The Python GTK+ 3 Tutorial

Release 3.4

Sebastian Pölsterl

03 mar. 2026

Sumário

Lançamento
3.4

Data
03 mar. 2026

Este tutorial fornece uma introdução à criação de aplicativos GTK+ 3 no Python.

Antes de trabalhar com este tutorial, é recomendável que você tenha uma compreensão razoável da linguagem de programação Python. A programação GUI introduz novos problemas em comparação com a interação com a saída padrão (console / terminal). É necessário que você saiba como criar e executar arquivos Python, entender os erros básicos do interpretador e trabalhar com strings, inteiros, floats e valores booleanos. Para os widgets mais avançados neste tutorial, serão necessários bons conhecimentos de listas e tuplas.

Although this tutorial describes the most important classes and methods within GTK+ 3, it is not supposed to serve as an API reference. Please refer to the GTK+ 3 Reference Manual for a detailed description of the API. Also there's a Python-specific reference available.

Conteúdo:

CAPÍTULO 1

Instalação

O primeiro passo antes de começarmos com a programação de fato consiste em configurar o PyGObject e suas dependências. PyGObject é um módulo Python que permite aos desenvolvedores acessar bibliotecas baseadas no GObject, como o GTK+, dentro do Python. Ele possui suporte exclusivamente ao GTK+ versão 3 ou posterior.

For full IDE support (including autocomplete) you will also need the type stubs provided by the PyGObject-stubs package.

1.1 Dependências

- GTK+3
- Python 2 (2.6 ou posterior) ou Python 3 (3.1 ou posterior)
- gobject-introspection

1.2 Pacotes pré-construídos

Versões recentes do PyGObject e suas dependências são empacotadas por quase todas as principais distribuições do Linux. Então, se você usa o Linux, você provavelmente pode começar instalando o pacote a partir do repositório oficial da sua distribuição.

1.3 Instalando a partir da origem

A maneira mais fácil de instalar o PyGObject a partir do código-fonte é usando o JHBuild. Ele é projetado para criar facilmente pacotes de código-fonte e descobrir quais dependências precisam ser construídas e em que ordem. Para configurar o JHBuild, por favor, siga o manual do JHBuild.

Depois de ter instalado o JHBuild com sucesso, baixe a configuração mais recente de[1]. Copie os arquivos com o sufixo *.modules* para o diretório de módulos do JHBuild e o arquivo *sample-tarball.jhbuildrc* para *~/.jhbuildrc*.

Se você não tiver feito isso antes, verifique se o seu ambiente de compilação está configurado corretamente executando:

```
$ jhbuild sanitycheck
```

It will print any applications and libraries that are currently missing on your system but required for building. You should install those using your distribution's package repository. A list of package names for different distributions is maintained on the GNOME wiki. Run the command above again to ensure the required tools are present.

Executando o seguinte comando irá construir o PyGObject e todas as suas dependências:

```
$ jhbuild build pygobject
```

Finalmente, você pode querer instalar o GTK+ a partir do código-fonte:

```
$ jhbuild build gtk+-3
```

Para iniciar um shell com o mesmo ambiente usado pelo JHBuild, execute:

```
$ jhbuild shell
```

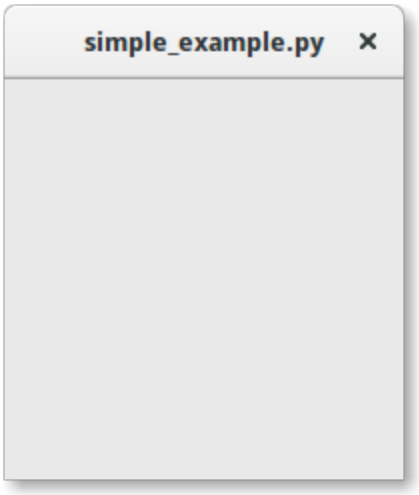
[1] https://download.gnome.org/teams/releng/

CAPÍTULO 2

Começando

2.1 Exemplo simples

Para começar com o nosso tutorial, criamos o programa mais simples possível. Este programa irá criar uma janela vazia de 200×200 pixels.




```
1 import gi
2
3 gi.require_version("Gtk", "3.0")
4 from gi.repository import Gtk
5
6 win = Gtk.Window()
7 win.connect("destroy", Gtk.main_quit)
8 win.show_all()
9 Gtk.main()
```

Vamos agora explicar cada linha do exemplo.

```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk
```

No começo, temos que importar o módulo Gtk para poder acessar as classes e funções do GTK+. Como o sistema de um usuário pode ter várias versões do GTK+ instaladas ao mesmo tempo, queremos ter certeza de que, quando importamos o Gtk, ele se refere ao GTK+ 3 e não a qualquer outra versão da biblioteca, que é o propósito da declaração `gi.require_version('Gtk', '3.0')`.

A próxima linha cria uma janela vazia.

```
win = Gtk.Window()
```

Seguido conectando-se ao evento de exclusão da janela para garantir que o aplicativo seja encerrado se clicarmos no *x* para fechar a janela.

```
win.connect("destroy", Gtk.main_quit)
```

Na próxima etapa, exibimos a janela.

```
win.show_all()
```

Finalmente, iniciamos o loop de processamento do GTK+, que encerramos quando a janela é fechada (veja a linha 6).

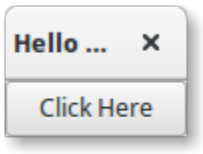
```
Gtk.main()
```

Para executar o programa, abra um terminal, mude para o diretório do arquivo e digite:

```
python simple_example.py
```

2.2 Exemplo estendido

Para algo um pouco mais útil, aqui está a versão PyGObject do programa clássico “Hello World”.




```
1  import gi
2
3  gi.require_version("Gtk", "3.0")
4  from gi.repository import Gtk
5
6
7  class MyWindow(Gtk.Window):
8      def __init__(self):
9          super().__init__(title="Hello World")
10
```

(continua na próxima página)

(continuação da página anterior)

```
        self.button = Gtk.Button(label="Click Here")
        self.button.connect("clicked", self.on_button_clicked)
        self.add(self.button)

    def on_button_clicked(self, widget):
        print("Hello World")


win = MyWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

Este exemplo difere do exemplo simples, pois subclassificamos `Gtk.Window` para definir nossa própria classe `MyWindow`.

```
class MyWindow(Gtk.Window):
```

No construtor da classe, temos que chamar o construtor da superclasse. Além disso, dizemos para definir o valor da propriedade *title* como *Hello World*.

```
        super().__init__(title="Hello World")
```

As próximas três linhas são usadas para criar um widget de botão, conectar ao seu sinal *clicked* e adicioná-lo como filho à janela de nível superior.

```
        self.button = Gtk.Button(label="Click Here")
        self.button.connect("clicked", self.on_button_clicked)
        self.add(self.button)
```

Assim, o método `on_button_clicked()` será chamado se você clicar no botão.

```
    def on_button_clicked(self, widget):
        print("Hello World")
```

O último bloco, fora da classe, é muito semelhante ao exemplo acima, mas ao invés de criar uma instância da classe genérica `Gtk.Window`, criamos uma instância de `MyWindow`.

CAPÍTULO 3

Noções básicas

Esta seção apresentará alguns dos aspectos mais importantes do GTK+.

3.1 Loop principal e sinais

Como a maioria dos toolkits de GUI, o GTK+ usa um modelo de programação orientada a eventos. Quando o usuário não está fazendo nada, o GTK+ fica no loop principal e aguarda a entrada. Se o usuário executar alguma ação – digamos, um clique do mouse — o loop principal "acorda" e entrega um evento para o GTK+.

Quando widgets recebem um evento, eles frequentemente emitem um ou mais sinais. Sinais notificam seu programa que "algo interessante aconteceu" invocando funções que você conectou ao sinal. Tais funções são comumente conhecidas como *callbacks* ou *retorno de chamada*. Quando seus retornos de chamada são invocados, você normalmente toma algumas ações – por exemplo, quando um botão Abrir é clicado, você pode exibir uma caixa de diálogo de seleção de arquivos. Depois que um retorno de chamada terminar, o GTK+ retornará ao loop principal e aguardará mais entrada do usuário.

Um exemplo genérico é:

```
handler_id = widget.connect("event", callback, data)
```

Em primeiro lugar, *widget* é uma instância de um widget que criamos anteriormente. Em seguida, o evento em que estamos interessados. Cada widget tem seus próprios eventos específicos que podem ocorrer. Por exemplo, se você tem um botão, geralmente deseja se conectar ao evento "clicked". Isso significa que quando o botão é clicado, o sinal é emitido. Em terceiro lugar, o argumento *callback* é o nome da função de retorno de chamada. Ele contém o código que é executado quando os sinais do tipo especificado são emitidos. Finalmente, o argumento *data* inclui todos os dados que devem ser passados quando o sinal é emitido. No entanto, esse argumento é completamente opcional e pode ser deixado de fora se não for necessário.

A função retorna um número que identifica esse par de sinal/retorno de chamada específico. É necessário desconectar de um sinal de modo que a função de retorno de chamada não seja chamada durante qualquer emissão futura ou atual do sinal ao qual está conectada.

```
widget.disconnect(handler_id)
```

Se você perdeu o "handler_id" por algum motivo (por exemplo, os manipuladores foram instalados usando `Gtk.Builder.connect_signals()`), você ainda pode desconectar um retorno de chamada específico usando a função `disconnect_by_func()`:

```
widget.disconnect_by_func(callback)
```

Os aplicativos devem se conectar ao sinal "destroy" da janela de nível superior. É emitido quando um objeto é destruído, portanto, quando um usuário solicita que uma janela de nível superior é fechada, o manipulador padrão para este sinal destrói a janela, mas não finaliza o aplicativo. Conectar o sinal "destroy" da janela de nível superior à função `Gtk.main_quit()` resultará no comportamento desejado.

```
window.connect("destroy", Gtk.main_quit)
```

Chamar `Gtk.main_quit()` faz o loop principal dentro do retorno de `Gtk.main()`.

3.2 Propriedades

Propriedades descrevem a configuração e o estado dos widgets. Quanto aos sinais, cada widget tem seu próprio conjunto particular de propriedades. Por exemplo, um botão tem a propriedade "label", que contém o texto do widget de etiqueta dentro do botão. Você pode especificar o nome e o valor de qualquer número de propriedades como argumentos nomeados ao criar uma instância de um widget. Para criar um rótulo alinhado à direita com o texto "Hello World" e um ângulo de 25 graus, use:

```
label = Gtk.Label(label="Hello World", angle=25, halign=Gtk.Align.END)
```

que é equivalente a

```
label = Gtk.Label()
label.set_label("Hello World")
label.set_angle(25)
label.set_halign(Gtk.Align.END)
```

Em vez de usar getters e setters, você também pode obter e definir as propriedades do gobject através da propriedade "props", como `widget.props.prop_name = valor`. Isto é equivalente ao mais detalhado `widget.get_property("prop-name")` e `widget.set_property("prop-name", valor)`.

Para ver quais propriedades estão disponíveis para um widget na versão em execução do GTK, você pode usar "dir" com a propriedade "props":

```
widget = Gtk.Box()
print(dir(widget.props))
```

Isto irá imprimir no console a lista de propriedades que um Gtk.Box possui.

CAPÍTULO 4

Como lidar com strings

Esta seção explica como as cadeias de caracteres são representadas no Python 2.x, no Python 3.x e no GTK+ e discute erros comuns que surgem ao trabalhar com strings.

4.1 Definições

Conceitualmente, uma string é uma lista de caracteres como "A", "B", "C" ou "É". **Caracteres** são representações abstratas e seu significado depende do idioma e do contexto em que são usados. O padrão Unicode descreve como os caracteres são representados por **pontos de código**. Por exemplo, os caracteres acima são representados com os pontos de código U+0041, U+0042, U+0043 e U+00C9, respectivamente. Basicamente, os pontos de código são números no intervalo de 0 a 0x10FFFF.

Como mencionado anteriormente, a representação de uma string como uma lista de pontos de código é abstrata. Para converter essa representação abstrata em uma sequência de bytes, a string Unicode deve ser **codificada**. A forma mais simples de codificação é ASCII e é executada da seguinte maneira:

1. Se o ponto de código for < 128, cada byte é o mesmo que o valor do ponto de código.
2. Se o ponto de código for 128 ou maior, a string Unicode não poderá ser representada nessa codificação. (Python dispara uma exceção `UnicodeEncodeError` neste caso.)

Embora a codificação ASCII seja simples de aplicar, ela só pode codificar 128 caracteres diferentes, o que não é suficiente. Uma das codificações mais usadas para resolver esse problema é o UTF-8 (ele pode manipular qualquer ponto de código Unicode). UTF significa "Formato de Transformação Unicode", do inglês "Unicode Transformation Format", e "8" significa que números de 8 bits são usados na codificação.

4.2 Python 2

4.2.1 Suporte a Unicode do Python 2.x

O Python 2 vem com dois tipos diferentes de objetos que podem ser usados para representar strings `str` e `unicode`. Instâncias do último são usadas para expressar strings Unicode, enquanto instâncias do tipo `str` são representações de byte (a string codificada). Sob o capô, Python representa strings Unicode como números inteiros de 16 ou 32 bits, dependendo de como o interpretador Python foi compilado. Strings Unicode podem ser convertidas em strings de 8 bits com `unicode.encode()`:

```
>>> unicode_string = u"Fu\u00dfb\u00e4lle"
>>> print unicode_string
Fußbälle
>>> type(unicode_string)
<type 'unicode'>
>>> unicode_string.encode("utf-8")
'Fu\xc3\x9fb\xc3\xa4lle'
```

As strings de 8 bits do Python têm um método `str.decode()` que interpreta a string usando a codificação fornecida:

```
>>> utf8_string = unicode_string.encode("utf-8")
>>> type(utf8_string)
<type 'str'>
>>> u2 = utf8_string.decode("utf-8")
>>> unicode_string == u2
True
```

Infelizmente, o Python 2.x permite que você misture `unicode` e `str` se a string de 8 bits contivesse apenas bytes de 7 bits (ASCII), mas obteria `UnicodeDecodeError` se contivesse valores não-ASCII:

```
>>> utf8_string = " sind rund"
>>> unicode_string + utf8_string
u'Fu\xdfb\xe4lle sind rund'
>>> utf8_string = " k\xc3\xb6nnten rund sein"
>>> print utf8_string
 könnten rund sein
>>> unicode_string + utf8_string
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 2:
ordinal not in range(128)
```

4.2.2 Unicode no GTK+

O GTK+ usa strings codificadas em UTF-8 para todo o texto. Isto significa que se você chamar um método que retorna uma string, você sempre obterá uma instância do tipo `str`. O mesmo se aplica aos métodos que esperam um ou mais strings como parâmetro, eles devem ser codificados em UTF-8. No entanto, por conveniência, o PyGObject converterá automaticamente qualquer instância `unicode` para `str` se fornecido como argumento:

```
>>> from gi.repository import Gtk
>>> label = Gtk.Label()
>>> unicode_string = u"Fu\u00dfb\u00e4lle"
```

(continua na próxima página)

(continuação da página anterior)

```
>>> label.set_text(unicode_string)
>>> txt = label.get_text()
>>> type(txt), txt
(<type 'str'>, 'Fu\xc3\x9fb\xc3\xa4lle')
>>> txt == unicode_string
__main__:1: UnicodeWarning: Unicode equal comparison failed to convert
both arguments to Unicode - interpreting them as being unequal
False
```

Observe o aviso no final. Apesar de chamarmos `Gtk.Label.set_text()` com uma instância de `unicode` como argumento, `Gtk.Label.get_text()` sempre retornará uma instância `str`. Assim, `txt` e `unicode_string` *não* são iguais.

Isto é especialmente importante se você quiser internacionalizar seu programa usando gettext. Você precisa ter certeza de que gettext retornará strings de 8 bits codificadas em UTF-8 para todos os idiomas. Em geral, recomenda-se não usar objetos `unicode` em aplicativos GTK+ e usar somente objetos codificados em UTF-8 `str`, já que o GTK+ não se integra totalmente a objetos `unicode`. Caso contrário, você teria que decodificar os valores de retorno para cadeias de caracteres Unicode cada vez que você chamar um método GTK+:

```
>>> txt = label.get_text().decode("utf-8")
>>> txt == unicode_string
True
```

4.3 Python 3

4.3.1 Suporte a Unicode do Python 3.x

Desde o Python 3.0, todas as strings são armazenadas como Unicode em uma instância do tipo `str`. Strings *codificadas*, por outro lado, são representadas como dados binários na forma de instâncias do tipo `bytes`. Conceitualmente, `str` refere-se a *texto*, enquanto `bytes` refere-se a *dados*. Use `str.encode()` para ir de `str` para `bytes` e `bytes.decode()` para ir de `bytes` para `str`.

Além disso, não é mais possível misturar strings Unicode com strings codificadas, porque resultará em um `TypeError`:

```
>>> text = "Fu\u00dfb\u00e4lle"
>>> data = b" sind rund"
>>> text + data
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'bytes' object to str implicitly
>>> text + data.decode("utf-8")
'Fußbälle sind rund'
>>> text.encode("utf-8") + data
b'Fu\xc3\x9fb\xc3\xa4lle sind rund'
```

4.3.2 Unicode no GTK+

Como consequência, as coisas são muito mais limpas e consistentes com o Python 3.x porque o PyGObject irá automaticamente codificar/decodificar para/de UTF-8 se você passar uma string para um método ou um método retornar uma string. Strings, ou *text*, sempre serão representados como instâncias de `str` apenas:

