# The Python GTK+ 3 Tutorial

*Utgåva 3.4*

**Sebastian Pölsterl**

**apr. 13, 2024**

# Innehåll

**Utgåva** 3.4

**Datum** apr. 13, 2024



Denna handledning ger en introduktion till att skriva GTK+ 3-program i Python.

Innan du arbetar dig genom denna handledning så är det rekommenderat att du har rimlig förståelse av programmeringsspråket Python. Programmering av grafiska användargränssnitt introducerar nya problem jämfört med att interagera med standardutmatning (konsol / terminal). Det är nödvändigt för dig att veta hur du skapar och kör Python-filer, förstår enkla tolkfel, hur du arbetar med strängar, heltal, flyttal och booleska värden. För de mer avancerade komponenterna i denna handledning kommer god kunskap om listor och tupler vara nödvändig.

Även om denna handledning beskriver de viktigaste klasserna och metoderna i GTK+ 3 så är den inte tänkt att tjäna som en API-referens. Se Referenshandboken för GTK+ 3 för en detaljerad beskrivning av API:t. Det finns även en Python-specifik referens tillgänglig.

Innehåll:

KAPITEL 1

# Installation

Det första steget innan vi börjar med faktisk kodning består i att konfigurera PyGObject och dess beroenden. PyGObject är en Python-modul som låter utvecklare komma åt GObject-baserade bibliotek så som GTK+ i Python. Dess stöd är begränsat till GTK+ version 3 eller senare.

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

## 1.1 Beroenden

- GTK+3
- Python 2 (2.6 eller senare) eller Python 3 (3.1 eller senare)
- gobject-introspection

## 1.2 Förbyggda paket

De senaste versionerna av PyGObject och dess beroenden paketeras av nästan alla stora Linux-distributioner. Så om du använder Linux kan du troligen komma igång genom att installera paketet från det officiella förrådet för din distribution.

## 1.3 Installera från källkod

Det lättaste sättet att installera PyGObject från källkod är genom att använda JHBuild. Det är designat för att lätt bygga källpaket och upptäcka vilka beroenden som måste byggas och i vilken ordning. För att konfigurera JHBuild, följ JHBuild-handboken.

När du har installerat JHBuild, hämta den senaste konfigurationen från[1]. Kopiera filer med filändelsen *.modules* to JHBuilds modulkatalog och filen *sample-tarball.jhbuildrc* till *~/.jhbuildrc*.

Om du inte har gjort det tidigare kan du bekräfta att din byggmiljö är konfigurerad riktigt genom att köra:

```
$ jhbuild sanitycheck
```

Det kommer skriva ut alla program och bibliotek som för närvarande saknas på ditt system men behövs för att bygga. Du bör installera dessa genom din distributions paketförråd. En lista över paketnamn för olika distributioner finns på GNOME-wikin. Kör kommandot ovan igen för att säkerställa att de nödvändiga verktygen är tillgängliga.

Att köra följande kommando kommer bygga PyGObject och alla dess beroenden:

```
$ jhbuild build pygobject
```

Slutligen kan du även vilja installera GTK+ från källkod:

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

För att starta ett skal med samma miljö som används av JHBuild, kör:

```
$ jhbuild shell
```

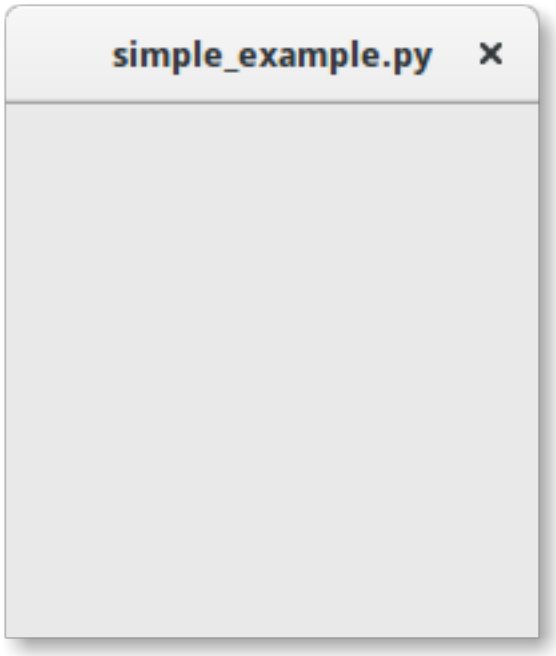
[1] https://download.gnome.org/teams/releng/

KAPITEL 2

# Komma igång

## 2.1 Enkelt exempel

För att starta med vår handledning skapar vi det enklast tänkbara programmet. Detta program kommer skapa ett tomt fönster som är 200 x 200 bildpunkter stort.




```
import gi

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

(continues on next page)

(fortsättning från föregående sida)

```
5
6  win = Gtk.Window()
7  win.connect("destroy", Gtk.main_quit)
8  win.show_all()
9  Gtk.main()
```

Vi kommer nu förklara varje rad i exemplet.

