statusbar_get_context_id(GTK_STATUSBAR(window), str), str);
```

Mostramos a mensagem na barra de status.

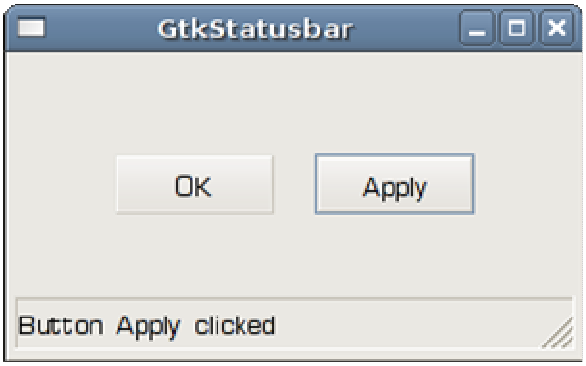


Figura: GtkStatusbar

## GtkIconView

O GtkIconView é um widget que mostra uma lista de ícones em uma grade.

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

enum
{
  COL_DISPLAY_NAME,
  COL_PIXBUF,
  NUM_COLS
};


GtkTreeModel * init_model(void)
{
  GtkListStore *list_store;
  GdkPixbuf *p1, *p2, *p3, *p4;
  GtkTreeIter iter;
  GError *err = NULL;


  p1 = gdk_pixbuf_new_from_file("ubuntu.png", &err);
  p2 = gdk_pixbuf_new_from_file("gnumeric.png", &err);
  p3 = gdk_pixbuf_new_from_file("blender.png", &err);
  p4 = gdk_pixbuf_new_from_file("inkscape.png", &err);

  assert(err==NULL);

  list_store = gtk_list_store_new(NUM_COLS,
      G_TYPE_STRING, GDK_TYPE_PIXBUF);

  int i = 0;
  for (i; i < 50; i++) {
    gtk_list_store_append(list_store, &iter);
    gtk_list_store_set(list_store, &iter, COL_DISPLAY_NAME,
        "ubuntu", COL_PIXBUF, p1, -1);
    gtk_list_store_append(list_store, &iter);
    gtk_list_store_set(list_store, &iter, COL_DISPLAY_NAME,
        "gnumeric", COL_PIXBUF, p2, -1);
    gtk_list_store_append(list_store, &iter);
    gtk_list_store_set(list_store, &iter, COL_DISPLAY_NAME,
        "blender", COL_PIXBUF, p3, -1);
    gtk_list_store_append(list_store, &iter);
    gtk_list_store_set(list_store, &iter, COL_DISPLAY_NAME,
        "inkscape", COL_PIXBUF, p4, -1);
```

```
  }

  return GTK_TREE_MODEL(list_store);
}

int main (int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *icon_view;
  GtkWidget *sw;

  gtk_init (&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);

  gtk_window_set_title(GTK_WINDOW (window), "Icon View");
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_container_set_border_width(GTK_CONTAINER(window), 10);
  gtk_widget_set_size_request(window, 350, 330);

  sw = gtk_scrolled_window_new(NULL, NULL);
  gtk_container_add(GTK_CONTAINER (window), sw);
  gtk_scrolled_window_set_policy(GTK_SCROLLED_WINDOW(sw),
      GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
  gtk_scrolled_window_set_shadow_type(GTK_SCROLLED_WINDOW(sw),
      GTK_SHADOW_IN);

  icon_view = gtk_icon_view_new_with_model(init_model());
  gtk_container_add(GTK_CONTAINER(sw), icon_view);

  gtk_icon_view_set_text_column(GTK_ICON_VIEW(icon_view),
      COL_DISPLAY_NAME);
  gtk_icon_view_set_pixbuf_column(GTK_ICON_VIEW(icon_view),
      COL_PIXBUF);
  gtk_icon_view_set_selection_mode(GTK_ICON_VIEW(icon_view),
      GTK_SELECTION_MULTIPLE);

  g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

O exemplo mostrará 200 ícones. Os ícones representam quatro importantes projetos de código aberto.

```
p1 = gdk_pixbuf_new_from_file("ubuntu.png", &err);
p2 = gdk_pixbuf_new_from_file("gnumeric.png", &err);
p3 = gdk_pixbuf_new_from_file("blender.png", &err);
p4 = gdk_pixbuf_new_from_file("inkscape.png", &err);
```

Carregamos 4 ícones do disco.

```
list_store = gtk_list_store_new(NUM_COLS,
    G_TYPE_STRING, GDK_TYPE_PIXBUF);
```

Nós iremos guardar dados textuais e pixbuf.

```
gtk_list_store_append(list_store, &iter);
gtk_list_store_set(list_store, &iter, COL_DISPLAY_NAME,
    "ubuntu", COL_PIXBUF, p1, -1);
```

Este código adiciona um novo ícone para a exibição de ícones.

```
  icon_view = gtk_icon_view_new_with_model(init_model());
  gtk_container_add(GTK_CONTAINER(sw), icon_view);