```
>>> from gi.repository import Gtk
>>> label = Gtk.Label()
>>> text = "Fu\u00dfb\u00e4lle"
>>> label.set_text(text)
>>> txt = label.get_text()
>>> type(txt), txt
(<class 'str'>, 'Fußbälle')
>>> txt == text
True
```

4.4 Referências

O que há de novo no Python 3.0 descreve os novos conceitos que distinguir claramente entre texto e dados.

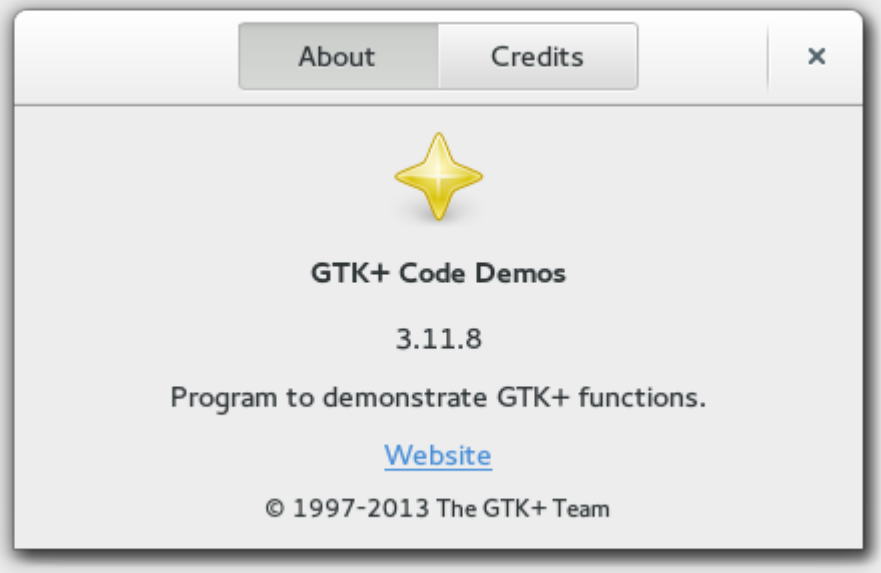
O Unicode HOWTO aborda o suporte do Python 2.x a Unicode e explica vários problemas que as pessoas comumente encontram ao tentar trabalhar com o Unicode.

O Unicode HOWTO for Python 3.x discute o suporte a Unicode no Python 3.x.

A tabela de codificação UTF-8 e os caracteres Unicode contém uma lista de pontos de código Unicode e sua respectiva codificação UTF-8.

CAPÍTULO 5

Galeria de widgets

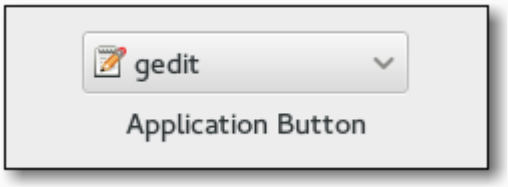
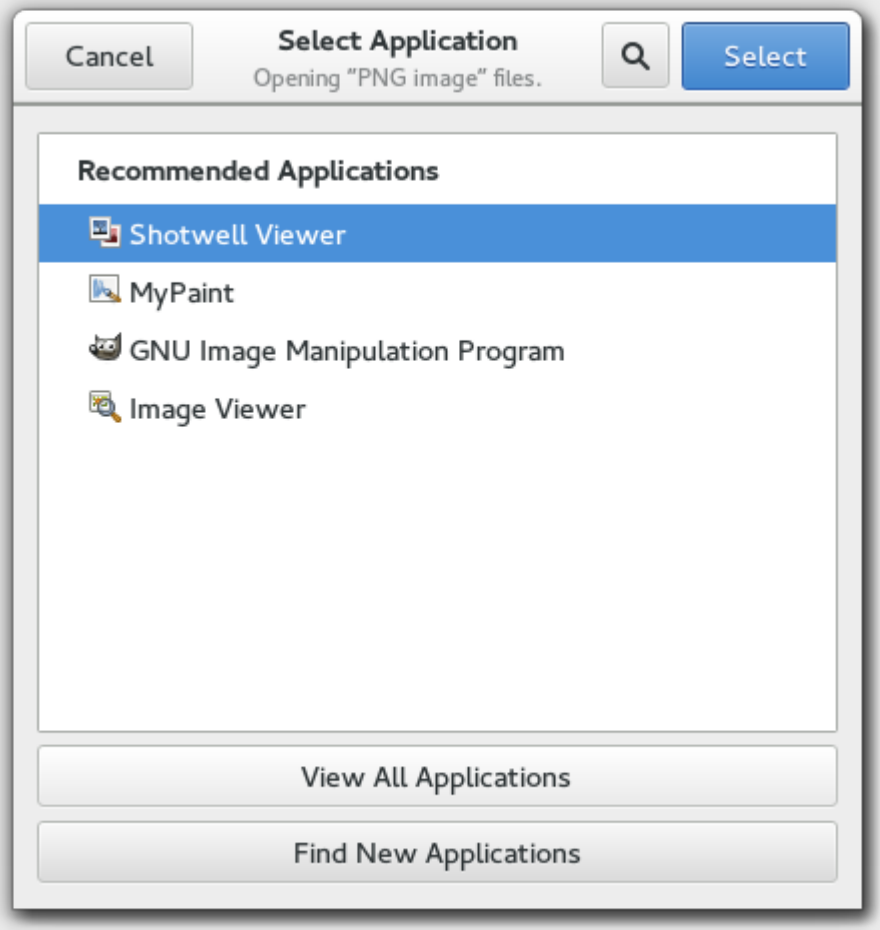


`Gtk.AboutDialog`

`Gtk.AccelLabel`

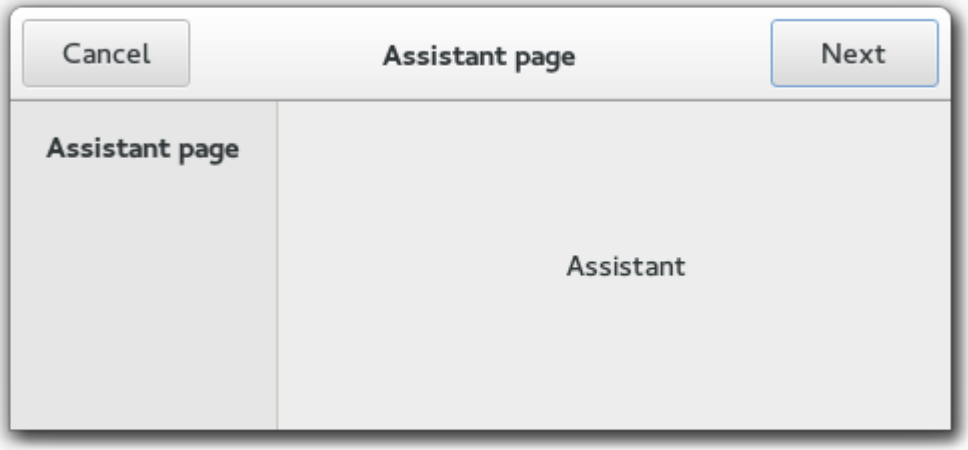
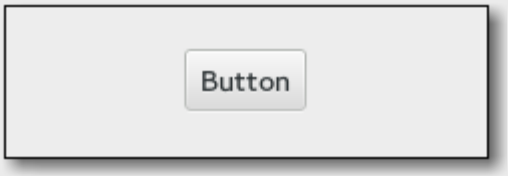
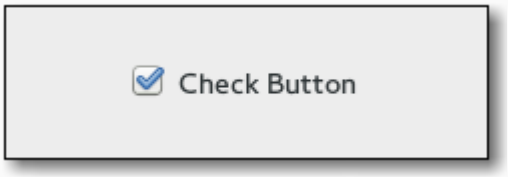
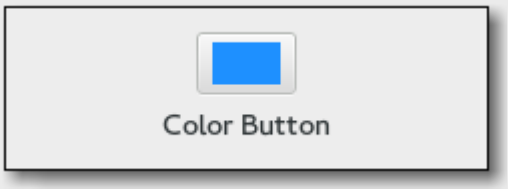
continua na p

Tabela 1 – continuação da página anterior

Gtk.ActionBar	
Gtk.AppChooserButton	
Gtk.AppChooserDialog	

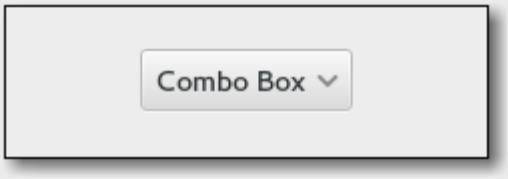
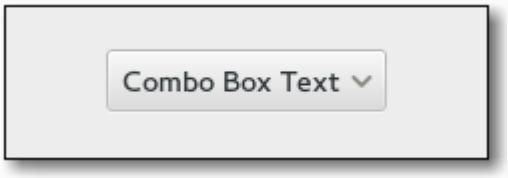
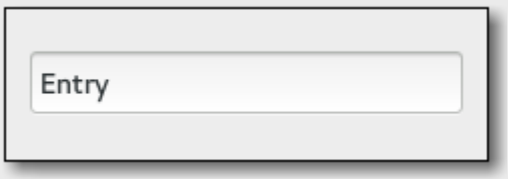
continua na p

Tabela 1 – continuação da página anterior

Gtk.Assistant	
Gtk.Button	
Gtk.CheckButton	
Gtk.ColorButton	

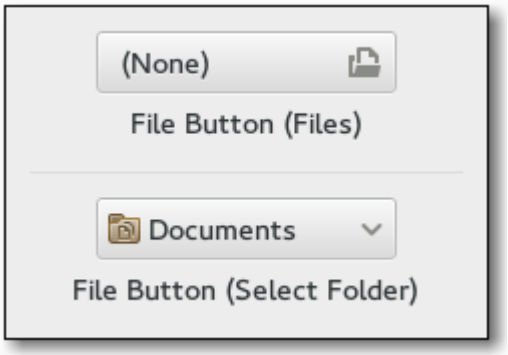
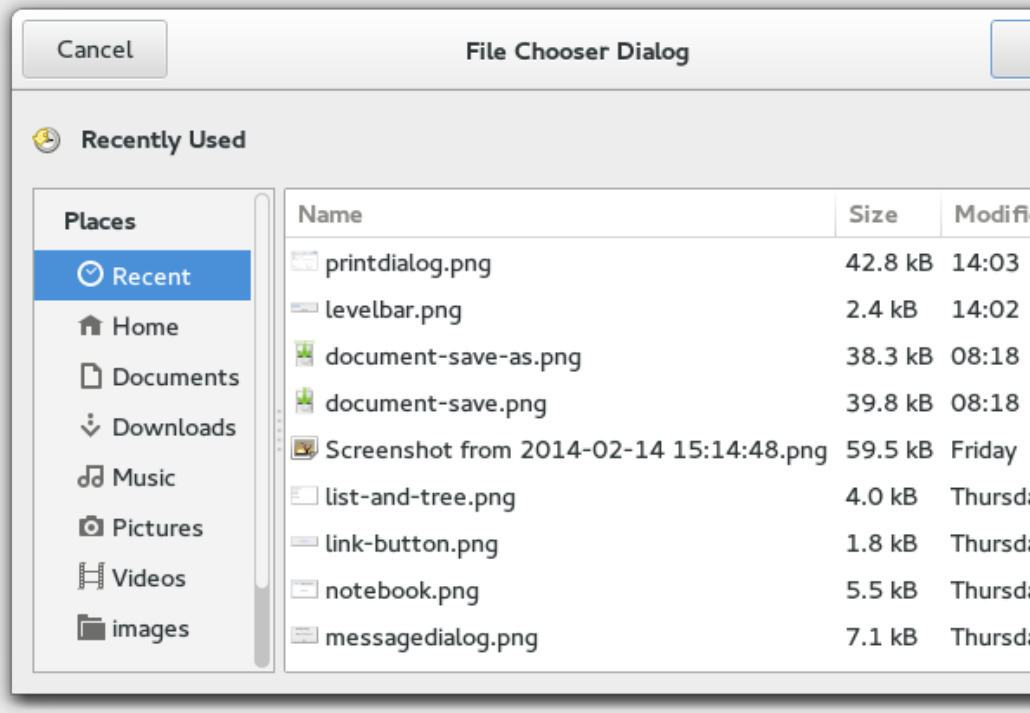
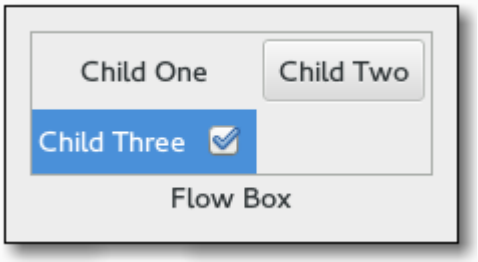
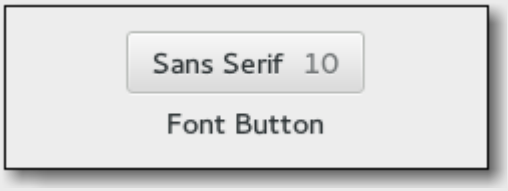
continua na p

Tabela 1 – continuação da página anterior

Gtk.ColorChooserDialog	
Gtk.ComboBox	
Gtk.ComboBoxText	
Gtk.Entry	

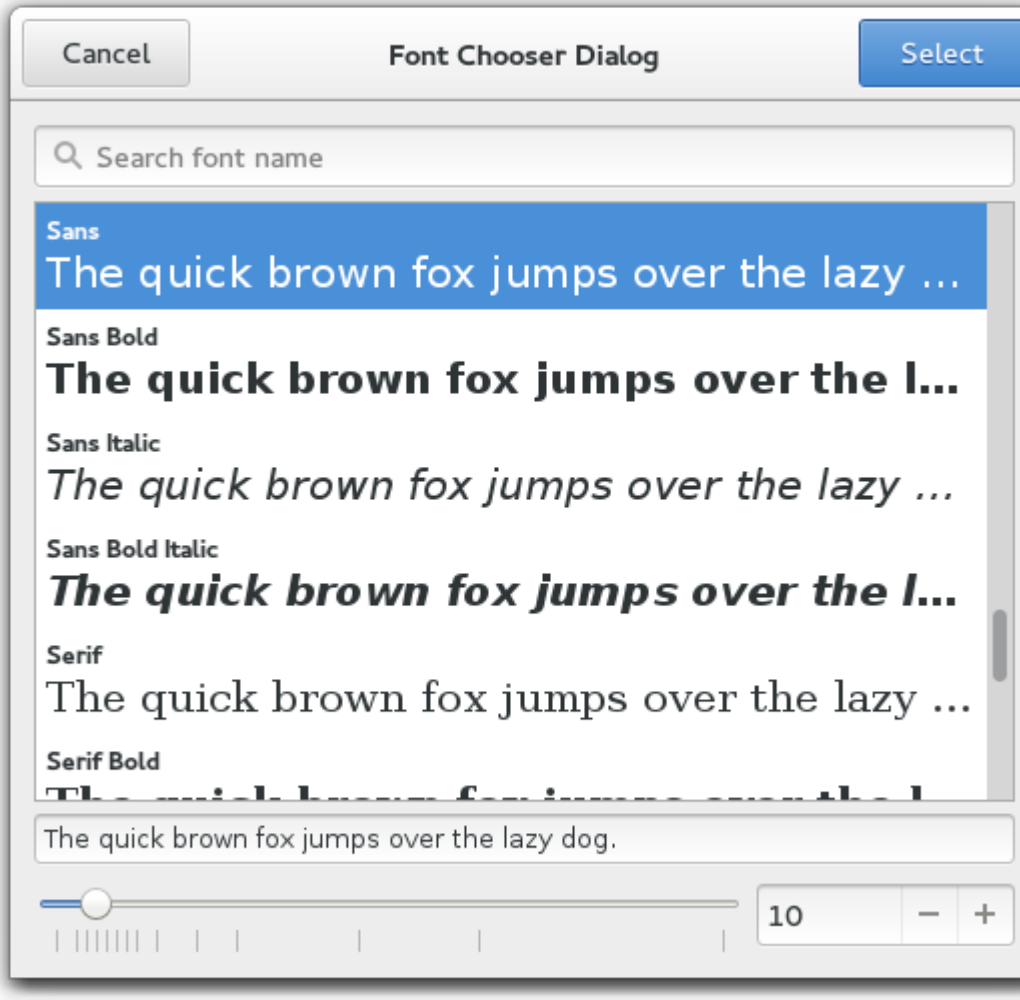
continua na p

Tabela 1 – continuação da página anterior

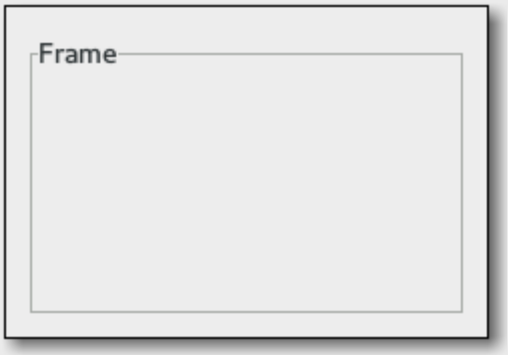
Gtk.FileChooserButton	
Gtk.FileChooserDialog	
Gtk.FlowBox	
Gtk.FontButton	

continua na p

Tabela 1 – continuação da página anterior



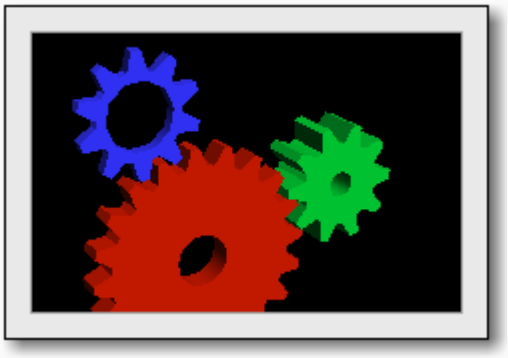
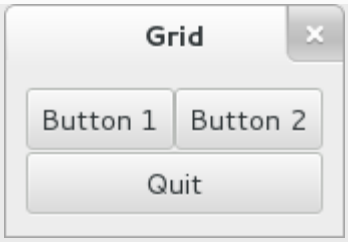
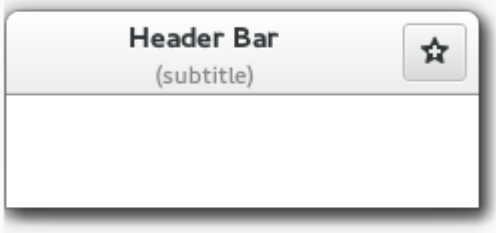
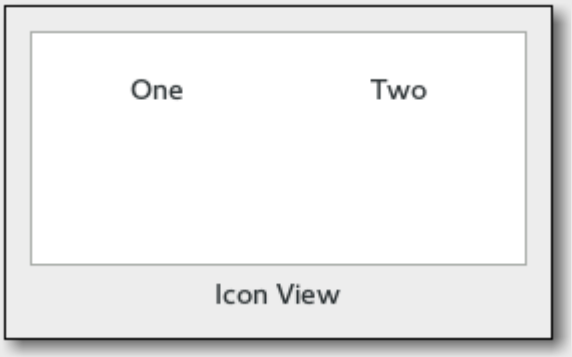
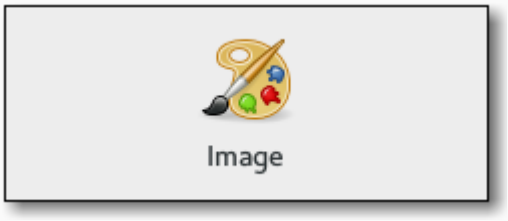
Gtk.FontChooserDialog



Gtk.Frame

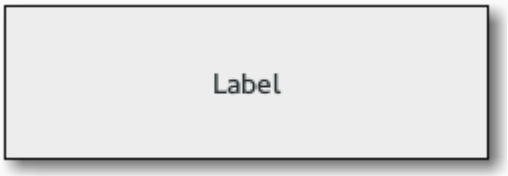
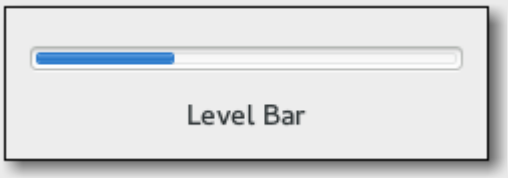
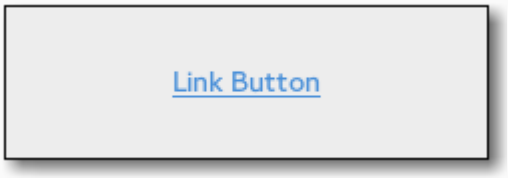
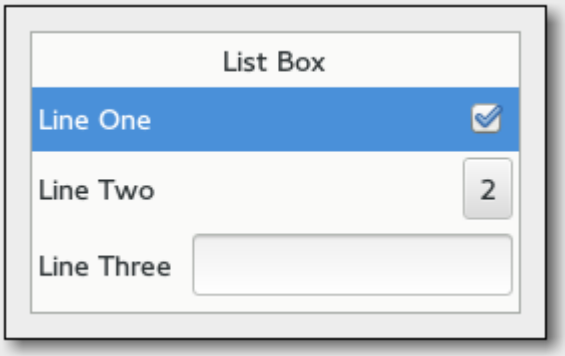
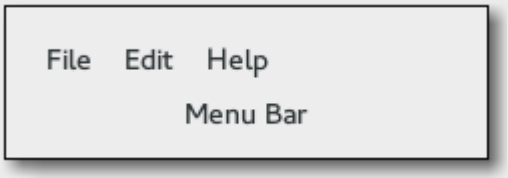
continua na p

Tabela 1 – continuação da página anterior

Gtk.GLArea	
Gtk.Grid	
Gtk.HeaderBar	
Gtk.IconView	
Gtk.Image	

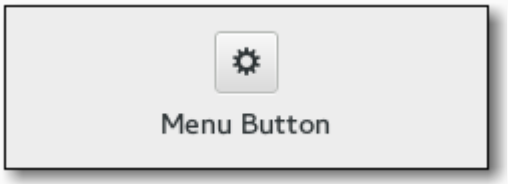
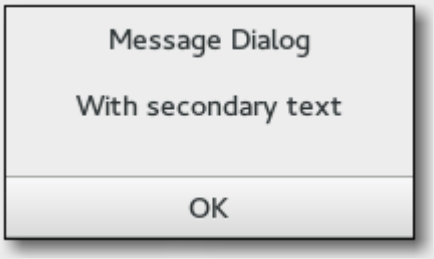
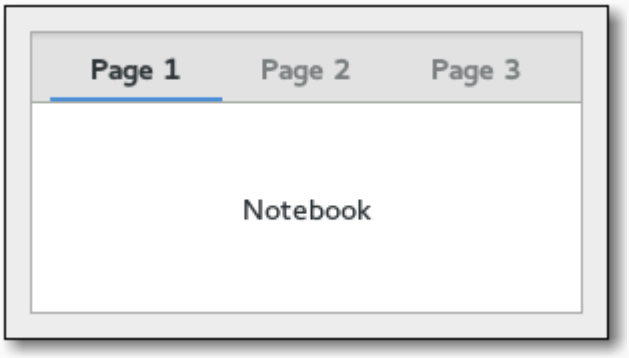
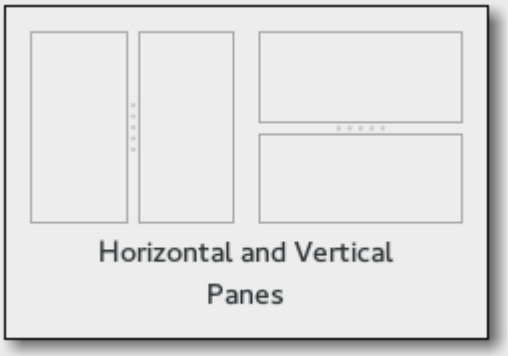
continua na p

Tabela 1 – continuação da página anterior

Gtk.InfoBar	
Gtk.Label	
Gtk.LevelBar	
Gtk.LinkButton	
Gtk.ListBox	
Gtk.LockButton	
Gtk.MenuBar	

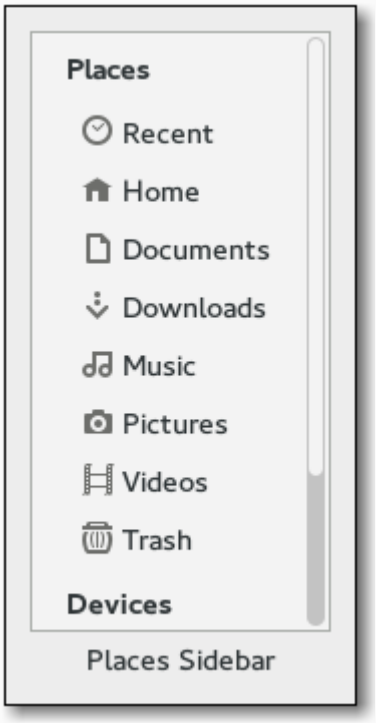
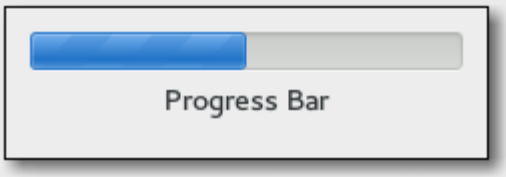
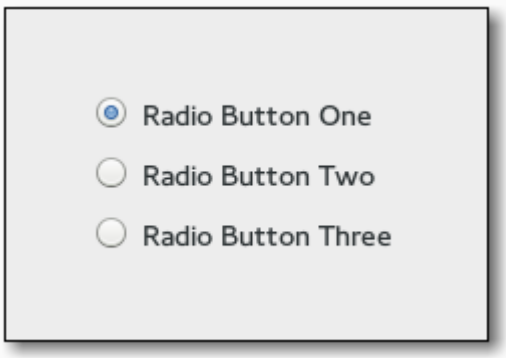
continua na p

Tabela 1 – continuação da página anterior

Gtk.MenuButton	
Gtk.MessageDialog	
Gtk.Notebook	
Gtk.Paned	

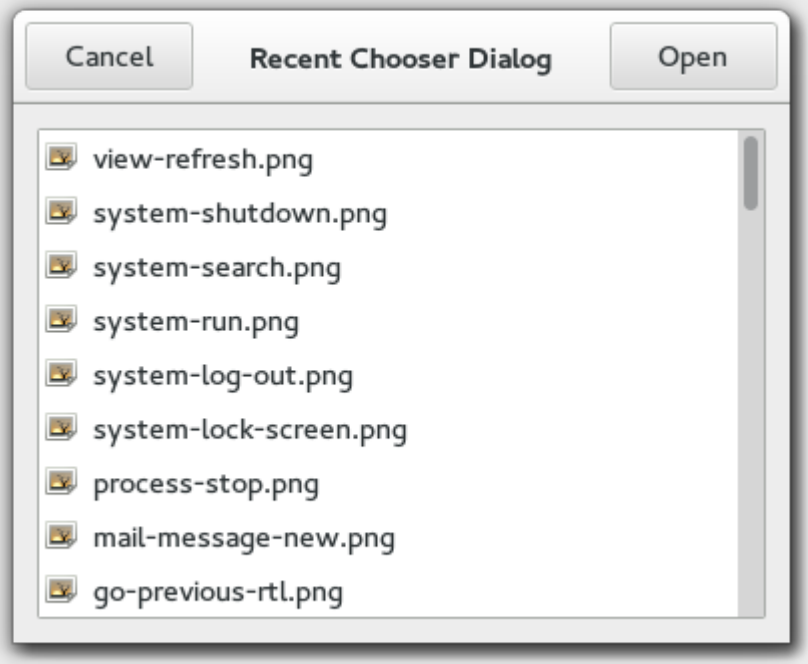
continua na p

Tabela 1 – continuação da página anterior

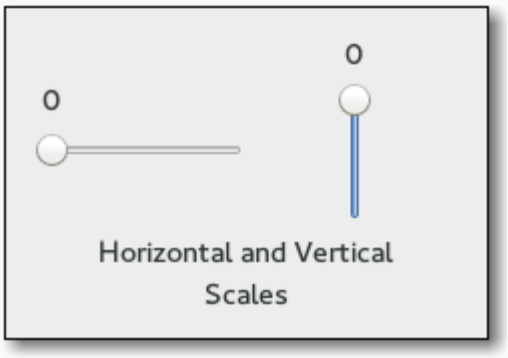
Gtk.PlacesSidebar	
Gtk.ProgressBar	
Gtk.RadioButton	

continua na p

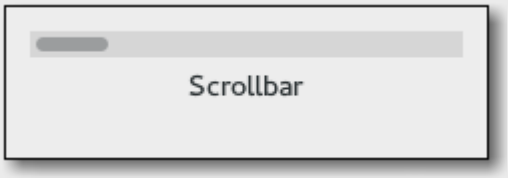
Tabela 1 – continuação da página anterior



Gtk.RecentChooserDialog



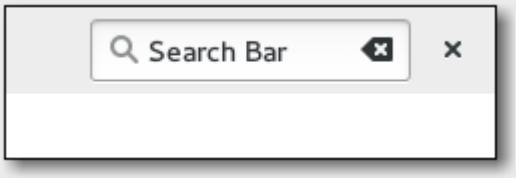
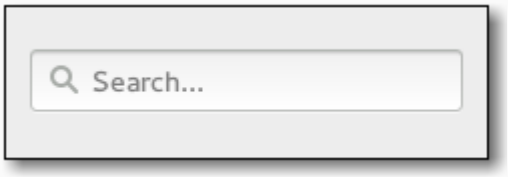
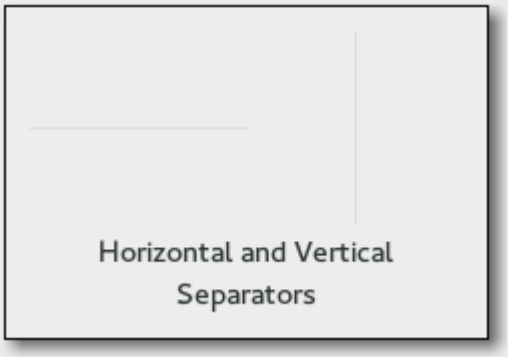
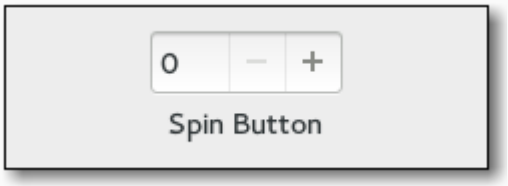
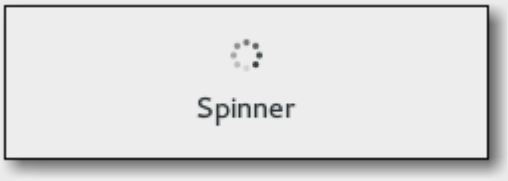
Gtk.Scale



Gtk.Scrollbar

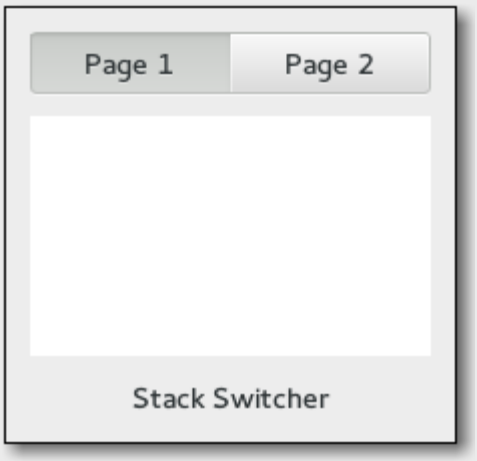
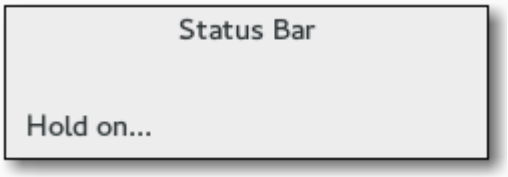
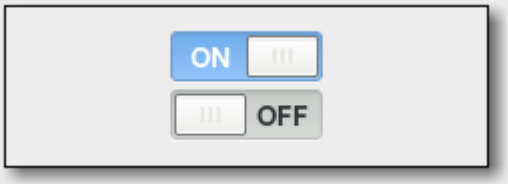
continua na p

Tabela 1 – continuação da página anterior

Gtk.ScrolledWindow	
Gtk.SearchBar	
Gtk.SearchEntry	
Gtk.Separator	
Gtk.SpinButton	
Gtk.Spinner	

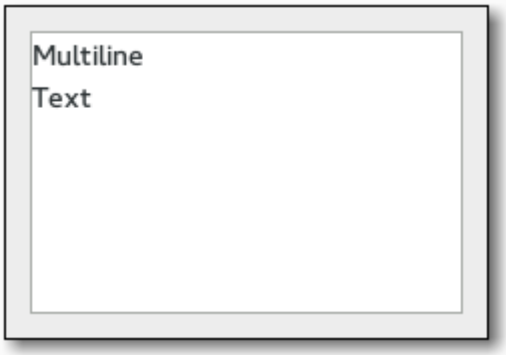
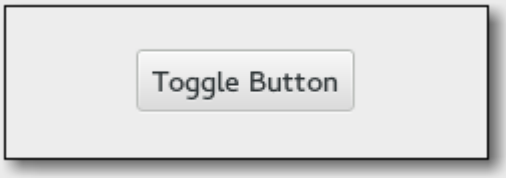
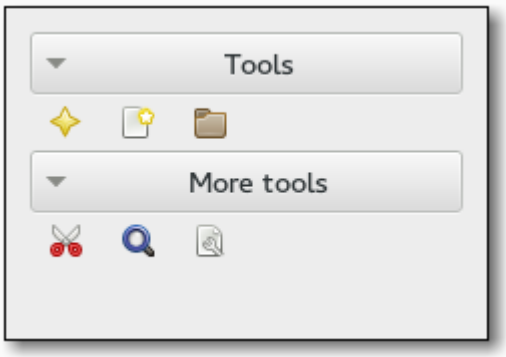
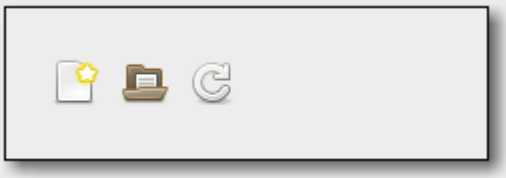
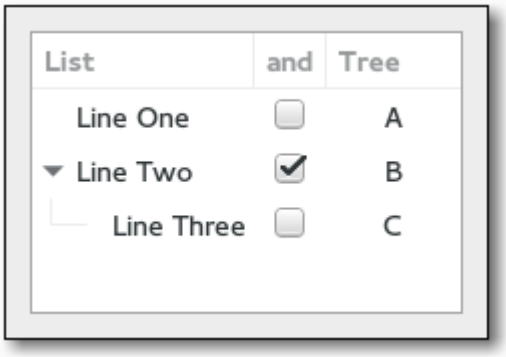
continua na p

Tabela 1 – continuação da página anterior

Gtk.Stack	
Gtk.StackSwitcher	
Gtk.Statusbar	
Gtk.Switch	

continua na p

Tabela 1 – continuação da página anterior

Gtk.TextView	
Gtk.ToggleButton	
Gtk.ToolPalette	
Gtk.Toolbar	
Gtk.TreeView	

continua na p

Tabela 1 – continuação da página anterior

Gtk.VolumeButton	
Gtk.Window	

CAPÍTULO 6

Contêineres de Layout

Enquanto muitos toolkits de GUI exigem que você coloque precisamente widgets em uma janela, usando posicionamento absoluto, o GTK+ usa uma abordagem diferente. Em vez de especificar a posição e o tamanho de cada widget na janela, você pode organizar seus widgets em linhas, colunas e/ou tabelas. O tamanho da sua janela pode ser determinado automaticamente, com base nos tamanhos dos widgets que ela contém. E os tamanhos dos widgets, por sua vez, são determinados pela quantidade de texto que eles contêm, ou os tamanhos mínimo e máximo que você especifica, e/ou como você solicitou que o espaço disponível seja compartilhado entre conjuntos de widgets. Você pode aperfeiçoar seu layout especificando a distância de preenchimento e os valores de centralização para cada um de seus widgets. O GTK+ usa todas essas informações para redimensionar e reposicionar tudo de maneira sensata e suave quando o usuário manipula a janela.

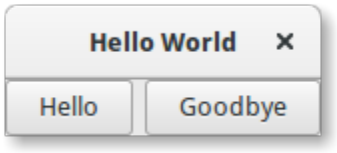
O GTK+ organiza widgets hierarquicamente, usando *contêineres*. Eles são invisíveis para o usuário-final e são inseridos em uma janela ou colocados entre si para os componentes do layout. Existem dois tipos de contêineres: contêineres filho único, todos descendentes de `Gtk.Bin`, e contêineres com vários filhos, que são descendentes de `Gtk.Container`. Os mais usados são caixas verticais ou horizontais (`Gtk.Box`) e grades (`Gtk.Grid`).

6.1 Box

As caixas *Box* são contêineres invisíveis nos quais podemos empacotar nossos widgets. Ao agrupar widgets em uma caixa horizontal, os objetos são inseridos horizontalmente da esquerda para a direita ou da direita para a esquerda, dependendo se `Gtk.Box.pack_start()` ou `Gtk.Box.pack_end()` for usado. Em uma caixa vertical, os widgets são empacotados de cima para baixo ou vice-versa. Você pode usar qualquer combinação de caixas dentro ou ao lado de outras caixas para criar o efeito desejado.

6.1.1 Exemplo

Vamos dar uma olhada em uma versão ligeiramente modificada do exemplo estendido com dois botões.




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class MyWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Hello World")

        self.box = Gtk.Box(spacing=6)
        self.add(self.box)

        self.button1 = Gtk.Button(label="Hello")
        self.button1.connect("clicked", self.on_button1_clicked)
        self.box.pack_start(self.button1, True, True, 0)

        self.button2 = Gtk.Button(label="Goodbye")
        self.button2.connect("clicked", self.on_button2_clicked)
        self.box.pack_start(self.button2, True, True, 0)

    def on_button1_clicked(self, widget):
        print("Hello")

    def on_button2_clicked(self, widget):
        print("Goodbye")


win = MyWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

Primeiro, criamos um contêiner de caixa orientado horizontalmente, onde 6 pixels são colocados entre os filhos. Esta caixa se torna o filho da janela de nível superior.

```
        self.box = Gtk.Box(spacing=6)
        self.add(self.box)
```

Posteriormente, adicionamos dois botões diferentes ao contêiner da caixa.

```
        self.button1 = Gtk.Button(label="Hello")
        self.button1.connect("clicked", self.on_button1_clicked)
        self.box.pack_start(self.button1, True, True, 0)
```

(continua na próxima página)

(continuação da página anterior)

```
        self.button2 = Gtk.Button(label="Goodbye")
        self.button2.connect("clicked", self.on_button2_clicked)
        self.box.pack_start(self.button2, True, True, 0)
```

Enquanto com os widgets `Gtk.Box.pack_start()` estão posicionados da esquerda para a direita, `Gtk.Box.pack_end()` os posiciona da direita para a esquerda.

6.2 Grade

`Gtk.Grid` é um contêiner que organiza seus widgets filhos em linhas e colunas, mas você não precisa especificar as dimensões no construtor. Os filhos são adicionados usando `Gtk.Grid.attach()`. Eles podem abranger várias linhas ou colunas. O método `Gtk.Grid.attach()` usa cinco parâmetros:

1. O parâmetro `child` é o `Gtk.Widget` para adicionar.
2. `left` é o número da coluna para anexar o lado esquerdo de `child` em.
3. `top` indica o número da linha para anexar o lado superior do `child`.
4. `width` e `height` indicam o número de colunas que o `child` irá abranger, e o número de linhas que o `child` irá abranger, respectivamente.

Também é possível adicionar um child ao lado de um child existente, usando `Gtk.Grid.attach_next_to()`, que também usa cinco parâmetros:

1. `child` é o `Gtk.Widget` para adicionar, como acima.
2. `sibling` é um widget filho existente de `self` (uma instância `Gtk.Grid`) ou `None`. O widget `child` será colocado próximo ao `sibling`, ou se `sibling` for `None`, no início ou no final da grade.
3. `side` é um `Gtk.PositionType` indicando o lado do `sibling` que `child` é posicionado ao lado de.
4. `width` e `height` indicam o número de colunas e linhas que o widget `child` abrangerá, respectivamente.

Finalmente, `Gtk.Grid` pode ser usado como `Gtk.Box` usando apenas `Gtk.Grid.add()`, que colocará os filhos um ao lado do outro na direção determinada pela propriedade "orientation" (o padrão é `Gtk.Orientation.HORIZONTAL`).

6.2.1 Exemplo


```
1 import gi
2
3 gi.require_version("Gtk", "3.0")
4 from gi.repository import Gtk
5
```

(continua na próxima página)

(continuação da página anterior)

```

class GridWindow(Gtk.Window):
    def __init__(self):

        super().__init__(title="Grid Example")

        button1 = Gtk.Button(label="Button 1")
        button2 = Gtk.Button(label="Button 2")
        button3 = Gtk.Button(label="Button 3")
        button4 = Gtk.Button(label="Button 4")
        button5 = Gtk.Button(label="Button 5")
        button6 = Gtk.Button(label="Button 6")

        grid = Gtk.Grid()
        grid.add(button1)
        grid.attach(button2, 1, 0, 2, 1)
        grid.attach_next_to(button3, button1, Gtk.PositionType.BOTTOM, 1, 2)
        grid.attach_next_to(button4, button3, Gtk.PositionType.RIGHT, 2, 1)
        grid.attach(button5, 1, 2, 1, 1)
        grid.attach_next_to(button6, button5, Gtk.PositionType.RIGHT, 1, 1)

        self.add(grid)


win = GridWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

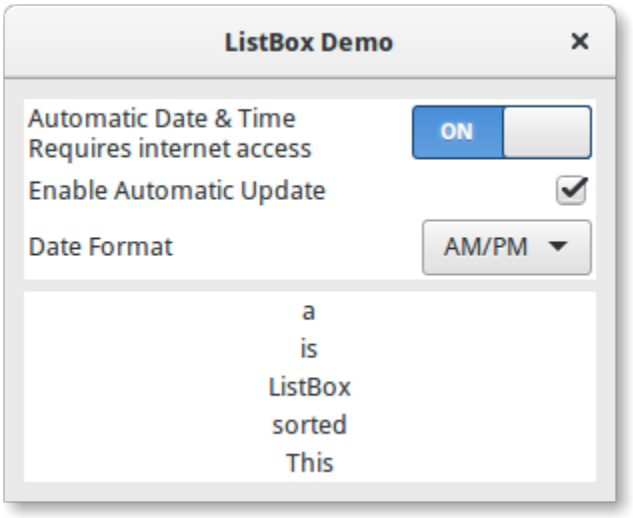
6.3 ListBox

A `Gtk.ListBox` é um contêiner vertical que contém `Gtk.ListBoxRow` filhos. Essas linhas podem ser classificadas e filtradas dinamicamente e os cabeçalhos podem ser adicionados dinamicamente, dependendo do conteúdo da linha. Também permite navegação e seleção de teclado e mouse como uma lista típica.

Usar `Gtk.ListBox` é muitas vezes uma alternativa para `Gtk.TreeView`, especialmente quando o conteúdo da lista tem um layout mais complicado do que o permitido por um `Gtk.CellRenderer`, ou quando o conteúdo é interativo (por exemplo, tem um botão).

Embora um `Gtk.ListBox` deva ter apenas `Gtk.ListBoxRow` filhos, você pode adicionar qualquer tipo de widget a ele via `Gtk.Container.add()` e um `Gtk.ListBoxRow` widget será automaticamente inserido entre a lista e o widget.

