Notas Básicas sobre GTK+ 3
(Parte I)

Samuel Eleutério
sme@tecnico.ulisboa.pt
Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

Dezembro de 2016

Resumo

Procura-se com esta pequena nota pôr ao dispor dos alunos de Programação do Mestrado em Engenharia Física Tecnológica alguns exemplos e indicações úteis para a escrita de programas em GTK+ 3 em **C**.

Este texto foi elaborado em articulação com um conjunto de exemplos de software que se encontram disponíveis na página da cadeira. Aconselha-se, pois, que a sua leitura seja acompanhada pela visualização desses programas nomeadamente do seu código fonte e da sua execução.

Nesta primeira parte serão apenas tratadas *windows*, no contexto das *boxes*, e os objectos gráficos mais frequentes. As representações gráficas do tipo *cairo* serão integradas na segunda parte destas notas bem como outros objectos de uso mais específico como sejam *lists* ou *trees*.

Conteúdo

1 Introdução

O GTK+ (GIMP toolkit) é um *toolkit*[1] multiplataforma para a criação de interfaces gráficas do utilizador e está licenciado sob a licença LGPL. É, pois, um *software* livre que permite que *softwares* livres ou proprietários o usem na sua construção.

Inicialmente desenvolvido para sistemas X11, está actualmente disponível também para Microsoft Windows (Windows API) e Mac OS X (Quartz).

O GTK foi desenvolvido por Spencer Kimball e Peter Mattis como ferramenta auxiliar para o desenvolvimento do GIMP (General Image Manipulation Program). Passou a designar-se por GTK+ depois de reescrito com técnicas de programação orientada a objectos.

Presentemente o pacote GTK+ é constituído por diversas bibliotecas:

- **GLib**[5]: inicialmente encontrava-se integrada no GTK+ mas, a partir da versão 2, passou a biblioteca autónoma. Não contém funções da interface gráfica mas sim funções e macros de uso geral. Na prática funciona como uma biblioteca auxiliar de uso geral;
- **GObject**[6] (GLib Object System): até à versão GTK+ 2.0 esteve integrada no GTK+. Depende apenas da biblioteca de **C** e da **GLib**;
- **GDK**[8]: (GIMP Drawing Kit): esta biblioteca contém as funções de baixo-nível sobre as quais assentam as funções de GTK+;
- **GDK-PixBuf**[9]: é uma biblioteca de ferramentas para o carregamento de imagens e manipulação do *buffer* de pixeis associado;
- **GTK+**[4]: esta biblioteca contém as funções de GTK+ propriamente ditas;
- **Pango**[7]: é uma biblioteca orientada para tratamento de texto. Dispõe de suporte multilingue;
- **ATK**[10] (Accessibility Toolkit): é uma interface para o desenvolvimento de aplicações;
- **Cairo**[11]: é uma biblioteca de software gráfico bidimensional baseada em gráficos vetoriais.

[1] Também designado por *widget tootkit*, é, uma biblioteca (ou conjunto de bibliotecas, como é o caso do GTK+) que disponibiliza um conjunto de elementos gráficos de controle usados na construção da interface gráfica do utilizador (GUI, Graphical User Interface) de programas.

Em apêndice existe uma lista das funções referidas ao longo do texto em que se faz uma descrição sumária da sua utilização pelo que se dispensa ao longo do texto repetir essas indicações.

A leitura deste texto deve ser acompanhada pela consulta e eventual execução dos programas referidos em cada ponto os quais ajudam a evidenciar os diversos aspectos da sua utilização.

2 Primeiros passos

O GTK+ é uma biblioteca de criação de interfaces gráficas para aplicações e, como a maioria dos sistemas deste tipo, dispõe da facilidade de poder ficar à espera das instruções do utilizador. Uma vez recebidas essas instruções a aplicação executa-as e fica novamente à espera de novas indicações. Essa espera e execução é realizada através de um ciclo interno que, em geral, só é quebrado no fim do programa. Um exemplo típicos destes sistemas é um *browser* de acesso à internet.

Essas instruções são executadas pelo programa através da execução de funções (funções de '*callback*') que se encontram associadas a objectos (widgets[2]) específicos existentes na aplicação (por exemplo, botões) e às acções particulares exercidas sobre eles (por exemplo, carregar no botão).

De um modo análogo, quando se inicia um programa, as instruções de criação dos objectos são executadas mas a sua realização efectiva só se dá depois de o programa entrar para o referido ciclo de espera.

Os nomes funções de GTK+ iniciam-se sempre por "gtk_" ao que se segue o nome do objecto em que actuam e finalmento o tipo de acção sobre esse objecto. Assim, se se quiser criar (new) uma *window* (janela) a função será:

```
gtk_window_new
```

depois seguem-se os seus respectivos argumentos.

Quando se escreve em programa em **C** utilizando as bibliotecas associadas ao GTK+ deve fazer-se uma prévia inicialização. Tal é feito com a função '*gtk_init*' que recebe como argumentos os ponteiros para as variáveis associadas aos dois primeiros argumentos da função *main*, a saber, o número de argumentos ('int argc') e os argumentos ('char **argv') associados à chamada do programa. Assim, a primeira instrução deverá ser:

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

[2]Ao longo do texto será usada a palavra *widget* para designar os elementos gráficos de controle, ao quais se associa, em **C**, o tipo 'GtkWidget'

Uma vez criado um objecto (widget) ele não fica visível. Para o tornar visível é preciso dar uma instrução explícita:

gtk_widget_show (widget);

é igualmente possível tornar visíveis todos o objectos contidos num dado objecto, por exemplo, se se cria uma *window* com objectos no seu interior, basta dar a instrução

gtk_widget_show_all (widget);

para tornar visíveis todos o objectos nela contidos.

Apesar do programa executar as instruções de criação dos objectos que nele se deseja colocar, a criação propriamente dita desses objectos só vai ser feita numa fase posterior em que o programa entra no ciclo de espera. Para entrar nesse ciclo deve executar-se a função

gtk_main ();

É igualmente possível quebrar este ciclo usando a função:

gtk_main_quit ();

Note-se que se se executar duas vezes a função 'gtk_main', para terminar o programa, dever-se-á igualmente executar duas vezes a função 'gtk_main_quit'. Assim, esta função comporta-se de um modo análogo à instrução 'break' em ciclos.

Os objectos que se colocam numa *window* são, em geral, ponteiros para o tipo 'GtkWidget'. No entanto, poderão ser de factos de outros tipos que herdaram as características da estrutura 'GtkWidget'. Ver-se-á isso, na utilização dos objectos depois da sua criação.

3 Funções de '*callback*'

Apesar de não ser usual começar uma apresentação de GTK+ pelas funções de *callback*, é útil e prático, referi-las desde o início. De facto, elas irão aparecer desde o primeiro programa que se irá apresentar.

Como se disse atrás, durante a execução dum programa o utilizador pode actuar de determinadas maneiras sobre os *widgets* com vista a obter determinados resultados. A realização prática deste mecanismo é feita associando a chamada de uma função (*callback*) à actuação sobre um determinado objecto produzindo um evento o qual emite um sinal. Para tal pode usar-se a macro definida na biblioteca *GObject*:

g_signal_connect (instance, detailed_signal, c_handler, data)

em que '*instance*' é o ponteiro para o objecto sobre o qual se actua, *detailed_signal*

é uma *string* com o nome do sinal (ver Apêndice *Events* e *Signals*), '*c_handler*' é a função a ser chamada (do tipo *GCallback*) e '*data*' é um ponteiro ao dispor do programador a enviar para a função (note-se que apenas se pode enviar um único ponteiro). O retorno desta macro é um número maior do que '0' se a chamada tiver sucesso e corresponde *id* associado a esta chamada.

Ao definir-se a função deve ter-se em conta que os argumentos que ela deve ter são definidos pelo GTK+, assim, no primeiro argumento é o ponteiro para o objecto sobre o qual se actuou (*instance*) e o último argumento é o ponteiro enviado em '*data*'. Poderão existir eventualmente argumentos intermédios de acordo com o tipo de objecto e o sinal em questão.

4 *Widgets*

O tipo *GtkWidget* é o tipo básico dos objectos gráficos de GTK+. Este tipo vai servir de base à criação de outros tipos. A declaração dos ponteiros para a grande maioria dos objectos gráficos são ponteiros para o tipo *GtkWidget*. De facto, essos objectos têm como primeiro membro um *GtkWidget* a que se seguem os seus restantes membros (herança).

Associadas a este tipo básico existe um conjunto de propriedades que os restentes objectos dele derivados também possuem (herança).

Nesta secção serão descritas algumas dessas propriedades bem como as funções que nos permitem fazer o seu controle. No entanto, numa primeira leitura, a parte que se segue desta secção pode ser ignorada e apenas consultada quando alguma dessas propriedades for referida. Assim, nas referências ao programas que as usam não será seguido um critério de complexidade crescente como no restante texto mas simplesmente indicados programas em que elas são utilizadas.

Quando uma *widget* é criada em GTK+ mantém-se invisível até que explicitamente seja dada uma ordem em contrário. Para tornar visível uma *widget* usa-se a função '*gtk_widget_show*'. É ainda possível dar indicação para o fazer não só para o objecto em causa mas também para todos os outros nele contidos. Tal é feito frequentemente quando se deseja tornar visíveis todas as *widgets* contidas numa *window*. A função a usar neste caso é '*gtk_widget_show_all*'. Inversamente, pode tornar-se invisível uma *widget* com a função '*gtk_widget_hide*'. Note-se que ao tornar invisível um objecto tudo o que se encontra no interior também deixa de ser visível. Ver programa 'Gtk3_04_01.c';

Às *widgets* criadas é atribuído um certo espaço que tem a ver com diversos factores nomeadamente o seu conteúdo, o conteúdo das *widgets* que a rodeiam e as propriedades de distribuição se espaço. Por isso, é por vezes conveniente poder garantir um tamanho mínimo aos objectos. Tal pode ser feito com a

função '*gtk_widget_set_size_request*' que associa a uma *widget* um comprimento e uma altura mínimos. Caso não se queria fazer essa imposição a alguma dessas direcções deve indicar-se o valor -1". É ainda necessário ter em conta que o ajuste em ambas as direcções só é possível desde que haja possibilidade de o fazer, isto é, se que quer, por exemplo, controlar a altura de uma *widget* ela não pode estar apenas numa *box* horizontal, é preciso também que a algum nível acima exista uma *box* vertical. Para se obterem os valores impostos por esta função, usa-se '*gtk_widget_get_size_request*'. Ver programas 'Gtk3_01_03.c' e 'Gtk3_02_03.c'.