  gtk_icon_view_set_text_column(GTK_ICON_VIEW(icon_view),
      COL_DISPLAY_NAME);
  gtk_icon_view_set_pixbuf_column(GTK_ICON_VIEW(icon_view),
      COL_PIXBUF);
```

Criamos um **GtkIconView** widget e definimos um ícone e o nome para a exibição de ícone.

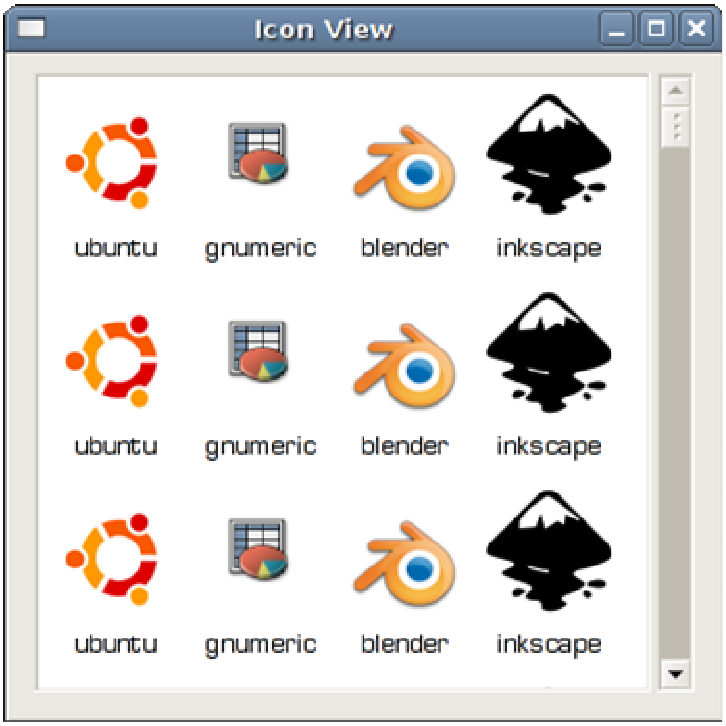


Figura: Exibição de ícone

## GtkTreeView Widget

Nesta parte do tutorial de programação GTK+ nós trabalharemos com um widget GtkTreeView (exibição em forma de árvore).

**GtkTreeView** widget é um complexo widget que pode ser usado para exibir listas e árvores. O widget pode ter uma ou múltiplas colunas. O GtkTreeView widget tem um MVC (Model View Controller) design de arquitetura. Isto significa que o dado é separado da exibição.

Existem vários outros objetos que são utilizados com o widget GtkTreeView. O **GtkCellRenderer** determina como que os dados serão exibidos no **GtkTreeViewColumn**. O **GtkListStore** e o **GtkTreeStore** representam o modelo. Eles lidam com dados, que são exibidos no widget GtkTreeView. **GtkTreeIter** é uma estrutura usada para referenciar a coluna no GtkTreeView. **GtkTreeSelection** é um objeto que lida com seleções.

# Lista Simples de exibição

O primeiro exemplo mostrará uma lista simples de exibição. Mostraremos dados textuais.

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

enum
{
  LIST_ITEM = 0,
  N_COLUMNS
};

static void
init_list(GtkWidget *list)
{

  GtkCellRenderer *renderer;
  GtkTreeViewColumn *column;
  GtkListStore *store;

  renderer = gtk_cell_renderer_text_new();
  column = gtk_tree_view_column_new_with_attributes("List Items",
          renderer, "text", LIST_ITEM, NULL);
  gtk_tree_view_append_column(GTK_TREE_VIEW(list), column);

  store = gtk_list_store_new(N_COLUMNS, G_TYPE_STRING);

  gtk_tree_view_set_model(GTK_TREE_VIEW(list),
      GTK_TREE_MODEL(store));

  g_object_unref(store);
}

static void
add_to_list(GtkWidget *list, const gchar *str)
{
  GtkListStore *store;
  GtkTreeIter iter;

  store = GTK_LIST_STORE(gtk_tree_view_get_model
      (GTK_TREE_VIEW(list)));

  gtk_list_store_append(store, &iter);
  gtk_list_store_set(store, &iter, LIST_ITEM, str, -1);
}


void  on_changed(GtkWidget *widget, gpointer label)
{
  GtkTreeIter iter;
  GtkTreeModel *model;
  char *value;


  if (gtk_tree_selection_get_selected(
      GTK_TREE_SELECTION(widget), &model, &iter)) {

    gtk_tree_model_get(model, &iter, LIST_ITEM, &value,  -1);
    gtk_label_set_text(GTK_LABEL(label), value);
    g_free(value);
  }

}

int main (int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *list;

  GtkWidget *vbox;
```

```
  GtkWidget *label;
  GtkTreeSelection *selection;

  gtk_init(&argc, &argv);


  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_container_set_border_width(GTK_CONTAINER(window), 10);
  gtk_widget_set_size_request(window, 270, 250);
  gtk_window_set_title(GTK_WINDOW(window), "List View");

  list = gtk_tree_view_new();
  gtk_tree_view_set_headers_visible(GTK_TREE_VIEW(list), FALSE);

  vbox = gtk_vbox_new(FALSE, 0);

  gtk_box_pack_start(GTK_BOX(vbox), list, TRUE, TRUE, 5);

  label = gtk_label_new("");
  gtk_label_set_justify(GTK_LABEL(label), GTK_JUSTIFY_CENTER);
  gtk_box_pack_start(GTK_BOX(vbox), label, FALSE, FALSE, 5);

  gtk_container_add(GTK_CONTAINER(window), vbox);

  init_list(list);
  add_to_list(list, "Aliens");
  add_to_list(list, "Leon");
  add_to_list(list, "Capote");
  add_to_list(list, "Saving private Ryan");
  add_to_list(list, "Der Untergang");


  selection = gtk_tree_view_get_selection(GTK_TREE_VIEW(list));

  g_signal_connect(selection, "changed",
      G_CALLBACK(on_changed), label);

  g_signal_connect(G_OBJECT (window), "destroy",
      G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main ();

  return 0;
}
```

No nosso exemplo de código, vamos mostrar 5 itens no GtkTreeView. Teremos apenas uma coluna e o cabeçalho da coluna será escondido. Vamos colocar o **GtkVBox** na janela. Esta caixa terá dois widgets. Um **GtkTreeView** e um **GtkLabel**.

```
  list = gtk_tree_view_new();
  gtk_tree_view_set_headers_visible(GTK_TREE_VIEW(list), FALSE);
```

O **GtkTreeView** é criado e as colunas são escondidas.

```
  label = gtk_label_new("");
  gtk_label_set_justify(GTK_LABEL(label), GTK_JUSTIFY_CENTER);
  gtk_box_pack_start(GTK_BOX(vbox), label, FALSE, FALSE, 5);
```

O **GtkLabel** é criado, centralizado e colocado abaixo do **GtkTreeView**

```
 init_list(list);
```

Esta função inicializa a lista.

```
renderer = gtk_cell_renderer_text_new();
column = gtk_tree_view_column_new_with_attributes("List Items",
        renderer, "text", LIST_ITEM, NULL);
gtk_tree_view_append_column(GTK_TREE_VIEW(list), column);
```

Dentro da função criamos e adicionamos uma simples coluna.

```
store = gtk_list_store_new(N_COLUMNS, G_TYPE_STRING);

gtk_tree_view_set_model(GTK_TREE_VIEW(list),
    GTK_TREE_MODEL(store));
```

Criamos um **GtkListStore** (um modelo) e setamos ele para a lista.

```
g_object_unref(store);
```

O modelo é destruído automaticamente com a exibição.

```
add_to_list(list, "Aliens");
```

Esta função de usuário adiciona uma opção à lista.

```
store = GTK_LIST_STORE(gtk_tree_view_get_model
    (GTK_TREE_VIEW(list)));

gtk_list_store_append(store, &iter);
gtk_list_store_set(store, &iter, LIST_ITEM, str, -1);
```

Dentro da função **add_to_list()** function, temos o modelo usando a chamada de função **gtk_tree_view_get_model()**. Adicionamos uma nova linha e definimos um valor para ela que é referenciado por um objeto **GtkTreeIter**.

```
selection = gtk_tree_view_get_selection(GTK_TREE_VIEW(list));
```

O **GtkTreeSelection** não precisa ser criado explicitamente. Ele é automaticamente criado com o widget **GtkTreeView**. A referência ao widget é obtida usando a chamada de função **gtk_tree_view_get_selection()**.

```
g_signal_connect(selection, "changed",
    G_CALLBACK(on_changed), label);
```

Ao sinal de mudança da **GtkTreeSelection**, chamamos o manipulador **on_changed()**.

```
gtk_tree_model_get(model, &iter, LIST_ITEM, &value,  -1);
gtk_label_set_text(GTK_LABEL(label), value);
```

Dentro da função de manipulador, obtemos o valor da célula na linha referenciada pelo objeto.