6.3.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class ListBoxRowWithData(Gtk.ListBoxRow):
    def __init__(self, data):
        super().__init__()
        self.data = data
        self.add(Gtk.Label(label=data))


class ListBoxWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="ListBox Demo")
        self.set_border_width(10)

        box_outer = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(box_outer)

        listbox = Gtk.ListBox()
        listbox.set_selection_mode(Gtk.SelectionMode.NONE)
        box_outer.pack_start(listbox, True, True, 0)

        row = Gtk.ListBoxRow()
        hbox = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=50)
        row.add(hbox)
        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
        hbox.pack_start(vbox, True, True, 0)

        label1 = Gtk.Label(label="Automatic Date & Time", xalign=0)
        label2 = Gtk.Label(label="Requires internet access", xalign=0)
```

(continua na próxima página)

(continuação da página anterior)

```
        vbox.pack_start(label1, True, True, 0)
        vbox.pack_start(label2, True, True, 0)

        switch = Gtk.Switch()
        switch.props.valign = Gtk.Align.CENTER
        hbox.pack_start(switch, False, True, 0)

        listbox.add(row)

        row = Gtk.ListBoxRow()
        hbox = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=50)
        row.add(hbox)
        label = Gtk.Label(label="Enable Automatic Update", xalign=0)
        check = Gtk.CheckButton()
        hbox.pack_start(label, True, True, 0)
        hbox.pack_start(check, False, True, 0)

        listbox.add(row)

        row = Gtk.ListBoxRow()
        hbox = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL, spacing=50)
        row.add(hbox)
        label = Gtk.Label(label="Date Format", xalign=0)
        combo = Gtk.ComboBoxText()
        combo.insert(0, "0", "24-hour")
        combo.insert(1, "1", "AM/PM")
        hbox.pack_start(label, True, True, 0)
        hbox.pack_start(combo, False, True, 0)

        listbox.add(row)

        listbox_2 = Gtk.ListBox()
        items = "This is a sorted ListBox Fail".split()

        for item in items:
            listbox_2.add(ListBoxRowWithData(item))

        def sort_func(row_1, row_2, data, notify_destroy):
            return row_1.data.lower() > row_2.data.lower()

        def filter_func(row, data, notify_destroy):
            return False if row.data == "Fail" else True

        listbox_2.set_sort_func(sort_func, None, False)
        listbox_2.set_filter_func(filter_func, None, False)

        def on_row_activated(listbox_widget, row):
            print(row.data)

        listbox_2.connect("row-activated", on_row_activated)

        box_outer.pack_start(listbox_2, True, True, 0)
```

(continua na próxima página)

(continuação da página anterior)

```
        listbox_2.show_all()


win = ListBoxWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

6.4 Stack e StackSwitcher

A `Gtk.Stack` é um contêiner que mostra apenas um de seus filhos por vez. Em contraste com `Gtk.Notebook`, `Gtk.Stack` não fornece um meio para os usuários alterarem o filho visível. Em vez disso, o widget `Gtk.StackSwitcher` pode ser usado com `Gtk.Stack` para fornecer essa funcionalidade.

Transições entre páginas podem ser animadas como slides ou fades. Isso pode ser controlado com `Gtk.Stack.set_transition_type()`. Essas animações respeitam a configuração "gtk-enable-animations".

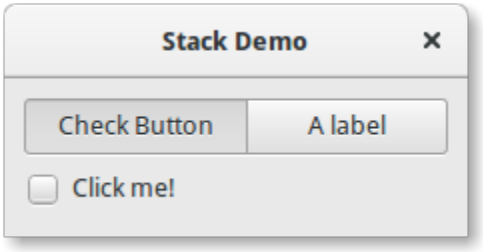
A velocidade de transição pode ser ajustada com `Gtk.Stack.set_transition_duration()`

O widget `Gtk.StackSwitcher` atua como um controlador para um `Gtk.Stack`; Ele mostra uma linha de botões para alternar entre as várias páginas do widget de pilha associado.

Todo o conteúdo para os botões vem das propriedades filho do `Gtk.Stack`.

É possível associar múltiplos widgets `Gtk.StackSwitcher` com o mesmo widget `Gtk.Stack`.

6.4.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class StackWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Stack Demo")
        self.set_border_width(10)

        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(vbox)

```

(continua na próxima página)

(continuação da página anterior)

```
        stack = Gtk.Stack()
        stack.set_transition_type(Gtk.StackTransitionType.SLIDE_LEFT_RIGHT)
        stack.set_transition_duration(1000)

        checkbutton = Gtk.CheckButton(label="Click me!")
        stack.add_titled(checkbutton, "check", "Check Button")

        label = Gtk.Label()
        label.set_markup("<big>A fancy label</big>")
        stack.add_titled(label, "label", "A label")

        stack_switcher = Gtk.StackSwitcher()
        stack_switcher.set_stack(stack)
        vbox.pack_start(stack_switcher, True, True, 0)
        vbox.pack_start(stack, True, True, 0)


win = StackWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

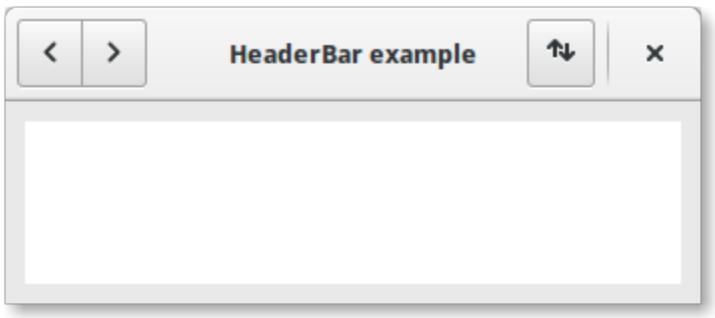
6.5 HeaderBar

A `Gtk.HeaderBar` é semelhante a uma horizontal `Gtk.Box`, permite colocar filhos no início ou no final. Além disso, permite que um título seja exibido. O título será centrado em relação à largura da caixa, mesmo que os filhos de ambos os lados ocupem diferentes quantidades de espaço.

Como o GTK+ agora tem suporte a Client Side Decoration, um `Gtk.HeaderBar` pode ser usado no lugar da barra de título (que é renderizada pelo Gerenciador de Janelas).

A `Gtk.HeaderBar` geralmente está localizado na parte superior de uma janela e deve conter controles comumente usados que afetam o conteúdo abaixo. Eles também fornecem acesso a controles de janela, incluindo o botão de fechar janela e o menu de janela.

6.5.1 Exemplo




```
import gi

```

(continua na próxima página)

(continuação da página anterior)

```
gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, Gio


class HeaderBarWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="HeaderBar Demo")
        self.set_border_width(10)
        self.set_default_size(400, 200)

        hb = Gtk.HeaderBar()
        hb.set_show_close_button(True)
        hb.props.title = "HeaderBar example"
        self.set_titlebar(hb)

        button = Gtk.Button()
        icon = Gio.ThemedIcon(name="mail-send-receive-symbolic")
        image = Gtk.Image.new_from_gicon(icon, Gtk.IconSize.BUTTON)
        button.add(image)
        hb.pack_end(button)

        box = Gtk.Box(orientation=Gtk.Orientation.HORIZONTAL)
        Gtk.StyleContext.add_class(box.get_style_context(), "linked")

        button = Gtk.Button()
        button.add(
            Gtk.Arrow(arrow_type=Gtk.ArrowType.LEFT, shadow_type=Gtk.ShadowType.NONE)
        )
        box.add(button)

        button = Gtk.Button.new_from_icon_name("pan-end-symbolic", Gtk.IconSize.MENU)
        box.add(button)

        hb.pack_start(box)

        self.add(Gtk.TextView())


win = HeaderBarWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

6.6 FlowBox

Nota: Este exemplo requer pelo menos GTK+ 3.12.

A `Gtk.FlowBox` é um contêiner que posiciona widgets filhos em sequência de acordo com sua orientação.

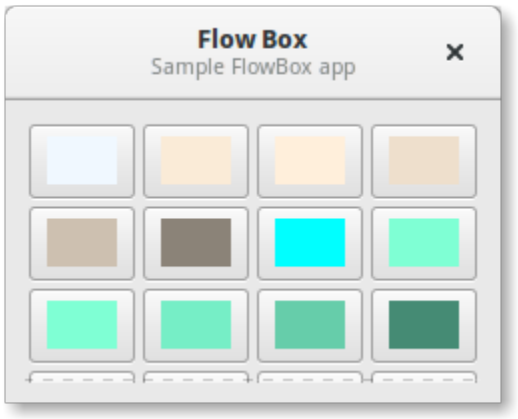
Por exemplo, com a orientação horizontal, os widgets serão organizados da esquerda para a direita, iniciando uma nova linha na linha anterior, quando necessário. Reduzir a largura neste caso exigirá mais linhas, portanto, uma altura maior será solicitada.

Da mesma forma, com a orientação vertical, os widgets serão organizados de cima para baixo, iniciando uma nova coluna à direita quando necessário. Reduzir a altura exigirá mais colunas, portanto será solicitada uma largura maior.

Os filhos de uma `Gtk.FlowBox` podem ser classificados e filtrados dinamicamente.

Embora uma `Gtk.FlowBox` deva ter apenas filhos `Gtk.FlowBoxChild`, você pode adicionar qualquer tipo de widget a ele via `Gtk.Container.add()`, e um widget `Gtk.FlowBoxChild` será automaticamente inserido entre a caixa e o widget.

6.6.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, Gdk


class FlowBoxWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="FlowBox Demo")
        self.set_border_width(10)
        self.set_default_size(300, 250)

        header = Gtk.HeaderBar(title="Flow Box")
        header.set_subtitle("Sample FlowBox app")
        header.props.show_close_button = True

        self.set_titlebar(header)
```

(continua na próxima página)

(continuação da página anterior)

```
        scrolled = Gtk.ScrolledWindow()
        scrolled.set_policy(Gtk.PolicyType.NEVER, Gtk.PolicyType.AUTOMATIC)

        flowbox = Gtk.FlowBox()
        flowbox.set_valign(Gtk.Align.START)
        flowbox.set_max_children_per_line(30)
        flowbox.set_selection_mode(Gtk.SelectionMode.NONE)

        self.create_flowbox(flowbox)

        scrolled.add(flowbox)

        self.add(scrolled)
        self.show_all()

    def on_draw(self, widget, cr, data):
        context = widget.get_style_context()

        width = widget.get_allocated_width()
        height = widget.get_allocated_height()
        Gtk.render_background(context, cr, 0, 0, width, height)

        r, g, b, a = data["color"]
        cr.set_source_rgba(r, g, b, a)
        cr.rectangle(0, 0, width, height)
        cr.fill()

    def color_swatch_new(self, str_color):
        rgba = Gdk.RGBA()
        rgba.parse(str_color)

        button = Gtk.Button()

        area = Gtk.DrawingArea()
        area.set_size_request(24, 24)
        area.connect("draw", self.on_draw, {"color": rgba})

        button.add(area)

        return button

    def create_flowbox(self, flowbox):
        colors = [
            "AliceBlue",
            "AntiqueWhite",
            "AntiqueWhite1",
            "AntiqueWhite2",
            "AntiqueWhite3",
            "AntiqueWhite4",
            "aqua",
            "aquamarine",
```

(continua na próxima página)

(continuação da página anterior)

```
            "aquamarine1",
            "aquamarine2",
            "aquamarine3",
            "aquamarine4",
            "azure",
            "azure1",
            "azure2",
            "azure3",
            "azure4",
            "beige",
            "bisque",
            "bisque1",
            "bisque2",
            "bisque3",
            "bisque4",
            "black",
            "BlanchedAlmond",
            "blue",
            "blue1",
            "blue2",
            "blue3",
            "blue4",
            "BlueViolet",
            "brown",
            "brown1",
            "brown2",
            "brown3",
            "brown4",
            "burlywood",
            "burlywood1",
            "burlywood2",
            "burlywood3",
            "burlywood4",
            "CadetBlue",
            "CadetBlue1",
            "CadetBlue2",
            "CadetBlue3",
            "CadetBlue4",
            "chartreuse",
            "chartreuse1",
            "chartreuse2",
            "chartreuse3",
            "chartreuse4",
            "chocolate",
            "chocolate1",
            "chocolate2",
            "chocolate3",
            "chocolate4",
            "coral",
            "coral1",
            "coral2",
            "coral3",
```

(continua na próxima página)

(continuação da página anterior)

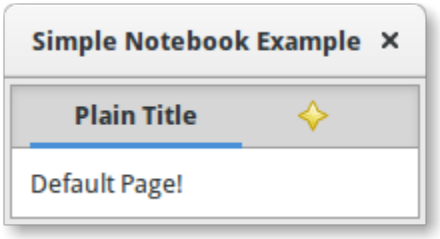
```
            "coral4",
        ]

        for color in colors:
            button = self.color_swatch_new(color)
            flowbox.add(button)


win = FlowBoxWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

6.7 Notebook

O widget `Gtk.Notebook` é um `Gtk.Container` cujos filhos são páginas que podem ser alternadas usando rótulos de guias ao longo de uma borda.

Existem muitas opções de configuração para o GtkNotebook. Entre outras coisas, você pode escolher em qual borda as abas aparecem (veja `Gtk.Notebook.set_tab_pos()`), se houver muitas abas para caber no notebook, elas devem ser maiores ou setas de rolagem serão adicionadas (veja `Gtk.Notebook.set_scrollable()`), e se haverá um menu pop-up que permita aos usuários trocar de página (veja `Gtk.Notebook.popup_enable()`, `Gtk.Notebook.popup_disable()`).

6.7.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class MyWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Simple Notebook Example")
        self.set_border_width(3)

        self.notebook = Gtk.Notebook()
        self.add(self.notebook)

```

(continua na próxima página)

(continuação da página anterior)

```
        self.page1 = Gtk.Box()
        self.page1.set_border_width(10)
        self.page1.add(Gtk.Label(label="Default Page!"))
        self.notebook.append_page(self.page1, Gtk.Label(label="Plain Title"))

        self.page2 = Gtk.Box()
        self.page2.set_border_width(10)
        self.page2.add(Gtk.Label(label="A page with an image for a Title."))
        self.notebook.append_page(
            self.page2, Gtk.Image.new_from_icon_name("help-about", Gtk.IconSize.MENU)
        )


win = MyWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 7

Label

Os rótulos (labels) são o principal método de colocar texto não editável nas janelas, por exemplo, para colocar um título ao lado de um widget `Gtk.Entry`. Você pode especificar o texto no construtor, ou mais tarde com os métodos `Gtk.Label.set_text()` ou `Gtk.Label.set_markup()`.

A largura da etiqueta será ajustada automaticamente. Você pode produzir rótulos de várias linhas colocando as quebras de linha ("\n") na sequência de rótulos.

Os rótulos podem ser feitos selecionáveis com `Gtk.Label.set_selectable()`. Rótulos selecionáveis permitem que o usuário copie o conteúdo do rótulo para a área de transferência. Somente os rótulos que contêm informações úteis para copiar, como mensagens de erro, devem ser selecionáveis.

O texto do rótulo pode ser justificado usando o método `Gtk.Label.set_justify()`. O widget também é capaz de quebra automática de palavras, que pode ser ativado com `Gtk.Label.set_line_wrap()`.

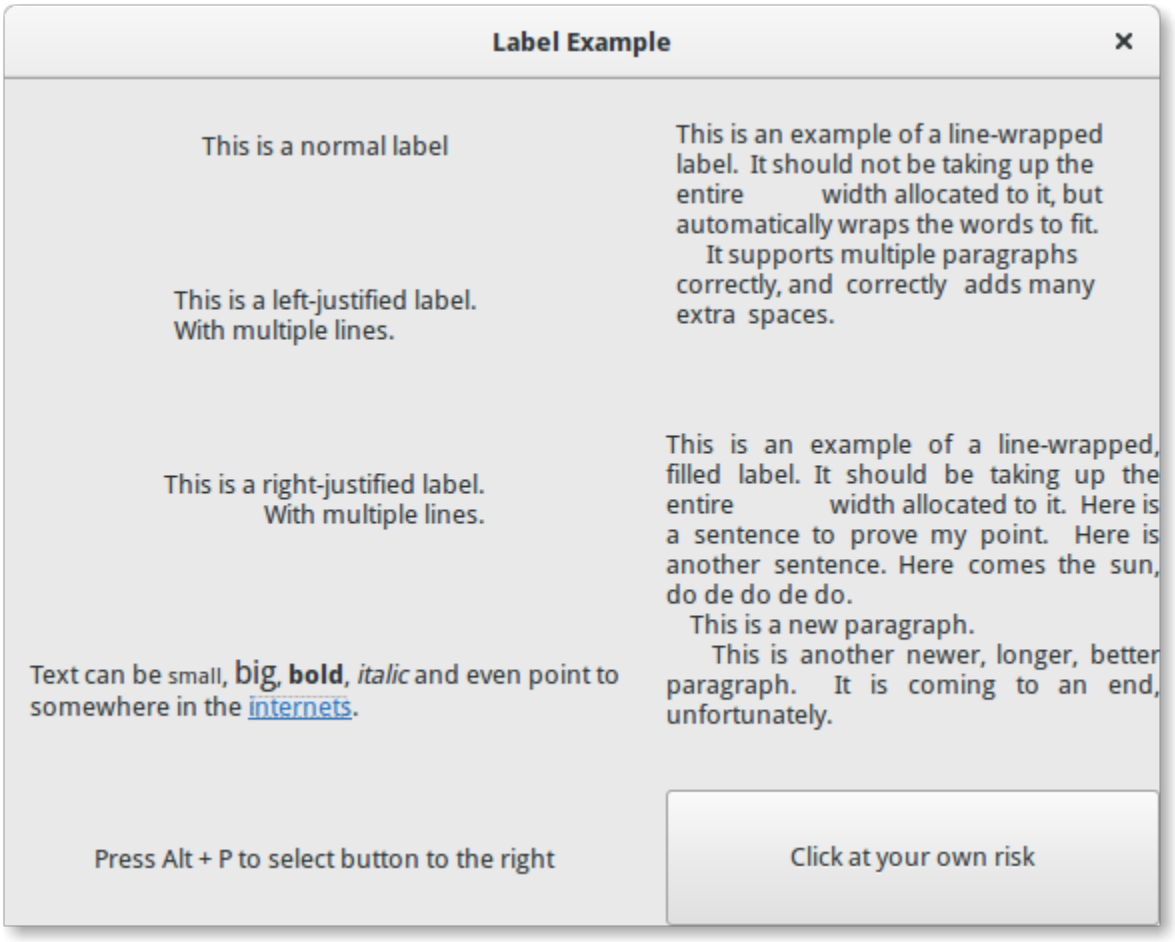
`Gtk.Label` possui suporte algumas a formatações simples, por exemplo, permitindo que você torne algum texto em negrito, colorido ou maior. Você pode fazer isso fornecendo uma string para `Gtk.Label.set_markup()`, usando a sintaxe de marcação do Pango[1]. Por exemplo, `<b>texto em negrito</b> e <s>texto tachado</s>`. Além disso, `Gtk.Label` possui suporte a hiperlinks clicáveis. A marcação para links é emprestada do HTML, usando os atributos de a com href e title. O GTK+ renderiza links semelhantes ao modo como aparecem nos navegadores da web, com texto colorido e sublinhado. O atributo de título é exibido como uma dica de ferramenta no link.

```
label.set_markup("Go to <a href=\"https://www.gtk.org\" "
                 "title=\"Our website\">GTK+ website</a> for more")
```

Os rótulos podem conter *mnemônicos*. Os mnemônicos são caracteres sublinhados no rótulo, usados para navegação pelo teclado. Os mnemônicos são criados fornecendo uma string com um sublinhado antes do caractere mnemônico, como "_File", para as funções `Gtk.Label.new_with_mnemonic()` ou `Gtk.Label.set_text_with_mnemonic()`. Os mnemônicos ativam automaticamente qualquer widget ativável em que o rótulo esteja dentro, como um `Gtk.Button`; se o rótulo não estiver dentro do widget de destino do mnemônico, você deve informar o rótulo sobre o destino usando `Gtk.Label.set_mnemonic_widget()`.

[1] Pango Markup Syntax, https://docs.gtk.org/Pango/pango_markup.html

7.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class LabelWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Label Example")

        hbox = Gtk.Box(spacing=10)
        hbox.set_homogeneous(False)
        vbox_left = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)
        vbox_left.set_homogeneous(False)
        vbox_right = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)
        vbox_right.set_homogeneous(False)

        hbox.pack_start(vbox_left, True, True, 0)
        hbox.pack_start(vbox_right, True, True, 0)

```

(continua na próxima página)

(continuação da página anterior)

```
        label = Gtk.Label(label="This is a normal label")
        vbox_left.pack_start(label, True, True, 0)

        label = Gtk.Label()
        label.set_text("This is a left-justified label.\nWith multiple lines.")
        label.set_justify(Gtk.Justification.LEFT)
        vbox_left.pack_start(label, True, True, 0)

        label = Gtk.Label(
            label="This is a right-justified label.\nWith multiple lines."
        )
        label.set_justify(Gtk.Justification.RIGHT)
        vbox_left.pack_start(label, True, True, 0)

        label = Gtk.Label(
            label="This is an example of a line-wrapped label.  It "
            "should not be taking up the entire              "
            "width allocated to it, but automatically "
            "wraps the words to fit.\n"
            "     It supports multiple paragraphs correctly, "
            "and  correctly   adds "
            "many          extra  spaces. "
        )
        label.set_line_wrap(True)
        label.set_max_width_chars(32)
        vbox_right.pack_start(label, True, True, 0)

        label = Gtk.Label(
            label="This is an example of a line-wrapped, filled label. "
            "It should be taking "
            "up the entire              width allocated to it.  "
            "Here is a sentence to prove "
            "my point.  Here is another sentence. "
            "Here comes the sun, do de do de do.\n"
            "    This is a new paragraph.\n"
            "    This is another newer, longer, better "
            "paragraph.  It is coming to an end, "
            "unfortunately."
        )
        label.set_line_wrap(True)
        label.set_justify(Gtk.Justification.FILL)
        label.set_max_width_chars(32)
        vbox_right.pack_start(label, True, True, 0)

        label = Gtk.Label()
        label.set_markup(
            "Text can be <small>small</small>, <big>big</big>, "
            "<b>bold</b>, <i>italic</i> and even point to "
            'somewhere in the <a href="https://www.gtk.org" '
            'title="Click to find out more">internets</a>.'
        )
        label.set_line_wrap(True)
```

(continua na próxima página)

(continuação da página anterior)

```
        label.set_max_width_chars(48)
        vbox_left.pack_start(label, True, True, 0)

        label = Gtk.Label.new_with_mnemonic(
            "_Press Alt + P to select button to the right"
        )
        vbox_left.pack_start(label, True, True, 0)
        label.set_selectable(True)

        button = Gtk.Button(label="Click at your own risk")
        label.set_mnemonic_widget(button)
        vbox_right.pack_start(button, True, True, 0)

        self.add(hbox)


window = LabelWindow()
window.connect("destroy", Gtk.main_quit)
window.show_all()
Gtk.main()
```

CAPÍTULO 8

Entry

Entry são widgets que permitem que o usuário insira texto. Você pode alterar o conteúdo com o método `Gtk.Entry.set_text()` e ler o conteúdo atual com o método `Gtk.Entry.get_text()`. Você também pode limitar o número de caracteres que a Entrada pode receber chamando `Gtk.Entry.set_max_length()`.

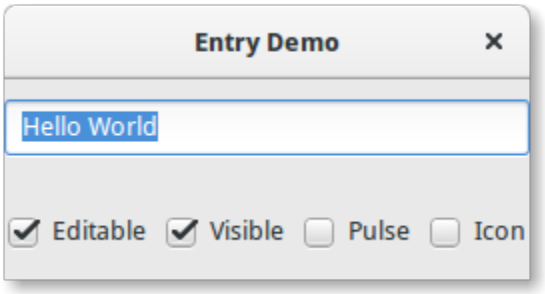
Ocasionalmente, você pode querer tornar um widget de Entrada somente leitura. Isto pode ser feito passando `False` para o método `Gtk.Entry.set_editable()`.

Os widgets de entrada também podem ser usados para recuperar senhas do usuário. É uma prática comum ocultar os caracteres digitados na entrada para evitar revelar a senha a terceiros. Chamando `Gtk.Entry.set_visibility()` com `False` fará com que o texto fique oculto.

`Gtk.Entry` tem a capacidade de exibir informações de progresso ou atividade por trás do texto. Isso é semelhante ao widget `Gtk.ProgressBar` e é comumente encontrado em navegadores web para indicar quanto de um download de página foi concluído. Para fazer uma entrada exibir tais informações, use `Gtk.Entry.set_progress_fraction()`, `Gtk.Entry.set_progress_pulse_step()` ou `Gtk.Entry.progress_pulse()`.

Além disso, uma entrada pode mostrar ícones em ambos os lados da entrada. Esses ícones podem ser ativados clicando, podem ser configurados como fonte de arrastar e podem ter dicas de ferramentas. Para adicionar um ícone, use `Gtk.Entry.set_icon_from_icon_name()` ou uma das várias outras funções que definem um ícone a partir de um nome de ícone, um pixbuf ou tema de ícone. Para definir uma dica de ferramenta em um ícone, use `Gtk.Entry.set_icon_tooltip_text()` ou a função correspondente para marcação.

8.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GLib


class EntryWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Entry Demo")
        self.set_size_request(200, 100)

        self.timeout_id = None

        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(vbox)

        self.entry = Gtk.Entry()
        self.entry.set_text("Hello World")
        vbox.pack_start(self.entry, True, True, 0)

        hbox = Gtk.Box(spacing=6)
        vbox.pack_start(hbox, True, True, 0)

        self.check_editable = Gtk.CheckButton(label="Editable")
        self.check_editable.connect("toggled", self.on_editable_toggled)
        self.check_editable.set_active(True)
        hbox.pack_start(self.check_editable, True, True, 0)

        self.check_visible = Gtk.CheckButton(label="Visible")
        self.check_visible.connect("toggled", self.on_visible_toggled)
        self.check_visible.set_active(True)
        hbox.pack_start(self.check_visible, True, True, 0)

        self.pulse = Gtk.CheckButton(label="Pulse")
        self.pulse.connect("toggled", self.on_pulse_toggled)
        self.pulse.set_active(False)
        hbox.pack_start(self.pulse, True, True, 0)

        self.icon = Gtk.CheckButton(label="Icon")
        self.icon.connect("toggled", self.on_icon_toggled)
```

(continua na próxima página)

(continuação da página anterior)

```
        self.icon.set_active(False)
        hbox.pack_start(self.icon, True, True, 0)

    def on_editable_toggled(self, button):
        value = button.get_active()
        self.entry.set_editable(value)

    def on_visible_toggled(self, button):
        value = button.get_active()
        self.entry.set_visibility(value)

    def on_pulse_toggled(self, button):
        if button.get_active():
            self.entry.set_progress_pulse_step(0.2)
            # Call self.do_pulse every 100 ms
            self.timeout_id = GLib.timeout_add(100, self.do_pulse, None)
        else:
            # Don't call self.do_pulse anymore
            GLib.source_remove(self.timeout_id)
            self.timeout_id = None
            self.entry.set_progress_pulse_step(0)

    def do_pulse(self, user_data):
        self.entry.progress_pulse()
        return True

    def on_icon_toggled(self, button):
        if button.get_active():
            icon_name = "system-search-symbolic"
        else:
            icon_name = None
        self.entry.set_icon_from_icon_name(Gtk.EntryIconPosition.PRIMARY, icon_name)


win = EntryWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 9

Widgets de botão

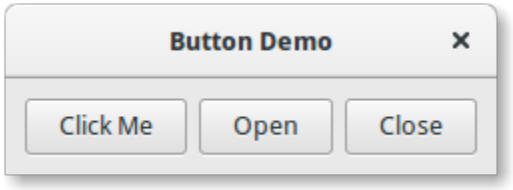
9.1 Button

O widget Button é outro widget comumente usado. Geralmente é usado para anexar uma função que é chamada quando o botão é pressionado.

O widget `Gtk.Button` pode conter qualquer widget filho válido. Isto é, ele pode conter praticamente qualquer outro padrão `Gtk.Widget`. Um filho mais comumente usado é a `Gtk.Label`.

Normalmente, você quer se conectar ao sinal “clicked” do botão que é emitido quando o botão foi pressionado e liberado.

9.1.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class ButtonWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Button Demo")
        self.set_border_width(10)
```

(continua na próxima página)

(continuação da página anterior)

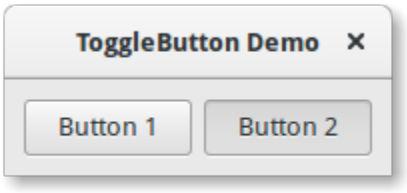
```
        hbox = Gtk.Box(spacing=6)
        self.add(hbox)

        button = Gtk.Button.new_with_label("Click Me")
        button.connect("clicked", self.on_click_me_clicked)
        hbox.pack_start(button, True, True, 0)

        button = Gtk.Button.new_with_mnemonic("_Open")
        button.connect("clicked", self.on_open_clicked)
        hbox.pack_start(button, True, True, 0)

        button = Gtk.Button.new_with_mnemonic("_Close")
        button.connect("clicked", self.on_close_clicked)
        hbox.pack_start(button, True, True, 0)

    def on_click_me_clicked(self, button):
        print('"Click me" button was clicked')

    def on_open_clicked(self, button):
        print('"Open" button was clicked')

    def on_close_clicked(self, button):
        print("Closing application")
        Gtk.main_quit()


win = ButtonWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

9.2 ToggleButton

A `Gtk.ToggleButton` (botão de alternância) é muito semelhante a um `Gtk.Button` normal, mas quando clicados eles permanecem ativados, ou pressionados, até serem clicados novamente. Quando o estado do botão é alterado, o sinal "toggled" é emitido.

Para recuperar o estado da `Gtk.ToggleButton`, você pode usar o método `Gtk.ToggleButton.get_active()`. Isso retorna `True` se o botão estiver "down" (inativo). Você também pode definir o estado do botão de alternância, com `Gtk.ToggleButton.set_active()`. Observe que, se você fizer isso e o estado realmente mudar, isso fará com que o sinal "toggled" seja emitido.

9.2.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class ToggleButtonWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="ToggleButton Demo")
        self.set_border_width(10)

        hbox = Gtk.Box(spacing=6)
        self.add(hbox)

        button = Gtk.ToggleButton(label="Button 1")
        button.connect("toggled", self.on_button_toggled, "1")
        hbox.pack_start(button, True, True, 0)

        button = Gtk.ToggleButton(label="B_utton 2", use_underline=True)
        button.set_active(True)
        button.connect("toggled", self.on_button_toggled, "2")
        hbox.pack_start(button, True, True, 0)

    def on_button_toggled(self, button, name):
        if button.get_active():
            state = "on"
        else:
            state = "off"
        print("Button", name, "was turned", state)


win = ToggleButtonWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

9.3 CheckButton

`Gtk.CheckButton` (botão de seleção) herda de `Gtk.ToggleButton`. A única diferença real entre os dois é como `Gtk.CheckButton` é apresentado. A `Gtk.CheckButton` coloca um discreto `Gtk.ToggleButton` ao lado de um widget, (geralmente um `Gtk.Label`). O sinal "toggled", `Gtk.ToggleButton.set_active()` e `Gtk.ToggleButton.get_active()` são herdados.

9.4 RadioButton

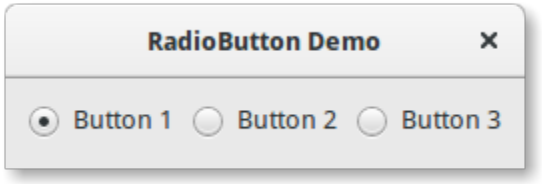
Assim como os botões de seleção, botões de opção também herdam de `Gtk.ToggleButton`, mas estes funcionam em grupos, e apenas um `Gtk.RadioButton` em um grupo pode ser selecionado de cada vez. Portanto, um `Gtk.RadioButton` é uma maneira de dar ao usuário uma escolha entre várias opções.