O espaço atribuído a uma *widget* varia por diversas razões mesmo quando se lhe atribuí um certo espaço. Basta ter presente que a alteração de tamanho duma *window* leva usualmente ao rearranjo dos objectos nessa *window*. É assim útil poder saber o tamanho e a posição de um objecto. Para tal pode usar-se a função '*gtk_widget_get_allocation*' que preenche um objecto do tipo 'GtkAllocation' que contém a informação referente à sua posição e tamanho. Noutros casos, deseja-se apenas saber o seu comprimento ou altura, então, é mais simples usar as funções que retornam esses valores. Elas são respectivamente '*gtk_widget_get_allocated_height*' e '*gtk_widget_get_allocated_width*'. Ver programa 'Gtk3_30_02.c'.

Em certos casos é desejável bloquear certos objectos a fim de os impedir de realizem as tarefas que lhes estão atribuídas. Isso pode ser feito de diversas formas de acordo com o que se deseja fazer ou mostrar. Situações deste tipo acontecem ao ser bloqueiado, num editor, o botão de guardar um ficheiro enquanto o ficheiro ainda não foi alterado. Outra situação possível dá-se quando o controle do crescimento ou decrescimento de uma variável numérica é feito por *buttons* e já de atingiu um dos seus valores limites. Então, para bloquear ou desbloquear uma dada *widget* pode usar-se a função '*gtk_widget_set_sensitive*' atribuindo à propriedade '*sensitive*' os valores TRUE ou FALSE. Para saber se uma *widget* está ou não bloqueada usa-se a função '*gtk_widget_get_sensitive*'. Ver programa 'Gtk3_02_05.c'.

A utilização de *Cascading Style Sheets* (CSS) para associar atributos a uma *widget* pode ser feito ligando-lhe o nome dum conjunto de propriedades CSS. A função que atribui esse nome à *widget* é '*gtk_widget_set_name*'. Ver programas 'Gtk3_09_03e4.c'.

Da mesma maneira como são criadas, as *widgets* também podem ser destruídas. Para o fazer usa-se a função *gtk_widget_destroy*.

5 *Windows*

A função que permite criar uma *window* (janela) é '*gtk_window_new*'. Ela recebe como argumento um membro do enumerado *GtkWindowType* que especifica o tipo de *window* a criar. Existem apenas dois tipo: '*GTK_WINDOW_TOPLEVEL*' a que corresponde uma *window* com as respectivas decorações e '*GTK_WINDOW_POPUP*' a que corresponde uma *window* sem decorações. Esta função retorna um ponteiro do tipo *GtkWidget*. Assim, para se criar uma *window* do tipo *toplevel*:

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

Uma vez criada a *window*, pode atribuir-se-lhe determinadas propriedades, por exemplo, um tamanho inicial, um título ou um posicionamento no ecran.

Para poder terminar uma aplicação usando o botão de fecho contido nas decorações da *window*, tem de se quebrar o ciclo em que o programa se encontra o que é feito, como se viu, com a função '*gtk_main_quit*'. Há pois que associar a chamada desta função à acção "*destroy*" da *window*. Esta associação é, como se viu anteriormente, feita com a função '*g_signal_connect*':

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

Finalmente, resta dar a indicação de mostrar a *window* e entrar no ciclo em que a *window* é, de facto, criada e fica à espera das indicações do utilizador:

```
gtk_widget_show (window);
gtk_main ();
```

Ver programa 'Gtk3_01_01.c'.

O programa 'Gtk3_01_02.c' é idêntico ao anterior, no entanto, pode receber como argumentos as alterações ao tipo de *window*, posicionamento no ecran, título e tamanho. Para ver as opções disponíveis fazer: 'Gtk3_01_02 -h'.

Quando se cria uma *window* pode impor-se-lhe um tamanho mínimo, tal pode ser feito com a função '*gtk_widget_set_size_request*'. Ver programa 'Gtk3_01_03.c'.

É ainda possível forçar a *window* a ter um seu tamanho fixo. Isso é feito com a função '*gtk_window_set_resizable*' que permitir ou não que o utilizador possa alterar o seu tamanho. Por defeito, uma *window* ao ser criada permite que o seu tamanho seja alterado. Ver programa 'Gtk3_01_04.c'.

6 *Boxes*

Uma *window* em GTK+ apenas pode conter um objecto no seu interior, por isso, é necessário colocar nela um objecto que possa depois conter outros objectos no

seu interior. Existem em GTK+ diversos tipo de contentores '*containers*' capazes de receber no seu interior mais do que uma *widget* no seu interior. Vai-se aqui começar pelo mais simples e flexível que é a *box* ('caixa').

Um *container* (tipo *GtkContainer*) é uma *widget* na qual se podem colocar outras *widgets*. Quando se coloca uma *widget* num *container* que só pode receber um objecto (exemplo, *windows*, *frames*, etc.) deve usar-se a função '*gtk_container_add*'. Esta função tem dois argumentos: o primeiro é o '*container*' e o segundo a *widget* nele colocado.

As *boxes* são áreas rectangulares onde as *widgets* nelas colocados se dispõem de um modo uni-direccional (horizontal ou verticalmente) de acordo com as características atribuídas à *box* na sua definição. Para criar uma *box* usa-se a função '*gtk_box_new*' em que primeiro argumento é a orientação da caixa (horizontal ou vertical) e o segundo argumento é o número de pixels que ficam entre os objectos nela contidos.

Quanto à distribuição dos objectos nas *boxes* elas poder ser homogéneas ou não homogéneas. No primeiro caso, o espaço é igualmente distribuído pelas diferentes *widgets* nela colocadas; no segundo, são-no de acordo com o espaço atribuído. Estas propriedades são definidas pela função '*gtk_box_set_homogeneous*' cujo primeiro argumento é a *box* em questão e o segundo é do tipo booleano. Por defeito, a *box* é criada a FALSE.

Para empacotar as *widgets* nelas contidas pode optar-se por acrescentar do princípio para fim ou do fim para o princípio. Existem duas funções para isso: 'gtk_box_pack_start' e 'gtk_box_pack_end'. Os dois primeiros argumentos destas função são respectivamente a *box* em questão e a *widget* a colocar nela. O terceiro argumento é de tipo boolean e no caso de ser TRUE a *widget* tende a estender-se pelo espaço disponível; se for FALSE ocupa apenas o espaço de que necessita. O quarto argumento apenas se aplica se o terceiro for TRUE, é igualmente booleano e ser for TRUE faz com que a *widget* se estenda por todo o espaço que lhe é dado; se for FALSE, apesar dispor do espaço não se alarga para além do que necessita. Finalmente, o quinto argumento é o número de pixels que são colocados entre a *widget* e as *widgets* à volta para além do espaçamento previamente existente.

Uma vez que as *boxes* são uma classe de *container*, as funções deste também podem ser usadas com elas. Assim, a função '*gtk_container_add*', atrás referida, também pode ser usada com elas.

O programa 'ShowGtkBox' permite simular a criação de *boxes* numa *window* e mostrar as instruções que se dão na sua criação.

7 *Buttons*

Um dos objectos mais usados em qualquer programa de interface gráfica é sem dúvida o *button* (botão). Apesar de haver vários tipos de *buttons* neste caso vai tratar-se do caso mais simples. Um *button* pode ser criado sem argumentos ou com um *label* e, neste caso, terá a função que o cria de receberá uma *string*. As funções em causa são '*gtk_button_new*' e '*gtk_button_new_with_label*'.

Nos programas 'Gtk3_02_01a4.c' mostra-se como se criam botões e como se lhes associam funções de *callback* quando se carrega neles, sinal "clicked". Mostra-se ainda como se pode colocar *buttons* dentro de *boxes* para se obterem os efeitos desejados. No programa 'Gtk3_02_04.c' é ainda utilizado um novo tipo de objecto, o '*label*', ver secção seguinte.

É ainda possível fazer o bloqueio, ou debloqueio, de um *button*, para tal usa-se a função '*gtk_widget_set_sensitive*'. Ver programa 'Gtk3_02_05.c'.

8 *Labels*

Um *label* é um objecto que contém simplesmente um certo texto. A função que cria um *label* é '*gtk_label_new*' que tem como único argumento o texto a ser apresentado. Uma vez criado um *label* é possível alterar o seu conteúdo usando a função '*gtk_label_set_text*' que tem dois argumentos, o *label* e uma *string* que irá ser o novo texto a ser mostrado.

No programa 'Gtk3_02_04.c', o *label* é usado para mostrar um valor numérico que se vai alterando de acordo com o *button* em que se carrega. A sua actualização é feita nas funções de '*callback*' associadas aos *buttons*.

9 *Menus*

A criação de uma *menu bar* é algo mais complexa do que os objectos que se mostraram até aqui. Ela exige a criação de cada um dos seu componentes. A sua criação pode ser sistematizada do seguinte modo:

- Cria-se uma *menu bar*, para tal usa-se a função '*gtk_menu_bar_new*';
- Cria-se um *menu item* com a função '*gtk_menu_item_new_with_label*' que recebe como argumento uma *string* contendo o texto a apresentar. Em seguida coloca-se o *menu item* agora criado na *menu bar* criada no item anterior com a função '*gtk_menu_shell_append*' que recebe como primeiro argumento a *menu bar* e como segundo o *menu item* agora criado;

- Cria-se agora um *menu* com a função sem argumentos '*gtk_menu_new*' que por sua vez é metido no *menu item* anteriormente criado usando a função '*gtk_menu_item_set_submenu*' cujo primeiro argumento é o *menu item* e o segundo o *menu* agora criado;

- Criam-se finalmente tantos *menu item* quantos os que se desejam usando novamente a função '*gtk_menu_item_new_with_label*' e colocam-se no *menu* anterior usando outra vez a função '*gtk_menu_shell_append*'.

Se se quiser construir mais *menus* repetem-se os dois últimos passos. Ver programas 'Gtk3_03_01a6.c'.

No programa 'Gtk3_03_02.c' mostra-se como se pode associar uma letra à abertura dum *menu* directamente do teclado, esta associação é feita com a função '*gtk_menu_item_new_with_mnemonic*' em que a *string* tem uma barra inferior antes da letra. A chamada é feita usando a tecla 'Alt' e a letra respectiva.