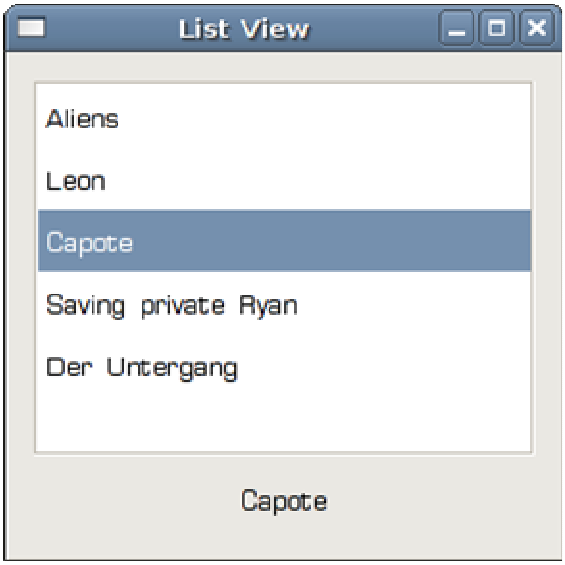


Figura: Lista de exibição

# Lista de exibição avançada

O segundo exemplo irá adicionar funcionalidade ao anterior. Seremos capazes de adicionar e remover os itens da lista.

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

enum
{
  LIST_ITEM = 0,
  N_COLUMNS
};

GtkWidget *list;


static void
append_item(GtkWidget *widget, gpointer entry)
{
  GtkListStore *store;
  GtkTreeIter  iter;

  const char *str = gtk_entry_get_text(entry);

  store = GTK_LIST_STORE(gtk_tree_view_get_model(
       GTK_TREE_VIEW(list)));

  gtk_list_store_append(store, &iter);
  gtk_list_store_set(store, &iter, LIST_ITEM, str, -1);
}

static void
remove_item(GtkWidget *widget, gpointer selection)
{
  GtkListStore *store;
  GtkTreeModel *model;
  GtkTreeIter  iter;
```

```
  store = GTK_LIST_STORE(gtk_tree_view_get_model(
      GTK_TREE_VIEW (list)));
  model = gtk_tree_view_get_model (GTK_TREE_VIEW (list));

  if (gtk_tree_model_get_iter_first(model, &iter) == FALSE)
      return;

  if (gtk_tree_selection_get_selected(GTK_TREE_SELECTION(selection),
      &model, &iter)) {
    gtk_list_store_remove(store, &iter);
  }
}

static void
remove_all(GtkWidget *widget, gpointer selection)
{
  GtkListStore *store;
  GtkTreeModel *model;
  GtkTreeIter  iter;


  store = GTK_LIST_STORE(gtk_tree_view_get_model(
      GTK_TREE_VIEW (list)));
  model = gtk_tree_view_get_model (GTK_TREE_VIEW (list));

  if (gtk_tree_model_get_iter_first(model, &iter) == FALSE)
      return;
  gtk_list_store_clear(store);
}

static void
init_list(GtkWidget *list)
{

  GtkCellRenderer    *renderer;
  GtkTreeViewColumn  *column;
  GtkListStore       *store;

  renderer = gtk_cell_renderer_text_new();
  column = gtk_tree_view_column_new_with_attributes("List Item",
          renderer, "text", LIST_ITEM, NULL);
  gtk_tree_view_append_column(GTK_TREE_VIEW (list), column);

  store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING);

  gtk_tree_view_set_model(GTK_TREE_VIEW (list),
      GTK_TREE_MODEL(store));

  g_object_unref(store);
}


int main (int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *sw;

  GtkWidget *remove;
  GtkWidget *add;
  GtkWidget *removeAll;
  GtkWidget *entry;

  GtkWidget *vbox;
  GtkWidget *hbox;

  GtkTreeSelection *selection;
```

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


  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  sw = gtk_scrolled_window_new(NULL, NULL);
  list = gtk_tree_view_new();

  gtk_window_set_title (GTK_WINDOW (window), "List View");
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);
  gtk_widget_set_size_request (window, 370, 270);

  gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW(sw),
            GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);

  gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW(sw),
            GTK_SHADOW_ETCHED_IN);

  gtk_tree_view_set_headers_visible (GTK_TREE_VIEW (list), FALSE);

  vbox = gtk_vbox_new(FALSE, 0);

  gtk_box_pack_start(GTK_BOX(vbox), sw, TRUE, TRUE, 5);

  hbox = gtk_hbox_new(TRUE, 5);

  add = gtk_button_new_with_label("Add");
  remove = gtk_button_new_with_label("Remove");
  removeAll = gtk_button_new_with_label("Remove All");
  entry = gtk_entry_new();

  gtk_box_pack_start(GTK_BOX(hbox), add, FALSE, TRUE, 3);
  gtk_box_pack_start(GTK_BOX(hbox), entry, FALSE, TRUE, 3);
  gtk_box_pack_start(GTK_BOX(hbox), remove, FALSE, TRUE, 3);
  gtk_box_pack_start(GTK_BOX(hbox), removeAll, FALSE, TRUE, 3);

  gtk_box_pack_start(GTK_BOX(vbox), hbox, FALSE, TRUE, 3);

  gtk_container_add(GTK_CONTAINER (sw), list);
  gtk_container_add(GTK_CONTAINER (window), vbox);

  init_list(list);

  selection  = gtk_tree_view_get_selection(GTK_TREE_VIEW(list));

  g_signal_connect(G_OBJECT(add), "clicked",
          G_CALLBACK(append_item), entry);

  g_signal_connect(G_OBJECT(remove), "clicked",
          G_CALLBACK(remove_item), selection);

  g_signal_connect(G_OBJECT(removeAll), "clicked",
          G_CALLBACK(remove_all), selection);

  g_signal_connect (G_OBJECT (window), "destroy",
          G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main ();

  return 0;
}
```

Em vez de um rótulo, nós criamos três botões e uma entrada de texto. Seremos capazes de adicionar uma nova opção de forma dinâmica e remova a opção selecionada atualmente ou remover todos os itens da lista.

```
sw = gtk_scrolled_window_new(NULL, NULL);
...

gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW(sw),
          GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);

gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW(sw),
          GTK_SHADOW_ETCHED_IN);

...
gtk_box_pack_start(GTK_BOX(vbox), sw, TRUE, TRUE, 5);
...
gtk_container_add(GTK_CONTAINER (sw), list);
```

O **GtkTreeView** é colocado dentro da janela.

```
if (gtk_tree_selection_get_selected(GTK_TREE_SELECTION(selection),
    &model, &iter)) {
  gtk_list_store_remove(store, &iter);
}
```

A função **gtk_list_store_remove()** remove um item da lista.

```
gtk_list_store_clear(store);
```

A **gtk_list_store_clear()** removerá todos os itens da lista.

```
if (gtk_tree_model_get_iter_first(model, &iter) == FALSE)


    return;
```

Este código verifica se há algum item sobrando na lista. Obviamente, podemos remover itens somente se houver pelo menos um na lista.