Botões de opção podem ser criados com um dos métodos estáticos `Gtk.RadioButton.new_from_widget()`, `Gtk.RadioButton.new_with_label_from_widget()` ou `Gtk.RadioButton.new_with_mnemonic_from_widget()`. O primeiro botão de opção de um grupo será criado passando o `None` como o argumento de *group*. Nas chamadas subsequentes, o grupo ao qual você deseja adicionar esse botão deve ser passado como um argumento.

Quando executado pela primeira vez, o primeiro botão de opção do grupo estará ativo. Isto pode ser alterado chamando `Gtk.ToggleButton.set_active()` com `True` como primeiro argumento.

Alterar o grupo de widgets `Gtk.RadioButton` após sua criação pode ser feito chamando `Gtk.RadioButton.join_group()`.

9.4.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class RadioButtonWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="RadioButton Demo")
        self.set_border_width(10)

        hbox = Gtk.Box(spacing=6)
        self.add(hbox)

        button1 = Gtk.RadioButton.new_with_label_from_widget(None, "Button 1")
        button1.connect("toggled", self.on_button_toggled, "1")
        hbox.pack_start(button1, False, False, 0)
```

(continua na próxima página)

(continuação da página anterior)

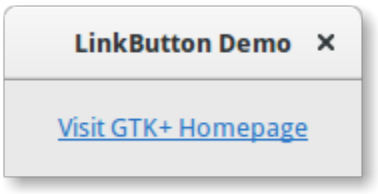
```
        button2 = Gtk.RadioButton.new_from_widget(button1)
        button2.set_label("Button 2")
        button2.connect("toggled", self.on_button_toggled, "2")
        hbox.pack_start(button2, False, False, 0)

        button3 = Gtk.RadioButton.new_with_mnemonic_from_widget(button1, "B_utton 3")
        button3.connect("toggled", self.on_button_toggled, "3")
        hbox.pack_start(button3, False, False, 0)

    def on_button_toggled(self, button, name):
        if button.get_active():
            state = "on"
        else:
            state = "off"
        print("Button", name, "was turned", state)


win = RadioButtonWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

9.5 LinkButton

A `Gtk.LinkButton` é um `Gtk.Button` com um hiperlink, similar ao usado pelos navegadores web, que aciona uma ação quando clicado. É útil mostrar links rápidos para recursos.

A URI vinculada a um `Gtk.LinkButton` pode ser configurada especificamente usando `Gtk.LinkButton.set_uri()` e sendo obtida usando `Gtk.LinkButton.get_uri()`.

9.5.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class LinkButtonWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="LinkButton Demo")
```

(continua na próxima página)

(continuação da página anterior)

```
        self.set_border_width(10)

        button = Gtk.LinkButton.new_with_label(
            uri="https://www.gtk.org",
            label="Visit GTK+ Homepage"
        )
        self.add(button)


win = LinkButtonWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

9.6 SpinButton

A `Gtk.SpinButton` (botão de rotação) é uma maneira ideal de permitir que o usuário defina o valor de algum atributo. Em vez de digitar diretamente um número em `Gtk.Entry`, `Gtk.SpinButton` permite que o usuário clique em uma das duas setas para incrementar ou decrementar o valor exibido. Um valor ainda pode ser digitado, com o bônus que pode ser verificado para garantir que esteja em um determinado intervalo. As propriedades principais de um `Gtk.SpinButton` são definidas através de `Gtk.Adjustment`.

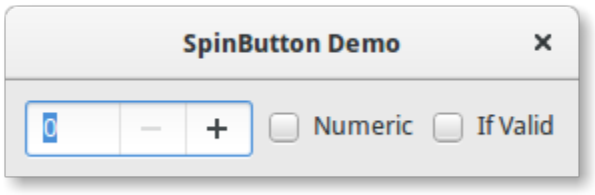
Para alterar o valor que `Gtk.SpinButton` está mostrando, use `Gtk.SpinButton.set_value()`. O valor digitado pode ser um número inteiro ou ponto flutuante, dependendo de seus requisitos, use `Gtk.SpinButton.get_value_as_int()` ou `Gtk.SpinButton.get_value()`, respectivamente.

Quando você permite a exibição de valores flutuantes no botão de rotação, você pode querer ajustar o número de espaços decimais exibidos chamando `Gtk.SpinButton.set_digits()`.

Por padrão, `Gtk.SpinButton` aceita dados textuais. Se você deseja limitar isso apenas a valores numéricos, chame `Gtk.SpinButton.set_numeric()` com `True` como argumento.

Também podemos ajustar a política de atualização de `Gtk.SpinButton`. Existem duas opções aqui; por padrão, o botão de rotação atualiza o valor mesmo se os dados inseridos forem inválidos. Alternativamente, podemos definir a política para apenas atualizar quando o valor inserido é válido chamando `Gtk.SpinButton.set_update_policy()`.

9.6.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk

```

(continua na próxima página)

(continuação da página anterior)

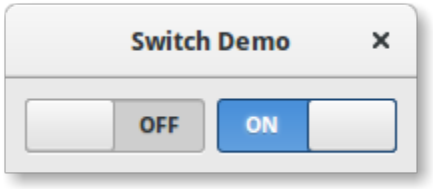
```
6
7  class SpinButtonWindow(Gtk.Window):
8      def __init__(self):
9          super().__init__(title="SpinButton Demo")
10         self.set_border_width(10)
11
12         hbox = Gtk.Box(spacing=6)
13         self.add(hbox)
14
15         adjustment = Gtk.Adjustment(upper=100, step_increment=1, page_increment=10)
16         self.spinbutton = Gtk.SpinButton()
17         self.spinbutton.set_adjustment(adjustment)
18         self.spinbutton.connect("value-changed", self.on_value_changed)
19         hbox.pack_start(self.spinbutton, False, False, 0)
20
21         check_numeric = Gtk.CheckButton(label="Numeric")
22         check_numeric.connect("toggled", self.on_numeric_toggled)
23         hbox.pack_start(check_numeric, False, False, 0)
24
25         check_ifvalid = Gtk.CheckButton(label="If Valid")
26         check_ifvalid.connect("toggled", self.on_ifvalid_toggled)
27         hbox.pack_start(check_ifvalid, False, False, 0)
28
29     def on_value_changed(self, scroll):
30         print(self.spinbutton.get_value_as_int())
31
32     def on_numeric_toggled(self, button):
33         self.spinbutton.set_numeric(button.get_active())
34
35     def on_ifvalid_toggled(self, button):
36         if button.get_active():
37             policy = Gtk.SpinButtonUpdatePolicy.IF_VALID
38         else:
39             policy = Gtk.SpinButtonUpdatePolicy.ALWAYS
40         self.spinbutton.set_update_policy(policy)
41
42
43 win = SpinButtonWindow()
44 win.connect("destroy", Gtk.main_quit)
45 win.show_all()
46 Gtk.main()
```

9.7 Switch

A `Gtk.Switch` (interruptor) é um widget que possui dois estados: ligado ou desligado. O usuário pode controlar qual estado deve estar ativo clicando na área vazia ou arrastando a alça.

Você não deve usar o sinal "activate" no Gtk.Switch que é um sinal de ação e emiti-lo faz com que o switch anime. Os aplicativos nunca devem se conectar a este sinal, mas use o sinal "notify::active", veja o exemplo abaixo.

9.7.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class SwitcherWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Switch Demo")
        self.set_border_width(10)

        hbox = Gtk.Box(spacing=6)
        self.add(hbox)

        switch = Gtk.Switch()
        switch.connect("notify::active", self.on_switch_activated)
        switch.set_active(False)
        hbox.pack_start(switch, True, True, 0)

        switch = Gtk.Switch()
        switch.connect("notify::active", self.on_switch_activated)
        switch.set_active(True)
        hbox.pack_start(switch, True, True, 0)

    def on_switch_activated(self, switch, gparam):
        if switch.get_active():
            state = "on"
        else:
            state = "off"
        print("Switch was turned", state)


win = SwitcherWindow()
win.connect("destroy", Gtk.main_quit)
```

(continua na próxima página)

(continuação da página anterior)

```
35 win.show_all()
36 Gtk.main()
```

CAPÍTULO 10

Expander

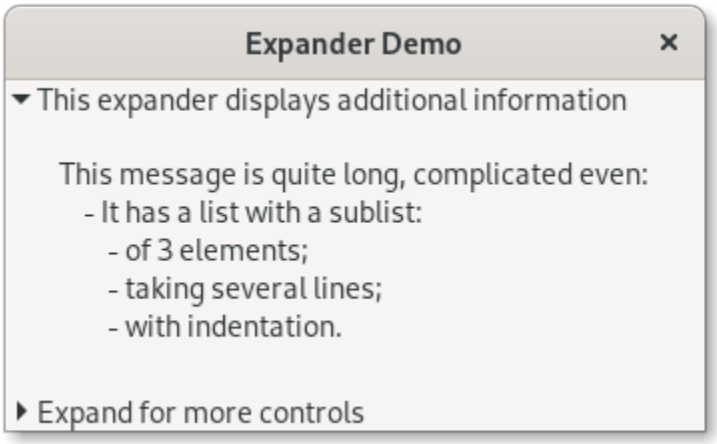
Os expansores permitem ocultar ou mostrar informações dinamicamente em uma janela ou caixa de diálogo. Um expansor pode conter um único widget que será exibido quando expandido.

Os expansores permanecem expandidos até serem clicados novamente. Quando o estado de um expansor é alterado, o sinal "activate" é emitido.

Um expansor pode ser expandido ou recolhido programaticamente passando *True* ou *False* para `Gtk.Expander.set_expanded()`. Observe que isso faz com que o sinal "activate" seja emitido.

Mais de um widget, como `Gtk.Label` e `Gtk.Button`, pode ser adicionado anexando-os a `Gtk.Box`.

10.1 Exemplo




```
1 import gi
2
```

(continua na próxima página)

(continuação da página anterior)

```
gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class ExpanderExample(Gtk.Window):
    def __init__(self):
        super().__init__(title="Expander Demo")

        self.set_size_request(350, 100)

        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(vbox)

        text_expander = Gtk.Expander(
                label="This expander displays additional information"
        )
        text_expander.set_expanded(True)
        vbox.add(text_expander)

        msg = """
This message is quite long, complicated even:
    - It has a list with a sublist:
        - of 3 elements;
        - taking several lines;
        - with indentation.
"""
        details = Gtk.Label(label=msg)
        text_expander.add(details)

        widget_expander = Gtk.Expander(label="Expand for more controls")
        vbox.add(widget_expander)

        expander_hbox = Gtk.HBox()
        widget_expander.add(expander_hbox)

        expander_hbox.add(Gtk.Label(label="Text message"))
        expander_hbox.add(Gtk.Button(label="Click me"))

        self.show_all()


win = ExpanderExample()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 11

ProgressBar

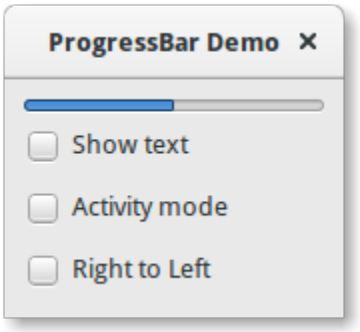
O `Gtk.ProgressBar` é normalmente usado para exibir o progresso de uma operação longa. Ele fornece uma pista visual de que o processamento está em andamento. O `Gtk.ProgressBar` pode ser usado em dois modos diferentes: *modo de porcentagem* e *modo de atividade*.

Quando um aplicativo pode determinar quanto trabalho precisa ocorrer (por exemplo, ler um número fixo de bytes de um arquivo) e monitorar seu progresso, ela pode usar `Gtk.ProgressBar` no modo *percentage* e o usuário vê uma barra crescente indicando a porcentagem do trabalho que foi concluído. Neste modo, o aplicativo é necessário para chamar `Gtk.ProgressBar.set_fraction()` periodicamente para atualizar a barra de progresso, passando um ponto flutuante entre 0 e 1 para fornecer o novo valor percentual.

Quando um aplicativo não tem uma maneira precisa de saber a quantidade de trabalho a ser feito, ele pode usar o *modo de atividade*, que mostra a atividade de um bloco se movendo para frente e para trás na área de progresso. Neste modo, o aplicativo é necessário para chamar `Gtk.ProgressBar.pulse()` periodicamente para atualizar a barra de progresso. Você também pode escolher o tamanho do passo, com o método `Gtk.ProgressBar.set_pulse_step()`.

Por padrão, `Gtk.ProgressBar` é horizontal e da esquerda para a direita, mas você pode alterá-lo para uma barra de progresso vertical usando o método `Gtk.ProgressBar.set_orientation()`. Mudar a direção da barra de progresso pode ser feito usando `Gtk.ProgressBar.set_inverted()`. `Gtk.ProgressBar` também pode conter texto que pode ser definido chamando `Gtk.ProgressBar.set_text()` e `Gtk.ProgressBar.set_show_text()`.

11.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GLib


class ProgressBarWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="ProgressBar Demo")
        self.set_border_width(10)

        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(vbox)

        self.progressbar = Gtk.ProgressBar()
        vbox.pack_start(self.progressbar, True, True, 0)

        button = Gtk.CheckButton(label="Show text")
        button.connect("toggled", self.on_show_text_toggled)
        vbox.pack_start(button, True, True, 0)

        button = Gtk.CheckButton(label="Activity mode")
        button.connect("toggled", self.on_activity_mode_toggled)
        vbox.pack_start(button, True, True, 0)

        button = Gtk.CheckButton(label="Right to Left")
        button.connect("toggled", self.on_right_to_left_toggled)
        vbox.pack_start(button, True, True, 0)

        self.timeout_id = GLib.timeout_add(50, self.on_timeout, None)
        self.activity_mode = False

    def on_show_text_toggled(self, button):
        show_text = button.get_active()
        if show_text:
            text = "some text"
        else:
            text = None
        self.progressbar.set_text(text)
```

(continua na próxima página)

(continuação da página anterior)

```
        self.progressbar.set_show_text(show_text)

    def on_activity_mode_toggled(self, button):
        self.activity_mode = button.get_active()
        if self.activity_mode:
            self.progressbar.pulse()
        else:
            self.progressbar.set_fraction(0.0)

    def on_right_to_left_toggled(self, button):
        value = button.get_active()
        self.progressbar.set_inverted(value)

    def on_timeout(self, user_data):
        """
        Update value on the progress bar
        """
        if self.activity_mode:
            self.progressbar.pulse()
        else:
            new_value = self.progressbar.get_fraction() + 0.01

            if new_value > 1:
                new_value = 0

            self.progressbar.set_fraction(new_value)

        # As this is a timeout function, return True so that it
        # continues to get called
        return True


win = ProgressBarWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

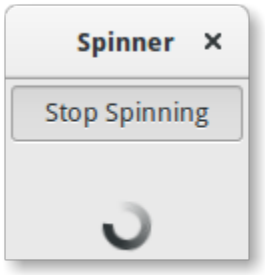
CAPÍTULO 12

Spinner

O `Gtk.Spinner` exibe uma animação giratória do tamanho de um ícone. É frequentemente usado como uma alternativa a `GtkProgressBar` para exibir atividade indefinida, em vez de progresso real.

Para iniciar a animação, use `Gtk.Spinner.start()`. Para pará-lo, use `Gtk.Spinner.stop()`.

12.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class SpinnerAnimation(Gtk.Window):
    def __init__(self):

        super().__init__(title="Spinner")
        self.set_border_width(3)
        self.connect("destroy", Gtk.main_quit)

```

(continua na próxima página)

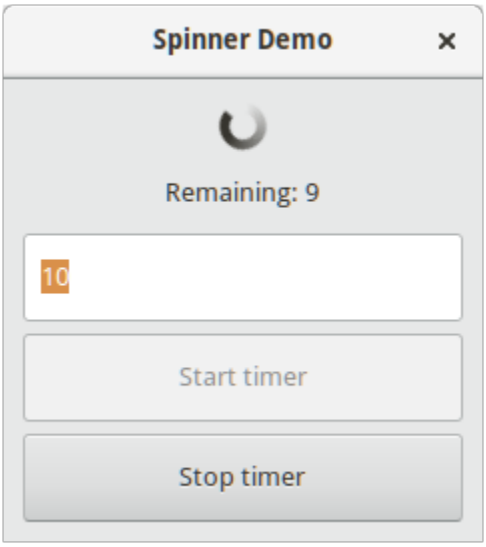
(continuação da página anterior)

```
        self.button = Gtk.ToggleButton(label="Start Spinning")
        self.button.connect("toggled", self.on_button_toggled)
        self.button.set_active(False)

        self.spinner = Gtk.Spinner()

        self.grid = Gtk.Grid()
        self.grid.add(self.button)
        self.grid.attach_next_to(
            self.spinner, self.button, Gtk.PositionType.BOTTOM, 1, 2
        )
        self.grid.set_row_homogeneous(True)

        self.add(self.grid)
        self.show_all()

    def on_button_toggled(self, button):

        if button.get_active():
            self.spinner.start()
            self.button.set_label("Stop Spinning")

        else:
            self.spinner.stop()
            self.button.set_label("Start Spinning")


myspinner = SpinnerAnimation()

Gtk.main()
```

12.2 Exemplo estendido

Um exemplo estendido que usa uma função de tempo limite para iniciar e parar a animação giratória. A função `on_timeout()` é chamada em intervalos regulares até retornar `False`, momento em que o tempo limite é destruído automaticamente e a função não será chamada novamente.

12.2.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GLib


class SpinnerWindow(Gtk.Window):
    def __init__(self, *args, **kwargs):
        super().__init__(title="Spinner Demo")
        self.set_border_width(10)

        mainBox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
        self.add(mainBox)

        self.spinner = Gtk.Spinner()
        mainBox.pack_start(self.spinner, True, True, 0)

        self.label = Gtk.Label()
        mainBox.pack_start(self.label, True, True, 0)

        self.entry = Gtk.Entry()
        self.entry.set_text("10")
        mainBox.pack_start(self.entry, True, True, 0)

        self.buttonStart = Gtk.Button(label="Start timer")
        self.buttonStart.connect("clicked", self.on_buttonStart_clicked)
        mainBox.pack_start(self.buttonStart, True, True, 0)

        self.buttonStop = Gtk.Button(label="Stop timer")
        self.buttonStop.set_sensitive(False)
        self.buttonStop.connect("clicked", self.on_buttonStop_clicked)
        mainBox.pack_start(self.buttonStop, True, True, 0)
```

(continua na próxima página)

(continuação da página anterior)

```
        self.timeout_id = None
        self.connect("destroy", self.on_SpinnerWindow_destroy)

    def on_buttonStart_clicked(self, widget, *args):
        """ Handles "clicked" event of buttonStart. """
        self.start_timer()

    def on_buttonStop_clicked(self, widget, *args):
        """ Handles "clicked" event of buttonStop. """
        self.stop_timer("Stopped from button")

    def on_SpinnerWindow_destroy(self, widget, *args):
        """ Handles destroy event of main window. """
        # ensure the timeout function is stopped
        if self.timeout_id:
            GLib.source_remove(self.timeout_id)
            self.timeout_id = None
        Gtk.main_quit()

    def on_timeout(self, *args, **kwargs):
        """ A timeout function.

        Return True to stop it.
        This is not a precise timer since next timeout
        is recalculated based on the current time."""
        self.counter -= 1
        if self.counter <= 0:
            self.stop_timer("Reached time out")
            return False
        self.label.set_label("Remaining: " + str(int(self.counter / 4)))
        return True

    def start_timer(self):
        """ Start the timer. """
        self.buttonStart.set_sensitive(False)
        self.buttonStop.set_sensitive(True)
        # time out will check every 250 milliseconds (1/4 of a second)
        self.counter = 4 * int(self.entry.get_text())
        self.label.set_label("Remaining: " + str(int(self.counter / 4)))
        self.spinner.start()
        self.timeout_id = GLib.timeout_add(250, self.on_timeout, None)

    def stop_timer(self, alabeltext):
        """ Stop the timer. """
        if self.timeout_id:
            GLib.source_remove(self.timeout_id)
            self.timeout_id = None
        self.spinner.stop()
        self.buttonStart.set_sensitive(True)
        self.buttonStop.set_sensitive(False)
        self.label.set_label(alabeltext)
```

(continua na próxima página)

(continuação da página anterior)

```
85
86
87 win = SpinnerWindow()
88 win.show_all()
89 Gtk.main()
```

CAPÍTULO 13

Widgets de árvore e lista

A `Gtk.TreeView` e seus widgets associados são uma maneira extremamente poderosa de exibir dados. Eles são usados em conjunto com um `Gtk.ListStore` ou `Gtk.TreeStore` e fornecem uma maneira de exibir e manipular dados de várias maneiras, incluindo:

- Atualizações automáticas quando os dados são adicionados, removidos ou editados
- Suporte a arrastar e soltar
- Ordenação de dados
- Incorporação de widgets, como caixas de seleção, barras de progresso, etc.
- Colunas reordenáveis e redimensionáveis
- Filtragem de dados

Com o poder e a flexibilidade de um `Gtk.TreeView` vem a complexidade. Geralmente, é difícil para os desenvolvedores iniciantes serem capazes de utilizá-lo corretamente devido ao número de métodos necessários.

13.1 O modelo

Cada `Gtk.TreeView` possui um `Gtk.TreeModel`, o qual contém os dados exibidos pelo TreeView. Cada `Gtk.TreeModel` pode ser usado por mais de um `Gtk.TreeView`. Por exemplo, isso permite que os mesmos dados subjacentes sejam exibidos e editados de duas maneiras diferentes ao mesmo tempo. Ou os 2 modos de exibição podem exibir colunas diferentes dos mesmos dados do modelo, da mesma forma que duas consultas SQL (ou "views") podem mostrar campos diferentes da mesma tabela de banco de dados.

Embora você possa teoricamente implementar seu próprio Model, você normalmente usará as classes de modelo `Gtk.ListStore` ou `Gtk.TreeStore`. `Gtk.ListStore` contém linhas simples de dados, e cada linha não tem filhos, enquanto `Gtk.TreeStore` contém linhas de dados, e cada linha pode ter linhas filhas.

Ao construir um modelo, você deve especificar os tipos de dados para cada coluna que o modelo contém.

```
store = Gtk.ListStore(str, str, float)
```

Isso cria um armazenamento de lista com três colunas, duas colunas de string e uma coluna flutuante.

A adição de dados ao modelo é feita usando Gtk.ListStore.append() ou Gtk.TreeStore.append(), dependendo de qual tipo de modelo foi criado.

Para um Gtk.ListStore:

```
treeiter = store.append(["The Art of Computer Programming",
                         "Donald E. Knuth", 25.46])
```

Para um Gtk.TreeStore você deve especificar uma linha existente para anexar a nova linha, usando um Gtk.TreeIter, ou None para o nível superior da árvore:

```
treeiter = store.append(None, ["The Art of Computer Programming",
                               "Donald E. Knuth", 25.46])
```

Ambos os métodos retornam uma instância Gtk.TreeIter, que aponta para a localização da linha recém-inserida. Você pode recuperar um Gtk.TreeIter chamando Gtk.TreeModel.get_iter().

Depois que os dados foram inseridos, você pode recuperar ou modificar dados usando o iterador de árvore e o índice de coluna.

```
print(store[treeiter][2]) # Prints value of third column
store[treeiter][2] = 42.15
```

Assim como no objeto interno `list` do Python, você pode usar `len()` para obter o número de linhas e usar fatias para recuperar ou definir valores.

```
# Print number of rows
print(len(store))
# Print all but first column
print(store[treeiter][1:])
# Print last column
print(store[treeiter][-1])
# Set last two columns
store[treeiter][1:] = ["Donald Ervin Knuth", 41.99]
```

Iterar sobre todas as linhas de um modelo de árvore é muito simples também.

```
for row in store:
    # Print values of all columns
    print(row[:])
```

Tenha em mente que, se você usar Gtk.TreeStore, o código acima irá apenas iterar sobre as linhas do nível superior, mas não os filhos dos nós. Para iterar sobre todas as linhas, use Gtk.TreeModel.foreach().

```
def print_row(store, treepath, treeiter):
    print("\t" * (treepath.get_depth() - 1), store[treeiter][:], sep="")

store.foreach(print_row)
```

Além de acessar valores armazenados em um Gtk.TreeModel com o método list-like mencionado acima, também é possível usar as instâncias Gtk.TreeIter ou Gtk.TreePath. Ambos fazem referência a uma linha específica em um modelo de árvore. Pode-se converter um caminho para um iterador chamando Gtk.TreeModel.get_iter(). Como Gtk.ListStore contém apenas um nível, ou seja, nós não têm nenhum nó filho, um caminho é essencialmente o índice da linha que você deseja acessar.

```
# Get path pointing to 6th row in list store
path = Gtk.TreePath(5)
treeiter = liststore.get_iter(path)
# Get value at 2nd column
value = liststore.get_value(treeiter, 1)
```

No caso de Gtk.TreeStore, um caminho é uma lista de índices ou uma string. O formulário de string é uma lista de números separados por dois pontos. Cada número refere-se ao deslocamento nesse nível. Assim, o caminho "0" refere-se ao nó raiz e o caminho "2:4" refere-se ao quinto filho do terceiro nó.

```
# Get path pointing to 5th child of 3rd row in tree store
path = Gtk.TreePath([2, 4])
treeiter = treestore.get_iter(path)
# Get value at 2nd column
value = treestore.get_value(treeiter, 1)
```

Instâncias de Gtk.TreePath podem ser acessadas como listas, `len(treepath)` retorna a profundidade do item `treepath` está apontando para, e `treepath[i]` retorna o índice do filho no nível *i*.

13.2 A visão

Embora existam vários modelos diferentes para escolher, há apenas um widget de visualização para lidar. Funciona com a lista ou com o armazenamento em árvore. Configurar um Gtk.TreeView não é uma tarefa difícil. Ele precisa de um Gtk.TreeModel para saber de onde recuperar seus dados, seja passando-o para o construtor Gtk.TreeView, ou chamando Gtk.TreeView.set_model().

```
tree = Gtk.TreeView(model=store)
```

Uma vez que o widget Gtk.TreeView possua um modelo, ele precisará saber como exibir o modelo. Ele faz isso com colunas e renderizadores de célula. headers_visible controla se exibe cabeçalhos de coluna.

Os renderizadores de célula são usados para desenhar os dados no modelo de árvore de uma maneira específica. Existem vários renderizadores de célula que vêm com o GTK+, por exemplo Gtk.CellRendererText, Gtk.CellRendererPixbuf e Gtk.CellRendererToggle. Além disso, é relativamente fácil escrever um renderizador personalizado criando uma subclasse de Gtk.CellRenderer e adicionando propriedades com `GObject.Property()`.

Um Gtk.TreeViewColumn é o objeto que usa Gtk.TreeView para organizar as colunas verticais na visualização em árvore e conter um ou mais renderizadores de células. Cada coluna pode ter um title que ficará visível se o Gtk.TreeView estiver mostrando os cabeçalhos das colunas. O modelo é mapeado para a coluna usando argumentos nomeados com propriedades do renderizador como identificadores e índices das colunas do modelo como argumentos.

```
renderer = Gtk.CellRendererPixbuf()
column = Gtk.TreeViewColumn(cell_renderer=renderer, icon_name=3)
tree.append_column(column)
```

Argumentos posicionais podem ser usados para o título da coluna e o renderizador.

```
renderer = Gtk.CellRendererText()
column = Gtk.TreeViewColumn("Title", renderer, text=0, weight=1)
tree.append_column(column)
```

Para renderizar mais de uma coluna de modelo em uma coluna de visão, você precisa criar uma instância Gtk.TreeViewColumn e usar Gtk.TreeViewColumn.pack_start() para adicionar as colunas de modelo a ela.

```
column = Gtk.TreeViewColumn("Title and Author")

title = Gtk.CellRendererText()
author = Gtk.CellRendererText()

column.pack_start(title, True)
column.pack_start(author, True)

column.add_attribute(title, "text", 0)
column.add_attribute(author, "text", 1)

tree.append_column(column)
```

13.3 A seleção

A maioria dos aplicativos precisará não apenas lidar com a exibição de dados, mas também receber eventos de entrada dos usuários. Para fazer isso, basta obter uma referência a um objeto de seleção e conectar-se ao sinal "changed".

```
select = tree.get_selection()
select.connect("changed", on_tree_selection_changed)
```

Em seguida, para recuperar dados para a linha selecionada:

```
def on_tree_selection_changed(selection):
    model, treeiter = selection.get_selected()
    if treeiter is not None:
        print("You selected", model[treeiter][0])
```

Você pode controlar quais seleções são permitidas chamando `Gtk.TreeSelection.set_mode()`. `Gtk.TreeSelection.get_selected()` não funciona se o modo de seleção estiver definido como `Gtk.SelectionMode.MULTIPLE`, use `Gtk.TreeSelection.get_selected_rows()`.

13.4 Classificação

A classificação é um recurso importante para as visualizações em árvore e é suportada pelos modelos de árvore padrão (`Gtk.TreeStore` e `Gtk.ListStore`), que implementam a interface `Gtk.TreeSortable`.

13.4.1 Classificando clicando em colunas

Uma coluna de um `Gtk.TreeView` pode ser facilmente ordenada com uma chamada para `Gtk.TreeViewColumn.set_sort_column_id()`. Depois, a coluna pode ser ordenada clicando no cabeçalho.

Primeiro precisamos de um simples `Gtk.TreeView` e um `Gtk.ListStore` como modelo.

```
model = Gtk.ListStore(str)
model.append(["Benjamin"])
model.append(["Charles"])
model.append(["alfred"])
model.append(["Alfred"])
```

(continua na próxima página)

(continuação da página anterior)

```
model.append(["David"])
model.append(["charles"])
model.append(["david"])
model.append(["benjamin"])

treeView = Gtk.TreeView(model=model)

cellRenderer = Gtk.CellRendererText()
column = Gtk.TreeViewColumn("Title", renderer, text=0)
```

O próximo passo é ativar a classificação. Note que o *column_id* (**0** no exemplo) refere-se à coluna do modelo e **não** à coluna do TreeView.

```
column.set_sort_column_id(0)
```

13.4.2 Definindo uma função de classificação personalizada

Também é possível definir uma função de comparação personalizada para alterar o comportamento de classificação. Como exemplo, criaremos uma função de comparação que classifica maiúsculas e minúsculas. No exemplo acima, a lista classificada parecia com:

```
alfred
Alfred
benjamin
Benjamin
charles
Charles
david
David
```

A lista classificada com distinção entre maiúsculas e minúsculas será semelhante a:

```
Alfred
Benjamin
Charles
David
alfred
benjamin
charles
david
```

Em primeiro lugar, é necessária uma função de comparação. Esta função obtém duas linhas e tem que retornar um inteiro negativo se o primeiro deve vir antes do segundo, zero se eles forem iguais e um inteiro positivo se o segundo vier antes do primeiro.