No programa 'Gtk3_03_03.c' mostra-se como se pode executar um item directamente usando o teclado (utilização de *accel_group*). Neste exemplo, usa-se para saída do programa CTRL Q associado a 'Quit' no *menu* 'File'. Para tal é necessário criar um '*accel_group*' com a função sem argumentos '*gtk_accel_group_new*', o tipo associado a este é 'GtkAccelGroup', em seguida adiciona-se-lo à *window* usando a função '*gtk_window_add_accel_group*' e, finalmente, deve usar-se a função '*gtk_widget_add_accelerator*' para associar o CTRL Q ao *menu item* respectivo.

No programa 'Gtk3_03_04.c' acrescenta-se aos items dos *menus* uns pequenos icons. Tal é feito criando uma *box* que se introduz no *menu_item* e nela se colocam o *label* e o icon[2] associado ao tipo de acção desejada. O icon pode ser obtido a partir da função '*gtk_image_new_from_icon_name*'.

Os programas 'Gtk3_03_05e6.c' integram os menus nos programas anteriores.

10 *Statusbar*

Uma '*statusbar*' é uma barra para a qual se podem enviar diversas mensagens. É criada usando a função sem argumentos '*gtk_statusbar_new*'. A '*statusbar*' permite ter uma pilha de mensagens e fazer a sua gestão. A função '*gtk_statusbar_push*' permite adicionar uma mensagem à pilha. Inversamente a função '*gtk_statusbar_pop*' permite remover a mensagem que se encontra no cimo da pilha.

Os programas 'Gtk3_04_01e2.c' mostram uma utilização elementar deste objecto.

11 *Toolbar*

Uma *toolbar* é uma barra na qual se podem colocar *items* aos quais se associam tarefas específicas.

A criação de uma *toolbar* faz-se usando a função '*gtk_toolbar_new*' sem argumentos. Por defeito, a *toolbar* tem uma direcção horizontal e permite 'icons' e texto (ver programa 'Gtk3_05_01.c').

Com a função '*gtk_toolbar_set_style*' pode definir-se se se permite que a '*toolbar*' possa integral icons ou apenas texto (ver programa 'Gtk3_05_02.c').

Se o tamanho da *window* não permite que os items caibam todos pode criada uma seta e um menu a ela associado em que se incluem os item que não cabem. A função que controla esta opção é '*gtk_toolbar_set_show_arrow*'.

Para alterar a direcção da '*toolbar*' usa-se o facto de que ela ser orientável. A função que altera a orientação é '*gtk_orientable_set_orientation* que recebe como argumentos o objecto a orientar e a direcção desejada. No programa 'Gtk3_05_03.c' mostra-se como se pode criar uma '*toolbar*' vertical. No programa 'Gtk3_05_04.c' mostra-se como, em tempo real, se pode trocar a orientação.

Os items que se podem colocar numa *toolbar* podem ter um texto e um icon associados. A função que os cria é '*gtk_tool_button_new*' que recebe dois argumentos, o primeiro é o icon ('widget') que, caso não se use deve ser posto a 'NULL' e o segundo argumento é o texto associado. Para além dos icons criados pelo utilizador, existe um conjunto de icons pre-definidos que se podem associar ao botão com a função '*gtk_tool_button_set_icon_name*'. A lista dos nomes destes botões pode ser encontrada no site *Icon Naming Specification*[2].

Ver programas 'Gtk3_05_01a5.c'.

12 *Buttons* com imagens

A função mais básica que criar uma *image* é *gtk_image_new*'. A *widget* associada à imagem pode ser obtida também pelas funções: '*gtk_image_new_from_file*' que a cria partir de um ficheiro ou por '*gtk_image_new_from_icon_name*' que a cria a partir de *icon*[2].

Ver programas 'Gtk3_06_01a4.c'.

13 *Entry*

A criação de uma *entry* pode ser feita com a função '*gtk_entry_new*'. É ainda possível ao criar uma *entry* associando-lhe um *buffer*.

Existe ao dispor do programador um elevado número de propriedades associadas a uma *entry*. Mostram-se aqui apenas as mais úteis e frequentes:

- **gtk_entry_set_max_length**: define o número máximo de caracteres que se pode escrever numa *entry*;
- **gtk_entry_set_width_chars**: define o tamanho duma *entry* em número de caracteres;
- **gtk_entry_set_max_width_chars**: estabelece um limite para o tamanho duma *entry* em número caracteres;
- **gtk_entry_set_visibility**: permite optar por tornar ou não visível o texto numa *entry*;
- **gtk_entry_get_visibility**: Retorna TRUE ou FALSE de acordo com o texto estar visível ou não. Por defeito é TRUE;
- **gtk_entry_set_icon_from_icon_name**: permite colocar icons nas posições iniciais e finais duma *entry*;
- **gtk_entry_set_has_frame**: põe ou retira a esquadria à volta duma *entry*. Por defeito é TRUE;
- **gtk_entry_get_has_frame**: Retorna TRUE ou FALSE de acordo com existir não de esquadria;

Ver programas 'Gtk3_07_01a3.c'.

Recorde-se ainda que é possível permitir ou não a escrita numa *entry* usando o atributo *sensitive* associado às *widgets*. Assim, se se desejar, pode usar-se a função '*gtk_widget_set_sensitive*'.

14 *Adjustment*

Um *adjustment* ('*GtkAdjustment*') é um objecto que tem associados um valor, os seus limites superior e inferior e ainda os seus incrementos e decrementos passo a passo e página a página. É usado em algumas *widgets* de GTK+ por exemplo no *SpinButton*. O controle dos valores é feito pelos objectos em que ele se integra.

Para criar um '*adjustment*' usa-se a função '*gtk_adjustment_new*' cujos argumentos são

- **value**: valor actual;

- **lower**: limite inferior que o valor pode ter;
- **upper**: limite superior que o valor pode ter;
- **step_increment**: incremento de cada passo;
- **page_increment**: incremento de cada página;
- **page_size**: tamanho de cada página;

Associado a este objecto existe um conjunto de funções que basicamente permitem ler e alterar os seus valores.

Exemplos deste objecto podem encontrar na secção *spin button*.

15 *Spin button*

Um *spin button* permite a entrada de dados e é constituído por uma linha de entrada de lados e dois botões que permitem incrementar de decrementar o seu valor (por herança recebe as propriedades das *entries*). Apenas de poder permitir a escrita de caracteres não numéricos, este objecto está especialmente orientado para a leitura de valores numéricos. Aliás, a sua leitura retorna uma variável do tipo 'gdouble'. Para criar um '*spin button*' usa-se a função '*gtk_spin_button_new*' cujo primeiro argumento é o *adjustment* que contém as características da entrada, o segundo argumento permite indicar o incremento/decremento quando se carrega nos botões e o terceiro é o número de casas decimais a serem mostradas. Pode ainda criar-se um '*spin button*' sem recorrer a um *adjustment* usando a função '*gtk_spin_button_new_with_range*'.

O incremento (ou decremento) depende do modo como se actua no rato, assim, um *click* na tecla esquerda do rato faz o incremento (ou decremento) passo a passo, um *click* na tecla centrar faz o incremento (ou decremento) página a página e um *click* na tecla direita faz o valor saltar para o limite superior (ou inferior).

Para a leitura e escrita de valores no '*spin button*' usa-se respectivamente as funções '*gtk_spin_button_get_value*' e '*gtk_spin_button_set_value*'. O valor é do tipo '*gdouble*'.

Para restringir a aceitação de caracteres apenas a valores numéricos usa-se a função '*gtk_spin_button_set_numeric*'. Por defeito o seu valor é FALSE. Para alterar o número de casas decimais a usa-se '*gtk_spin_button_set_digits*'.

Associado a esta *widget*, existe um conjunto de funções que permitem alterar os valores associados aos limites e e incrementos. É igualmente possível obter o *adjustment* usado e mudá-lo para outro.

No programa 'Gtk3_08_01.c' é criado um *spin button* e usa-se um *button* com uma função de *callback* associada que faz a sua leitura e mostra o seu valor num *label*. Em 'Gtk3_08_02.c' mostra-se, em paralelo, o funcionamento de uma *entry* e de um *spin button* sendo a sua utilização idêntica à do problema anterior. No programa 'Gtk3_08_03.c' associou-se a chamada de uma função à alteração do valor do *spin button* (o que corresponde ao desencadear do o sinal "*changed*"). Assim, o *label* em que se escreve o valor é imediatamente actualizado. Existe ainda um botão que permite alterar o número de casas decimais mostradas.

16 *Toggle button*

Um *toggle button* é um botão que guarda o seu estado de carregado. Quando se carrega nele, alternamente fica ou não carregado, de resto, comporta-se como um botão simples.

A criação dum *toggle button* pode ser feita com uma das seguintes funções: '*gtk_toggle_button_new*', '*gtk_toggle_button_new_with_label*' colocando um *label* ou '*gtk_toggle_button_new_with_mnemonic*' associando uma mnemónica. Recorde-se que a criação duma mnemónica é feita precedendo dum traço em baixo, "_", o caracter desejado do *label*.

Um botão *toggle button* pode ser activado ou desactivado com a função '*gtk_toggle_button_set_mode*' de um modo idêntico pode ser lido o seu estado com a função '*gtk_toggle_button_get_mode*'.

Ver programa 'Gtk3_12_01.c'.

17 *Check button*

Um *check button* é um *toggle button* usualmente representado por um quadrado em que se coloca um "visto" e o seu *label* é colocado ao lado. Para a sua criação usam-se funções idênticas às que se usaram para o *toggle button* em que se substitui no nome *toggle* por *check*. Para a alteração do seu estado usam-se as funções definidas para o *toggle button*.

Ver programa 'Gtk3_13_01.c'.

18 *Radio button*

Um *radio button* é um *check button* que permite ligar *buttons* entre si de tal maneira que quando se liga um outro se desligar de modo que haja apenas um *button* ligado de cada vez.

Uma vez que os *buttons* criados vão estar ligados entre si, há necessidade de os relacionar. Para o fazer começa-se por criar *radio button* de um modo análogo ao que se faz para os outros *buttons*. Também aqui se pode criar simplesmento um *radio button* ou fazê-lo acreacentando um 'label' ou usando uma mnemónica. Se se usual *labels* então a função usada será '*gtk_radio_button_new_with_label*'. Para criar os restantes *radio buttons* relaciona-se o novo *button* com um dos anteriores com a função '*gtk_radio_button_new_with_label_from_widget*'.

Para activar ou desactivar um *button* usam-se as funções dos *toggle buttons*.

Ver programa 'Gtk3_14_01.c'.

19 CSS (Cascading Style Sheets)

A partir da versão 3.18 do GTK+ as atribuições de cores, tipos de letra e muitos outros atributos das *widgets* passaram a dever serem feitas usando o mecanismo dos CSS (Cascading Style Sheets).