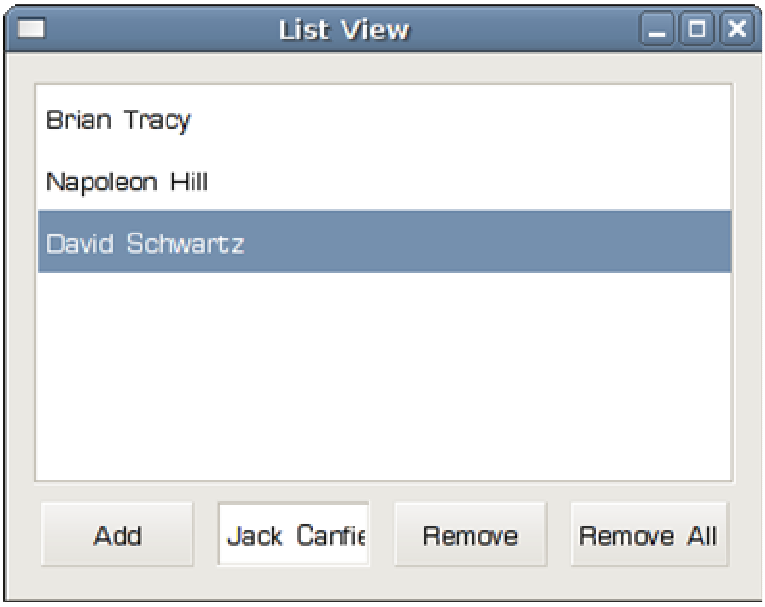


Figura: Lista de exibição avançada

# Exibição de árvore

Em seguidao vamos usar **GtkTreeView** widget para exibir dados hierárquicos. Nos dois exemplos anteriores, usamos a exibição de lista, agora vamos utilizar a vista de árvore.

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

enum
{
  COLUMN = 0,
  NUM_COLS
} ;

void  on_changed(GtkWidget *widget, gpointer statusbar)
{
  GtkTreeIter iter;
  GtkTreeModel *model;
  char *value;


  if (gtk_tree_selection_get_selected(
      GTK_TREE_SELECTION(widget), &model, &iter)) {

    gtk_tree_model_get(model, &iter, COLUMN, &value,  -1);
    gtk_statusbar_push(GTK_STATUSBAR(statusbar),
        gtk_statusbar_get_context_id(GTK_STATUSBAR(statusbar),
            value), value);
    g_free(value);
  }
}


static GtkTreeModel *
create_and_fill_model (void)
{
  GtkTreeStore *treestore;
  GtkTreeIter toplevel, child;

  treestore = gtk_tree_store_new(NUM_COLS,
                 G_TYPE_STRING);

  gtk_tree_store_append(treestore, &toplevel, NULL);
  gtk_tree_store_set(treestore, &toplevel,
                     COLUMN, "Scripting languages",
                     -1);

  gtk_tree_store_append(treestore, &child, &toplevel);
  gtk_tree_store_set(treestore, &child,
                     COLUMN, "Python",
                     -1);
  gtk_tree_store_append(treestore, &child, &toplevel);
  gtk_tree_store_set(treestore, &child,
                     COLUMN, "Perl",
                     -1);
  gtk_tree_store_append(treestore, &child, &toplevel);
  gtk_tree_store_set(treestore, &child,
                     COLUMN, "PHP",
                     -1);

  gtk_tree_store_append(treestore, &toplevel, NULL);
  gtk_tree_store_set(treestore, &toplevel,
                     COLUMN, "Compiled languages",
                     -1);

  gtk_tree_store_append(treestore, &child, &toplevel);
```

```
  gtk_tree_store_set(treestore, &child,
                     COLUMN, "C",
                     -1);

  gtk_tree_store_append(treestore, &child, &toplevel);
  gtk_tree_store_set(treestore, &child,
                     COLUMN, "C++",
                     -1);

  gtk_tree_store_append(treestore, &child, &toplevel);
  gtk_tree_store_set(treestore, &child,
                     COLUMN, "Java",
                     -1);

  return GTK_TREE_MODEL(treestore);
}



static GtkWidget *
create_view_and_model (void)
{
  GtkTreeViewColumn *col;
  GtkCellRenderer *renderer;
  GtkWidget *view;
  GtkTreeModel *model;

  view = gtk_tree_view_new();

  col = gtk_tree_view_column_new();
  gtk_tree_view_column_set_title(col, "Programming languages");
  gtk_tree_view_append_column(GTK_TREE_VIEW(view), col);

  renderer = gtk_cell_renderer_text_new();
  gtk_tree_view_column_pack_start(col, renderer, TRUE);
  gtk_tree_view_column_add_attribute(col, renderer,
      "text", COLUMN);

  model = create_and_fill_model();
  gtk_tree_view_set_model(GTK_TREE_VIEW(view), model);
  g_object_unref(model);

  return view;
}


int
main (int argc, char **argv)
{
  GtkWidget *window;
  GtkWidget *view;
  GtkTreeSelection *selection;
  GtkWidget *vbox;
  GtkWidget *statusbar;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_title(GTK_WINDOW(window), "Tree View");
  gtk_widget_set_size_request (window, 350, 300);


  vbox = gtk_vbox_new(FALSE, 2);
  gtk_container_add(GTK_CONTAINER(window), vbox);
```

```
  view = create_view_and_model();
  selection = gtk_tree_view_get_selection(GTK_TREE_VIEW(view));

  gtk_box_pack_start(GTK_BOX(vbox), view, TRUE, TRUE, 1);

  statusbar = gtk_statusbar_new();
  gtk_box_pack_start(GTK_BOX(vbox), statusbar, FALSE, TRUE, 1);

  g_signal_connect(selection, "changed",
      G_CALLBACK(on_changed), statusbar);

  g_signal_connect (G_OBJECT (window), "destroy",
          G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

No nosso exemplo, dividimos as linguagens de programação em dois grupos. Linguagens de scripting e linguagens compiladas. As categorias de linguagem servem como nós de nível superior para a sua lista de itens. O nó atualmente selecionado ou item é mostrado na barra de estado.

Os passos para criar uma exibição de árvore é muito semelhante à criação de uma exibição de lista.

```
GtkTreeStore *treestore;
```

Aqui usamos um diferente modelo. Um **GtkTreeStore**.

```
treestore = gtk_tree_store_new(NUM_COLS,
                G_TYPE_STRING);
```

Criamos um **GtkTreeStore** com apenas uma coluna.

```
gtk_tree_store_append(treestore, &toplevel, NULL);
gtk_tree_store_set(treestore, &toplevel,
                   COLUMN, "Scripting languages",
                   -1);
```

Isto cria um nó de nível superior.

```
gtk_tree_store_append(treestore, &child, &toplevel);
gtk_tree_store_set(treestore, &child,
                   COLUMN, "Python",
                   -1);
```

Este código cria um item filho.