```
import gi

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

I början måste vi importera Gtk-modulen för att kunna komma åt GTK+:s klasser och funktioner. Då en användares system kan ha flera versioner av GTK+ installerade samtidigt så vill vi säkerställa att då vi importerar Gtk så refererar det till GTK+ 3 och inte någon annan version av biblioteket, vilket är syftet med satsen `gi.require_version('Gtk', '3.0')`.

Nästa rad skapar ett tomt fönster.

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

Följt av att ansluta till fönstrets borttagningshändelse för att säkerställa att programmet stängs om vi klickar på *x* för att stänga fönstret.

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

I nästa steg visar vi fönstret.

```
win.show_all()
```

Slutligen startar vi GTK+-bearbetningsslingan som vi avslutar när fönstret stängs (se rad 6).

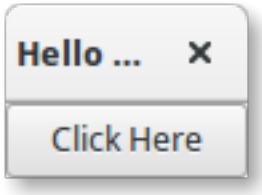
```
Gtk.main()
```

För att köra programmet, öppna en terminal, ändra till filens katalog, och mata in:

```
python simple_example.py
```

## 2.2 Utökat exempel

För något som är lite mer användbart kommer här PyGObject-versionen av det klassiska ”Hej världen”-programmet.




```
1  import gi
2
3  gi.require_version("Gtk", "3.0")
```

(continues on next page)

(fortsättning från föregående sida)

```
from gi.repository import Gtk


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

        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()
```

Detta exempel skiljer sig från det enkla exemplet då vi gör en underklass till `Gtk.Window` för att skapa vår egen klass `MyWindow`.

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

I klassens konstruktor måste vi anropa superklassens konstruktor. Dessutom säger vi till den att ställa in värdet på egenskapen *title* till *Hello World*.

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

De nästa tre raderna används för att skapa en knappkomponent, ansluta till dess *clicked*-signal och lägga till den som barn till toppnivåfönstret.

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

På det viset kommer metoden `on_button_clicked()` att anropas om du klickar på knappen.

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

Det sista blocket, utanför klassen, är väldigt likt det enkla exemplet ovan, men istället för att skapa en instans av den generiska klassen `Gtk.Window`, så skapar vi en instans av `MyWindow`.

KAPITEL 3

# Grunder

Detta avsnitt kommer introducera några av de viktigaste aspekterna i GTK+.

## 3.1 Huvudslinga och signaler

Som de flesta verktygslådor för grafiska användargränssnitt använder GTK+ en händelsedriven programmeringsmodell. Då användaren gör något sitter GTK+ i huvudslingan och väntar på inmatning. Om användaren utför någon åtgärd - exempelvis ett musklick - så ”vaknar” huvudslingan och levererar en händelse till GTK+.

När komponenter tar emot en händelse så sänder de ofta ut en eller flera signaler. Signaler meddelar ditt program att ”något intressant inträffade” genom att anropa funktioner som du anslutit till signalen. Sådana funktioner kallas vanligen *återanrop*. Då dina återanrop anropas skulle du vanligen utföra någon åtgärd - då exempelvis en Öppna-knapp klickas på kanske du visar en filväljardialog. Efter att ett återanrop slutförs kommer GTK+ återgå till huvudslingan och vänta på mer användarinmatning.

Ett generiskt exempel är:

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

Först så är *widget* en instans av en komponent vi skapade tidigare. Härnäst är händelsen ”event” som vi är intresserade av. Varje komponent har sina egna specifika händelser som kan inträffa. Om du exempelvis har en knapp så skulle du vanligen vilja ansluta den till händelsen ”clicked”. Det här betyder att när knappen klickas på så utfärdas signalen. Som trea kommer argumentet *callback* som är namnet på återanropsfunktionen. Den innehåller koden som körs när signaler av den angivna typen utfärdas. Slutligen inkluderar argumentet *data* de data som ska skickas när signalen utfärdas. Detta argument är dock fullständigt valfritt och kan utelämnas om det inte behövs.

Funktionen returnerar ett tal som identifierar detta specifika signal-återanropspar. Det krävs för att koppla bort från en signal så att återanropsfunktionen inte kommer anropas under framtida eller aktuellt pågående utsändningar av signalen den har anslutits till.

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

Om du förlorat ”handler_id” av någon anledning (till exempel för att hanterarna installerades med `Gtk.Builder.connect_signals()`), så kan du fortfarande koppla från ett specifikt återanrop med funktionen `disconnect_by_func()`:

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

Program bör ansluta till toppnivåfönstrets ”destroy”-signal. Den sänds ut när ett objekt förstörs, så när en användare begär att ett toppnivåfönster stängs så förstör standardhanteraren för denna signal fönstret, men avslutar inte programmet. Att ansluta ”destroy”-signalen för toppnivåfönstret till funktionen `Gtk.main_quit()` kommer att resultera i det önskade beteendet.