Para integrar as entradas tipo CSS no estilo das *widgets* cria-se um objecto do tipo '*GtkCssProvider*' e faz-se a sua associação ao *screen* do *display* do programa. Para tal:

```
GtkCssProvider *provider = gtk_css_provider_new ();
GdkDisplay *display = gdk_display_get_default ();
GdkScreen *screen = gdk_display_get_default_screen (display);
gtk_style_context_add_provider_for_screen (screen,
    GTK_STYLE_PROVIDER(provider),
    GTK_STYLE_PROVIDER_PRIORITY_USER);
```

Seguidamente leem-se os dados CSS propriamente ditos. Tal leitura pode ser partir de um ficheiro ou a partir de uma string que os contenha.

Para fazer a leitura a partir dum ficheiro usa-se uma das funções de leitura a partir de ficheiros, por exemplo, '*gtk_css_provider_load_from_path*'.

No caso de se introduzirem os dados CSS a partir de uma *string*, a função a usar será '*gtk_css_provider_load_from_data*'.

Finalmente, deverá libertar-se o *provider* criado usando a função '*g_object_unref* e depois limpando a variável com '*g_clear_object*'.

As instruções e exemplos da utilização dos CSS encontram-se em 'Apêndice CSS'. Os programas 'Gtk3_09_01a15.c', fazem uma primeira introdução à utilização deste mecanismo.

20 *Combo boxes*

Uma *combo_box* é um objecto que permite ao utilizador escolher entre uma lista de opções previamente definidas. Ela mostra a opção selecionada. Quando é activada mostra uma *window* do tipo *pop-up* com as opções válidas a fim de permitir ao utilizador fazer a escolha que deseja.

A *combo box* mais simples de usar é a *combo box text* em que as opções de escolha são do tipo texto.

Para criar uma *combo box text* usa-se a função '*gtk_combo_box_text_new*' sem argumentos. Para introduzir as opções acrescenta-se um a um os respectivos textos. Há várias funções para executar esta tarefa e têm basicamente a ver com a posição em que se insere cada opção. A título de referência indicam-se aqui duas dessas funções '*gtk_combo_box_text_insert_text*' que permite inserir o texto numa dada posição e '*gtk_combo_box_text_append_text*' que o insere no final.

Para fazer a escolha duma opção usa-se '*gtk_combo_box_set_active*' indicando qual a posição dessa opção na lista. Para a ler usa-se '*gtk_combo_box_get_active*' que retorna a posição escolhida.

No programa 'Gtk3_10_01.c' cria-se uma *combo box text* com os meses do ano, indica-se inicialmente qual escolhida e associa-se a chamada de uma função à alteração da escolha. A nova escolha é mostrada usando um *label*. No programa 'Gtk3_10_02.c' altera-se a cor e a fonte da *combo box text*.

Um outro processo de criar uma *combo box* é usando um *model* (modelo). Para tal recorre-se ao objecto *list store* (ver funções associadas). No programa 'Gtk3_10_03.c' exemplifica-se a sua utilização.

Note-se ainda que a alteração da entrada activada pode ser feita usando a roda do terceiro botão do rato.

Nos programa 'Gtk3_10_04e5.c'. Alteram-se as cores da *combo box*.

21 *Scales*

Uma maneira prática de introduzir valores desde que se não exiga uma grande precisão é através de uma *scale* (cursor). As *scales* podem funcionar horizontal ou verticalmente e para a caracterização do seu valor pode usar-se um '*adjustment*' ou dar explicitamente os seus limites e o seu passo. Quando se associa um *adjustment* à *scale*, a função a utilizar é '*gtk_scale_new*' e quando se dão os limites é '*gtk_scale_new_with_range*'. A *scale* (*GtkScale*) é um objecto que herdar as propriedades do objecto '*range*' (*GtkRange*), assim, para a atribuição e leitura dos seus valores usam-se as funções '*gtk_range_set_value*' e '*gtk_range_get_value*'

respectivamente.

A apresentação do valor actual duma *scale* pode ser controlado de diversos modos. O número de casas decimais a apresentar é controlado pela função '*gtk_scale_set_digits*'; a posição em que o valor actual é mostrado em relação à *scale* é fixado por '*gtk_scale_set_value_pos*' sendo os valores possíveis GTK_POS_LEFT, GTK_POS_RIGHT, GTK_POS_TOP e GTK_POS_BOTTOM; é ainda possível mostrar ou não o seu valor actual, para isso, usa-se a função '*gtk_scale_set_draw_value*' com o valor TRUE ou FALSE.

No programa 'Gtk3_11_01.c' mostra-se como se pode criar uma *scale* e alterar a posição e o número de casas decimais. É ainda possível mostrar ou não o seu valor actual. Note-se que se pode alterar o *label* de um botão com a função '*gtk_button_set_label*'. Mostra-se mais uma vez como se pode chamar uma função sempre que o valor da *scale* é alterado.

No programa 'Gtk3_11_02.c' acrescentam-se cores ao programa 'Gtk3_11_01.c'.

22 *Frames*

O objecto '*frame*' (*GtkFrame*) é *container* usualmente integrado no grupo dos objectos decorativos que mostra uma cercadura à volta do seu conteúdo e pode apresentar um título. De um modo análogo à *window* só pode receber um único objecto no seu interior, usualmente, uma *box* que por sua vez poderá ter outros objectos no seu interior.

A função que cria um *frame* é '*gtk_frame_new*' a qual recebe ainda o seu título. Caso não se deseje pôr um título no *frame* esse argumento deve ser posto a NULL. No entanto, é sempre possível a qualquer momento atribuir ou alterar o título usando a função '*gtk_frame_set_label*'; inversamente se se desejar saber qual o título de um *frame* pode usar-se a função '*gtk_frame_get_label*'. É ainda possível no título dum frame colocar-se uma *widget*, para isso usa-se a função '*gtk_frame_set_label_widget*'.

Para ajustar o alinhamento do título usa-se a função '*gtk_frame_set_label_align*'. Os ajustes horizontais e verticais são feitos por valores reais ('gfloat') no intervalo [0.,1.]. Segundo a horizontal, '0.' corresponde o alinhamento à esquerda, '1.' à direita e '0.5' a centrar. O comportamento na vertical é análogo correspondendo '1.' à posição mais alta, '0.' à mais baixa e '0.5' a centrar. Para obter estes valores do alinhamento do título usa-se a função '*gtk_frame_get_label_align*'. Os valores, por defeito, são respectivamente '0.' e '0.5';

Ver programas 'Gtk3_15_01a5.c'. No programa 'Gtk3_15_04.c' altera-se o programa 'Gtk3_15_02.c' para incluir cores.

23 *Images* a partir de *pixbufs*

No ponto referente aos *menus* já se referiu a criação de *images* a partir de *icons*. Do mesmo modo já se referiu que se podem criar *images* para serem usadas em *buttons*.

A função '*gdk_pixbuf_new_from_file*' permite criar um *pixbuf* a partir de uma *image* contida num ficheiro. É igualmente possível obter o tamanho de uma *image* contida num ficheiro usando a função '*gdk_pixbuf_get_file_info*'. Uma ver criado um *pixbuf* pode criar-se uma *image* a partir dele. Uma vez criada a *image*, pode libertar-se o *pixbuf* usando a função *g_object_unref*:

```
GdkPixbuf *pixbuf = gdk_pixbuf_new_from_file (fileName, NULL);
GtkWidget *image = gtk_image_new_from_pixbuf (pixbuf);
g_object_unref (pixbuf);
```

No programa 'Gtk3_18_01.c' mostra-se como se cria a *image* a partir de um *pixbuf* e depois se coloca num *button*.

Pode igualmente criar-se uma *image* a partir duma *string*. Neste caso a criação do *pixbuf* exige várias etapes: é preciso criar um *pixbuf loader*, depois é preciso escrever os dados no *pixbuf loader*, em seguida, obtém-se um *pixbuf* a partir do *pixbuf loader*, depois aumenta-se o contador do *pixbuf* e finalmente fecha o *pixbuf loader*. Vejamos então como se processam estes passos:

```
GdkPixbufLoader *loader = gdk_pixbuf_loader_new ();
gdk_pixbuf_loader_write (loader, (guchar *) data, len, NULL);
GdkPixbuf *pixbuf = gdk_pixbuf_loader_get_pixbuf (loader);
g_object_ref (pixbuf);
gdk_pixbuf_loader_close (loader, &error);
GtkWidget *image = gtk_image_new_from_pixbuf (pixbuf);
g_object_unref (pixbuf);
```

O programa 'Gtk3_18_02.c' é idêntido a 'Gtk3_18_01.c' mas aqui os ficheiros contendo as *images* são previamente lidos para uma *string*.

24 Tamanho e posições das *widget*

Apesar de já ter sido anteriormente referido quando se falou de *widgets* a primeira vez, vale a pena analisar com um pouco mais de detalhe como se pode obter a sua posição e o seu tamanho duma *widget* ou de uma *window*.

Para se obter o tamanho duma *window* usa-se o função '*gtk_window_get_size*'.

As coordenadas do canto superior esquerdo duma *window* no ecran são obtidas com a função '*gtk_window_get_position*'. Note-se, no entanto, que tal só acontece caso não tenha sido alterada a '*gravity*' inicial da *window* (Norte-Oeste). Se a '*gravity*' tiver sido alterada os valores obtidos correspondem ao dessa nova situação.

Se se deseja saber qual a *window* na qual uma dada *widget* se encontra, usa-se a função '*gtk_widget_get_toplevel*'.

A função '*gtk_widget_get_allocated_width*' usa-se para obter o comprimento duma *widget*, enquanto com '*gtk_widget_get_allocated_height*' obtém-se a sua altura. É ainda possível usar a função '*gtk_widget_get_allocation*' que preenche uma estrutura do tipo *GtkAllocation* que contém as coordenadas de canto superior esquerda da *widget* e o seu comprimento e largura.

Nos programas 'Gtk3_30_01e2.c' são apresentados os valores anteriormente referidos para as posições e tamanhos de uma *window* e de um *button*. Para a actualização, em tempo real, dos valores usou-se uma função de *callback* associada à *window* e ao *event* "configure-event" que mostra os valores actuais.

25 Leitura da posição do rato e do teclado

Para controlar o movimento do rato e os seus "clicks" deve associar-se os respectivos *events* à *window*. Note-se que é necessário activar a máscara a eles associada.