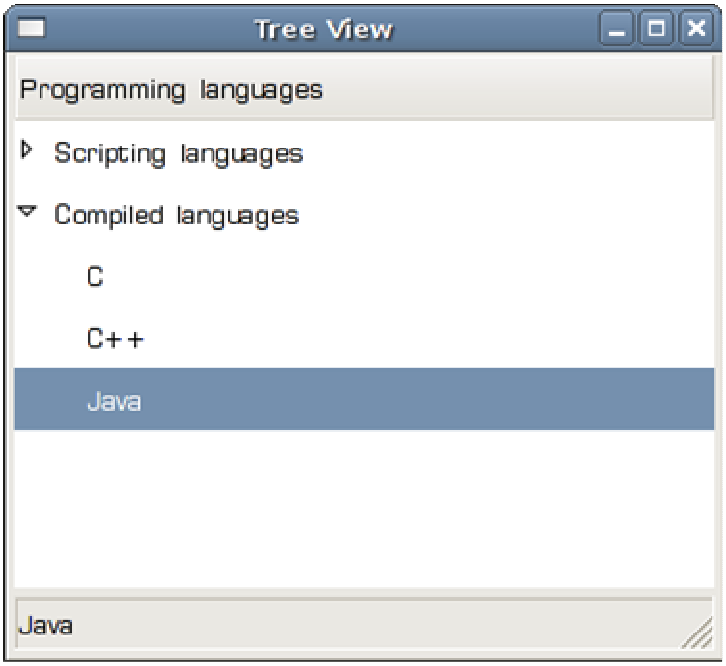


Figura: Exibição de árvore

## GtkTexView Widget

Nesta parte do tutorial de programação GTK +, vamos trabalhar com um widget GtkTexView.

**GtkTexView** widget é usado para exibição e edição de texto com várias linhas. GtkTexView widget possui também o design MVC. O GtkTextView representa componente dee exibição e **GtkTexBuffer** representa o componente modelo. O GtkTexBuffer é usado para manipular os dados de texto. **GtkTextTag** é um atributo que pode ser aplicado ao texto. O **GtkTextIter** representa uma posição entre dois caracteres no texto. Toda manipulação com o texto é feita com iteradores de texto.

## Exemplo simples

Em nosso primeiro exemplo, vamos mostrar algumas das funcionalidades do GtkTexView. Nós mostramos como aplicar várias marcas de texto com os dados de texto no GtkTextView.

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


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *view;
  GtkWidget *vbox;

  GtkTextBuffer *buffer;
  GtkTextIter start, end;
  GtkTextIter iter;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
```

```
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 200);
  gtk_window_set_title(GTK_WINDOW(window), "TextView");
  gtk_container_set_border_width(GTK_CONTAINER(window), 5);
  GTK_WINDOW(window)->allow_shrink = TRUE;

  vbox = gtk_vbox_new(FALSE, 0);
  view = gtk_text_view_new();
  gtk_box_pack_start(GTK_BOX(vbox), view, TRUE, TRUE, 0);

  buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(view));

  gtk_text_buffer_create_tag(buffer, "gap",
        "pixels_above_lines", 30, NULL);

  gtk_text_buffer_create_tag(buffer, "lmarg",
      "left_margin", 5, NULL);
  gtk_text_buffer_create_tag(buffer, "blue_fg",
      "foreground", "blue", NULL);
  gtk_text_buffer_create_tag(buffer, "gray_bg",
      "background", "gray", NULL);
  gtk_text_buffer_create_tag(buffer, "italic",
      "style", PANGO_STYLE_ITALIC, NULL);
  gtk_text_buffer_create_tag(buffer, "bold",
      "weight", PANGO_WEIGHT_BOLD, NULL);

  gtk_text_buffer_get_iter_at_offset(buffer, &iter, 0);

  gtk_text_buffer_insert(buffer, &iter, "Plain text\n", -1);
  gtk_text_buffer_insert_with_tags_by_name(buffer, &iter,
        "Colored Text\n", -1, "blue_fg", "lmarg",  NULL);
  gtk_text_buffer_insert_with_tags_by_name (buffer, &iter,
        "Text with colored background\n", -1, "lmarg", "gray_bg", NULL);

  gtk_text_buffer_insert_with_tags_by_name (buffer, &iter,
        "Text in italics\n", -1, "italic", "lmarg",  NULL);

  gtk_text_buffer_insert_with_tags_by_name (buffer, &iter,
        "Bold text\n", -1, "bold", "lmarg",  NULL);

  gtk_container_add(GTK_CONTAINER(window), vbox);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

O exemplo mostra um texto com diferentes **GtkTextTags** aplicados.

```
view = gtk_text_view_new();
```

O **GtkTextView** é criado.

```
gtk_text_buffer_create_tag(buffer, "blue_fg",
    "foreground", "blue", NULL);
```

Este é um exemplo de **GtkTextTag**. A marca muda a cor do texto para azul.

```
gtk_text_buffer_insert_with_tags_by_name(buffer, &iter,
      "Colored Text\n", -1, "blue_fg", "lmarg",  NULL);
```

Este código insere um texto na marca específica de texto **blue_fg**.

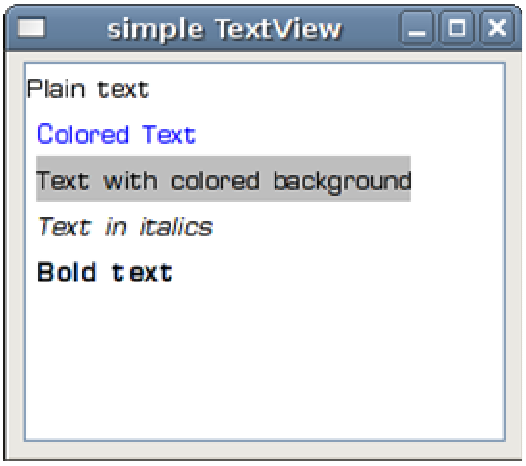



Figura: Exibição de texto simples

## Linhas e colunas

O exemplo a seguir irá mostrar a linha e coluna atuais do cursor de texto.

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

update_statusbar(GtkTextBuffer *buffer,
    GtkStatusbar  *statusbar)
{
  gchar *msg;
  gint row, col;
  GtkTextIter iter;

  gtk_statusbar_pop(statusbar, 0);

  gtk_text_buffer_get_iter_at_mark(buffer,
      &iter, gtk_text_buffer_get_insert(buffer));

  row = gtk_text_iter_get_line(&iter);
  col = gtk_text_iter_get_line_offset(&iter);

  msg = g_strdup_printf("Col %d Ln %d", col+1, row+1);

  gtk_statusbar_push(statusbar, 0, msg);

  g_free(msg);
}

static void
mark_set_callback(GtkTextBuffer *buffer,
    const GtkTextIter *new_location, GtkTextMark *mark,
    gpointer data)
{
  update_statusbar(buffer, GTK_STATUSBAR(data));
}


int main( int argc, char *argv[])
```

```
{

  GtkWidget *window;
  GtkWidget *vbox;

  GtkWidget *toolbar;
  GtkWidget *view;
  GtkWidget *statusbar;
  GtkToolItem *exit;
  GtkTextBuffer *buffer;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 200);
  gtk_window_set_title(GTK_WINDOW(window), "lines & cols");

  vbox = gtk_vbox_new(FALSE, 0);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  toolbar = gtk_toolbar_new();
  gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_ICONS);

  exit = gtk_tool_button_new_from_stock(GTK_STOCK_QUIT);
  gtk_toolbar_insert(GTK_TOOLBAR(toolbar), exit, -1);

  gtk_box_pack_start(GTK_BOX(vbox), toolbar, FALSE, FALSE, 5);

  view = gtk_text_view_new();
  gtk_box_pack_start(GTK_BOX(vbox), view, TRUE, TRUE, 0);
  gtk_widget_grab_focus(view);

  buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(view));

  statusbar = gtk_statusbar_new();
  gtk_box_pack_start(GTK_BOX(vbox), statusbar, FALSE, FALSE, 0);

  g_signal_connect(G_OBJECT(exit), "clicked",
        G_CALLBACK(gtk_main_quit), NULL);

  g_signal_connect(buffer, "changed",
        G_CALLBACK(update_statusbar), statusbar);

  g_signal_connect_object(buffer, "mark_set",
        G_CALLBACK(mark_set_callback), statusbar, 0);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), NULL);

  gtk_widget_show_all(window);

  update_statusbar(buffer, GTK_STATUSBAR (statusbar));

  gtk_main();

  return 0;
}
```

Neste exemplo de código, mostramos a posição atual do cursor de texto na barra de estado.