```
def compare(model, row1, row2, user_data):
    sort_column, _ = model.get_sort_column_id()
    value1 = model.get_value(row1, sort_column)
    value2 = model.get_value(row2, sort_column)
    if value1 < value2:
        return -1
```

(continua na próxima página)

(continuação da página anterior)

```
    elif value1 == value2:
        return 0
    else:
        return 1
```

Então a função sort deve ser definida por `Gtk.TreeSortable.set_sort_func()`.

```
model.set_sort_func(0, compare, None)
```

13.5 Filtragem

Ao contrário da classificação, a filtragem não é tratada pelos dois modelos que vimos anteriormente, mas pela classe `Gtk.TreeModelFilter`. Esta classe, como `Gtk.TreeStore` e `Gtk.ListStore`, é uma `Gtk.TreeModel`. Ele age como uma camada entre o modelo "real" (a `Gtk.TreeStore` ou a `Gtk.ListStore`), ocultando alguns elementos para a view. Na prática, ele fornece o `Gtk.TreeView` com um subconjunto do modelo subjacente. Instâncias de `Gtk.TreeModelFilter` podem ser empilhadas umas sobre as outras, para usar múltiplos filtros no mesmo modelo (da mesma forma que você usaria cláusulas "AND" em uma requisição SQL). Eles também podem ser encadeados com instâncias `Gtk.TreeModelSort`.

Você pode criar uma nova instância de `Gtk.TreeModelFilter` e dar a ela um modelo para filtrar, mas a maneira mais fácil é gerá-lo diretamente do modelo filtrado, usando o método `Gtk.TreeModel.filter_new()` método.

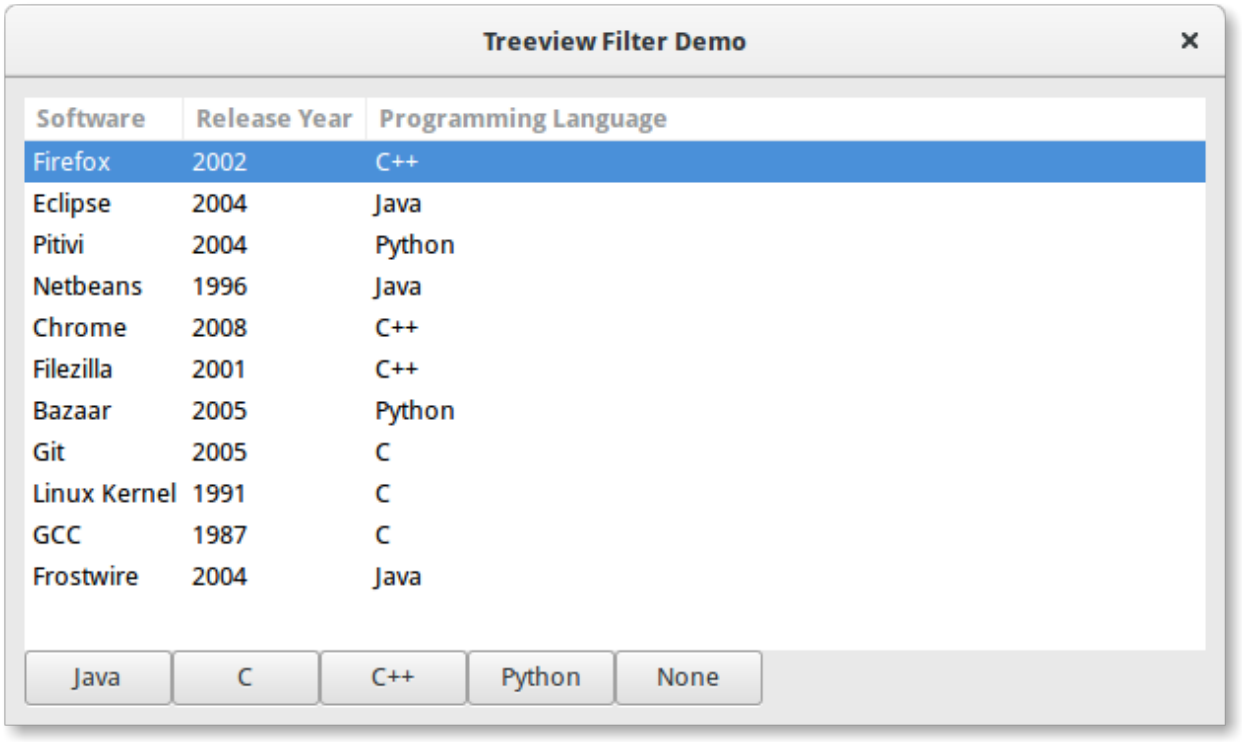
```
filter = model.filter_new()
```

Da mesma forma que funciona a função de classificação, o `Gtk.TreeModelFilter` usa de uma função "visibility", que, dada uma linha do modelo subjacente, retornará um booleano indicando se essa linha deve ser filtrada ou não. É definido por `Gtk.TreeModelFilter.set_visible_func()`:

```
filter.set_visible_func(filter_func, data=None)
```

A alternativa para uma função de "visibilidade" é usar uma coluna booleana no modelo para especificar quais linhas filtrar. Escolha qual coluna com `Gtk.TreeModelFilter.set_visible_column()`.

Vejamos um exemplo completo que usa a pilha inteira `Gtk.ListStore` – `Gtk.TreeModelFilter` – `Gtk.TreeModelFilter` — `Gtk.TreeView`.




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk

# list of tuples for each software, containing the software name, initial release, and␣
↪main programming languages used
software_list = [
    ("Firefox", 2002, "C++"),
    ("Eclipse", 2004, "Java"),
    ("Pitivi", 2004, "Python"),
    ("Netbeans", 1996, "Java"),
    ("Chrome", 2008, "C++"),
    ("Filezilla", 2001, "C++"),
    ("Bazaar", 2005, "Python"),
    ("Git", 2005, "C"),
    ("Linux Kernel", 1991, "C"),
    ("GCC", 1987, "C"),
    ("Frostwire", 2004, "Java"),
]


class TreeViewFilterWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Treeview Filter Demo")
        self.set_border_width(10)

        # Setting up the self.grid in which the elements are to be positioned
        self.grid = Gtk.Grid()
```

(continua na próxima página)

(continuação da página anterior)

```
        self.grid.set_column_homogeneous(True)
        self.grid.set_row_homogeneous(True)
        self.add(self.grid)

        # Creating the ListStore model
        self.software_liststore = Gtk.ListStore(str, int, str)
        for software_ref in software_list:
            self.software_liststore.append(list(software_ref))
        self.current_filter_language = None

        # Creating the filter, feeding it with the liststore model
        self.language_filter = self.software_liststore.filter_new()
        # setting the filter function, note that we're not using the
        self.language_filter.set_visible_func(self.language_filter_func)

        # creating the treeview, making it use the filter as a model, and adding the␣
↪columns
        self.treeview = Gtk.TreeView(model=self.language_filter)
        for i, column_title in enumerate(
            ["Software", "Release Year", "Programming Language"]
        ):
            renderer = Gtk.CellRendererText()
            column = Gtk.TreeViewColumn(column_title, renderer, text=i)
            self.treeview.append_column(column)

        # creating buttons to filter by programming language, and setting up their events
        self.buttons = list()
        for prog_language in ["Java", "C", "C++", "Python", "None"]:
            button = Gtk.Button(label=prog_language)
            self.buttons.append(button)
            button.connect("clicked", self.on_selection_button_clicked)

        # setting up the layout, putting the treeview in a scrollwindow, and the buttons␣
↪in a row
        self.scrollable_treelist = Gtk.ScrolledWindow()
        self.scrollable_treelist.set_vexpand(True)
        self.grid.attach(self.scrollable_treelist, 0, 0, 8, 10)
        self.grid.attach_next_to(
            self.buttons[0], self.scrollable_treelist, Gtk.PositionType.BOTTOM, 1, 1
        )
        for i, button in enumerate(self.buttons[1:]):
            self.grid.attach_next_to(
                button, self.buttons[i], Gtk.PositionType.RIGHT, 1, 1
            )
        self.scrollable_treelist.add(self.treeview)

        self.show_all()

    def language_filter_func(self, model, iter, data):
        """Tests if the language in the row is the one in the filter"""
        if (
            self.current_filter_language is None
```

(continua na próxima página)

(continuação da página anterior)

```
            or self.current_filter_language == "None"
        ):
            return True
        else:
            return model[iter][2] == self.current_filter_language

    def on_selection_button_clicked(self, widget):
        """Called on any of the button clicks"""
        # we set the current language filter to the button's label
        self.current_filter_language = widget.get_label()
        print("%s language selected!" % self.current_filter_language)
        # we update the filter, which updates in turn the view
        self.language_filter.refilter()


win = TreeViewFilterWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 14

CellRenderers

Os widgets `Gtk.CellRenderer` (renderizadores de célula) são usados para exibir informações dentro de widgets como `Gtk.TreeView` ou `Gtk.ComboBox`. Eles trabalham de perto com os widgets associados e são muito poderosos, com muitas opções de configuração para exibir uma grande quantidade de dados de diferentes maneiras. Há sete widgets `Gtk.CellRenderer` que podem ser usados para diferentes propósitos:

- `Gtk.CellRendererText`
- `Gtk.CellRendererToggle`
- `Gtk.CellRendererPixbuf`
- `Gtk.CellRendererCombo`
- `Gtk.CellRendererProgress`
- `Gtk.CellRendererSpinner`
- `Gtk.CellRendererSpin`
- `Gtk.CellRendererAccel`

14.1 CellRendererText

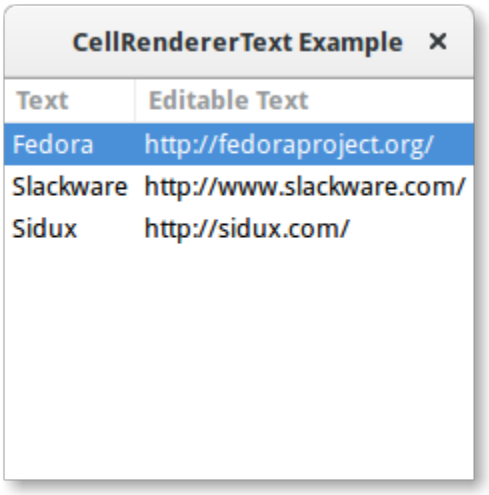
A `Gtk.CellRendererText` processa um dado texto em sua célula, usando as informações de fonte, cor e estilo fornecidas por suas propriedades. O texto será reticulado se for muito longo e a propriedade "ellipsize" permitir.

Por padrão, o texto em `Gtk.CellRendererText` widgets não é editável. Isso pode ser alterado, definindo o valor da propriedade "editable" como `True`:

```
cell.set_property("editable", True)
```

Você pode então se conectar ao sinal "edited" e atualizar seu `Gtk.TreeModel` de acordo.

14.1.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class CellRendererTextWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="CellRendererText Example")

        self.set_default_size(200, 200)

        self.liststore = Gtk.ListStore(str, str)
        self.liststore.append(["Fedora", "https://fedoraproject.org/"])
        self.liststore.append(["Slackware", "http://www.slackware.com/"])
        self.liststore.append(["Sidux", "http://sidux.com/"])

        treeview = Gtk.TreeView(model=self.liststore)

        renderer_text = Gtk.CellRendererText()
        column_text = Gtk.TreeViewColumn("Text", renderer_text, text=0)
        treeview.append_column(column_text)

        renderer_editabletext = Gtk.CellRendererText()
        renderer_editabletext.set_property("editable", True)

        column_editabletext = Gtk.TreeViewColumn(
            "Editable Text", renderer_editabletext, text=1
        )
        treeview.append_column(column_editabletext)

        renderer_editabletext.connect("edited", self.text_edited)

        self.add(treeview)
```

(continua na próxima página)

(continuação da página anterior)

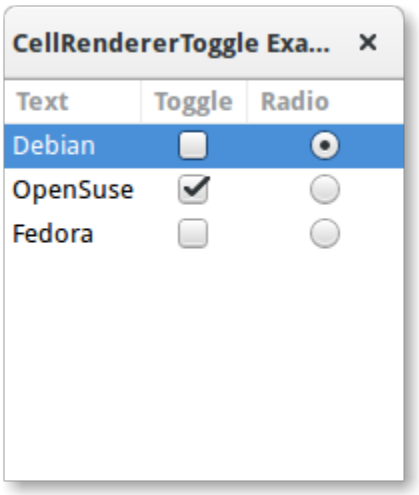
```
35
36      def text_edited(self, widget, path, text):
37          self.liststore[path][1] = text
38
39
40  win = CellRendererTextWindow()
41  win.connect("destroy", Gtk.main_quit)
42  win.show_all()
43  Gtk.main()
```

14.2 CellRendererToggle

`Gtk.CellRendererToggle` renderiza um botão de alternância em uma célula. O botão é desenhado como um botão de rádio ou de verificação, dependendo da propriedade “radio”. Quando ativado, emite o sinal “toggled”.

Como um `Gtk.CellRendererToggle` pode ter dois estados, ativos e não ativos, você provavelmente deseja vincular a propriedade “active” no renderizador de célula a um valor booleano no modelo, fazendo com que o botão de seleção reflita o estado do modelo.

14.2.1 Exemplo




```
1   import gi
2
3   gi.require_version("Gtk", "3.0")
4   from gi.repository import Gtk
5
6
7   class CellRendererToggleWindow(Gtk.Window):
8       def __init__(self):
9           super().__init__(title="CellRendererToggle Example")
10
11          self.set_default_size(200, 200)
```

(continua na próxima página)

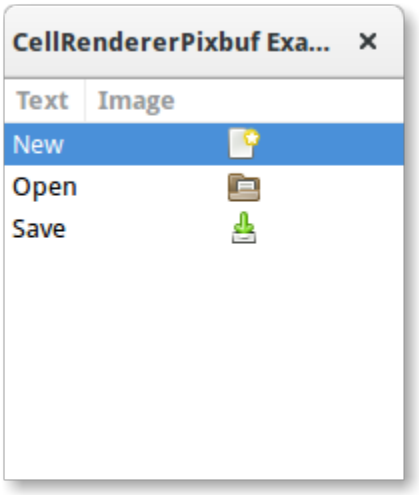
(continuação da página anterior)

```
        self.liststore = Gtk.ListStore(str, bool, bool)
        self.liststore.append(["Debian", False, True])
        self.liststore.append(["OpenSuse", True, False])
        self.liststore.append(["Fedora", False, False])

        treeview = Gtk.TreeView(model=self.liststore)

        renderer_text = Gtk.CellRendererText()
        column_text = Gtk.TreeViewColumn("Text", renderer_text, text=0)
        treeview.append_column(column_text)

        renderer_toggle = Gtk.CellRendererToggle()
        renderer_toggle.connect("toggled", self.on_cell_toggled)

        column_toggle = Gtk.TreeViewColumn("Toggle", renderer_toggle, active=1)
        treeview.append_column(column_toggle)

        renderer_radio = Gtk.CellRendererToggle()
        renderer_radio.set_radio(True)
        renderer_radio.connect("toggled", self.on_cell_radio_toggled)

        column_radio = Gtk.TreeViewColumn("Radio", renderer_radio, active=2)
        treeview.append_column(column_radio)

        self.add(treeview)

    def on_cell_toggled(self, widget, path):
        self.liststore[path][1] = not self.liststore[path][1]

    def on_cell_radio_toggled(self, widget, path):
        selected_path = Gtk.TreePath(path)
        for row in self.liststore:
            row[2] = row.path == selected_path


win = CellRendererToggleWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

14.3 CellRendererPixbuf

A `Gtk.CellRendererPixbuf` pode ser usado para renderizar uma imagem em uma célula. Ele permite renderizar um dado `Gdk.Pixbuf` (definido através da propriedade "pixbuf") ou um ícone nomeado (configurado através da propriedade "icon-name").

14.3.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class CellRendererPixbufWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="CellRendererPixbuf Example")

        self.set_default_size(200, 200)

        self.liststore = Gtk.ListStore(str, str)
        self.liststore.append(["New", "document-new"])
        self.liststore.append(["Open", "document-open"])
        self.liststore.append(["Save", "document-save"])

        treeview = Gtk.TreeView(model=self.liststore)

        renderer_text = Gtk.CellRendererText()
        column_text = Gtk.TreeViewColumn("Text", renderer_text, text=0)
        treeview.append_column(column_text)

        renderer_pixbuf = Gtk.CellRendererPixbuf()

        column_pixbuf = Gtk.TreeViewColumn("Image", renderer_pixbuf, icon_name=1)
        treeview.append_column(column_pixbuf)
```

(continua na próxima página)

(continuação da página anterior)

```
        self.add(treeview)


win = CellRendererPixbufWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

14.4 CellRendererCombo

`Gtk.CellRendererCombo` processa texto em uma célula como `Gtk.CellRendererText` do qual é derivado. Mas enquanto o último oferece uma entrada simples para editar o texto, `Gtk.CellRendererCombo` oferece um widget `Gtk.ComboBox` para editar o texto. Os valores a serem exibidos na caixa de combinação são obtidos de `Gtk.TreeModel` especificado na propriedade "model".

O CellRendererCombo cuida de adicionar um renderizador de célula de texto à caixa de combinação e o configura para exibir a coluna especificada por sua propriedade "text-column".

A `Gtk.CellRendererCombo` pode operar em dois modos. Ele pode ser usado com e sem um widget associado `Gtk.Entry`, dependendo do valor da propriedade "has-entry".

14.4.1 Exemplo


```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class CellRendererComboWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="CellRendererCombo Example")
```

(continua na próxima página)

(continuação da página anterior)

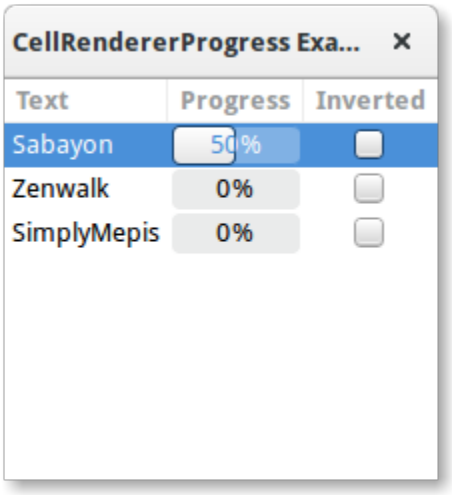
```
        self.set_default_size(200, 200)

        liststore_manufacturers = Gtk.ListStore(str)
        manufacturers = ["Sony", "LG", "Panasonic", "Toshiba", "Nokia", "Samsung"]
        for item in manufacturers:
            liststore_manufacturers.append([item])

        self.liststore_hardware = Gtk.ListStore(str, str)
        self.liststore_hardware.append(["Television", "Samsung"])
        self.liststore_hardware.append(["Mobile Phone", "LG"])
        self.liststore_hardware.append(["DVD Player", "Sony"])

        treeview = Gtk.TreeView(model=self.liststore_hardware)

        renderer_text = Gtk.CellRendererText()
        column_text = Gtk.TreeViewColumn("Text", renderer_text, text=0)
        treeview.append_column(column_text)

        renderer_combo = Gtk.CellRendererCombo()
        renderer_combo.set_property("editable", True)
        renderer_combo.set_property("model", liststore_manufacturers)
        renderer_combo.set_property("text-column", 0)
        renderer_combo.set_property("has-entry", False)
        renderer_combo.connect("edited", self.on_combo_changed)

        column_combo = Gtk.TreeViewColumn("Combo", renderer_combo, text=1)
        treeview.append_column(column_combo)

        self.add(treeview)

    def on_combo_changed(self, widget, path, text):
        self.liststore_hardware[path][1] = text


win = CellRendererComboWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

14.5 CellRendererProgress

`Gtk.CellRendererProgress` renderiza um valor numérico como uma barra de progresso em uma célula. Além disso, pode exibir um texto na parte superior da barra de progresso.

O valor percentual da barra de progresso pode ser modificado alterando a propriedade “value”. Semelhante a `Gtk.ProgressBar`, você pode ativar o *modo de atividade* incrementando a propriedade “pulse” em vez da propriedade “value”.

14.5.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, GLib


class CellRendererProgressWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="CellRendererProgress Example")

        self.set_default_size(200, 200)

        self.liststore = Gtk.ListStore(str, int, bool)
        self.current_iter = self.liststore.append(["Sabayon", 0, False])
        self.liststore.append(["Zenwalk", 0, False])
        self.liststore.append(["SimplyMepis", 0, False])

        treeview = Gtk.TreeView(model=self.liststore)

        renderer_text = Gtk.CellRendererText()
        column_text = Gtk.TreeViewColumn("Text", renderer_text, text=0)
        treeview.append_column(column_text)

        renderer_progress = Gtk.CellRendererProgress()
        column_progress = Gtk.TreeViewColumn(
            "Progress", renderer_progress, value=1, inverted=2
        )
        treeview.append_column(column_progress)

        renderer_toggle = Gtk.CellRendererToggle()
        renderer_toggle.connect("toggled", self.on_inverted_toggled)
        column_toggle = Gtk.TreeViewColumn("Inverted", renderer_toggle, active=2)
        treeview.append_column(column_toggle)

```

(continua na próxima página)

(continuação da página anterior)

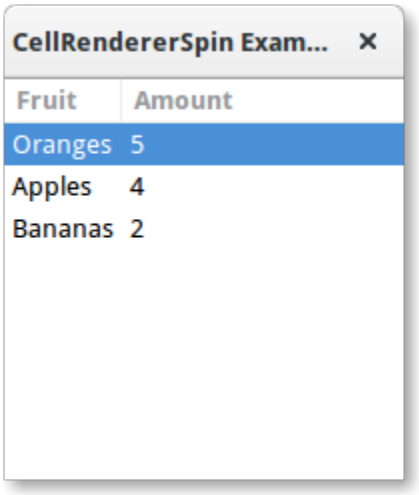
```
        self.add(treeview)

        self.timeout_id = GLib.timeout_add(100, self.on_timeout, None)

    def on_inverted_toggled(self, widget, path):
        self.liststore[path][2] = not self.liststore[path][2]

    def on_timeout(self, user_data):
        new_value = self.liststore[self.current_iter][1] + 1
        if new_value > 100:
            self.current_iter = self.liststore.iter_next(self.current_iter)
            if self.current_iter is None:
                self.reset_model()
            new_value = self.liststore[self.current_iter][1] + 1

        self.liststore[self.current_iter][1] = new_value
        return True

    def reset_model(self):
        for row in self.liststore:
            row[1] = 0
        self.current_iter = self.liststore.get_iter_first()


win = CellRendererProgressWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

14.6 CellRendererSpin

`Gtk.CellRendererSpin` processa o texto em uma célula como `Gtk.CellRendererText` do qual é derivado. Mas enquanto o último oferece uma entrada simples para editar o texto, `Gtk.CellRendererSpin` oferece um widget `Gtk.SpinButton`. Claro, isso significa que o texto deve ser analisado como um número de ponto flutuante.

O intervalo do botão de rotação é obtido da propriedade de ajuste do renderizador de célula, que pode ser definido explicitamente ou mapeado para uma coluna no modelo de árvore, como todas as propriedades dos renderizadores de célula. `Gtk.CellRendererSpin` também possui propriedades para a taxa de subida e o número de dígitos a serem exibidos.

14.6.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class CellRendererSpinWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="CellRendererSpin Example")

        self.set_default_size(200, 200)

        self.liststore = Gtk.ListStore(str, int)
        self.liststore.append(["Oranges", 5])
        self.liststore.append(["Apples", 4])
        self.liststore.append(["Bananas", 2])

        treeview = Gtk.TreeView(model=self.liststore)

        renderer_text = Gtk.CellRendererText()
        column_text = Gtk.TreeViewColumn("Fruit", renderer_text, text=0)
        treeview.append_column(column_text)

        renderer_spin = Gtk.CellRendererSpin()
        renderer_spin.connect("edited", self.on_amount_edited)
        renderer_spin.set_property("editable", True)

        adjustment = Gtk.Adjustment(
            value=0,
            lower=0,
            upper=100,
            step_increment=1,
            page_increment=10,
            page_size=0,
```

(continua na próxima página)

(continuação da página anterior)

```
        )
        renderer_spin.set_property("adjustment", adjustment)

        column_spin = Gtk.TreeViewColumn("Amount", renderer_spin, text=1)
        treeview.append_column(column_spin)

        self.add(treeview)

    def on_amount_edited(self, widget, path, value):
        self.liststore[path][1] = int(value)


win = CellRendererSpinWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 15

ComboBox

A Gtk.ComboBox permite a seleção de um item em um menu suspenso. Eles são preferíveis a ter muitos botões de opção na tela, pois ocupam menos espaço. Se apropriado, ele pode mostrar informações extras sobre cada item, como texto, uma imagem, uma caixa de seleção ou uma barra de progresso.

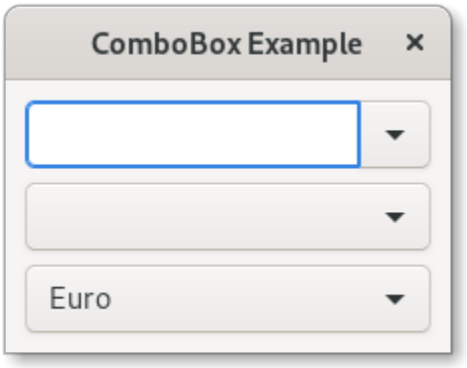
Gtk.ComboBox é muito similar a Gtk.TreeView, já que ambos usam o padrão model-view; A lista de opções válidas é especificada na forma de um modelo de árvore, e a exibição das opções pode ser adaptada aos dados no modelo usando *renderizadores de célula*. Se a caixa de combinação contiver um grande número de itens, talvez seja melhor exibi-los em uma grade em vez de em uma lista. Isso pode ser feito chamando Gtk.ComboBox.set_wrap_width().

Um valor padrão pode ser definido chamando Gtk.ComboBox.set_active() com o índice do valor desejado.

O widget Gtk.ComboBox geralmente restringe o usuário às opções disponíveis, mas ele pode opcionalmente ter um Gtk.Entry, permitindo que o usuário insira texto arbitrário se nenhuma das opções disponíveis for adequada. Para fazer isso, use um dos métodos estáticos Gtk.ComboBox.new_with_entry() ou Gtk.ComboBox.new_with_model_and_entry() para criar uma instância Gtk.ComboBox.

Para uma lista simples de escolhas textuais, a API de visão de modelo de Gtk.ComboBox pode ser um pouco avassaladora. Neste caso, Gtk.ComboBoxText oferece uma alternativa simples. Ambos Gtk.ComboBox e Gtk.ComboBoxText podem conter uma entrada.

15.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class ComboBoxWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="ComboBox Example")

        self.set_border_width(10)

        name_store = Gtk.ListStore(int, str)
        name_store.append([1, "Billy Bob"])
        name_store.append([11, "Billy Bob Junior"])
        name_store.append([12, "Sue Bob"])
        name_store.append([2, "Joey Jojo"])
        name_store.append([3, "Rob McRoberts"])
        name_store.append([31, "Xavier McRoberts"])

        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)

        name_combo = Gtk.ComboBox.new_with_model_and_entry(name_store)
        name_combo.connect("changed", self.on_name_combo_changed)
        name_combo.set_entry_text_column(1)
        vbox.pack_start(name_combo, False, False, 0)

        country_store = Gtk.ListStore(str)
        countries = [
            "Austria",
            "Brazil",
            "Belgium",
            "France",
            "Germany",
            "Switzerland",
            "United Kingdom",
```

(continua na próxima página)

(continuação da página anterior)

```
            "United States of America",
            "Uruguay",
        ]
        for country in countries:
            country_store.append([country])

        country_combo = Gtk.ComboBox.new_with_model(country_store)
        country_combo.connect("changed", self.on_country_combo_changed)
        renderer_text = Gtk.CellRendererText()
        country_combo.pack_start(renderer_text, True)
        country_combo.add_attribute(renderer_text, "text", 0)
        vbox.pack_start(country_combo, False, False, True)

        currencies = [
            "Euro",
            "US Dollars",
            "British Pound",
            "Japanese Yen",
            "Russian Ruble",
            "Mexican peso",
            "Swiss franc",
        ]
        currency_combo = Gtk.ComboBoxText()
        currency_combo.set_entry_text_column(0)
        currency_combo.connect("changed", self.on_currency_combo_changed)
        for currency in currencies:
            currency_combo.append_text(currency)

        currency_combo.set_active(0)
        vbox.pack_start(currency_combo, False, False, 0)

        self.add(vbox)

    def on_name_combo_changed(self, combo):
        tree_iter = combo.get_active_iter()
        if tree_iter is not None:
            model = combo.get_model()
            row_id, name = model[tree_iter][:2]
            print("Selected: ID=%d, name=%s" % (row_id, name))
        else:
            entry = combo.get_child()
            print("Entered: %s" % entry.get_text())

    def on_country_combo_changed(self, combo):
        tree_iter = combo.get_active_iter()
        if tree_iter is not None:
            model = combo.get_model()
            country = model[tree_iter][0]
            print("Selected: country=%s" % country)

    def on_currency_combo_changed(self, combo):
        text = combo.get_active_text()
```

(continua na próxima página)

(continuação da página anterior)

```
        if text is not None:
            print("Selected: currency=%s" % text)


win = ComboBoxWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 16

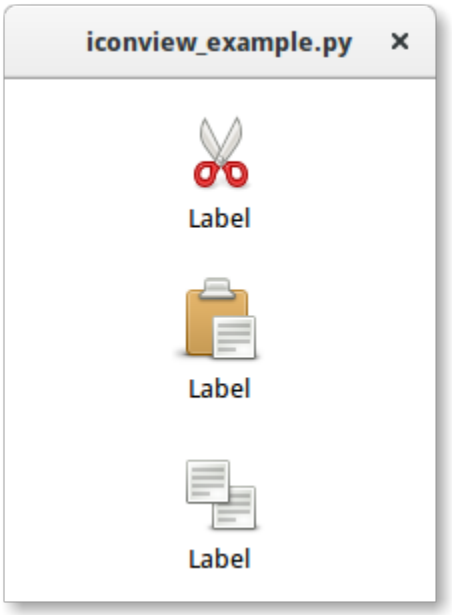
IconView

A `Gtk.IconView` é um widget que exibe uma coleção de ícones em uma visualização de grade. Ele possui suporte a recursos como arrastar e soltar, seleções múltiplas e reordenação de itens.

Similarmente a `Gtk.TreeView`, `Gtk.IconView` usa um `Gtk.ListStore` para seu modelo. Em vez de usar *renderizadores de célula*, `Gtk.IconView` requer que uma das colunas em seu `Gtk.ListStore` contenha objetos `GdkPixbuf.Pixbuf`.

`Gtk.IconView` possui suporte a vários modos de seleção para permitir a seleção de vários ícones por vez, restringindo seleções para apenas um item ou desaprovando a seleção de itens completamente. Para especificar um modo de seleção, o método `Gtk.IconView.set_selection_mode()` é usado com um dos modos de seleção `Gtk.SelectionMode`.