No programa 'Gtk3_32_01.c' associou-se uma função aos diferentes eventos do rato e a premir uma tecla do teclado. A acção dentro dessa função é depois escolhida a partir do tipo de *event* usando o elemento da estrutura *GdkEventExpose* que indica o tipo de *event*, 'type'. Depois, usando a informação contida na estrutura de cada um: *GdkEventButton* para o rato e *GdkEventKey* para o teclado, obtêm-se as respectivas informações. O programa 'Gtk3_32_02.c' é idêntico ao anterior com cores.

A título de exemplo mostra-se como se activam as máscaras e de associam as funções de *callback*:

```
gtk_widget_set_events (window, GDK_POINTER_MOTION_MASK |
    GDK_BUTTON_PRESS_MASK | GDK_KEY_PRESS_MASK |
    GDK_BUTTON_RELEASE_MASK);
g_signal_connect (window, "motion-notify-event",
    G_CALLBACK (cb_motion_notify), NULL);
g_signal_connect (window, "button_press_event",
    G_CALLBACK (cb_motion_notify), NULL);
```

```
g_signal_connect (window, "button_release_event",
    G_CALLBACK (cb_motion_notify), NULL);
g_signal_connect (window, "key_press_event",
    G_CALLBACK (cb_motion_notify), NULL);
```

Para mais detalhes ver Apêndice de *events*.

26 *Dialogs*

Uma *dialog* é uma *window* predefinida para interacção com o utilizador. Pode servir, por exemplo, para dar uma informação, fazer uma pergunta e obter uma resposta, etc. Uma *dialog* pode ser criada com a função '*gtk_dialog_new*' ou com a função '*ggtk_dialog_new_with_buttons*'. No primeiro caso a função não tem argumentos e deverá ser o programador a fazer a sua caracterização; no segundo caso, são indicadas já algumas especificações, em especial, o número e tipo de *buttons* que ela apresentará.

A função '*gtk_window_set_modal*', permite atribuir o atribuito 'modal' à *window*. Uma *window modal* evita a interacção com outras *windows* da mesma aplicação. Para a manter no cimo da *window* da aplicação usa-se a função *gtk_window_set_transient_for*.

Usualmente, acrescenta-se um *label* com o texto desejado à *dialog* a fim de especificar a informação desejada.

A função '*gtk_dialog_run*' permite realizar a *dialog* e ficar num ciclo de espera até obter uma resposta que é retornada por essa função.

No programa 'Gtk3_61_01.c' é criada uma *dialog* apenas com um *button* que deverá ser premido para terminar a *dialog*. No programa 'Gtk3_61_02.c' a *dialog* criada tem dois *buttons* o que permite duas respostas diferentes. A escolha é obtida a partir do retorno da função '*gtk_dialog_run*' ao terminar.

A Apêndices

A.1 Funções de GTK+

A.1.1 Funções Iniciais

- void **gtk_init** (int *argc, char ***argv);
  Inicializa o ambiente GTK+;

- void **gtk_main** (void);
  Executa o ciclo até ser dada a ordem de o terminar (ver gtk_main_quit());

- void **gtk_main_quit** (void);
  Termina um ciclo 'gtk_main';

A.1.2 *Accel groups*

- GtkAccelGroup ***gtk_accel_group_new** (void);
  Cria um 'AccelGroup';

A.1.3 *Adjustments*

- GtkAdjustment ***gtk_adjustment_new** (gdouble value, gdouble lower,
  gdouble upper, gdouble step_increment,
  gdouble page_increment, gdouble page_size);
  Cria um '*adjustment*' com valor inicial 'value', limite inferior 'lower', limite superior 'upper', incremento 'step_increment', incremento de página 'page_increment' e tamanho de página 'page_size';

A.1.4 *Boxes*

- GtkWidget ***gtk_box_new** (GtkOrientation orientation, gint spacing);
  Cria uma 'box' com orientação dada por 'orientation' que toma os valores: 'GTK_ORIENTATION_HORIZONTAL' ou 'GTK_ORIENTATION_VERTICAL';

- void **gtk_box_pack_start** (GtkBox *box, GtkWidget *child,
  gboolean expand, gboolean fill, guint padding);
  Acrescenta 'child' a 'box' empacotando-a a partir do princípio;

- void **gtk_box_pack_end** (GtkBox *box, GtkWidget *child,
  gboolean expand, gboolean fill, guint padding);
  Acrescenta 'child' a 'box' empacotando-a a partir do fim;

- GtkWidget ***gtk_box_set_homogeneous*** (GtkBox *box,
  gboolean homogeneous);
  Indica a homogeneidade da 'box';

A.1.5 *Buttons*

- GtkWidget ***gtk_button_new*** (void);
  Cria um botão;

- GtkWidget ***gtk_button_new_with_label*** (const gchar *label);
  Cria um botão com o texto 'label';

- void **gtk_button_set_label** (GtkButton *button, const gchar *label);
  Altera o 'label' de um botão;

A.1.6 *Cell layout*

- void **gtk_cell_layout_add_attribute** (GtkCellLayout *cell_layout,
  GtkCellRenderer *cell, const gchar *attribute, gint column);
  Adiciona à 'cell' o 'attribute' em 'column';

- void **gtk_cell_layout_pack_end** (GtkCellLayout *cell_layout,
  GtkCellRenderer *cell, gboolean expand);
  Coloca a 'cell' no final do 'cell_layout'. Se 'expand' for TRUE irá ocupar o espaço disponível;

- void **gtk_cell_layout_pack_start** (GtkCellLayout *cell_layout,
  GtkCellRenderer *cell, gboolean expand);
  Coloca a 'cell' no início do 'cell_layout'. Se 'expand' for TRUE irá ocupar o espaço disponível;

- void **gtk_cell_layout_set_attributes** (GtkCellLayout *cell_layout,
  GtkCellRenderer *cell, ...);
  Adiciona à 'cell' os atributos que se seguem.
  Ver '*gtk_cell_layout_add_attribute*';

A.1.7 *Cell renderer*

- GtkCellRenderer ***gtk_cell_renderer_text_new*** (void);
  Cria um *cell renderer text*.

A.1.8 *Check buttons*

- GtkWidget ***gtk_check_button_new*** (void);
  Cria um *check button*;

- GtkWidget ***gtk_check_button_new_with_label*** (const gchar *label);
  Cria um *check button* com um 'label';

- GtkWidget ***gtk_check_button_new_with_mnemonic***
  (const gchar *label);
  Cria um *check button* com um 'label' com mnemónica. O caracter para a mnemónica deve ser precedido de um traço em baixo, "_";

A.1.9 *Combo boxes*

- GtkWidget ***gtk_combo_box_text_new*** (void);
  Cria uma '*combo_box_text*';

- gint **gtk_combo_box_get_active** (GtkComboBox *combo_box);
  Retorna a posição da opção escolhida;

- void **gtk_combo_box_set_active** (GtkComboBox *combo_box,
  gint index);
  Indica a opção a mostrar. 'index' é a sua posição na lista introduzida (começando por '0')

- void **gtk_combo_box_text_append_text**
  (GtkComboBoxText *combo_box, const gchar *text);
  Insere o texto da opção 'text' no final da 'combo_box';

- void **gtk_combo_box_text_insert_text** (GtkComboBoxText *combo_box,
  gint position, const gchar *text);
  Insere o texto da opção 'text' na posição 'position' de 'combo_box'. Se 'position' for negativo acrescenta no final;

A.1.10 *Containers*

- void **gtk_container_add** (GtkContainer *container,
  GtkWidget *widget);
  Acrescenta a 'widget' ao 'container';

- void **gtk_container_set_border_width** (GtkContainer *container,
  guint border_width);
  Define a largura do 'border' de um 'container';

A.1.11 *CSS provider*

- GtkCssProvider ***gtk_css_provider_new** (void);
  Cria e um novo *css_provider*;

- gboolean **gtk_css_provider_load_from_file**
  (GtkCssProvider *css_provider, GFile *file, GError **error);
  Faz a leitura dos dados contidos no ficheiro 'css_provider', limpa qualquer informação que previamente tenha sido lida e preenche 'error' caso ele não seja NULL. Retorna TRUE, o retorno FALSE deixou de ser usado;

- gboolean **gtk_css_provider_load_from_path**
  (GtkCssProvider *css_provider, GFile *file, GError **error);
  Idêntida a *gtk_css_provider_load_from_file* mas em que 'file' é substituída pela 'path';

- gboolean **gtk_css_provider_load_from_data**
  (GtkCssProvider *css_provider, const gchar *data,
  gssize length, GError **error);
  Faz a leitura dos dados contidos na string 'data', limpa qualquer informação que previamente tenha sido lida e preenche 'error' caso ele não seja NULL. 'length' é o comprimento dos dados a ler ou '-1' se se usar o mecanismo do terminador NULL das strings;

A.1.12 *Dialogs*

- GtkWidget ***gtk_dialog_new** (void);
  Cria uma *dialog*;

- GtkWidget ***ggtk_dialog_new_with_buttons** (const gchar *title,
  GtkWindow *parent, GtkDialogFlags flags,
  const gchar *first_button_text, ...);
  Cria uma *dialog* com um título, a *window* em relação à qual é depende, com opções contidas em *GtkDialogFlags* e finalmente com os *buttons* que devem ser criados e que pertencem ao enumerado *GtkResponseType*.