```
 view = gtk_text_view_new();
```

O **GtkTextView** widget é criado.

```
g_signal_connect(buffer, "changed",
      G_CALLBACK(update_statusbar), statusbar);
```

Quando alteramos o texto, chamamos o manipulador **update_statusbar()**.

```
g_signal_connect_object(buffer, "mark_set",
      G_CALLBACK(mark_set_callback), statusbar, 0);
```

O sinal **mark_set** é emitido quando o cursos se move.

```
gtk_statusbar_pop(statusbar, 0);
```

Esta linha dede código limpa qualquer mensagem anterior a partir da barra de status.

```
gtk_text_buffer_get_iter_at_mark(buffer,
    &iter, gtk_text_buffer_get_insert(buffer));

row = gtk_text_iter_get_line(&iter);
col = gtk_text_iter_get_line_offset(&iter);
```

Estas linhas determinam a linha / coluna atual e coluna.

```
msg = g_strdup_printf("Col %d Ln %d", col+1, row+1);
```

Este código cria o texto a ser exibido na barra de estatus.

```
gtk_statusbar_push(statusbar, 0, msg);
```

Mostramos o texto na barra de status.

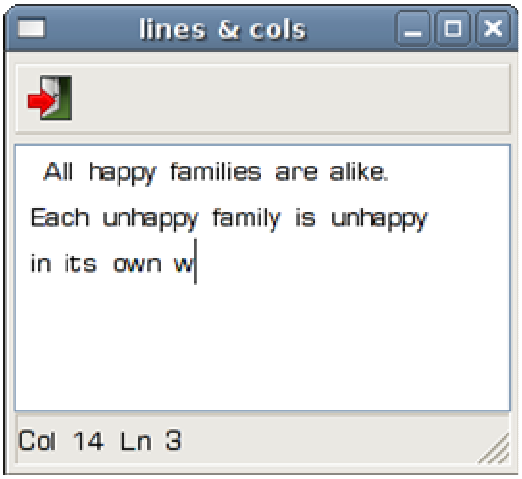



Figura: Linhas & colunas

# Pesquisa e realce

No próximo exemplo, vamos fazer algumas pesquisas na **GtkTextBuffer**. Vamos destacar alguns padrões de texto no buffer de texto.

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

```
gboolean key_pressed(GtkWidget * window,
    GdkEventKey* event, GtkTextBuffer *buffer) {

  GtkTextIter start_sel, end_sel;
  GtkTextIter start_find, end_find;
  GtkTextIter start_match, end_match;
  gboolean selected;
  gchar *text;



  if ((event->type == GDK_KEY_PRESS) &&
     (event->state & GDK_CONTROL_MASK)) {

    switch (event->keyval)
    {
      case GDK_m :
        selected = gtk_text_buffer_get_selection_bounds(buffer,
            &start_sel, &end_sel);
      if (selected) {
        gtk_text_buffer_get_start_iter(buffer, &start_find);
        gtk_text_buffer_get_end_iter(buffer, &end_find);

        gtk_text_buffer_remove_tag_by_name(buffer, "gray_bg",
            &start_find, &end_find);
        text = (char *) gtk_text_buffer_get_text(buffer, &start_sel,
            &end_sel, FALSE);

        while ( gtk_text_iter_forward_search(&start_find, text,
                GTK_TEXT_SEARCH_TEXT_ONLY |
                GTK_TEXT_SEARCH_VISIBLE_ONLY,
                &start_match, &end_match, NULL) ) {

          gtk_text_buffer_apply_tag_by_name(buffer, "gray_bg",
              &start_match, &end_match);
          int offset = gtk_text_iter_get_offset(&end_match);
          gtk_text_buffer_get_iter_at_offset(buffer,
              &start_find, offset);
        }

        g_free(text);
      }

      break;

      case GDK_r:
        gtk_text_buffer_get_start_iter(buffer, &start_find);
        gtk_text_buffer_get_end_iter(buffer, &end_find);

        gtk_text_buffer_remove_tag_by_name(buffer, "gray_bg",
            &start_find, &end_find);
      break;
    }
  }

  return FALSE;
}


int main( int argc, char *argv[])
{

  GtkWidget *window;
  GtkWidget *view;
  GtkWidget *vbox;
```

```
  GtkTextBuffer *buffer;
  GtkTextIter start, end;
  GtkTextIter iter;

  gtk_init(&argc, &argv);

  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 250, 200);
  gtk_window_set_title(GTK_WINDOW(window), "Search & Highlight");
  gtk_container_set_border_width(GTK_CONTAINER(window), 5);
  GTK_WINDOW(window)->allow_shrink = TRUE;

  vbox = gtk_vbox_new(FALSE, 0);
  view = gtk_text_view_new();
  gtk_widget_add_events(view, GDK_BUTTON_PRESS_MASK);
  gtk_box_pack_start(GTK_BOX(vbox), view, TRUE, TRUE, 0);

  buffer = gtk_text_view_get_buffer(GTK_TEXT_VIEW(view));
  gtk_text_buffer_create_tag(buffer, "gray_bg",
      "background", "gray", NULL);
  gtk_container_add(GTK_CONTAINER(window), vbox);

  g_signal_connect_swapped(G_OBJECT(window), "destroy",
        G_CALLBACK(gtk_main_quit), G_OBJECT(window));

  g_signal_connect(G_OBJECT(window), "key-press-event",
        G_CALLBACK(key_pressed), buffer);

  gtk_widget_show_all(window);

  gtk_main();

  return 0;
}
```

No nosso exemplo o código que usamos atalhos de teclado. O atalho Ctrl + M destaca a todas as ocorrências do texto selecionado. O Ctrl + R remove os destaques do texto.

```
gtk_text_buffer_create_tag(buffer, "gray_bg",
    "background", "gray", NULL);
```

Este é o **GtkTextTag** que usamos em nosso exemplo. A marca cria um fundo do texto cinza.

```
selected = gtk_text_buffer_get_selection_bounds(buffer,
    &start_sel, &end_sel);
```

Aqui temos a posições inicial e final do texto selecionado.

```
gtk_text_buffer_get_start_iter(buffer, &start_find);
gtk_text_buffer_get_end_iter(buffer, &end_find);
```

Ficamos com a primeira e última posição no buffer de texto.