16.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk
from gi.repository.GdkPixbuf import Pixbuf

icons = ["edit-cut", "edit-paste", "edit-copy"]


class IconViewWindow(Gtk.Window):
    def __init__(self):
        super().__init__()
        self.set_default_size(200, 200)

        liststore = Gtk.ListStore(Pixbuf, str)
        iconview = Gtk.IconView.new()
        iconview.set_model(liststore)
        iconview.set_pixbuf_column(0)
        iconview.set_text_column(1)

        for icon in icons:
            pixbuf = Gtk.IconTheme.get_default().load_icon(icon, 64, 0)
            liststore.append([pixbuf, "Label"])

        self.add(iconview)


win = IconViewWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
```

(continua na próxima página)

(continuação da página anterior)

```
31 Gtk.main()
```

CAPÍTULO 17

Editor de Texto Multilinha

O widget `Gtk.TextView` pode ser usado para exibir e editar grandes quantidades de texto formatado. Como o `Gtk.TreeView`, ele possui um design de modelo/visualização. Neste caso, o `Gtk.TextBuffer` é o modelo que representa o texto que está sendo editado. Isto permite que dois ou mais widgets `Gtk.TextView` compartilhem o mesmo `Gtk.TextBuffer`, e permite que os buffers de texto sejam exibidos de forma ligeiramente diferente. Ou você pode manter vários buffers de texto e optar por exibir cada um deles em momentos diferentes no mesmo widget `Gtk.TextView`.

17.1 A visão

O `Gtk.TextView` é o frontend com o qual o usuário pode adicionar, editar e excluir dados textuais. Eles são comumente usados para editar várias linhas de texto. Ao criar um `Gtk.TextView` ele contém seu próprio padrão `Gtk.TextBuffer`, que você pode acessar através do método `Gtk.TextView.get_buffer()`.

Por padrão, o texto pode ser adicionado, editado e removido da `Gtk.TextView`. Você pode desabilitar isso chamando `Gtk.TextView.set_editable()`. Se o texto não for editável, você geralmente deseja ocultar o cursor de texto com `Gtk.TextView.set_cursor_visible()` também. Em alguns casos, pode ser útil definir a justificação do texto com `Gtk.TextView.set_justification()`. O texto pode ser exibido na borda da esquerda, (`Gtk.Justification.LEFT`), na borda da direita (`Gtk.Justification.RIGHT`), centralizado (`Gtk.Justification.CENTER`) ou distribuído em toda a largura (`Gtk.Justification.FILL`).

Outra configuração padrão do widget `Gtk.TextView` é que linhas longas de texto continuarão horizontalmente até que uma quebra seja inserida. Para encapsular o texto e impedir que ele saia das bordas da tela, chame `Gtk.TextView.set_wrap_mode()`.

17.2 O modelo

O Gtk.TextBuffer é o núcleo do widget Gtk.TextView e é usado para armazenar qualquer texto que esteja sendo exibido na Gtk.TextView. Definir e recuperar o conteúdo é possível com Gtk.TextBuffer.set_text() e Gtk.TextBuffer.get_text(). No entanto, a maior parte da manipulação de texto é realizada com *iteradores*, representados por um Gtk.TextIter. Um iterador representa uma posição entre dois caracteres no buffer de texto. Iteradores não são válidos indefinidamente; sempre que o buffer é modificado de uma maneira que afeta o conteúdo do buffer, todos os iteradores pendentes se tornam inválidos.

Por causa disso, os iteradores não podem ser usados para preservar posições nas modificações do buffer. Para preservar uma posição, use Gtk.TextMark. Um buffer de texto contém duas marcas internas; uma marca "insert" (que é a posição do cursor) e a marca "selection_bound". Ambos podem ser recuperados usando Gtk.TextBuffer.get_insert() e Gtk.TextBuffer.get_selection_bound(), respectivamente. Por padrão, a localização de um Gtk.TextMark não é mostrada. Isso pode ser alterado chamando Gtk.TextMark.set_visible().

Existem muitos métodos para recuperar um Gtk.TextIter. Por exemplo, Gtk.TextBuffer.get_start_iter() retorna um iterador apontando para a primeira posição no buffer de texto, enquanto Gtk.TextBuffer.get_end_iter() retorna um iterador apontando após o último caractere válido. A recuperação dos limites do texto selecionado pode ser obtida chamando Gtk.TextBuffer.get_selection_bounds().

Para inserir texto em uma posição específica use Gtk.TextBuffer.insert(). Outro método útil é Gtk.TextBuffer.insert_at_cursor() que insere texto onde quer que o cursor esteja posicionado no momento. Para remover partes do buffer de texto, use Gtk.TextBuffer.delete().

Além disso, Gtk.TextIter pode ser usado para localizar correspondências textuais no buffer usando Gtk.TextIter.forward_search() e Gtk.TextIter.backward_search(). Os iters inicial e final são usados como ponto de partida da pesquisa e avançam/retrocedem dependendo dos requisitos.

17.3 Tags

O texto em um buffer pode ser marcado com tags. Uma tag é um atributo que pode ser aplicado a um intervalo de texto. Por exemplo, uma tag pode ser chamada de "negrito" e tornar o texto dentro da tag em negrito. No entanto, o conceito de tag é mais geral do que isso; as tags não precisam afetar a aparência. Eles podem afetar o comportamento de pressionamentos de mouse e de tecla, "bloquear" um intervalo de texto para que o usuário não possa editá-lo ou inúmeras outras coisas. Uma tag é representada por um objeto Gtk.TextTag. Um Gtk.TextTag pode ser aplicado a qualquer número de intervalos de texto em qualquer número de buffers.

Cada tag é armazenada em Gtk.TextTagTable. Uma tabela de tags define um conjunto de tags que podem ser usadas juntas. Cada buffer tem uma tabela de tags associada a ele; somente tags dessa tabela de tags podem ser usadas com o buffer. No entanto, uma única tabela de tags pode ser compartilhada entre vários buffers.

Para especificar que algum texto no buffer deve ter uma formatação específica, você deve definir uma tag para manter as informações de formatação e, em seguida, aplicar essa tag à região do texto usando Gtk.TextBuffer.create_tag() e Gtk.TextBuffer.apply_tag():

```
tag = textbuffer.create_tag("orange_bg", background="orange")
textbuffer.apply_tag(tag, start_iter, end_iter)
```

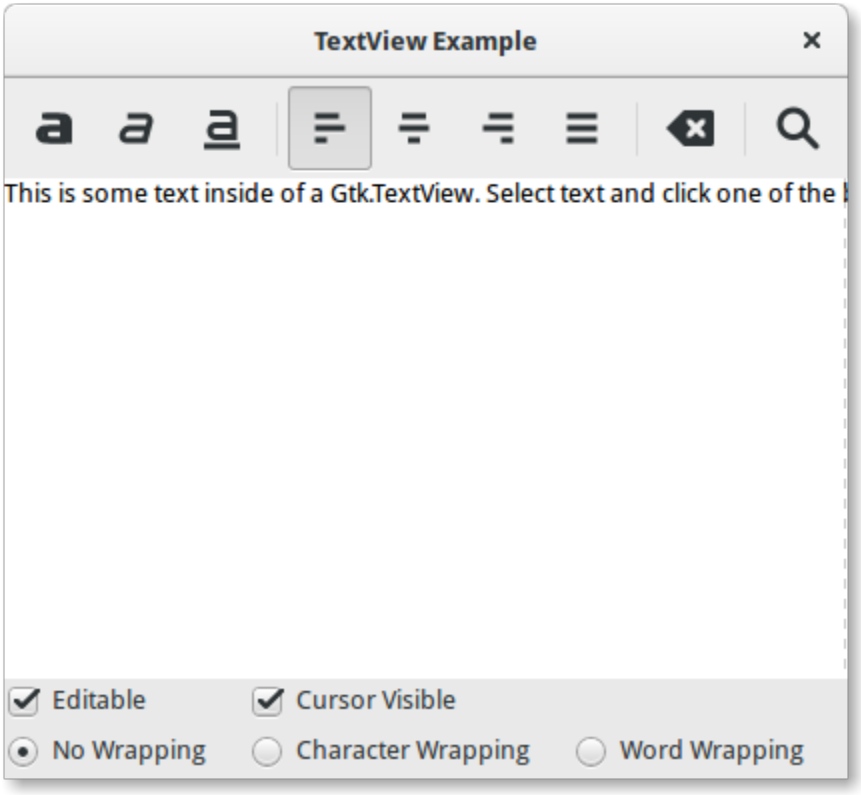
A seguir estão alguns dos estilos comuns aplicados ao texto:

- Cor de fundo (propriedade "background")
- Cor de primeiro plano (propriedade "background")
- Sublinhado (propriedade "underline")
- Negrito (propriedade "weight")

- Itálico (propriedade “style”)
- Tachado (propriedade “strikethrough”)
- Justificação (propriedade de “justification”)
- Tamanho (propriedades “size” e “size-points”)
- Quebra automática de texto (propriedade “wrap-mode”)

Você também pode excluir tags particulares posteriormente usando `Gtk.TextBuffer.remove_tag()` ou excluir todas as tags em uma determinada região chamando `Gtk.TextBuffer.remove_all_tags()`.

17.4 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, Pango


class SearchDialog(Gtk.Dialog):
    def __init__(self, parent):
        super().__init__(title="Search", transient_for=parent, modal=True)
        self.add_buttons(
            Gtk.STOCK_FIND,
            Gtk.ResponseType.OK,
            Gtk.STOCK_CANCEL,
```

(continua na próxima página)

(continuação da página anterior)

```
            Gtk.ResponseType.CANCEL,
        )

        box = self.get_content_area()

        label = Gtk.Label(label="Insert text you want to search for:")
        box.add(label)

        self.entry = Gtk.Entry()
        box.add(self.entry)

        self.show_all()


class TextViewWindow(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self, title="TextView Example")

        self.set_default_size(-1, 350)

        self.grid = Gtk.Grid()
        self.add(self.grid)

        self.create_textview()
        self.create_toolbar()
        self.create_buttons()

    def create_toolbar(self):
        toolbar = Gtk.Toolbar()
        self.grid.attach(toolbar, 0, 0, 3, 1)

        button_bold = Gtk.ToolButton()
        button_bold.set_icon_name("format-text-bold-symbolic")
        toolbar.insert(button_bold, 0)

        button_italic = Gtk.ToolButton()
        button_italic.set_icon_name("format-text-italic-symbolic")
        toolbar.insert(button_italic, 1)

        button_underline = Gtk.ToolButton()
        button_underline.set_icon_name("format-text-underline-symbolic")
        toolbar.insert(button_underline, 2)

        button_bold.connect("clicked", self.on_button_clicked, self.tag_bold)
        button_italic.connect("clicked", self.on_button_clicked, self.tag_italic)
        button_underline.connect("clicked", self.on_button_clicked, self.tag_underline)

        toolbar.insert(Gtk.SeparatorToolItem(), 3)

        radio_justifyleft = Gtk.RadioToolButton()
        radio_justifyleft.set_icon_name("format-justify-left-symbolic")
        toolbar.insert(radio_justifyleft, 4)
```

(continua na próxima página)

(continuação da página anterior)

```
        radio_justifycenter = Gtk.RadioToolButton.new_from_widget(radio_justifyleft)
        radio_justifycenter.set_icon_name("format-justify-center-symbolic")
        toolbar.insert(radio_justifycenter, 5)

        radio_justifyright = Gtk.RadioToolButton.new_from_widget(radio_justifyleft)
        radio_justifyright.set_icon_name("format-justify-right-symbolic")
        toolbar.insert(radio_justifyright, 6)

        radio_justifyfill = Gtk.RadioToolButton.new_from_widget(radio_justifyleft)
        radio_justifyfill.set_icon_name("format-justify-fill-symbolic")
        toolbar.insert(radio_justifyfill, 7)

        radio_justifyleft.connect(
            "toggled", self.on_justify_toggled, Gtk.Justification.LEFT
        )
        radio_justifycenter.connect(
            "toggled", self.on_justify_toggled, Gtk.Justification.CENTER
        )
        radio_justifyright.connect(
            "toggled", self.on_justify_toggled, Gtk.Justification.RIGHT
        )
        radio_justifyfill.connect(
            "toggled", self.on_justify_toggled, Gtk.Justification.FILL
        )

        toolbar.insert(Gtk.SeparatorToolItem(), 8)

        button_clear = Gtk.ToolButton()
        button_clear.set_icon_name("edit-clear-symbolic")
        button_clear.connect("clicked", self.on_clear_clicked)
        toolbar.insert(button_clear, 9)

        toolbar.insert(Gtk.SeparatorToolItem(), 10)

        button_search = Gtk.ToolButton()
        button_search.set_icon_name("system-search-symbolic")
        button_search.connect("clicked", self.on_search_clicked)
        toolbar.insert(button_search, 11)

    def create_textview(self):
        scrolledwindow = Gtk.ScrolledWindow()
        scrolledwindow.set_hexpand(True)
        scrolledwindow.set_vexpand(True)
        self.grid.attach(scrolledwindow, 0, 1, 3, 1)

        self.textview = Gtk.TextView()
        self.textbuffer = self.textview.get_buffer()
        self.textbuffer.set_text(
            "This is some text inside of a Gtk.TextView. "
            + "Select text and click one of the buttons 'bold', 'italic', "
            + "or 'underline' to modify the text accordingly."
```

(continua na próxima página)

(continuação da página anterior)

```
        )
        scrolledwindow.add(self.textview)

        self.tag_bold = self.textbuffer.create_tag("bold", weight=Pango.Weight.BOLD)
        self.tag_italic = self.textbuffer.create_tag("italic", style=Pango.Style.ITALIC)
        self.tag_underline = self.textbuffer.create_tag(
            "underline", underline=Pango.Underline.SINGLE
        )
        self.tag_found = self.textbuffer.create_tag("found", background="yellow")

    def create_buttons(self):
        check_editable = Gtk.CheckButton(label="Editable")
        check_editable.set_active(True)
        check_editable.connect("toggled", self.on_editable_toggled)
        self.grid.attach(check_editable, 0, 2, 1, 1)

        check_cursor = Gtk.CheckButton(label="Cursor Visible")
        check_cursor.set_active(True)
        check_editable.connect("toggled", self.on_cursor_toggled)
        self.grid.attach_next_to(
            check_cursor, check_editable, Gtk.PositionType.RIGHT, 1, 1
        )

        radio_wrapnone = Gtk.RadioButton.new_with_label_from_widget(None, "No Wrapping")
        self.grid.attach(radio_wrapnone, 0, 3, 1, 1)

        radio_wrapchar = Gtk.RadioButton.new_with_label_from_widget(
            radio_wrapnone, "Character Wrapping"
        )
        self.grid.attach_next_to(
            radio_wrapchar, radio_wrapnone, Gtk.PositionType.RIGHT, 1, 1
        )

        radio_wrapword = Gtk.RadioButton.new_with_label_from_widget(
            radio_wrapnone, "Word Wrapping"
        )
        self.grid.attach_next_to(
            radio_wrapword, radio_wrapchar, Gtk.PositionType.RIGHT, 1, 1
        )

        radio_wrapnone.connect("toggled", self.on_wrap_toggled, Gtk.WrapMode.NONE)
        radio_wrapchar.connect("toggled", self.on_wrap_toggled, Gtk.WrapMode.CHAR)
        radio_wrapword.connect("toggled", self.on_wrap_toggled, Gtk.WrapMode.WORD)

    def on_button_clicked(self, widget, tag):
        bounds = self.textbuffer.get_selection_bounds()
        if len(bounds) != 0:
            start, end = bounds
            self.textbuffer.apply_tag(tag, start, end)

    def on_clear_clicked(self, widget):
        start = self.textbuffer.get_start_iter()
```

(continua na próxima página)

(continuação da página anterior)

```
        end = self.textbuffer.get_end_iter()
        self.textbuffer.remove_all_tags(start, end)

    def on_editable_toggled(self, widget):
        self.textview.set_editable(widget.get_active())

    def on_cursor_toggled(self, widget):
        self.textview.set_cursor_visible(widget.get_active())

    def on_wrap_toggled(self, widget, mode):
        self.textview.set_wrap_mode(mode)

    def on_justify_toggled(self, widget, justification):
        self.textview.set_justification(justification)

    def on_search_clicked(self, widget):
        dialog = SearchDialog(self)
        response = dialog.run()
        if response == Gtk.ResponseType.OK:
            cursor_mark = self.textbuffer.get_insert()
            start = self.textbuffer.get_iter_at_mark(cursor_mark)
            if start.get_offset() == self.textbuffer.get_char_count():
                start = self.textbuffer.get_start_iter()

            self.search_and_mark(dialog.entry.get_text(), start)

        dialog.destroy()

    def search_and_mark(self, text, start):
        end = self.textbuffer.get_end_iter()
        match = start.forward_search(text, 0, end)

        if match is not None:
            match_start, match_end = match
            self.textbuffer.apply_tag(self.tag_found, match_start, match_end)
            self.search_and_mark(text, match_end)


win = TextViewWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 18

Diálogos

As janelas de caixa de diálogo são muito semelhantes às janelas padrão e são usadas para fornecer ou recuperar informações do usuário. Eles são frequentemente usados para fornecer uma janela de preferências, por exemplo. A principal diferença que uma caixa de diálogo tem é alguns widgets pré-empacotados que organizam a caixa de diálogo automaticamente. A partir daí, podemos simplesmente adicionar rótulos, botões, botões de seleção, etc. Outra grande diferença é o tratamento de respostas para controlar como o aplicativo deve se comportar após a interação com a caixa de diálogo.

Existem várias classes de diálogo derivadas que você pode achar útil. `Gtk.MessageDialog` é usado para notificações mais simples. Porém, em outras ocasiões, você pode precisar derivar sua própria classe de diálogo para fornecer uma funcionalidade mais complexa.

18.1 Dialogos personalizados

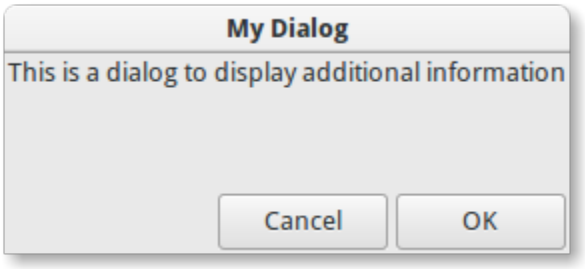
Para empacotar widgets em um diálogo personalizado, você deve empacotá-los no `Gtk.Box`, disponível via `Gtk.Dialog.get_content_area()`. Para adicionar apenas um `Gtk.Button` ao final do diálogo, você poderia usar o método `Gtk.Dialog.add_button()`.

Um diálogo "modal" (isto é, um que congela o resto do aplicativo da entrada do usuário), pode ser criado chamando `Gtk.Dialog.set_modal` no diálogo ou setando o argumento `flags` do o construtor `Gtk.Dialog` para incluir o sinalizador `Gtk.DialogFlags.MODAL`.

Clicar em um botão irá emitir um sinal chamado "response". Se você quiser bloquear a espera de um diálogo para retornar antes do retorno do fluxo de controle para o seu código, você pode chamar `Gtk.Dialog.run()`. Este método retorna um int que pode ser um valor de `Gtk.ResponseType` ou pode ser o valor de resposta personalizado que você especificou no construtor `Gtk.Dialog` ou `Gtk.Dialog.add_button()`.

Finalmente, existem duas maneiras de remover um diálogo. O método `Gtk.Widget.hide()` remove a caixa de diálogo da visualização, mas mantém armazenada na memória. Isso é útil para evitar a necessidade de construir a caixa de diálogo novamente se precisar ser acessada posteriormente. Alternativamente, o método `Gtk.Widget.destroy()` pode ser usado para excluir o diálogo da memória, uma vez que não é mais necessário. Deve ser notado que se o diálogo precisar ser acessado depois de ter sido destruído, ele precisará ser construído novamente, caso contrário a janela de diálogo estará vazia.

18.1.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class DialogExample(Gtk.Dialog):
    def __init__(self, parent):
        super().__init__(title="My Dialog", transient_for=parent, flags=0)
        self.add_buttons(
            Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL, Gtk.STOCK_OK, Gtk.ResponseType.OK
        )

        self.set_default_size(150, 100)

        label = Gtk.Label(label="This is a dialog to display additional information")

        box = self.get_content_area()
        box.add(label)
        self.show_all()


class DialogWindow(Gtk.Window):
    def __init__(self):
        Gtk.Window.__init__(self, title="Dialog Example")

        self.set_border_width(6)

        button = Gtk.Button(label="Open dialog")
        button.connect("clicked", self.on_button_clicked)

        self.add(button)

    def on_button_clicked(self, widget):
        dialog = DialogExample(self)
        response = dialog.run()

        if response == Gtk.ResponseType.OK:
            print("The OK button was clicked")
        elif response == Gtk.ResponseType.CANCEL:
            print("The Cancel button was clicked")
```

(continua na próxima página)

(continuação da página anterior)

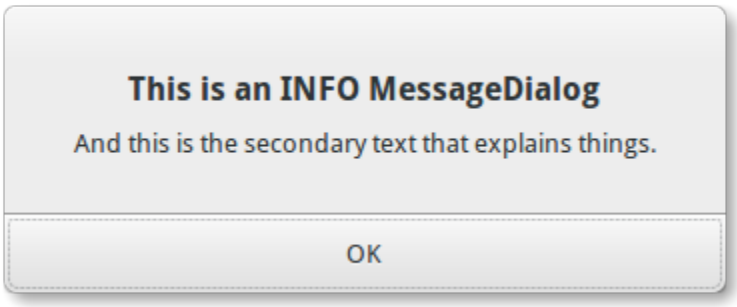
```
42
43          dialog.destroy()
44
45
46  win = DialogWindow()
47  win.connect("destroy", Gtk.main_quit)
48  win.show_all()
49  Gtk.main()
```

18.2 MessageDialog

`Gtk.MessageDialog` é uma classe de conveniência, usada para criar diálogos de mensagem simples e padrão, com uma mensagem, um ícone e botões para resposta do usuário. Você pode especificar o tipo de mensagem e o texto no construtor `Gtk.MessageDialog`, além de especificar botões padrão.

Em alguns diálogos que requerem alguma explicação adicional do que aconteceu, um texto secundário pode ser adicionado. Nesse caso, a mensagem principal inserida ao criar a caixa de diálogo da mensagem é maior e definida como texto em negrito. A mensagem secundária pode ser definida chamando `Gtk.MessageDialog.format_secondary_text()`.

18.2.1 Exemplo




```
1   import gi
2
3   gi.require_version("Gtk", "3.0")
4   from gi.repository import Gtk
5
6
7   class MessageDialogWindow(Gtk.Window):
8       def __init__(self):
9           super().__init__(title="MessageDialog Example")
10
11          box = Gtk.Box(spacing=6)
12          self.add(box)
13
14          button1 = Gtk.Button(label="Information")
15          button1.connect("clicked", self.on_info_clicked)
16          box.add(button1)
```

(continua na próxima página)

(continuação da página anterior)

```
        button2 = Gtk.Button(label="Error")
        button2.connect("clicked", self.on_error_clicked)
        box.add(button2)

        button3 = Gtk.Button(label="Warning")
        button3.connect("clicked", self.on_warn_clicked)
        box.add(button3)

        button4 = Gtk.Button(label="Question")
        button4.connect("clicked", self.on_question_clicked)
        box.add(button4)

    def on_info_clicked(self, widget):
        dialog = Gtk.MessageDialog(
            transient_for=self,
            flags=0,
            message_type=Gtk.MessageType.INFO,
            buttons=Gtk.ButtonsType.OK,
            text="This is an INFO MessageDialog",
        )
        dialog.format_secondary_text(
            "And this is the secondary text that explains things."
        )
        dialog.run()
        print("INFO dialog closed")

        dialog.destroy()

    def on_error_clicked(self, widget):
        dialog = Gtk.MessageDialog(
            transient_for=self,
            flags=0,
            message_type=Gtk.MessageType.ERROR,
            buttons=Gtk.ButtonsType.CANCEL,
            text="This is an ERROR MessageDialog",
        )
        dialog.format_secondary_text(
            "And this is the secondary text that explains things."
        )
        dialog.run()
        print("ERROR dialog closed")

        dialog.destroy()

    def on_warn_clicked(self, widget):
        dialog = Gtk.MessageDialog(
            transient_for=self,
            flags=0,
            message_type=Gtk.MessageType.WARNING,
            buttons=Gtk.ButtonsType.OK_CANCEL,
            text="This is an WARNING MessageDialog",
```

(continua na próxima página)

(continuação da página anterior)

```
        )
        dialog.format_secondary_text(
            "And this is the secondary text that explains things."
        )
        response = dialog.run()
        if response == Gtk.ResponseType.OK:
            print("WARN dialog closed by clicking OK button")
        elif response == Gtk.ResponseType.CANCEL:
            print("WARN dialog closed by clicking CANCEL button")

        dialog.destroy()

    def on_question_clicked(self, widget):
        dialog = Gtk.MessageDialog(
            transient_for=self,
            flags=0,
            message_type=Gtk.MessageType.QUESTION,
            buttons=Gtk.ButtonsType.YES_NO,
            text="This is an QUESTION MessageDialog",
        )
        dialog.format_secondary_text(
            "And this is the secondary text that explains things."
        )
        response = dialog.run()
        if response == Gtk.ResponseType.YES:
            print("QUESTION dialog closed by clicking YES button")
        elif response == Gtk.ResponseType.NO:
            print("QUESTION dialog closed by clicking NO button")

        dialog.destroy()


win = MessageDialogWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

18.3 FileChooserDialog

O `Gtk.FileChooserDialog` é adequado para uso com itens de menu “Arquivo/Abrir” ou “Arquivo/Salvar”. Você pode usar todos os métodos `Gtk.FileChooser` no diálogo do seletor de arquivos, assim como aqueles para `Gtk.Dialog`.

Ao criar um `Gtk.FileChooserDialog` você precisa definir o propósito do diálogo:

- Para selecionar um arquivo para abertura, como para um comando Arquivo/Abrir, use `Gtk.FileChooserAction.OPEN`
- Para salvar um arquivo pela primeira vez, como para um comando Arquivo/Salvar, use `Gtk.FileChooserAction.SAVE` e sugira um nome como “Untitled” com `Gtk.FileChooser.set_current_name()`.

- Para salvar um arquivo com um nome diferente, como para um comando Arquivo/Salvar como, use `Gtk.FileChooserAction.SAVE` e defina o nome do arquivo existente como `Gtk.FileChooser.set_filename()`.
- Para escolher uma pasta em vez de um arquivo, use `Gtk.FileChooserAction.SELECT_FOLDER`.

`Gtk.FileChooserDialog` herda de `Gtk.Dialog`, então os botões possuem IDs de resposta como `Gtk.ResponseType.ACCEPT` e `Gtk.ResponseType.CANCEL`, que pode ser especificado no construtor `Gtk.FileChooserDialog`. Em contraste com `Gtk.Dialog`, você não pode usar códigos de resposta customizados com `Gtk.FileChooserDialog`. Espera que pelo menos um botão tenha os seguintes IDs de resposta:

- `Gtk.ResponseType.ACCEPT`
- `Gtk.ResponseType.OK`
- `Gtk.ResponseType.YES`
- `Gtk.ResponseType.APPLY`

Quando o usuário terminar de selecionar arquivos, seu programa pode obter os nomes selecionados como nomes de arquivos (`Gtk.FileChooser.get_filename()`) ou como URIs (`Gtk.FileChooser.get_uri()`).

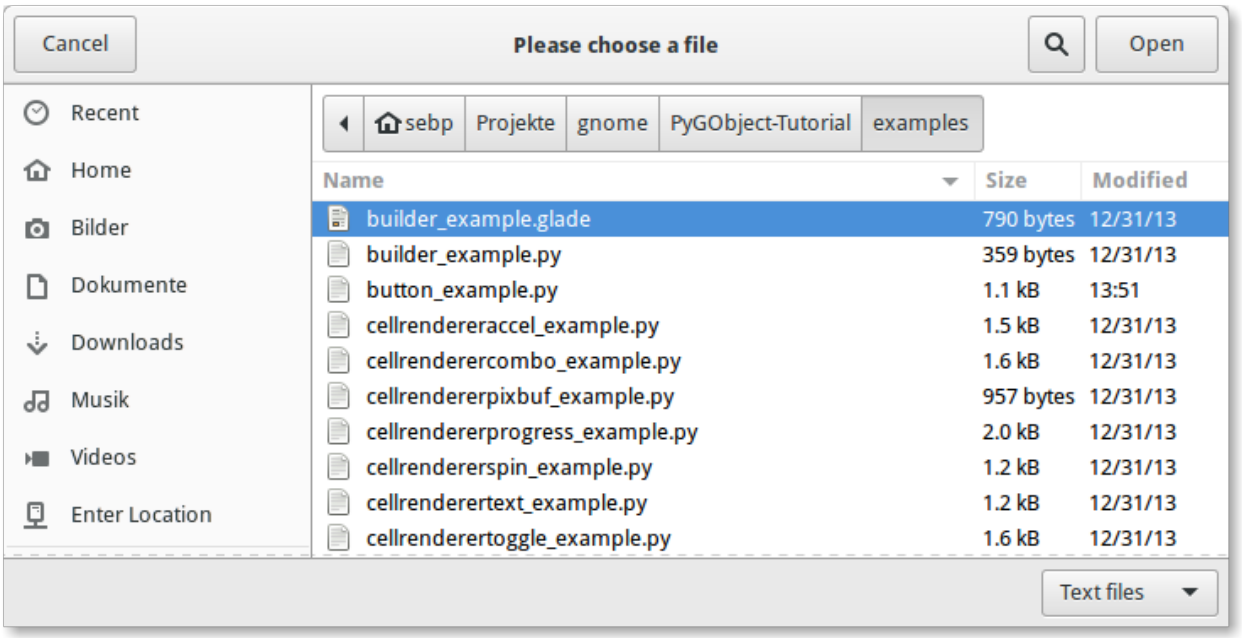
Por padrão, `Gtk.FileChooser` permite apenas que um único arquivo seja selecionado por vez. Para permitir que vários arquivos sejam selecionados, use `Gtk.FileChooser.set_select_multiple()`. Recuperar uma lista de arquivos selecionados é possível com `Gtk.FileChooser.get_filenames()` ou `Gtk.FileChooser.get_uris()`.

`Gtk.FileChooser` também possui suporte a uma variedade de opções que tornam os arquivos e pastas mais configuráveis e acessíveis.

- `Gtk.FileChooser.set_local_only()`: Somente arquivos locais podem ser selecionados.
- `Gtk.FileChooser.show_hidden()`: Arquivos e pastas ocultos são exibidos.
- `Gtk.FileChooser.set_do_overwrite_confirmation()`: Se o seletor de arquivos foi configurado no modo `Gtk.FileChooserAction.SAVE`, ele apresentará um diálogo de confirmação se o usuário digitar um nome de arquivo que já existe.

Além disso, você pode especificar quais tipos de arquivos são exibidos criando objetos `Gtk.FileFilter` e chamando `Gtk.FileChooser.add_filter()`. O usuário pode selecionar um dos filtros adicionados em uma caixa de combinação na parte inferior do seletor de arquivos.

18.3.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class FileChooserWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="FileChooser Example")

        box = Gtk.Box(spacing=6)
        self.add(box)

        button1 = Gtk.Button(label="Choose File")
        button1.connect("clicked", self.on_file_clicked)
        box.add(button1)

        button2 = Gtk.Button(label="Choose Folder")
        button2.connect("clicked", self.on_folder_clicked)
        box.add(button2)

    def on_file_clicked(self, widget):
        dialog = Gtk.FileChooserDialog(
            title="Please choose a file", parent=self, action=Gtk.FileChooserAction.OPEN
        )
        dialog.add_buttons(
            Gtk.STOCK_CANCEL,
            Gtk.ResponseType.CANCEL,
            Gtk.STOCK_OPEN,
            Gtk.ResponseType.OK,
```

(continua na próxima página)

(continuação da página anterior)

```
        )

        self.add_filters(dialog)

        response = dialog.run()
        if response == Gtk.ResponseType.OK:
            print("Open clicked")
            print("File selected: " + dialog.get_filename())
        elif response == Gtk.ResponseType.CANCEL:
            print("Cancel clicked")

        dialog.destroy()

    def add_filters(self, dialog):
        filter_text = Gtk.FileFilter()
        filter_text.set_name("Text files")
        filter_text.add_mime_type("text/plain")
        dialog.add_filter(filter_text)

        filter_py = Gtk.FileFilter()
        filter_py.set_name("Python files")
        filter_py.add_mime_type("text/x-python")
        dialog.add_filter(filter_py)

        filter_any = Gtk.FileFilter()
        filter_any.set_name("Any files")
        filter_any.add_pattern("*")
        dialog.add_filter(filter_any)

    def on_folder_clicked(self, widget):
        dialog = Gtk.FileChooserDialog(
            title="Please choose a folder",
            parent=self,
            action=Gtk.FileChooserAction.SELECT_FOLDER,
        )
        dialog.add_buttons(
            Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL, "Select", Gtk.ResponseType.OK
        )
        dialog.set_default_size(800, 400)

        response = dialog.run()
        if response == Gtk.ResponseType.OK:
            print("Select clicked")
            print("Folder selected: " + dialog.get_filename())
        elif response == Gtk.ResponseType.CANCEL:
            print("Cancel clicked")

        dialog.destroy()


win = FileChooserWindow()
win.connect("destroy", Gtk.main_quit)
```

(continua na próxima página)

(continuação da página anterior)

```
83 win.show_all()
84 Gtk.main()
```

CAPÍTULO 19

Popovers

O `Gtk.Popover` é uma janela separada usada para exibir informações adicionais e é frequentemente usada como parte de menus de botão e menus de contexto. Os popovers estão visualmente conectados a um widget relacionado com um pequeno triângulo. Seus usos são semelhantes aos das janelas de diálogo, com a vantagem de ser menos prejudicial e ter uma conexão com o widget para o qual o popover está apontando.