- gint **gtk_dialog_run** (GtkDialog *dialog);
  Entra num ciclo até que seja obtida uma resposta ou a *dialog* seja destruída. Neste último caso a resposta será GTK_RESPONSE_NONE, caso contrário será o elemento de *GtkResponseType* associado ao *button* activado;

A.1.13 *Entries*

- GtkWidget ***gtk_entry_new*** (void);
  Cria uma *entry*;

- gboolean **gtk_entry_get_has_frame** (GtkEntry *entry);
  Retorna TRUE ou FALSE de acordo com existir ou não um *frame* à sua volta;

- gchar ***gtk_entry_get_text*** (GtkEntry *entry, const gchar *text);
  Retorna um ponteiro para o texto contido na *entry*;

- gboolean **gtk_entry_get_visibility** (GtkEntry *entry);
  Retorna TRUE ou FALSE de acordo com o texto estar visível ou não;

- void **gtk_entry_set_has_frame** (GtkEntry *entry, gboolean setting);
  Permite pôr ou retirar um *frame* à volta da *entry*. Por defeito é verdadeiro;

- void **gtk_entry_set_icon_from_icon_name** (GtkEntry *entry,
  GtkEntryIconPosition icon_pos, const gchar *icon_name);
  Acrescenta um 'icon' com o nome 'icon_name', na posição 'icon_pos' que pode ser o início da *entry*, '*GTK_ENTRY_ICON_PRIMARY*', ou o seu final, '*GTK_ENTRY_ICON_SECONDARY*';

- void **gtk_entry_set_max_length** (GtkEntry *entry, gint max);
  Define o comprimento máximo permitido para o conteúdo da *entry*. Se o conteúdo actual é maior que o comprimento 'max', então será truncado;

- void **gtk_entry_set_max_width_chars** (GtkEntry *entry, gint n_chars);
  Estabelece o limite para o tamanho da *entry* em 'n_chars' caracteres;

- void **gtk_entry_set_text** (GtkEntry *entry, const gchar *text);
  Coloca o texto 'text' na *entry*;

- void **gtk_entry_set_visibility** (GtkEntry *entry, gboolean visible);
  De acordo com 'visible' mostra ou não o texto escrito na *entry*;

- void **gtk_entry_set_width_chars** (GtkEntry *entry, gint n_chars);
  Fixa o tamanho da *entry* em 'n_chars' caracteres. Se o valor for '-1' é usado o valor de defeito;

A.1.14 *Frames*

- GtkWidget ***gtk_frame_new*** (const gchar *label);
  Cria um *frame* com um título 'label'. No caso de não se desejar pôr um título, 'label' deverá ser posto a NULL;

- const gchar ***gtk_frame_get_label*** (GtkFrame *frame);
  Recebe um ponteiro para a *string* que contém o título do *frame*;

- void **gtk_frame_set_label** (GtkFrame *frame, const gchar *label);
  Altera o título de um *frame*;

- void **gtk_frame_set_label_align** (GtkFrame *frame,
  gfloat xalign, gfloat yalign);
  Ajusta o título do 'frame' horizontalmente com o valor de 'xalign' em que '0.0' corresponde ao alinhamento à esquerda e '1.0' o alinhamento à direita os restantes valores entre eles correspondem a posições intermédias, para centrar o valor é '0.5'. O alignamento vertical é idêntico. Por defeito, os valores são respectivamente '0.0' e '0.5';

- void **gtk_frame_set_label_widget** (GtkFrame *frame,
  GtkWidget *label_widget);
  Põe o objecto 'widget' no título do frame;

A.1.15 *Images*

- GtkWidget ***gtk_image_new*** (void);
  Limita-se a criar um objecto vazio do tipo *GtkImage*, moldado em *GtkWidget*. Deverá ser posteriormente preenchido;

- GtkWidget ***gtk_image_new_from_file*** (const gchar *filename);
  Cria uma *image* a partir do ficheiro 'filename';

- GtkWidget ***gtk_image_new_from_icon_name*** (const gchar *icon_name,
  GtkIconSize size);
  Cria uma *image* a partir do icon com o nome '*icon_name*'. A lista dos nomes dos icons pode encontrar em '*Icon Naming Specification*'[2]. 'size' especifica o tamanho do icon desejado;

- GtkWidget ***gtk_image_new_from_pixbuf*** (GdkPixbuf *pixbuf);
  Cria uma *image* a partir de um *pixbuf*;

- void **gtk_image_set_from_icon_name** (GtkImage *image,
  const gchar *icon_name, GtkIconSize size);

Acrescenta ao objecto *image* a imagem do *icon* com o tamanho dado por 'size'. O enumerado '*GtkIconSize*' pode tomar, dependendo de 'size', os seguintes valores: GTK_ICON_SIZE_MENU GTK_ICON_SIZE_SMALL_TOOLBAR GTK_ICON_SIZE_BUTTON a que correspondem *icons* de tamanho igual a 16px, GTK_ICON_SIZE_LARGE_TOOLBAR de 24px, GTK_ICON_SIZE_DND para 32px e GTK_ICON_SIZE_DIALOG para 48px;

A.1.16 *Labels*

- GtkWidget ***gtk_label_new** (const gchar *str);
  Cria um 'label' com o texto 'str';

- const gchar ***gtk_label_get_label** (GtkLabel *label);
  Retorna um ponteiro para o texto mostrado no 'label' incluindo marcas de mnemónicas ou do Pango;

- const gchar ***gtk_label_get_text** (GtkLabel *label);
  Retorna um ponteiro para o texto mostrado no 'label';

- void **gtk_label_set_text** (GtkLabel *label, const gchar *str);
  Altera o texto do 'label' para 'str';

A.1.17 *List store*

- GtkListStore ***gtk_list_store_new** (gint n_columns, ...);
  Cria uma '*list_store*' com 'n_columns' indicando-se a seguir os seus tipos específicos, por exemplo, G_TYPE_INT, G_TYPE_STRING, GDK_TYPE_PIXBUF;

- void **gtk_list_store_insert_with_values** (GtkListStore *list_store,
  GtkTreeIter *iter, gint position, ...);
  Cria uma nova linha em 'position.iter' à qual se acrescenta o seu conteúdo;

A.1.18 *Menus*

- GtkWidget ***gtk_menu_new** (void);
  Cria um *menu*;

- GtkWidget ***gtk_menu_bar_new** (void);
  Cria uma *menu bar*;

- GtkWidget ***gtk_menu_item_new** (void);
  Cria um *menu item*;

- GtkWidget ***gtk_menu_item_new_with_label** (const gchar *label);
  Cria um *menu item* com o texto contido em 'label';

- GtkWidget ***gtk_menu_item_new_with_mnemonic** (const gchar *label);
  Cria um *menu item* com o texto contido em 'label'. Um traço em baixo no label indica a mnemónica para esse *menu item*.

- void **gtk_menu_shell_append** (GtkMenuShell *menu_shell,
  GtkWidget *child);
  Adiciona o *menu item* 'child' a 'menu_shell';

- void **gtk_menu_item_set_submenu** (GtkMenuItem *menu_item,
  GtkWidget *submenu);
  Coloca o *menu* 'submenu' em 'menu_item';

- void **gtk_menu_item_set_label** (GtkMenuItem *menu_item,
  const gchar *label);
  Coloca o texto 'label' no 'menu_item';

A.1.19 *Orientable*

- void **gtk_orientable_set_orientation** (GtkOrientable *orientable,
  GtkOrientation orientation);
  Aplica-se a um objecto que seja orientável e dá-lhe a orientação desejável. Os valores possíveis para a orientação são *GTK_ORIENTATION_HORIZONTAL* e *GTK_ORIENTATION_VERTICAL*;

A.1.20 *Radio buttons*

- GtkWidget ***gtk_radio_button_new** (void);
  Cria um *radio button*;

- GtkWidget ***gtk_radio_button_new_with_label** (const gchar *label);
  Cria um *radio button* com um 'label';

- GtkWidget ***gtk_radio_button_new_with_mnemonic**
  (const gchar *label);
  Cria um *radio button* com um 'label' com mnemónica. O caracter para a mnemónica deve ser precedido de um traço em baixo, "_";

- GtkWidget ***gtk_radio_button_new_from_widget**
  (GtkRadioButton *radio_group_member);
  Cria um *radio button* associado ao *radio button* 'radio_group_member';

- GtkWidget ***gtk_radio_button_new_with_label_from_widget**
  (GtkRadioButton *radio_group_member, const gchar *label);
  Cria um *radio button* com um 'label' associado a 'radio_group_member';

- GtkWidget ***gtk_radio_button_new_with_mnemonic_from_widget**
  (GtkRadioButton *radio_group_member, const gchar *label);
  Cria um *radio button* com um 'label' com mnemónica associado ao *radio button* 'radio_group_member';

A.1.21 *Ranges*

- gdouble **gtk_range_get_value** (GtkRange *range);
  Retorna o valor de 'range';

- void **gtk_range_set_value** (GtkRange *range, gdouble value);
  Atribui o valor 'value' a 'range';

A.1.22 *Scales*

- GtkWidget ***gtk_scale_new** (GtkOrientation orientation,
  GtkAdjustment *adjustment);
  Cria uma '*scale*' com os dados definidos em 'adjustment' e com a orientação dada por 'orientation' que, de acordo com o que se pretende, pode ser GTK_ORIENTATION_HORIZONTAL ou GTK_ORIENTATION_VERTICAL;

- GtkWidget ***gtk_scale_new_with_range** (GtkOrientation orientation,
  gdouble min, gdouble max, gdouble step);
  Cria uma '*scale*' com valor mínimo 'min', valor máximo 'max' e com passo 'step'. A orientação é dada por 'orientation' que pode tomar os valores GTK_ORIENTATION_HORIZONTAL ou GTK_ORIENTATION_VERTICAL;

- gint **gtk_scale_get_digits** (GtkScale *scale);
  Retorna o número de dígitos que estão sendo mostrados;

- GtkPositionType **gtk_scale_get_value_pos** (GtkScale *scale);
  Retorna a posição, em relação à escala, em que se coloca o valor actual;

- void **gtk_scale_set_digits** (GtkScale *scale, gint digits);
  Fixa o número de dígitos a ser mostrado;

- void **gtk_scale_set_draw_value** (GtkScale *scale, gboolean draw_value);
  Fixa se o valor actual é ou não mostrado de acordo com 'draw_value' ser respectivamente TRUE ou FALSE;

- void **gtk_scale_set_value_pos** (GtkScale *scale, GtkPositionType pos);
  Define a posição em que é posto o valor actual. 'pos' pode tomar os valores: GTK_POS_LEFT, GTK_POS_RIGHT, GTK_POS_TOP e GTK_POS_BOTTOM;

A.1.23 *Spin buttons*

- GtkWidget ***gtk_spin_button_new** (GtkAdjustment *adjustment,
  gdouble climb_rate, guint digits);
  Cria um '*spin button*' em que 'adjustment' contém o seu valor inicial e as suas características, 'climb_rate' é o incremento/decremento quando se carrega nos botões e 'digits' o número de casas decimais a mostrar;

- GtkWidget ***gtk_spin_button_new_with_range** (gdouble min,
  gdouble max, gdouble step);
  Cria um '*spin button*' indicando os valores mínimo e máximo válidos e ainda incremento/decremento quando se carrega nos botões;

- guint **gtk_spin_button_get_digits** (GtkSpinButton *spin_button); Retorna o número de casas decimais mostrados por 'spin_button';

- gboolean **gtk_spin_button_get_numeric** (GtkSpinButton *spin_button);
  Retorna se caracteres não numéricos podem ser escritos no *spin button*;

- gdouble **gtk_spin_button_get_value** (GtkSpinButton *spin_button);
  Retorna o valor contido no 'spin button';