```
gtk_text_buffer_remove_tag_by_name(buffer, "gray_bg",
    &start_find, &end_find);
```

Nós removemos qualquer marca de texto anterior.

```
text = (char *) gtk_text_buffer_get_text(buffer, &start_sel,
    &end_sel, FALSE);
```

Obtemos o texto selecionado. Este é o texto que vamos procurar.

```
while ( gtk_text_iter_forward_search(&start_find, text,
        GTK_TEXT_SEARCH_TEXT_ONLY |
        GTK_TEXT_SEARCH_VISIBLE_ONLY,
        &start_match, &end_match, NULL) ) {

  gtk_text_buffer_apply_tag_by_name(buffer, "gray_bg",
      &start_match, &end_match);
   int offset = gtk_text_iter_get_offset(&end_match);
  gtk_text_buffer_get_iter_at_offset(buffer,
      &start_find, offset);
}
```

Este código procura todas as ocorrências do nosso texto selecionado. Se encontrar qualquer partida, nós aplicamos a marca de texto. Após a partida, o ponto final da palavra torna-se o ponto de partida para a próxima pesquisa.

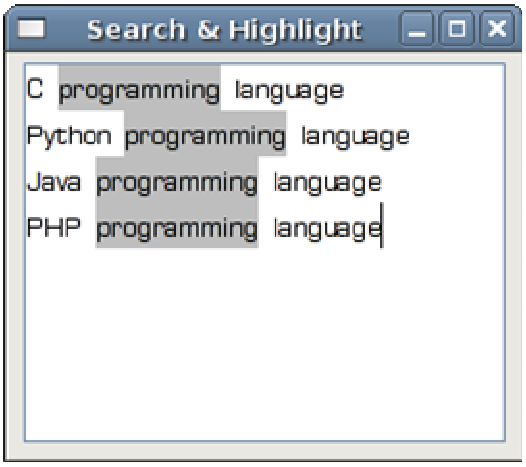


Figura: Search & Highlight

## GTK+ widget customizado

Nesta parte do tutorial de programação GTK+, criaremos um GTK+ widget customizado, onde nós usarema a biblioteca gráfica Cairo.

## CPU widget

No próximo exemplo vamos criar um widget CPU.

```
/* cpu.h */

#ifndef __CPU_H
#define __CPU_H

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

G_BEGIN_DECLS


#define GTK_CPU(obj) GTK_CHECK_CAST(obj, gtk_cpu_get_type (), GtkCpu)
#define GTK_CPU_CLASS(klass) GTK_CHECK_CLASS_CAST(klass,
gtk_cpu_get_type(), GtkCpuClass)
#define GTK_IS_CPU(obj) GTK_CHECK_TYPE(obj, gtk_cpu_get_type())
```

```
typedef struct _GtkCpu GtkCpu;
typedef struct _GtkCpuClass GtkCpuClass;


struct _GtkCpu {
  GtkWidget widget;

  gint sel;
};

struct _GtkCpuClass {
  GtkWidgetClass parent_class;
};


GtkType gtk_cpu_get_type(void);
void gtk_cpu_set_sel(GtkCpu *cpu, gint sel);
GtkWidget * gtk_cpu_new();


G_END_DECLS

#endif /* __CPU_H */
/* cpu.c */

#include "cpu.h"


static void gtk_cpu_class_init(GtkCpuClass *klass);
static void gtk_cpu_init(GtkCpu *cpu);
static void gtk_cpu_size_request(GtkWidget *widget,
    GtkRequisition *requisition);
static void gtk_cpu_size_allocate(GtkWidget *widget,
    GtkAllocation *allocation);
static void gtk_cpu_realize(GtkWidget *widget);
static gboolean gtk_cpu_expose(GtkWidget *widget,
    GdkEventExpose *event);
static void gtk_cpu_paint(GtkWidget *widget);
static void gtk_cpu_destroy(GtkObject *object);


GtkType
gtk_cpu_get_type(void)
{
  static GtkType gtk_cpu_type = 0;


  if (!gtk cpu type) {
      static const GtkTypeInfo gtk_cpu_info = {
          "GtkCpu",
          sizeof(GtkCpu),
          sizeof(GtkCpuClass),
          (GtkClassInitFunc) gtk_cpu_class_init,
          (GtkObjectInitFunc) gtk_cpu_init,
          NULL,
          NULL,
          (GtkClassInitFunc) NULL
      };
```

```
      gtk_cpu_type = gtk_type_unique(GTK_TYPE_WIDGET, >k_cpu_info);
  }


  return gtk_cpu_type;
}

void
gtk_cpu_set_state(GtkCpu *cpu, gint num)
{
   cpu->sel = num;
   gtk_cpu_paint(GTK_WIDGET(cpu));
}


GtkWidget * gtk_cpu_new()
{
   return GTK_WIDGET(gtk_type_new(gtk_cpu_get_type()));
}


static void
gtk_cpu_class_init(GtkCpuClass *klass)
{
  GtkWidgetClass *widget_class;
  GtkObjectClass *object_class;


  widget_class = (GtkWidgetClass *) klass;
  object_class = (GtkObjectClass *) klass;

  widget_class->realize = gtk_cpu_realize;
  widget_class->size_request = gtk_cpu_size_request;
  widget_class->size_allocate = gtk_cpu_size_allocate;
  widget_class->expose_event = gtk_cpu_expose;

  object_class->destroy = gtk_cpu_destroy;
}


static void
gtk_cpu_init(GtkCpu *cpu)
{
   cpu->sel = 0;
}


static void
gtk_cpu_size_request(GtkWidget *widget,
    GtkRequisition *requisition)
{
  g_return_if_fail(widget != NULL);
  g_return_if_fail(GTK_IS_CPU(widget));
  g_return_if_fail(requisition != NULL);

  requisition->width = 80;
  requisition->height = 100;
}
```

```
static void
gtk_cpu_size_allocate(GtkWidget *widget,
    GtkAllocation *allocation)
{
  g_return_if_fail(widget != NULL);
  g_return_if_fail(GTK_IS_CPU(widget));
  g_return_if_fail(allocation != NULL);

  widget->allocation = *allocation;

  if (GTK_WIDGET_REALIZED(widget)) {
     gdk_window_move_resize(
         widget->window,
         allocation->x, allocation->y,
         allocation->width, allocation->height
     );
   }
}


static void
gtk_cpu_realize(GtkWidget *widget)
{
  GdkWindowAttr attributes;
  guint attributes_mask;

  g_return_if_fail(widget != NULL);
  g_return_if_fail(GTK_IS_CPU(widget));

  GTK_WIDGET_SET_FLAGS(widget, GTK_REALIZED);

  attributes.window_type = GDK_WINDOW_CHILD;
  attributes.x = widget->allocation.x;
  attributes.y = widget->allocation.y;
  attributes.width = 80;
  attributes.height = 100;

  attributes.wclass = GDK_INPUT_OUTPUT;
  attributes.event_mask = gtk_widget_get_events(widget) |
GDK_EXPOSURE_MASK;

  attributes_mask = GDK_WA_X | GDK_WA_Y;

  widget->window = gdk_window_new(
     gtk_widget_get_parent_window (widget),
     & attributes, attributes_mask
  );

  gdk_window_set_user_data(widget->window, widget);

  widget->style = gtk_style_attach(widget->style, widget->window);
  gtk_style_set_background(widget->style, widget->window,
GTK_STATE_NORMAL);
}


static gboolean
gtk_cpu_expose(GtkWidget *widget,
    GdkEventExpose *event)
{
```