Um Popover pode ser criado com `Gtk.Popover`; para abrir o popover, use `Gtk.Popover.popup()`.

19.1 Popover Personalizado

Um widget pode ser adicionado a um popover usando o `Gtk.Container.add()`.

19.1.1 Exemplo

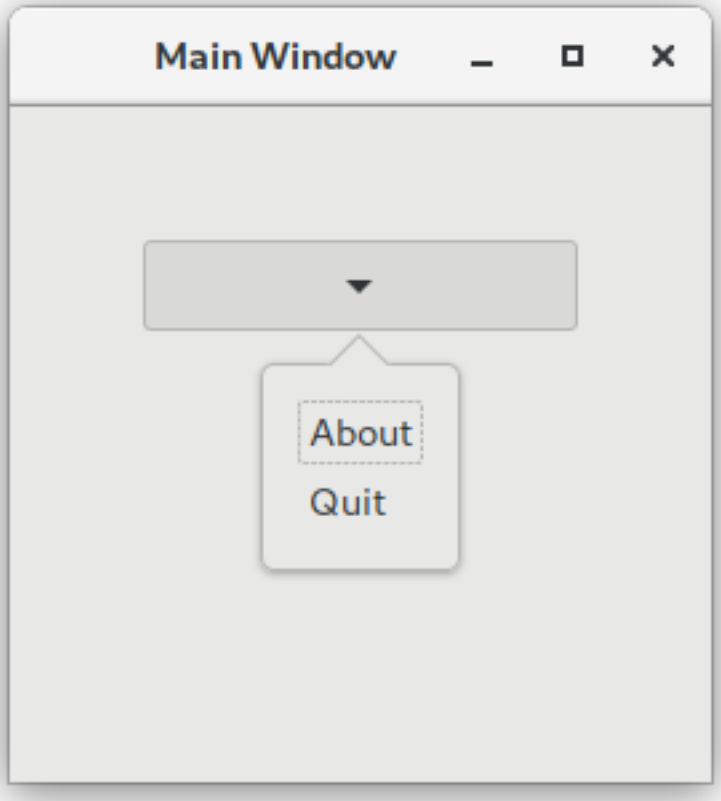

```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class PopoverWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Popover Demo")
        self.set_border_width(10)
        self.set_default_size(300, 200)

        outerbox = Gtk.Box(spacing=6, orientation=Gtk.Orientation.VERTICAL)
        self.add(outerbox)

        self.popover = Gtk.Popover()
        vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
        vbox.pack_start(Gtk.ModelButton(label="Item 1"), False, True, 10)
        vbox.pack_start(Gtk.Label(label="Item 2"), False, True, 10)
        vbox.show_all()
        self.popover.add(vbox)
        self.popover.set_position(Gtk.PositionType.BOTTOM)

        button = Gtk.MenuButton(label="Click Me", popover=self.popover)
        outerbox.pack_start(button, False, True, 0)


win = PopoverWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

19.2 Popover de menu

Um popover pode ser criado a partir de `Gio.MenuModel` usando `Gtk.Popover.new_from_model()` e pode ser alterado após a criação com `Gtk.Popover.bind_model()`.

Passar um `Gio.MenuModel` como um argumento `menu_model` para o construtor de `Gtk.MenuButton` cria implicitamente um popover.

19.2.1 Exemplo




```
import sys

import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gio, Gtk

# This would typically be its own file
MENU_XML = """
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <menu id="app-menu">
    <section>
```

(continua na próxima página)

(continuação da página anterior)

```
        <item>
            <attribute name="label">About</attribute>
            <attribute name="action">app.about</attribute>
        </item>
        <item>
            <attribute name="label">Quit</attribute>
            <attribute name="action">app.quit</attribute>
        </item>
    </section>
  </menu>
</interface>
"""


class AppWindow(Gtk.ApplicationWindow):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.set_default_size(300, 200)

        outerbox = Gtk.Box(spacing=6, orientation=Gtk.Orientation.VERTICAL)
        self.add(outerbox)
        outerbox.show()

        builder = Gtk.Builder.new_from_string(MENU_XML, -1)
        menu = builder.get_object("app-menu")

        button = Gtk.MenuButton(menu_model=menu)

        outerbox.pack_start(button, False, True, 0)
        button.show()
        self.set_border_width(50)


class Application(Gtk.Application):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, application_id="org.example.myapp", **kwargs)
        self.window = None

    def do_startup(self):
        Gtk.Application.do_startup(self)

        action = Gio.SimpleAction(name="about")
        action.connect("activate", self.on_about)
        self.add_action(action)

        action = Gio.SimpleAction(name="quit")
        action.connect("activate", self.on_quit)
        self.add_action(action)

    def do_activate(self):
        # We only allow a single window and raise any existing ones
        if not self.window:
```

(continua na próxima página)

(continuação da página anterior)

```
            # Windows are associated with the application
            # when the last one is closed the application shuts down
            self.window = AppWindow(application=self, title="Main Window")

        self.window.present()

    def on_about(self, action, param):
        about_dialog = Gtk.AboutDialog(transient_for=self.window, modal=True)
        about_dialog.present()

    def on_quit(self, action, param):
        self.quit()


if __name__ == "__main__":
    app = Application()
    app.run(sys.argv)
```

19.3 Veja também

- https://developer.gnome.org/hig/patterns/containers/popovers.html

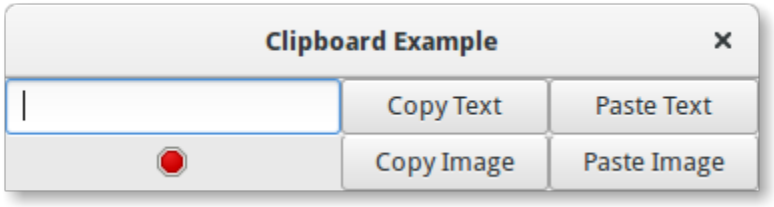
CAPÍTULO 20

Clipboard

`Gtk.Clipboard` fornece uma área de armazenamento para uma variedade de dados, incluindo texto e imagens. O uso de uma área de transferência permite que esses dados sejam compartilhados entre aplicativos por meio de ações como copiar, cortar e colar. Essas ações geralmente são feitas de três maneiras: usando atalhos de teclado, usando um `Gtk.MenuItem` e conectando as funções aos widgets `Gtk.Button`.

Existem várias seleções da área de transferência para finalidades diferentes. Na maioria das circunstâncias, a seleção chamada `CLIPBOARD` é usada para copiar e colar todos os dias. `PRIMARY` é outra seleção comum que armazena texto selecionado pelo usuário com o cursor.

20.1 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, Gdk


class ClipboardWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Clipboard Example")

        grid = Gtk.Grid()
```

(continua na próxima página)

(continuação da página anterior)

```
        self.clipboard = Gtk.Clipboard.get(Gdk.SELECTION_CLIPBOARD)
        self.entry = Gtk.Entry()
        self.image = Gtk.Image.new_from_icon_name("process-stop", Gtk.IconSize.MENU)

        button_copy_text = Gtk.Button(label="Copy Text")
        button_paste_text = Gtk.Button(label="Paste Text")
        button_copy_image = Gtk.Button(label="Copy Image")
        button_paste_image = Gtk.Button(label="Paste Image")

        grid.add(self.entry)
        grid.attach(self.image, 0, 1, 1, 1)
        grid.attach(button_copy_text, 1, 0, 1, 1)
        grid.attach(button_paste_text, 2, 0, 1, 1)
        grid.attach(button_copy_image, 1, 1, 1, 1)
        grid.attach(button_paste_image, 2, 1, 1, 1)

        button_copy_text.connect("clicked", self.copy_text)
        button_paste_text.connect("clicked", self.paste_text)
        button_copy_image.connect("clicked", self.copy_image)
        button_paste_image.connect("clicked", self.paste_image)

        self.add(grid)

    def copy_text(self, widget):
        self.clipboard.set_text(self.entry.get_text(), -1)

    def paste_text(self, widget):
        text = self.clipboard.wait_for_text()
        if text is not None:
            self.entry.set_text(text)
        else:
            print("No text on the clipboard.")

    def copy_image(self, widget):
        if self.image.get_storage_type() == Gtk.ImageType.PIXBUF:
            self.clipboard.set_image(self.image.get_pixbuf())
        else:
            print("No image has been pasted yet.")

    def paste_image(self, widget):
        image = self.clipboard.wait_for_image()
        if image is not None:
            self.image.set_from_pixbuf(image)


win = ClipboardWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 21

Arrastar e soltar

Nota: As versões do PyGObject < 3.0.3 contêm um bug que não permite arrastar e soltar para funcionar corretamente. Portanto, uma versão do PyGObject >= 3.0.3 é necessária para os exemplos a seguir funcionarem.

Configurar arrastar e soltar entre widgets consiste em selecionar uma fonte de arrasto (o widget do qual o usuário começa a arrastar) com o método `Gtk.Widget.drag_source_set()`, selecionando um destino de arrasto (o widget que o usuário coloca em) com o método `Gtk.Widget.drag_dest_set()` e depois manipular os sinais relevantes em ambos os widgets.

Em vez de usar `Gtk.Widget.drag_source_set()` e `Gtk.Widget.drag_dest_set()` alguns widgets especializados requerem o uso de funções específicas (como `Gtk.TreeView` e `Gtk.IconView`).

Um arrastar e soltar básico requer apenas que a fonte se conecte ao sinal "drag-data-get" e que o destino se conecte ao sinal "drag-data-received". Coisas mais complexas, como áreas de queda específicas e ícones de arrastar personalizados, exigirão que você se conecte a *sinais adicionais* e interaja com o objeto `Gdk.DragContext` que fornece.

Para transferir dados entre a origem e o destino, você deve interagir com a variável `Gtk.SelectionData` fornecida nos sinais *"drag-data-get"* e *"drag-data-received"* usando os métodos get e set de `Gtk.SelectionData`.

21.1 Entradas de alvo

Para permitir que a origem e o destino do arrastar saibam quais dados estão recebendo e enviando, uma lista comum de `Gtk.TargetEntry`s é necessária. A `Gtk.TargetEntry` descreve um dado que será enviado pela fonte de arrasto e recebido pelo destino do arrasto.

Existem duas maneiras de adicionar `Gtk.TargetEntry`s a uma origem e destino. Se o arrastar e soltar for simples e cada entrada de destino for de um tipo diferente, você pode usar o grupo de métodos mencionado aqui <`Gtk.Widget.drag_source_add_text_targets`>`()`.

Se você precisar de mais de um tipo de dados ou quiser fazer coisas mais complexas com os dados, você precisará criar o `Gtk.TargetEntry`s usando o método `Gtk.TargetEntry.new()`.

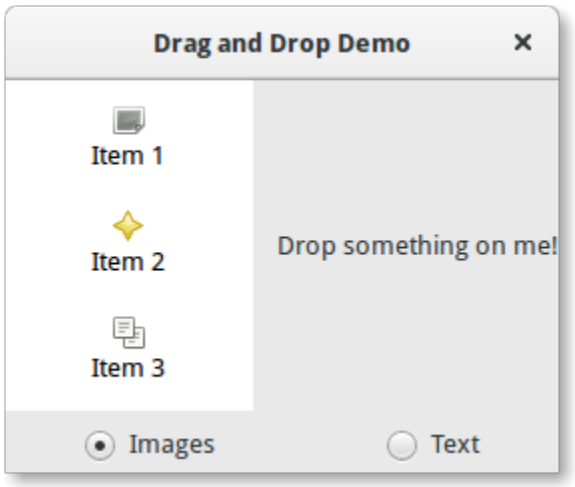
21.2 Sinais de origem do arrasto

Nome	Quando é emitido	Propósito comum
drag-begin	Usuário inicia um arrasto	Configurar ícone de arrasto
drag-data-get	Quando dados do arrasto são solicitados pelo destino	Transferir dados do arrasto da origem para o destino
drag-data-delete	Quando um arrasto com a ação Gdk.DragAction.MOVE é concluído	Excluir dados da origem para concluir o "movimento"
drag-end	Quando o arrasto estiver concluído	Desfazer qualquer coisa feita no drag-begin

21.3 Sinais de destino do arrasto

Nome	Quando é emitido	Propósito comum
drag-motion	O ícone de arrasto se move sobre uma área de soltar	Permitir que apenas algumas áreas sejam soltas
drag-drop	O ícone é solto em uma área de arrasto	Permitir que apenas algumas áreas sejam soltas
drag-data-received	Quando dados do arrasto são recebidos pelo destino	Transferir dados do arrasto da origem para o destino

21.4 Exemplo




```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk, Gdk, GdkPixbuf

(TARGET_ENTRY_TEXT, TARGET_ENTRY_PIXBUF) = range(2)
(COLUMN_TEXT, COLUMN_PIXBUF) = range(2)
```

(continua na próxima página)

(continuação da página anterior)

```
8
9  DRAG_ACTION = Gdk.DragAction.COPY
10
11
12 class DragDropWindow(Gtk.Window):
13     def __init__(self):
14         super().__init__(title="Drag and Drop Demo")
15
16         vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
17         self.add(vbox)
18
19         hbox = Gtk.Box(spacing=12)
20         vbox.pack_start(hbox, True, True, 0)
21
22         self.iconview = DragSourceIconView()
23         self.drop_area = DropArea()
24
25         hbox.pack_start(self.iconview, True, True, 0)
26         hbox.pack_start(self.drop_area, True, True, 0)
27
28         button_box = Gtk.Box(spacing=6)
29         vbox.pack_start(button_box, True, False, 0)
30
31         image_button = Gtk.RadioButton.new_with_label_from_widget(None, "Images")
32         image_button.connect("toggled", self.add_image_targets)
33         button_box.pack_start(image_button, True, False, 0)
34
35         text_button = Gtk.RadioButton.new_with_label_from_widget(image_button, "Text")
36         text_button.connect("toggled", self.add_text_targets)
37         button_box.pack_start(text_button, True, False, 0)
38
39         self.add_image_targets()
40
41     def add_image_targets(self, button=None):
42         targets = Gtk.TargetList.new([])
43         targets.add_image_targets(TARGET_ENTRY_PIXBUF, True)
44
45         self.drop_area.drag_dest_set_target_list(targets)
46         self.iconview.drag_source_set_target_list(targets)
47
48     def add_text_targets(self, button=None):
49         self.drop_area.drag_dest_set_target_list(None)
50         self.iconview.drag_source_set_target_list(None)
51
52         self.drop_area.drag_dest_add_text_targets()
53         self.iconview.drag_source_add_text_targets()
54
55
56 class DragSourceIconView(Gtk.IconView):
57     def __init__(self):
58         Gtk.IconView.__init__(self)
59         self.set_text_column(COLUMN_TEXT)
```

(continua na próxima página)

(continuação da página anterior)

```
        self.set_pixbuf_column(COLUMN_PIXBUF)

        model = Gtk.ListStore(str, GdkPixbuf.Pixbuf)
        self.set_model(model)
        self.add_item("Item 1", "image-missing")
        self.add_item("Item 2", "help-about")
        self.add_item("Item 3", "edit-copy")

        self.enable_model_drag_source(Gdk.ModifierType.BUTTON1_MASK, [], DRAG_ACTION)
        self.connect("drag-data-get", self.on_drag_data_get)

    def on_drag_data_get(self, widget, drag_context, data, info, time):
        selected_path = self.get_selected_items()[0]
        selected_iter = self.get_model().get_iter(selected_path)

        if info == TARGET_ENTRY_TEXT:
            text = self.get_model().get_value(selected_iter, COLUMN_TEXT)
            data.set_text(text, -1)
        elif info == TARGET_ENTRY_PIXBUF:
            pixbuf = self.get_model().get_value(selected_iter, COLUMN_PIXBUF)
            data.set_pixbuf(pixbuf)

    def add_item(self, text, icon_name):
        pixbuf = Gtk.IconTheme.get_default().load_icon(icon_name, 16, 0)
        self.get_model().append([text, pixbuf])


class DropArea(Gtk.Label):
    def __init__(self):
        Gtk.Label.__init__(self)
        self.set_label("Drop something on me!")
        self.drag_dest_set(Gtk.DestDefaults.ALL, [], DRAG_ACTION)

        self.connect("drag-data-received", self.on_drag_data_received)

    def on_drag_data_received(self, widget, drag_context, x, y, data, info, time):
        if info == TARGET_ENTRY_TEXT:
            text = data.get_text()
            print("Received text: %s" % text)

        elif info == TARGET_ENTRY_PIXBUF:
            pixbuf = data.get_pixbuf()
            width = pixbuf.get_width()
            height = pixbuf.get_height()

            print("Received pixbuf with width %spx and height %spx" % (width, height))


win = DragDropWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 22

Glade e Gtk.Builder

A classe `Gtk.Builder` oferece a você a oportunidade de projetar interfaces de usuário sem escrever uma única linha de código. Isso é alcançado definindo a interface em um arquivo XML e, em seguida, carregando aquela definição XML de UI em tempo de execução usando a classe Builder que cria os objetos automaticamente. Para evitar a escrita do XML manualmente, use o aplicativo Glade, o qual permite criar a interface do usuário de uma maneira WYSIWYG (o que você vê é o que obtém)

Esse método possui várias vantagens:

- Menos código precisa ser escrito.
- As mudanças da interface do usuário podem ser vistas mais rapidamente, para que as interfaces de usuário possam melhorar.
- Designers sem habilidades de programação podem criar e editar interfaces de usuário.
- A descrição da interface do usuário é independente da linguagem de programação utilizada.

Ainda existe código necessário para lidar com mudanças de interface acionadas pelo usuário, mas `Gtk.Builder` permite que você se concentre em implementar essa funcionalidade.

22.1 Criando e carregando o arquivo .glade

Primeiro de tudo você tem que baixar e instalar o Glade. Existem vários tutoriais sobre o Glade, então isso não é explicado aqui em detalhes. Vamos começar criando uma janela com um botão e salvando-a em um arquivo chamado *example.glade*. O arquivo XML resultante deve se parecer com isso.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <!-- interface-requires gtk+ 3.0 -->
  <object class="GtkWindow" id="window1">
    <property name="can_focus">False</property>
    <child>
      <object class="GtkButton" id="button1">
```

(continua na próxima página)

(continuação da página anterior)

```
        <property name="label" translatable="yes">button</property>
        <property name="use-action-appearance">False</property>
        <property name="visible">True</property>
        <property name="can-focus">True</property>
        <property name="receives-default">True</property>
      </object>
    </child>
  </object>
</interface>
```

Para carregar este arquivo em Python, precisamos de um objeto `Gtk.Builder`.

```
builder = Gtk.Builder()
builder.add_from_file("example.glade")
```

A segunda linha carrega todos os objetos definidos em *example.glade* no objeto Builder.

Também é possível carregar apenas alguns dos objetos. A linha a seguir adicionaria apenas os objetos (e seus objetos filhos) fornecidos na tupla.

```
# we don't really have two buttons here, this is just an example
builder.add_objects_from_file("example.glade", ("button1", "button2"))
```

Esses dois métodos também existem para o carregamento de uma string, em vez de um arquivo. Seus nomes correspondentes são `Gtk.Builder.add_from_string()` e `Gtk.Builder.add_objects_from_string()` e eles simplesmente pegam uma string XML em vez de um nome de arquivo.

22.2 Acessando widgets

Agora que a janela e o botão estão carregados, também queremos mostrá-los. Portanto, o método `Gtk.Window.show_all()` deve ser chamado na janela. Mas como acessamos o objeto associado?

```
window = builder.get_object("window1")
window.show_all()
```

Cada widget pode ser recuperado do construtor pelo método `Gtk.Builder.get_object()` e pelo *id* do widget. É realmente *isso* simples.

Também é possível obter uma lista de todos os objetos com

```
builder.get_objects()
```

22.3 Conectando sinais

O Glade também permite definir sinais que você pode conectar a manipuladores em seu código sem extrair todos os objetos do construtor e conectar-se aos sinais manualmente. A primeira coisa a fazer é declarar os nomes dos sinais no Glade. Para este exemplo, vamos agir quando a janela é fechada e quando o botão foi pressionado, então damos o nome “onDestroy” para o retorno de chamada manipulando o sinal “destroy” da janela e “onButtonPressed” para o retorno de chamada manipulando o sinal “pressed” do botão. Agora o arquivo XML deve ficar assim.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <!-- interface-requires gtk+ 3.0 -->
  <object class="GtkWindow" id="window1">
    <property name="can-focus">False</property>
    <signal name="destroy" handler="onDestroy" swapped="no"/>
    <child>
      <object class="GtkButton" id="button1">
        <property name="label" translatable="yes">button</property>
        <property name="use-action-appearance">False</property>
        <property name="visible">True</property>
        <property name="can-focus">True</property>
        <property name="receives-default">True</property>
        <property name="use-action-appearance">False</property>
        <signal name="pressed" handler="onButtonPressed" swapped="no"/>
      </object>
    </child>
  </object>
</interface>
```

Agora temos que definir as funções do manipulador em nosso código. O *onDestroy* deve simplesmente resultar em uma chamada para `Gtk.main_quit()`. Quando o botão é pressionado, gostaríamos de imprimir a string "Hello World!", Então definimos o manipulador da seguinte maneira

```
def hello(button):
    print("Hello World!")
```

Em seguida, temos que conectar os sinais e as funções do manipulador. A maneira mais fácil de fazer isso é definir um *dict* com um mapeamento dos nomes para os manipuladores e então passá-lo para o método `Gtk.Builder.connect_signals()`.

```
handlers = {
    "onDestroy": Gtk.main_quit,
    "onButtonPressed": hello
}
builder.connect_signals(handlers)
```

Uma abordagem alternativa é criar uma classe que tenha métodos que sejam chamados como os sinais. Em nosso exemplo, o último snippet de código pode ser reescrito como:

```
1  from gi.repository import Gtk
2
3
4  class Handler:
5      def onDestroy(self, *args):
6          Gtk.main_quit()
7
8      def onButtonPressed(self, button):
9          print("Hello World!")
10 builder.connect_signals(Handler())
```

22.4 Exemplo

O código final do exemplo

```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class Handler:
    def onDestroy(self, *args):
        Gtk.main_quit()

    def onButtonPressed(self, button):
        print("Hello World!")


builder = Gtk.Builder()
builder.add_from_file("builder_example.glade")
builder.connect_signals(Handler())

window = builder.get_object("window1")
window.show_all()

Gtk.main()
```

22.5 Gtk.Template

Gtk.WidgetClass allows UI definition files to be used to extend a widget, PyGObject provides Gtk.Template as a way of accessing this from Python.

O arquivo de definição de UI usado no exemplo precisa de uma pequena alteração para incluir um elemento *<template>*:

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <!-- interface-requires gtk+ 3.0 -->
  <template class="window1" parent="GtkWindow">
    <signal name="destroy" handler="onDestroy" swapped="no"/>
    <child>
      <object class="GtkButton" id="button1">
        <property name="label" translatable="yes">button</property>
        <property name="use-action-appearance">False</property>
        <property name="visible">True</property>
        <property name="can-focus">True</property>
        <property name="receives-default">True</property>
        <property name="use-action-appearance">False</property>
        <signal name="pressed" handler="onButtonPressed" swapped="no"/>
      </object>
    </child>
  </template>
</interface>
```

Então ele pode ser usado para implementar o exemplo com uma subclasse `Gtk.Window`:

```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


@Gtk.Template(filename="template_example.ui")
class Window1(Gtk.Window):
    __gtype_name__ = "window1"

    @Gtk.Template.Callback()
    def onDestroy(self, *args):
        Gtk.main_quit()

    @Gtk.Template.Callback()
    def onButtonPressed(self, button):
        print("Hello World!")


window = Window1()
window.show()

Gtk.main()
```

More information can be found at the PyGObject website.

CAPÍTULO 23

Objetos

O GObject é o tipo fundamental que fornece os atributos e métodos comuns para todos os tipos de objeto no GTK+, no Pango e em outras bibliotecas baseadas no GObject. A classe *GObject.GObject* fornece métodos para construção e destruição de objetos, métodos de acesso a propriedades e suporte a sinais.

Esta seção apresentará alguns aspectos importantes sobre a implementação do GObject no Python.

23.1 Herdar de GObject.GObject

Um GObject nativo é acessível via *GObject.GObject*. É raramente instanciado diretamente, geralmente usamos classes herdadas. A `Gtk.Widget` é uma classe herdada de um *GObject.GObject*. Pode ser interessante criar uma classe herdada para criar um novo widget, como uma caixa de diálogo de configurações.

Para herdar de *GObject.GObject*, você deve chamar `GObject.GObject.__init__()` em seu construtor (se a classe herdar de `Gtk.Button`, deve chamar `Gtk.Button.__init__()` por exemplo), como no exemplo abaixo:

```
from gi.repository import GObject

class MyObject(GObject.GObject):

    def __init__(self):
        GObject.GObject.__init__(self)
```

23.2 Sinais

Os sinais conectam eventos específicos de aplicativos arbitrários com qualquer número de ouvintes. Por exemplo, no GTK+, cada evento de usuário (pressionamento de tecla ou mouse) é recebido do servidor X e gera um evento GTK+ sob a forma de uma emissão de sinal em uma determinada instância de objeto.

Cada sinal é registrado no sistema de tipos junto com o tipo no qual ele pode ser emitido: os usuários do tipo são conectados ao sinal em uma determinada instância de tipo quando registram uma função a ser invocada na emissão do sinal. Os usuários também podem emitir o sinal sozinhos ou interromper a emissão do sinal de dentro de uma das funções conectadas ao sinal.

23.2.1 Receba sinais

Veja *Loop principal e sinais*

23.2.2 Crie novos sinais

Novos sinais podem ser criados adicionando-os a *GObject.GObject.__gsignals__*, um dicionário:

Quando um novo sinal é criado, um manipulador de método também pode ser definido, ele será chamado toda vez que o sinal for emitido. É chamado do_nome_sinal.

```
class MyObject(GObject.GObject):
    __gsignals__ = {
        'my_signal': (GObject.SIGNAL_RUN_FIRST, None,
                      (int,))
    }

    def do_my_signal(self, arg):
        print("method handler for `my_signal' called with argument", arg)
```

GObject.SIGNAL_RUN_FIRST indica que este sinal invocará o manipulador do método de objeto (`do_my_signal()` aqui) no primeiro estágio de emissão. As alternativas são *GObject.SIGNAL_RUN_LAST* (o manipulador de método será invocado no terceiro estágio de emissão) e *GObject.SIGNAL_RUN_CLEANUP* (invoca o manipulador de método no último estágio de emissão).

A segunda parte, `None`, indica o tipo de retorno do sinal, geralmente `None`.

`(int,)` indica os argumentos do sinal, aqui, o sinal só receberá um argumento, cujo tipo é int. Os tipos de argumentos exigidos pelo sinal são declarados como uma sequência, aqui é uma tupla de um elemento.

Os sinais podem ser emitidos usando *GObject.GObject.emit()*:

```
my_obj.emit("my_signal", 42) # emit the signal "my_signal", with the
                             # argument 42
```

23.3 Propriedades

Um dos ótimos recursos do GObject é seu mecanismo get/set genérico para propriedades de objetos. Cada classe herdada de *GObject.GObject* pode definir novas propriedades. Cada propriedade tem um tipo que nunca muda (por exemplo, str, float, int ...). Por exemplo, eles são usados para Gtk.Button onde existe uma propriedade "label" que contém o texto do botão.

23.3.1 Use propriedades existentes

A classe *GObject.GObject* fornece várias funções úteis para gerenciar propriedades existentes, *GObject.GObject.get_property()* e *GObject.GObject.set_property()*.

Algumas propriedades também possuem funções dedicadas a elas, chamadas de getter e setter. Para a propriedade "label" de um botão, existem duas funções para obter e configurá-las, `Gtk.Button.get_label()` e `Gtk.Button.set_label()`.

23.3.2 Crie novas propriedades

Uma propriedade é definida com um nome e um tipo. Mesmo se o próprio Python for digitado dinamicamente, você não poderá alterar o tipo de uma propriedade depois que ela for definida. Uma propriedade pode ser criada usando `GObject.Property`.