- void **gtk_spin_button_set_digits** (GtkSpinButton *spin_button,
  guint digits);
  Fixa o número de casas decimais a mostrar no *spin button*;

- void **gtk_spin_button_set_numeric** (GtkSpinButton *spin_button,
  gboolean numeric);
  No caso de 'numeric' ser TRUE apenas são válidos valores numéricos;

- void **gtk_spin_button_set_value** (GtkSpinButton *spin_button,
  gdouble value);
  Atribui o valor 'value' ao *spin button*;

A.1.24 *Status bars*

- GtkWidget ***gtk_statusbar_new** (void);
  Cria uma *status bar*;

- guint **gtk_statusbar_get_context_id** (GtkStatusbar *statusbar,
  const gchar *context_description);
  Retorna um novo identificador dada a descrição do contexto actual;

- void **gtk_statusbar_pop** (GtkStatusbar *statusbar,
  guint context_id);
  Remove a primeira mensagem da pilha da *status bar* com o identificador '*context_id*';

- guint **gtk_statusbar_push** (GtkStatusbar *statusbar,
  guint context_id, const gchar *text);
  Coloca uma nova mensagem na pilha da 'statusbar';

- void **gtk_statusbar_remove_all** (GtkStatusbar *statusbar,
  guint context_id);
  Remove todas as mensagens da pilha da *status bar* com o identificador '*context_id*';

A.1.25 *Style context*

- void **gtk_style_context_add_provider_for_screen** (GdkScreen *screen,
  GtkStyleProvider *provider, guint priority);
  Acrescenta o 'provider' global de 'style' ao 'screen' que será usado em todos os *style context* nesse 'screen' e em que 'priority' um valor compreendido entre GTK_STYLE_PROVIDER_PRIORITY_FALLBACK, que toma o valor mínimo e GTK_STYLE_PROVIDER_PRIORITY_USER que toma o valor máximo;

A.1.26 *Toggle buttons*

- GtkWidget ***gtk_toggle_button_new** (void);
  Cria um *toggle button*;

- GtkWidget ***gtk_toggle_button_new_with_label** (const gchar *label);
  Cria um *toggle button* com um 'label';

- GtkWidget ***gtk_toggle_button_new_with_mnemonic**
  (const gchar *label);
  Cria um *toggle button* com um 'label' e uma mnemónica associada. O caracter para a mnemónica deve ser precedido de um traço em baixo, "_";

- gboolean **gtk_toggle_button_get_mode**
  (GtkToggleButton *toggle_button);
  Retorna o estado em que se encontra o *toggle button* (activado ou não);

- void **gtk_toggle_button_set_mode** (GtkToggleButton *toggle_button,
  gboolean draw_indicator);
  Fixa o estado do *toggle button* em activado ou desactivado de acordo com o valor de 'draw_indicator';

A.1.27 *Tool bars*

- GtkWidget ***gtk_toolbar_new** (void);
  Cria um a '*toolbar*';

- void **gtk_toolbar_insert** (GtkToolbar *toolbar,
  GtkToolItem *item, gint pos);
  Insere o 'item' na *toolbar*. 'pos' indica a posição em que é colocada, '0' significa no início e negativo que é colocado em último lugar;

- void **gtk_toolbar_set_show_arrow** (GtkToolbar *toolbar,
  gboolean show_arrow);
  Se o tamanho da *window* não permite que os items caibam todos pode ser criada uma seta e um *menu*, a ela associado, em que se incluem os items que não cabem na *toolbar*. Para tal deve marcar-se 'show_arrow' em 'TRUE';

- void **gtk_toolbar_set_style** (GtkToolbar *toolbar,
  GtkToolbarStyle style);
  Permite definir o tipo de objectos que a toolbar pode integrar (icons ou texto).

A.1.28 *Tool button*

- GtkToolItem ***gtk_tool_button_new** (GtkWidget *icon_widget,
  const gchar *label);
  Cria um *tool button* para a ser integrado numa '*toolbar*'. O primeiro argumento é um icon (*widget*) que, caso não se use, deve ser posto a 'NULL' e o segundo argumento é o texto associado.

- void **gtk_tool_button_set_icon_name** (GtkToolButton *button,
  const gchar *icon_name);
  Associa ao *tool button* um *icon* com o nome '*icon_name*'. A lista dos nomes dos *icons* pode ser encontrada em '*Icon Naming Specification*'[2];

A.1.29 *Widgets*

- void **gtk_widget_add_accelerator** (GtkWidget *widget,
  const gchar *accel_signal, GtkAccelGroup *accel_group,

const guint accel_key, GdkModifierType accel_mods,
GtkAccelFlags accel_flags);
Instala o acelador 'accel_group' na 'widget' que dá origem ao sinal 'accel_signal' para ser emitido se o acelerador for activado, 'accel_key' é a tecla a associar, 'accel_mods' é o modificador da combinação de teclas e 'accel_flags' é uma das opções de 'GtkAccelFlags'. O acelerador tem de ser associado à *window* usando a função '*gtk_window_add_accel_group*'.

- gint **gtk_widget_get_allocated_height** (GtkWidget *widget);
  Retorna a altura de 'widget';

- gint **gtk_widget_get_allocated_width** (GtkWidget *widget);
  Retorna o comprimento de 'widget';

- void **gtk_widget_get_allocation** (GtkWidget *widget,
  GtkAllocation *allocation);
  Preenche a estrutura 'allocation' em que os componentes 'x' e 'y' são as coordenadas da posição da 'widget' na *window* e 'width' e 'height' são respectivamente o seu comprimento e altura;

- void **gtk_widget_destroy** (GtkWidget *widget);
  Destroi 'widget';

- gboolean **gtk_widget_get_sensitive** (GtkWidget *widget);
  Retorna TRUE ou FALSE de acordo com 'widget' estar ou não bloqueado;

- GtkWidget ***gtk_widget_get_toplevel** (GtkWidget *widget);
  Retorna a *widget* mais acima da sua hierarquia, em geral, é a *widget* associada à *window*.

- void **gtk_widget_get_size_request** (GtkWidget *widget,
  gint *width, gint *height);
  Retorna os valores do comprimento e altura de '*size request*' que foi explicitamente posto pela função '*gtk_widget_set_size_request*'. Caso não tenha sido feito, returna '-1';

- void **gtk_widget_hide** (GtkWidget *widget);
  Esconde (torna invisível) 'widget';

- void **gtk_widget_set_sensitive** (GtkWidget *widget,
  gboolean sensitive);
  De acordo com o valor de 'sensitive' (TRUE ou FALSE), a 'widget' fica ou não bloqueada;

- void **gtk_widget_set_size_request** GtkWidget *widget,
  gint width, gint height);
  Fixa o tamanho mínimo (comprimento e altura) da 'widget'. Pode usar-se '-1' para não se fixar algum dos valores;

- void **gtk_widget_set_name** (GtkWidget *widget, const gchar *name);
  As *widgets* podem ter nomes associados o que permite a sua ligação às propriedades contidas nas definições CSS. Com esta função atribui-se o nome 'name' à *widget*;

- void **gtk_widget_show** (GtkWidget *widget);
  Torna visível 'widget';

- void **gtk_widget_show_all** (GtkWidget *widget);
  Torna visível 'widget' e todas as *widgets* nela contidas;

A.1.30 *Windows*

- GtkWidget ***gtk_window_new** (GtkWindowType type);
  Cria uma *window*. 'type' deve ser 'GTK_WINDOW_TOPLEVEL' para uma *window* com decorações e 'GTK_WINDOW_POPUP' para uma sem elas;

- void **gtk_window_add_accel_group** (GtkWindow *window,
  GtkAccelGroup *accel_group);
  Associa um '*accel group*' à *window*;

- void **gtk_window_get_position** (GtkWindow *window,
  gint *root_x, gint *root_y);
  Caso não tenha sido alterada a sua '*gravity*' inicial (Norte Oeste), esta função preenche 'root_x' e 'root_y' com as coordenadas do ponto superior esquerdo da *window*;

- void **gtk_window_get_size** (GtkWindow *window,
  gint *width, gint *height);
  Preenche as variáveis 'width' e 'height' com o comprimento e altura da *window*;

- void **gtk_window_set_default_size** (GtkWindow *window,
  gint width, gint height);
  Define o tamanho, por defeito, da *window*;

- void **gtk_window_set_modal** (GtkWindow *window, gboolean modal);
  Dá ou retira o atributo 'modal' à *window*. Uma *window modal* evita a interacção com outras *window* da mesma aplicação. Para manter uma

window modal sobre as outras *windows* da aplicação pode usar-se a função '*gtk_window_set_transient_for*';

- void **gtk_window_set_position** (GtkWindow *window,
  GtkWindowPosition position);
  Define a posição em que a *window* é colocada. As opções disponíveis são: não dar indicação ('GTK_WIN_POS_NONE'), central a nova *window* no ecran ('GTK_WIN_POS_CENTER'), pô-la na posição em que se encontra o rato ('GTK_WIN_POS_MOUSE'), central sempre mesmo quando se altera o tamanho ('GTK_WIN_POS_CENTER_ALWAYS') e, para *windows* chamadas por *windows*, centrá-la com a anterior ('GTK_WIN_POS_CENTER_ON_PARENT');

- void **gtk_window_set_resizable** (GtkWindow *window,
  gboolean resizable);
  Permite ou não que se possa alterar o tamanho duma *window* de acordo com o valor de 'resizable' ser FALSE ou TRUE. Por defeito, uma *window* ao ser criada permite que o seu tamanho seja alterado;

- void **gtk_window_set_title** (GtkWindow *window, const gchar *title);
  Define o título da *window*;

- void **gtk_window_set_transient_for** (GtkWindow *window,
  GtkWindow *parent);
  Assegura que a *window* fica sobre a *window* da aplicação;

A.2 Listagem das funções de GDK

- GdkDisplay ***gdk_display_get_default** (void);
  Retorna o *display* de defeito ou NULL se não existir nenhum;

- GdkScreen ***gdk_display_get_default_screen** (GdkDisplay *display);
  Retorna o *screen* de defeito associado ao 'display';

- GdkPixbuf ***gdk_pixbuf_new_from_file** (const char *filename,
  GError **error);
  Cria um pixbuf ao carregar uma imagem a partir de um ficheiro. O formato de arquivo é detectada automaticamente. Se falhar retorna NULL e através de 'error' retorna o erro em causa;

- GdkPixbufFormat ***gdk_pixbuf_get_file_info** (const gchar *filename,
  gint *width, gint *height);
  Permite obter o comprimento de largura e uma imagem. Para além disso retorna um ponteiro para a estrutura *GdkPixbufFormat* que descreve o formato dessa imagem.
  Nota: o ponteiro que retorna é para um elemento do *GdkPixbuf* e, por isso, não deve ser libertado (free);

- GdkPixbufLoader ***gdk_pixbuf_loader_new** (void);
  Cria um *pixbuf loader*.