```
  g_return_val_if_fail(widget != NULL, FALSE);
  g_return_val_if_fail(GTK_IS_CPU(widget), FALSE);
  g_return_val_if_fail(event != NULL, FALSE);

  gtk_cpu_paint(widget);

  return FALSE;
}


static void
gtk_cpu_paint(GtkWidget *widget)
{
  cairo_t *cr;

  cr = gdk_cairo_create(widget->window);

  cairo_translate(cr, 0, 7);

  cairo_set_source_rgb(cr, 0, 0, 0);
  cairo_paint(cr);

  gint pos = GTK_CPU(widget)->sel;
  gint rect = pos / 5;

  cairo_set_source_rgb(cr, 0.2, 0.4, 0);

  gint i;
  for ( i = 1; i <= 20; i++) {
      if (i > 20 - rect) {
          cairo_set_source_rgb(cr, 0.6, 1.0, 0);
      } else {
          cairo_set_source_rgb(cr, 0.2, 0.4, 0);
      }
      cairo_rectangle(cr, 8, i*4, 30, 3);
      cairo_rectangle(cr, 42, i*4, 30, 3);
      cairo_fill(cr);
  }

  cairo_destroy(cr);
}


static void
gtk_cpu_destroy(GtkObject *object)
{
  GtkCpu *cpu;
  GtkCpuClass *klass;

  g return if fail(object != NULL);
  g_return_if_fail(GTK_IS_CPU(object));

  cpu = GTK_CPU(object);

  klass = gtk_type_class(gtk_widget_get_type());

  if (GTK_OBJECT_CLASS(klass)->destroy) {
     (* GTK_OBJECT_CLASS(klass)->destroy) (object);
  }
}
```

```
/* main.c */

#include "cpu.h"


static void set_value(GtkWidget * widget, gpointer data)
{
  GdkRegion *region;

  GtkRange *range = (GtkRange *) widget;
  GtkWidget *cpu = (GtkWidget *) data;
  GTK_CPU(cpu)->sel = gtk_range_get_value(range);

  region = gdk_drawable_get_clip_region(cpu->window);
  gdk_window_invalidate_region(cpu->window, region, TRUE);
  gdk_window_process_updates(cpu->window, TRUE);
}


int main (int argc, char ** argv)
{
  GtkWidget *window;
  GtkWidget *cpu;
  GtkWidget *fixed;
  GtkWidget *scale;

  gtk_init(&argc, &argv);


  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "CPU widget");
  gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
  gtk_window_set_default_size(GTK_WINDOW(window), 200, 180);


  g_signal_connect(G_OBJECT(window), "destroy",
       G_CALLBACK(gtk_main_quit), NULL);

  fixed = gtk_fixed_new();
  gtk_container_add(GTK_CONTAINER(window), fixed);

  cpu = gtk_cpu_new();
  gtk_fixed_put(GTK_FIXED(fixed), cpu, 30, 40);


  scale = gtk_vscale_new_with_range(0.0, 100.0, 1.0);
  gtk_range_set_inverted(GTK_RANGE(scale), TRUE);
  gtk_scale_set_value_pos(GTK_SCALE(scale), GTK_POS_TOP);
  gtk_widget_set_size_request(scale, 50, 120);
  gtk fixed put(GTK FIXED(fixed), scale, 130, 20);

  g_signal_connect(G_OBJECT(scale), "value_changed",
G_CALLBACK(set_value), (gpointer) cpu);

  gtk_widget_show(cpu);
  gtk_widget_show(fixed);
  gtk_widget_show_all(window);
  gtk_main();

  return 0;
```

```
}
```

O Widget de CPU é um **GtkWidget**, sobre a qual chamamos de Cairo API. Chamamos a um fundo preto e 40 pequenos retângulos. Os retângulos são desenhados em duas cores. Cor verde escuro e verde brilhante. O Widget **GtkVScale** controla o número de retângulos verde brilhante desenhado no widget.

O exemplo pode parecer difícil à primeira vista. Mas não é assim tão difícil, afinal. A maioria do código é clichê, ele sempre repete, quando criamos um novo widget.

O desenho é feito dentro da função **gtk_cpu_paint()**.

```
  cairo_t *cr;

  cr = gdk_cairo_create(widget->window);

  cairo_translate(cr, 0, 7);

  cairo_set_source_rgb(cr, 0, 0, 0);
  cairo_paint(cr);
```

Como de costume, nós criamos um contexto Cairo. Mudamos a origem 7 unidades para baixo. Em seguida pintamos o fundo do widget de cor preta.

```
gint pos = GTK_CPU(widget)->sel;
gint rect = pos / 5;
```

Aqui vamos recuperar o número sel. É o número que temos a partir do widget escala. O slider tem 100 números. O parâmetro rect faz uma conversão de valores em retângulos deslizante, que será desenhado na cor verde brilhante.

```
gint i;
for ( i = 1; i <= 20; i++) {
    if (i > 20 - rect) {
        cairo_set_source_rgb(cr, 0.6, 1.0, 0);
    } else {
        cairo_set_source_rgb(cr, 0.2, 0.4, 0);
    }
    cairo_rectangle(cr, 8, i*4, 30, 3);
    cairo_rectangle(cr, 42, i*4, 30, 3);
    cairo_fill(cr);
}
```

Dependendo do número rect, chamamos 40 retângulos em duas cores verde escuro ou na cor verde brilhante. Lembre-se que estamos a desenhar esses retângulos de cima para baixo.

```
GtkRange *range = (GtkRange *) widget;
GtkWidget *cpu = (GtkWidget *) data;
GTK_CPU(cpu)->sel = gtk_range_get_value(range);
```

Na chamada **set_value()**, temos a referência para o widget do CPU e definir o valor sel ao valor atual, selecionada no widget escala.

```
GdkRegion *region;
...
region = gdk_drawable_get_clip_region(cpu->window);
gdk_window_invalidate_region(cpu->window, region, TRUE);
gdk_window_process_updates(cpu->window, TRUE);
```

Este código invalida a janela completa do widget da CPU e, portanto, redesenha ela mesma.

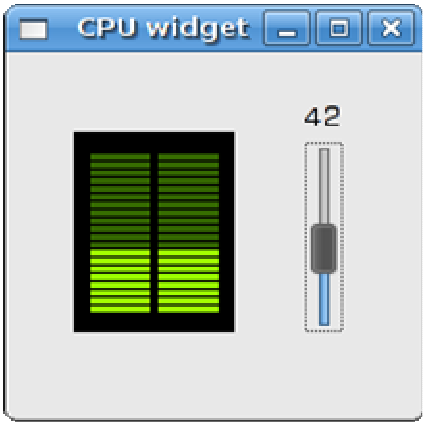


Figura: CPU widget