```
from gi.repository import GObject

class MyObject(GObject.GObject):

    foo = GObject.Property(type=str, default='bar')
    property_float = GObject.Property(type=float)
    def __init__(self):
        GObject.GObject.__init__(self)
```

As propriedades também podem ser somente leitura, se você quiser que algumas propriedades sejam legíveis, mas não graváveis. Para fazer isso, você pode adicionar alguns sinalizadores à definição da propriedade, para controlar o acesso de leitura/gravação. Sinalizadores são *GObject.ParamFlags.READABLE* (somente acesso de leitura para código externo), *GObject.ParamFlags.WRITABLE* (somente acesso de gravação), *GObject.ParamFlags.READWRITE* (publico):

```
foo = GObject.Property(type=str, flags = GObject.ParamFlags.READABLE) # not writable
bar = GObject.Property(type=str, flags = GObject.ParamFlags.WRITABLE) # not readable
```

Você também pode definir novas propriedades somente leitura com um novo método decorado com `GObject.Property`:

```
from gi.repository import GObject

class MyObject(GObject.GObject):

    def __init__(self):
        GObject.GObject.__init__(self)

    @GObject.Property
```

(continua na próxima página)

(continuação da página anterior)

```
    def readonly(self):
        return 'This is read-only.'
```

Você pode obter essa propriedade usando:

```
my_object = MyObject()
print(my_object.readonly)
print(my_object.get_property("readonly"))
```

A API de `GObject.Property` é semelhante ao construído em `property()`. Você pode criar o setter de propriedades de maneira semelhante à propriedade Python:

```
class AnotherObject(GObject.Object):
    value = 0

    @GObject.Property
    def prop(self):
        'Read only property.'
        return 1

    @GObject.Property(type=int)
    def propInt(self):
        'Read-write integer property.'
        return self.value

    @propInt.setter
    def propInt(self, value):
        self.value = value
```

Há também uma maneira de definir valores mínimos e máximos para números, usando um formulário mais detalhado:

```
from gi.repository import GObject

class MyObject(GObject.GObject):

    __gproperties__ = {
        "int-prop": (int, # type
                     "integer prop", # nick
                     "A property that contains an integer", # blurb
                     1, # min
                     5, # max
                     2, # default
                     GObject.ParamFlags.READWRITE # flags
                    ),
    }

    def __init__(self):
        GObject.GObject.__init__(self)
        self.int_prop = 2

    def do_get_property(self, prop):
        if prop.name == 'int-prop':
            return self.int_prop
```

(continua na próxima página)

(continuação da página anterior)

```
        else:
            raise AttributeError('unknown property %s' % prop.name)

    def do_set_property(self, prop, value):
        if prop.name == 'int-prop':
            self.int_prop = value
        else:
            raise AttributeError('unknown property %s' % prop.name)
```

As propriedades devem ser definidas em *GObject.GObject.__gproperties__*, um dicionário e manipulado em do_get_property e do_set_property.

23.3.3 Veja as propriedades

Quando uma propriedade é modificada, um sinal é emitido, cujo nome é "notify::property-name":

```
my_object = MyObject()

def on_notify_foo(obj, gparamstring):
    print("foo changed")

my_object.connect("notify::foo", on_notify_foo)

my_object.set_property("foo", "bar") # on_notify_foo will be called
```

Note que você tem que usar o nome da propriedade canônica ao se conectar aos sinais de notificação, como explicado em `GObject.Object.signals.notify()`. Por exemplo, para uma propriedade Python *foo_bar_baz* você conectaria ao sinal *notify::foo-bar-baz* usando

```
my_object = MyObject()

def on_notify_foo_bar_baz(obj, gparamstring):
    print("foo_bar_baz changed")

my_object.connect("notify::foo-bar-baz", on_notify_foo_bar_baz)
```

23.4 API

class GObject.**GObject**

get_property(*property_name*)

Recupera um valor de propriedade.

set_property(*property_name*, *value*)

Configura a propriedade *property_name* para *valor*.

emit(*signal_name*, ...)

Emite sinal *signal_name*. Argumentos de sinal devem seguir, p. ex., se o seu sinal é do tipo `(int,)`, deve ser emitido com:

```
self.emit(signal_name, 42)
```

freeze_notify()

Este método congela todos os sinais "notify::" (que são emitidos quando qualquer propriedade é alterada) até que o método *thaw_notify()* seja chamado.

Recomenda-se usar a instrução *with* ao chamar *freeze_notify()*, dessa forma é assegurado que *thaw_notify()* é chamado implicitamente no final do bloco:

```
with an_object.freeze_notify():
    # Do your work here
    ...
```

thaw_notify()

Descongela todos os sinais "notify::" que foram congelados por *freeze_notify()*.

Recomenda-se não chamar *thaw_notify()* explicitamente mas use *freeze_notify()* juntamente com a instrução *with*.

handler_block(*handler_id*)

Bloqueia um manipulador de uma instância para que ele não seja chamado durante qualquer emissão de sinal, a menos que *handler_unblock()* seja chamado para aquele *handler_id*. Assim, "bloquear" um manipulador de sinal significa desativá-lo temporariamente, um manipulador de sinal precisa ser desbloqueado exatamente na mesma quantidade de vezes que foi bloqueado antes de se tornar ativo novamente.

Recomenda-se usar *handler_block()* em conjunto com a instrução *with* que irá chamar *handler_unblock()* implicitamente no final do bloco:

```
with an_object.handler_block(handler_id):
    # Do your work here
    ...
```

handler_unblock(*handler_id*)

Desfaz o efeito de *handler_block()*. Um manipulador bloqueado é ignorado durante as emissões do sinal e não será chamado até que tenha sido desbloqueado exatamente a quantidade de vezes que foi bloqueado antes.

É recomendado não chamar explicitamente *handler_unblock()* mas use *handler_block()* junto com a instrução *with*.

__gsignals__

Um dicionário onde a classe herdada pode definir novos sinais.

Cada elemento no dicionário é um novo sinal. A chave é o nome do sinal. O valor é uma tupla, com o formato:

```
(GObject.SIGNAL_RUN_FIRST, None, (int,))
```

GObject.SIGNAL_RUN_FIRST pode ser substituído por *GObject.SIGNAL_RUN_LAST* ou *GObject.SIGNAL_RUN_CLEANUP*. `None` é o tipo de retorno do sinal. `(int,)` é a tupla dos parâmetros do sinal.

__gproperties__

O dicionário *__gproperties__* é uma propriedade de classe onde você define as propriedades do seu objeto. Esta não é a maneira recomendada de definir novas propriedades, o método escrito acima é muito menos detalhado. Os benefícios desse método é que uma propriedade pode ser definida com mais configurações, como o mínimo ou o máximo para números.

A chave é o nome da propriedade

O valor é uma tupla que descreve a propriedade. O número de elementos dessa tupla depende de seu primeiro elemento, mas a tupla sempre conterá pelo menos os seguintes itens:

O primeiro elemento é o tipo da propriedade (por exemplo, `int`, `float`...).

O segundo elemento é o apelido da propriedade, que é uma string com uma breve descrição da propriedade. Isso geralmente é usado por programas com fortes recursos de introspecção, como o construtor de interface gráfica de usuário Glade.

A terceira é a descrição da propriedade ou sinopse, que é outra string com uma descrição mais longa da propriedade. Também usado pelo Glade e programas similares.

O último (que não é necessariamente o último, como veremos mais adiante) é o sinalizador da propriedade `GObject.PARAM_READABLE`, `GObject.PARAM_WRITABLE`, `GObject.PARAM_READWRITE`.

O comprimento absoluto da tupla depende do tipo de propriedade (o primeiro elemento da tupla). Assim, temos as seguintes situações:

Se o tipo for `bool` ou `str`, o quarto elemento é o valor padrão da propriedade.

Se o tipo for `int` ou `float`, o quarto elemento é o valor mínimo aceito, o quinto elemento é o valor máximo aceito e o sexto elemento é o valor padrão.

Se o tipo não for um desses, não há elemento extra.

`GObject.`**`SIGNAL_RUN_FIRST`**

Invoca o manipulador de método de objeto no primeiro estágio de emissão.

`GObject.`**`SIGNAL_RUN_LAST`**

Invoca o manipulador de método de objeto no terceiro estágio de emissão.

`GObject.`**`SIGNAL_RUN_CLEANUP`**

Invoca o manipulador do método de objeto no último estágio de emissão.

`GObject.ParamFlags.`**`READABLE`**

A propriedade é legível.

`GObject.ParamFlags.`**`WRITABLE`**

A propriedade é gravável.

`GObject.ParamFlags.`**`READWRITE`**

A propriedade é legível e gravável.

CAPÍTULO 24

Application

`Gtk.Application` abrange muitas tarefas repetitivas que um aplicativo moderno precisa, como manipular várias instâncias, ativação do D-Bus, abertura de arquivos, análise de linha de comando, inicialização/desligamento, gerenciamento de menus, gerenciamento de janelas e muito mais.

24.1 Ações

`Gio.Action` é uma maneira de expor qualquer tarefa que seu aplicativo ou widget fizer por um nome. Essas ações podem ser desabilitadas/habilitadas no tempo de execução e podem ser ativadas ou ter um estado alterado (se elas contiverem estado).

O motivo para usar ações é separar a lógica da interface do usuário. Por exemplo, isso permite usar uma barra de menu no OSX e um menu de engrenagem no GNOME, simplesmente referenciando o nome de uma ação. A principal implementação que você estará usando é `Gio.SimpleAction` que será mostrado mais tarde.

Muitas classes, como `Gio.MenuItem` e `Gtk.ModelButton` suportam propriedades para definir um nome de ação.

Estas ações podem ser agrupadas em um `Gio.ActionGroup` e quando esses grupos são adicionados a um widget com `Gtk.Widget.insert_action_group()`, eles ganharão um prefixo. Tal como "win" quando adicionado a um `Gtk.ApplicationWindow`. Você usará o nome completo da ação ao fazer referência a ele, como "app.about", mas ao criar a ação, ela ficará "about" até ser adicionada ao aplicativo.

Você também pode facilmente criar keybindings para ações definindo a propriedade *accel* no arquivo `Gio.Menu` ou usando `Gtk.Application.set_accels_for_action()`.

24.2 Menus

Seus menus devem ser definidos em XML usando `Gio.Menu` e referenciam as ações mencionadas anteriormente que você definiu. `Gtk.Application` permite que você defina um menu via `Gtk.Application.set_app_menu()` ou `Gtk.Application.set_menubar()`. Se você faz uso de `Gio.Resource` isto pode usar automaticamente o menu correto baseado na plataforma, caso contrário você pode configurá-los manualmente. Um exemplo detalhado é mostrado abaixo.

24.3 Linha de comando

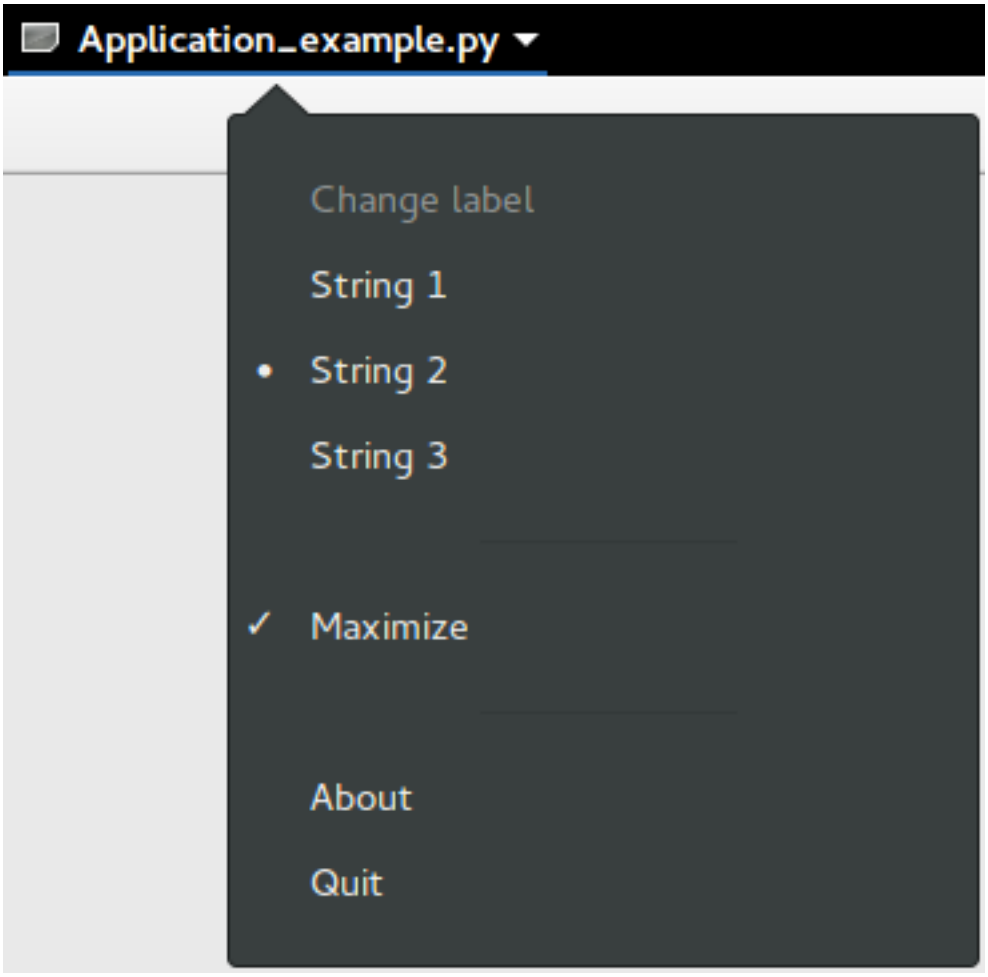
Ao criar seu aplicativo, ele recebe uma propriedade de flag de `Gio.ApplicationFlags`. Usando isso, você pode permitir que ele manipule tudo sozinho ou tenha um comportamento mais personalizado.

Você pode usar o *HANDLES_COMMAND_LINE* para permitir um comportamento customizado em `Gio.Application.do_command_line()`. Em combinação com `Gio.Application.add_main_option()` para adicionar opções personalizadas.

Usar *HANDLES_OPEN* fará o trabalho de simplesmente pegar argumentos de arquivo para você e permitir que você os manipule em `Gio.Application.do_open()`.

Se o seu aplicativo já estiver aberto, todos serão enviados para a instância existente, a menos que você use *NON_UNIQUE* para permitir várias instâncias.

24.4 Exemplo




```
import sys

import gi

gi.require_version("Gtk", "3.0")
from gi.repository import GLib, Gio, Gtk

# This would typically be its own file
MENU_XML = """
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <menu id="app-menu">
    <section>
      <attribute name="label" translatable="yes">Change label</attribute>
      <item>
        <attribute name="action">win.change_label</attribute>
        <attribute name="target">String 1</attribute>
        <attribute name="label" translatable="yes">String 1</attribute>
      </item>
```

(continua na próxima página)

(continuação da página anterior)

```
      <item>
        <attribute name="action">win.change_label</attribute>
        <attribute name="target">String 2</attribute>
        <attribute name="label" translatable="yes">String 2</attribute>
      </item>
      <item>
        <attribute name="action">win.change_label</attribute>
        <attribute name="target">String 3</attribute>
        <attribute name="label" translatable="yes">String 3</attribute>
      </item>
    </section>
    <section>
      <item>
        <attribute name="action">win.maximize</attribute>
        <attribute name="label" translatable="yes">Maximize</attribute>
      </item>
    </section>
    <section>
      <item>
        <attribute name="action">app.about</attribute>
        <attribute name="label" translatable="yes">_About</attribute>
      </item>
      <item>
        <attribute name="action">app.quit</attribute>
        <attribute name="label" translatable="yes">_Quit</attribute>
        <attribute name="accel">&lt;Primary&gt;q</attribute>
    </item>
    </section>
  </menu>
</interface>
"""


class AppWindow(Gtk.ApplicationWindow):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        # This will be in the windows group and have the "win" prefix
        max_action = Gio.SimpleAction.new_stateful(
            "maximize", None, GLib.Variant.new_boolean(False)
        )
        max_action.connect("change-state", self.on_maximize_toggle)
        self.add_action(max_action)

        # Keep it in sync with the actual state
        self.connect(
            "notify::is-maximized",
            lambda obj, pspec: max_action.set_state(
                GLib.Variant.new_boolean(obj.props.is_maximized)
            ),
        )

```

(continua na próxima página)

(continuação da página anterior)

```
        lbl_variant = GLib.Variant.new_string("String 1")
        lbl_action = Gio.SimpleAction.new_stateful(
            "change_label", lbl_variant.get_type(), lbl_variant
        )
        lbl_action.connect("change-state", self.on_change_label_state)
        self.add_action(lbl_action)

        self.label = Gtk.Label(label=lbl_variant.get_string(), margin=30)
        self.add(self.label)
        self.label.show()

    def on_change_label_state(self, action, value):
        action.set_state(value)
        self.label.set_text(value.get_string())

    def on_maximize_toggle(self, action, value):
        action.set_state(value)
        if value.get_boolean():
            self.maximize()
        else:
            self.unmaximize()


class Application(Gtk.Application):
    def __init__(self, *args, **kwargs):
        super().__init__(
            *args,
            application_id="org.example.myapp",
            flags=Gio.ApplicationFlags.HANDLES_COMMAND_LINE,
            **kwargs
        )
        self.window = None

        self.add_main_option(
            "test",
            ord("t"),
            GLib.OptionFlags.NONE,
            GLib.OptionArg.NONE,
            "Command line test",
            None,
        )

    def do_startup(self):
        Gtk.Application.do_startup(self)

        action = Gio.SimpleAction.new("about", None)
        action.connect("activate", self.on_about)
        self.add_action(action)

        action = Gio.SimpleAction.new("quit", None)
        action.connect("activate", self.on_quit)
        self.add_action(action)
```

(continua na próxima página)

(continuação da página anterior)

```
        builder = Gtk.Builder.new_from_string(MENU_XML, -1)
        self.set_app_menu(builder.get_object("app-menu"))

    def do_activate(self):
        # We only allow a single window and raise any existing ones
        if not self.window:
            # Windows are associated with the application
            # when the last one is closed the application shuts down
            self.window = AppWindow(application=self, title="Main Window")

        self.window.present()

    def do_command_line(self, command_line):
        options = command_line.get_options_dict()
        # convert GVariantDict -> GVariant -> dict
        options = options.end().unpack()

        if "test" in options:
            # This is printed on the main instance
            print("Test argument recieved: %s" % options["test"])

        self.activate()
        return 0

    def on_about(self, action, param):
        about_dialog = Gtk.AboutDialog(transient_for=self.window, modal=True)
        about_dialog.present()

    def on_quit(self, action, param):
        self.quit()


if __name__ == "__main__":
    app = Application()
    app.run(sys.argv)
```

24.5 Veja também

- https://wiki.gnome.org/HowDoI/GtkApplication
- https://wiki.gnome.org/HowDoI/GAction
- https://wiki.gnome.org/HowDoI/ApplicationMenu
- https://wiki.gnome.org/HowDoI/GMenu

CAPÍTULO 25

Menus

Nota: `Gtk.UIManager`, `Gtk.Action` e `Gtk.ActionGroup` foram descontinuados desde o GTK+ versão 3.10 e não devem ser usados em código recém-escrito. Use o framework *Application*.

O GTK+ vem com dois tipos diferentes de menus `Gtk.MenuBar` e `Gtk.Toolbar`. `Gtk.MenuBar` é uma barra de menus padrão que contém uma ou mais instâncias `Gtk.MenuItem` ou uma de suas subclasses. Os widgets `Gtk.Toolbar` são usados para acessibilidade rápida às funções comumente usadas de um aplicativo. Exemplos incluem criar um novo documento, imprimir uma página ou desfazer uma operação. Ele contém uma ou mais instâncias de `Gtk.ToolItem` ou uma de suas subclasses.

25.1 Ações

Embora existam APIs específicas para criar menus e barras de ferramentas, você deve usar `Gtk.UIManager` e criar instâncias `Gtk.Action`. As ações são organizadas em grupos. A `Gtk.ActionGroup` é essencialmente um mapa de nomes para objetos `Gtk.Action`. Todas as ações que fazem sentido usar em um contexto particular devem estar em um único grupo. Vários grupos de ação podem ser usados para uma interface de usuário específica. Na verdade, espera-se que a maioria dos aplicativos não triviais faça uso de vários grupos. Por exemplo, em um aplicativo que pode editar vários documentos, um grupo mantém ações globais (por exemplo, sair, sobre, novo) e um grupo por documento que contém ações que atuam nesse documento (por exemplo, salvar, recortar/copiar/colar etc.). Os menus de cada janela seriam construídos a partir de uma combinação de dois grupos de ação.

Existem classes diferentes representando diferentes tipos de ações:

- `Gtk.Action`: Uma ação que pode ser acionada por um item de menu ou barra de ferramentas
- `Gtk.ToggleAction`: Uma ação que pode ser alternada entre dois estados
- `Gtk.RadioAction`: Uma ação da qual apenas um em um grupo pode estar ativo
- `Gtk.RecentAction`: Uma ação que representa uma lista de arquivos usados recentemente

Ações representam operações que o usuário pode executar, juntamente com algumas informações sobre como ele deve ser apresentado na interface, incluindo seu nome (não para exibição), seu rótulo (para exibição), um acelerador, se um rótulo também indica uma dica de ferramenta como o retorno que é chamado quando a ação é ativada.

Você pode criar ações chamando um dos construtores diretamente e adicionando-os a um `Gtk.ActionGroup` chamando `Gtk.ActionGroup.add_action()` ou `Gtk.ActionGroup.add_action_with_accel()`, ou chamando uma das funções de conveniência:

- `Gtk.ActionGroup.add_actions()`,
- `Gtk.ActionGroup.add_toggle_actions()`
- `Gtk.ActionGroup.add_radio_actions()`.

Observe que você deve especificar ações para submenus e itens de menu.

25.2 Gerenciador de interface de usuário

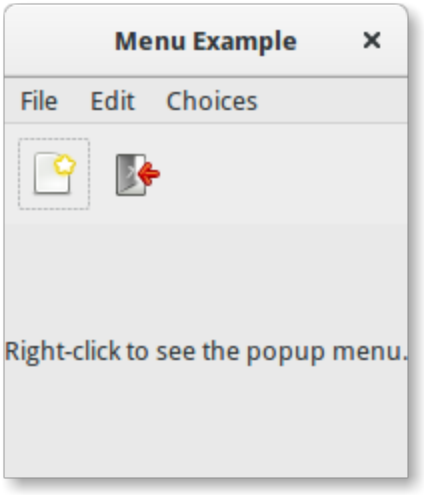
`Gtk.UIManager` provides an easy way of creating menus and toolbars using an XML-like description.

Primeiro de tudo, você deve adicionar o `Gtk.ActionGroup` ao UI Manager com `Gtk.UIManager.insert_action_group()`. Neste ponto também é uma boa ideia dizer à janela pai para responder aos atalhos de teclado especificados, usando `Gtk.UIManager.get_accel_group()` e `Gtk.Window.add_accel_group()`.

Em seguida, você pode definir o layout visível real dos menus e barras de ferramentas e adicionar o layout da interface do usuário. Essa "string de ui" usa um formato XML, no qual você deve mencionar os nomes das ações que você já criou. Lembre-se de que esses nomes são apenas os identificadores que usamos ao criar as ações. Eles não são o texto que o usuário verá nos menus e nas barras de ferramentas. Fornecemos esses nomes legíveis quando criamos as ações.

Finalmente, você obtém o widget raiz com `Gtk.UIManager.get_widget()` e adiciona o widget a um contêiner como `Gtk.Box`.

25.3 Exemplo




```
1 import gi
2
3 gi.require_version("Gtk", "3.0")
```

(continua na próxima página)

(continuação da página anterior)

```
from gi.repository import Gtk, Gdk

UI_INFO = """
<ui>
  <menubar name='MenuBar'>
    <menu action='FileMenu'>
      <menu action='FileNew'>
        <menuitem action='FileNewStandard' />
        <menuitem action='FileNewFoo' />
        <menuitem action='FileNewGoo' />
      </menu>
      <separator />
      <menuitem action='FileQuit' />
    </menu>
    <menu action='EditMenu'>
      <menuitem action='EditCopy' />
      <menuitem action='EditPaste' />
      <menuitem action='EditSomething' />
    </menu>
    <menu action='ChoicesMenu'>
      <menuitem action='ChoiceOne'/>
      <menuitem action='ChoiceTwo'/>
      <separator />
      <menuitem action='ChoiceThree'/>
    </menu>
  </menubar>
  <toolbar name='ToolBar'>
    <toolitem action='FileNewStandard' />
    <toolitem action='FileQuit' />
  </toolbar>
  <popup name='PopupMenu'>
    <menuitem action='EditCopy' />
    <menuitem action='EditPaste' />
    <menuitem action='EditSomething' />
  </popup>
</ui>
"""


class MenuExampleWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Menu Example")

        self.set_default_size(200, 200)

        action_group = Gtk.ActionGroup(name="my_actions")

        self.add_file_menu_actions(action_group)
        self.add_edit_menu_actions(action_group)
        self.add_choices_menu_actions(action_group)

        uimanager = self.create_ui_manager()
```

(continua na próxima página)

(continuação da página anterior)

```
        uimanager.insert_action_group(action_group)

        menubar = uimanager.get_widget("/MenuBar")

        box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
        box.pack_start(menubar, False, False, 0)

        toolbar = uimanager.get_widget("/ToolBar")
        box.pack_start(toolbar, False, False, 0)

        eventbox = Gtk.EventBox()
        eventbox.connect("button-press-event", self.on_button_press_event)
        box.pack_start(eventbox, True, True, 0)

        label = Gtk.Label(label="Right-click to see the popup menu.")
        eventbox.add(label)

        self.popup = uimanager.get_widget("/PopupMenu")

        self.add(box)

    def add_file_menu_actions(self, action_group):
        action_filemenu = Gtk.Action(name="FileMenu", label="File")
        action_group.add_action(action_filemenu)

        action_filenewmenu = Gtk.Action(name="FileNew", stock_id=Gtk.STOCK_NEW)
        action_group.add_action(action_filenewmenu)

        action_new = Gtk.Action(
            name="FileNewStandard",
            label="_New",
            tooltip="Create a new file",
            stock_id=Gtk.STOCK_NEW,
        )
        action_new.connect("activate", self.on_menu_file_new_generic)
        action_group.add_action_with_accel(action_new, None)

        action_group.add_actions(
            [
                (
                    "FileNewFoo",
                    None,
                    "New Foo",
                    None,
                    "Create new foo",
                    self.on_menu_file_new_generic,
                ),
                (
                    "FileNewGoo",
                    None,
                    "_New Goo",
                    None,
```

(continua na próxima página)

(continuação da página anterior)

```
                    "Create new goo",
                    self.on_menu_file_new_generic,
                ),
            ]
        )

        action_filequit = Gtk.Action(name="FileQuit", stock_id=Gtk.STOCK_QUIT)
        action_filequit.connect("activate", self.on_menu_file_quit)
        action_group.add_action(action_filequit)

    def add_edit_menu_actions(self, action_group):
        action_group.add_actions(
            [
                ("EditMenu", None, "Edit"),
                ("EditCopy", Gtk.STOCK_COPY, None, None, None, self.on_menu_others),
                ("EditPaste", Gtk.STOCK_PASTE, None, None, None, self.on_menu_others),
                (
                    "EditSomething",
                    None,
                    "Something",
                    "<control><alt>S",
                    None,
                    self.on_menu_others,
                ),
            ]
        )

    def add_choices_menu_actions(self, action_group):
        action_group.add_action(Gtk.Action(name="ChoicesMenu", label="Choices"))

        action_group.add_radio_actions(
            [
                ("ChoiceOne", None, "One", None, None, 1),
                ("ChoiceTwo", None, "Two", None, None, 2),
            ],
            1,
            self.on_menu_choices_changed,
        )

        three = Gtk.ToggleAction(name="ChoiceThree", label="Three")
        three.connect("toggled", self.on_menu_choices_toggled)
        action_group.add_action(three)

    def create_ui_manager(self):
        uimanager = Gtk.UIManager()

        # Throws exception if something went wrong
        uimanager.add_ui_from_string(UI_INFO)

        # Add the accelerator group to the toplevel window
        accelgroup = uimanager.get_accel_group()
        self.add_accel_group(accelgroup)
```

(continua na próxima página)

(continuação da página anterior)

```
        return uimanager

    def on_menu_file_new_generic(self, widget):
        print("A File|New menu item was selected.")

    def on_menu_file_quit(self, widget):
        Gtk.main_quit()

    def on_menu_others(self, widget):
        print("Menu item " + widget.get_name() + " was selected")

    def on_menu_choices_changed(self, widget, current):
        print(current.get_name() + " was selected.")

    def on_menu_choices_toggled(self, widget):
        if widget.get_active():
            print(widget.get_name() + " activated")
        else:
            print(widget.get_name() + " deactivated")

    def on_button_press_event(self, widget, event):
        # Check if right mouse button was preseed
        if event.type == Gdk.EventType.BUTTON_PRESS and event.button == 3:
            self.popup.popup(None, None, None, None, event.button, event.time)
            return True  # event has been handled


window = MenuExampleWindow()
window.connect("destroy", Gtk.main_quit)
window.show_all()
Gtk.main()
```

CAPÍTULO 26

Tabela

Nota: `Gtk.Table` foi descontinuado desde o GTK+ versão 3.4 e não deve ser usado em código recém-escrito. Use a classe de *Grade*.

Tabelas nos permite colocar widgets em uma grade similar a `Gtk.Grid`.

As dimensões da grade precisam ser especificadas no construtor `Gtk.Table`. Para colocar um widget em uma caixa, use `Gtk.Table.attach()`.

`Gtk.Table.set_row_spacing()` e `Gtk.Table.set_col_spacing()` definem o espaçamento entre as linhas na linha ou coluna especificada. Observe que, para colunas, o espaço vai para a direita da coluna e, para linhas, o espaço fica abaixo da linha.

Você também pode definir um espaçamento consistente para todas as linhas e/ou colunas com `Gtk.Table.set_row_spacings()` e `Gtk.Table.set_col_spacings()`. Observe que, com essas chamadas, a última linha e a última coluna não recebem espaçamento.

Obsoleto desde a versão 3.4: É recomendado que você use o `Gtk.Grid` para o novo código.

26.1 Exemplo

```
import gi

gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


class TableWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Table Example")

        table = Gtk.Table(n_rows=3, n_columns=3, homogeneous=True)
        self.add(table)

        button1 = Gtk.Button(label="Button 1")
        button2 = Gtk.Button(label="Button 2")
        button3 = Gtk.Button(label="Button 3")
        button4 = Gtk.Button(label="Button 4")
        button5 = Gtk.Button(label="Button 5")
        button6 = Gtk.Button(label="Button 6")

        table.attach(button1, 0, 1, 0, 1)
        table.attach(button2, 1, 3, 0, 1)
        table.attach(button3, 0, 1, 1, 3)
        table.attach(button4, 1, 3, 1, 2)
        table.attach(button5, 1, 2, 2, 3)
        table.attach(button6, 2, 3, 2, 3)


win = TableWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

CAPÍTULO 27

Índices e tabelas

- search

Índice

Símbolos

E

F

G

H

R

S

T

W