- gboolean **gdk_pixbuf_loader_write** (GdkPixbufLoader *loader,
  const guchar *buf, gsize count, GError **error);
  Irá carregador no *pixbuf loader* uma imagem a partir de 'buf' com o comprimento 'count'. Se tiver sucesso retorna TRUE; caso contrário retorna em 'error' o erro em causa;

- GdkPixbuf ***gdk_pixbuf_loader_get_pixbuf** (GdkPixbufLoader *loader);
  Retorna o *pixbuf* a partir do *pixbuf loader*;

- gboolean **gdk_pixbuf_loader_close** (GdkPixbufLoader *loader,
  GError **error);
  Informa o *pixbuf loader* que não há mais nada a escrever com a função '*gdk_pixbuf_loader_write*'.

A.3 Listagem das funções de GLib e GObject

- void **g_clear_object** (volatile GObject **object_ptr);
  Limpa a referência de 'object_ptr';

- #define **g_signal_connect** (instance, detailed_signal, c_handler, data);
  Liga a função 'c_handler' ao sinal 'detailed_signal' do objecto 'instance' e envia 'data' para essa função. Se tiver sucesso, retorna um inteiro positivo que é o '*id*' associado;

- gpointer **g_object_ref** (gpointer object);
  Incrementa o contador associado ao objecto 'object' e retorna próprio ponteiro do objecto.

- void **g_object_unref** (gpointer object);
  Decrementa o contador associado ao objecto 'object'. Quando o contador se encontra a "0", o objecto é finalizado, isto é, a memória é libertada. Ter em atenção que se a variável vier a ser utilizada novamente é aconselhável pô-la a NULL usando a função '*g_clear_object*';

- gchar ***g_filename_to_utf8** (const gchar *opsysstring, gssize len,
  gsize *bytes_read, gsize *bytes_written, GError **error);
  Retorna a conversão da *string* 'opsysstring' de comprimento 'len' para o nome dum ficheiro em UTF-8. Preenche ainda 'bytes_read' com o número de bytes lidos com sucesso, 'bytes_written' com o número de bytes da *string* retornada e, no caso de erro, retorna-o em 'error' caso este seja diferente de NULL;

A.4 Apêndice CSS (Cascading Style Sheets)

A organização dos dados CSS é feita por uma sucessão de declarações de propriedades que se encontrar entre chavetas. Essas declarações são precedidas de um cabeçado o qual pode ser:

- **Tipo de um objecto:** nesse caso todos os objectos criados com esse tipo adquirem as propriedades nele incluídas, por exemplo, que se quiser que o fundo de uma *window* seja 'Wheat1' pode fazer-se:

  GtkWindow {background-color: Wheat1;}

  Ver programa 'Gtk3_09_01.c'.

  Do mesmo modo pode atribuir-se um tipo de letra e uma cor a um *label*, tipo '*GtkLabel*. A declaração é feita de um modo análogo à anterior. Admitamos que queremos que os *labels* sejam escritos com a cor "#ab0567"e com a tipo de letra "Tahoma bold 16". Então o comando será:

  GtkLabel {color: #ab0567; font: Tahoma bold 16;}

  Ver programa 'Gtk3_09_02.c'. Note-se que ao definir estas características para o *label*, os *labels* dos *buttons* também foram alterados.

- **Nome de objecto:** uma vez que o tipo é demasiado geral para se poder fazer um controle correcto, pode associar-se um nome a um conjunto de propriedades, tal é feito usado como cabeçalho um nome começado por um cardinal, '#' e depois associa-se esse nome às *widgets* pretendidas com a função '*gtk_widget_set_name*'. Assim, se se designar o conjunto de propriedades por 'label1', a linha CSS seria:

  #label1 {color: #ab0567; font: Tahoma bold 16;}

  Ver programa 'Gtk3_09_03.c'.

- **Tipo e Nome de objecto:** É ainda possível quando se define um nome restringí-lo a um tipo específico. Assim, por exemplo, no caso de '#label1' que se usou para objectos do tipo *GtkLabel* pode fazer-se essa especificação na definição da entrada:

  GtkLabel#label1 {color: #ab0567; font: Tahoma bold 16;}

  para o caso dos objectos do tipo *GtkLabel* nada se altera, no entanto, com a primeira definição esse nome era válido para todos os objectos, com a nova definição apenas é válida para *GtkLabel*. Ver programa 'Gtk3_09_07.c'. Nele se aplica a propriedade '#Prop01' aos botões.

- **Definição de uma classe e sua atribuição:** É ainda possível criar classes, associadas ou não a objectos epecíficos:

  .label_Main color: #ab0567; font: Tahoma bold 16;

  A atribuição da classe é feita adicionando a classe específica ao style do objecto em causa, que se designa aqui por 'label':

  GtkStyleContext *context = gtk_widget_get_style_context (label);

  gtk_style_context_add_class (context, "label_Main");

  Ver programa 'Gtk3_09_07.c'.

Como se disse atrás, um programa pode obter a informação CSS basicamente de duas formas diferente: ou através de um ficheiro no qual se encontram as propriedades ou através de uma *string* na qual esse mesmo código se encontra. Foi essa a opção usada nos programas anteriores. No programa 'Gtk3_09_11.c' usam-se as definições escritas num ficheiro chamado 'Gtk3_09_11.css'.

No programa 'Gtk3_09_10.c' mostra-se como se pode colocar uma imagem de fundo numa *window*; no programa 'Gtk3_09_11.c' coloca-se um gradiente de cor no fundo da *window*.

A.5 Apêndice *Events* e *Signals*

Neste apêndice são apresentados os *events* mais frequentemente usados em GTK+. Para uma informação completa ver '*GTK+ 3 Reference Manual*'[4]. Os *events* encontram-se associados a cada objecto e é ainda indicada a estrutura argumental da função de *callback* a eles associada.

Lista dos principais eventos associado ao objecto *GtkWidget*:

- **button-press-event**: este *signal* é emitido quando se dá o *event* carregar no botão do rato. Para receber este *signal*, deve ser activada para a *window* a máscara GDK_BUTTON_PRESS_MASK;
- **button-release-event**: este *signal* é emitido quando se dá o *event* libertar uma tecla do botão do rato que estava carregada. Para receber este *signal*, deve ser activada para a *window* a máscara GDK_BUTTON_RELEASE_MASK;
- **configure-event**: é emitido sempre que o tamanho ou a posição de uma *window* se altera. Para receber este *signal*, deve ser activada para a *window* a máscara GDK_STRUCTURE_MASK;
- **delete-event**: é emitido se um utilizador solicita que uma *window* do tipo *top level* seja fechada. Por defeito, isto conduz à destruição da *window*. Quando ligado à função '*gtk_widget_hide_on_delete*', deixa a *window* invisível;
- **destroy**: este *signal* indica que todas as referências associadas serão destruídas e libertadas. Ao contrário de '*delete-event*' não permite a recuperação da *window*.
- **draw**: este *signal* permite desenhar a área associada e envia para a função de *callback* um objecto de *cairo* que manipulará a área em questão;
- **hide**: este *signal* é emitido quando uma *widget* é posta escondida, por exemplo, com '*gtk_widget_hide*';
- **key-press-event**: este *signal* é emitido quando se dá o *event* carregar numa tecla do teclado. Para receber este *signal*, deve ser activada para a *window* a máscara GDK_KEY_PRESS_MASK;
- **key-release-event**: este *signal* é emitido quando se dá o *event* libertar uma tecla do teclado que estava sendo carregada. Para receber este *signal*, deve ser activada para a *window* a máscara GDK_KEY_RELEASE_MASK;

- **motion-notify-event**: este *signal* é emitido quando o rato se move sobre a *window*. Para receber este *signal*, deve ser activada para a *window* a máscara GDK_POINTER_MOTION_MASK;
- **scroll-event**: este *signal* é emitido quando um botão do rato da gama 4 a 7 (roda) é pressionado. Para receber este *signal*, deve ser activada para a *window* a máscara GDK_SCROLL_MASK;
- **show**: este *signal* é emitido quando uma *widget* é mostrada, por exemplo, com '*gtk_widget_show*';

Lista dos principais eventos associado ao objecto *GtkButton*:

- **clicked**: é emitido sempre que o *button* é activado (premido ou largado);

Lista dos principais eventos associado ao objecto *GtkEditable*:

- **changed**: é emitido sempre que se dá uma alteração do seu conteúdo;
- **delete-text**: é emitido sempre que é apagado texto;
- **insert-text**: é emitido sempre que é inserido texto;

Lista dos principais eventos associado ao objecto *GtkComboBox*:

- **changed**: é emitido sempre que se dá uma alteração do seu conteúdo;
- **popdown**: é emitido sempre que se dá o fecho da lista da *combo box*;
- **popup**: é emitido sempre que se dá a abertura da lista da *combo box*;

Lista dos principais eventos associado ao objecto *GtkSpinButton*:

- **value-changed**: é emitido sempre que se dá uma alteração do seu conteúdo;

Referências

[1] Donald E. Knuth. *The TeXbook*. Addison-Wesley, Reading, Massachusetts: 1984. ISBN 0-201-13448-9.

[2] Site *Icon Naming Specification* da autoria de Rodney Dawes. URL: *https://specifications.freedesktop.org/icon-naming-spec/icon-naming-spec-latest.html*;

[3] Site *Fillster* URL: *http://www.fillster.com/*. Uma lista de cores pode ser encontrada na página *http://www.fillster.com/colorcodes/colorchart.html* deste stite;

[4] GTK+ 3 Reference Manual *https://developer.gnome.org/gtk3/stable*;

[5] GLib Reference Manual *https://developer.gnome.org/glib/stable/*;

[6] GObject Reference Manual *https://developer.gnome.org/gobject/stable/*;

[7] Pango Reference Manual *https://developer.gnome.org/pango/stable/*;

[8] GDK Reference Manual *https://developer.gnome.org/gdk3/stable/*;

[9] GDK-PixBuf Reference Manual *https://developer.gnome.org/gdk-pixbuf/unstable/*;

[10] ATK - Accessibility Toolkit Reference Manual *https://developer.gnome.org/atk/stable/*;

[11] Cairo Documentation *http://cairographics.org/documentation/*;