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

Att anropa `Gtk.main_quit()` får huvudslingan i `Gtk.main()` att returnera.

## 3.2 Egenskaper

Egenskaper beskriver konfigurationen och tillståndet för komponenter. Som för signaler så har varje komponent sin specifika uppsättning egenskaper. Exempelvis har en knapp egenskapen ”label” som innehåller texten för label-komponenten i knappen. Du kan ange namnet och värdet på valfritt antal egenskaper som nyckelordsargument då du skapar en instans av en komponent. För att skapa en högerjusterad etikett med texten ”Hello World” och en vinkel på 25 grader, använd:

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

vilket är ekvivalent med

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

Istället för att använda get- och set-metoder kan du också erhålla och ställa in gobjekt-egenskaperna genom egenskapen ”props” som `widget.props.egenskapsnamn = värde`. Detta är ekvivalent med det mer utförliga `widget.get_property("egenskapsnamn")` och `widget.set_property("egenskapsnamn", värde)`.

För att se vilka egenskaper som är tillgängliga för en komponent i versionen som körs av GTK kan du köra ”dir” på egenskapen ”props”:

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

Detta kommer i konsolen skriva listan av egenskaper som en Gtk.Box har.

KAPITEL 4

# Hur du hanterar strängar

Detta avsnitt förklarar hur strängar representeras i Python 2.x, Python 3.x och GTK+ samt diskuterar vanliga fel som uppstår vid arbete med strängar.

## 4.1 Definitioner

Som ett koncept är en sträng en lista tecken så som ”A”, ”B”, ”C” or ”É”. **Tecken** är abstrakta representationer och deras betydelse beror på språket och sammanhanget de används i. Unicode-standarden beskriver hur tecken representeras med **kodpunkter**. Exempelvis representeras tecknen ovan med kodpunkterna U+0041, U+0042, U+0043 respektive U+00C9. Kodpunkter är helt enkelt tal i intervallet 0 till 0x10FFFF.

Som nämndes tidigare är representationen av en sträng som en lista med kodpunkter abstrakt. För att konvertera denna abstrakta representation till en sekvens byte så behöver Unicode-strängen **kodas**. Den enklaste formen av kodning är ASCII och utförs som följer:

1. Om kodpunkten är < 128 så är varje byte detsamma som värdet på kodpunkten.
2. Om kodpunkten är 128 eller större, så kan Unicode-strängen inte representeras i denna kodning. (Python utfärdar ett `UnicodeEncodeError`-undantag i detta fall.)

Även om ASCII-kodning är lätt att tillämpa så kan den bara koda 128 olika tecken vilket knappast är tillräckligt. En av de vanligast använda kodningarna som tar itu med detta problem är UTF-8 (den kan hantera alla Unicode-kodpunkter). UTF står för ”Unicode Transformation Format”, och ”8” står för att 8-bitars tal används i kodningen.

## 4.2 Python 2

### 4.2.1 Unicode-stöd i Python 2.x

Python 2 kommer med två olika sorters objekt som kan användas för att representera strängar, `str` och `unicode`. Instanser av den senare används för att uttrycka Unicode-strängar, medan instanser av typen `str` är byterepresentationer (den kodade strängen). Under huven representerar Python Unicode-strängar antingen som 16- eller 32-bitars heltal, beroende på hur Python-tolken kompilerades. Unicode-strängar kan konverteras till 8-bitars strängar med `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'
```

Pythons 8-bitars strängar har en `str.decode()`-metod som tolkar strängen med given kodning:

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

Tyvärr låter Python 2.x dig mixa `unicode` och `str` om 8-bitarssträngen råkade innehålla endast 7-bitars byte (ASCII), men skulle få `UnicodeDecodeError` om den innehöll värden som inte var 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 i GTK+

GTK+ använder UTF-8-kodade strängar för all text. Detta betyder att om du anropar en metod som returnerar en sträng kommer du alltid erhålla en instans av typen `str`. Detsamma gäller metoder som förväntar sig en eller flera strängar som parameter, de måste vara UTF-8-kodade. För bekvämlighet kommer dock PyGObject automatiskt konvertera alla `unicode`-instanser till `str` om de tillhandahålls som argument:

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

(continues on next page)

(fortsättning från föregående sida)

```
>>> 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
```

Observera varningen i slutet. Även om vi anropade `Gtk.Label.set_text()` med en `unicode`-instans som argument, så kommer `Gtk.Label.get_text()` alltid returnera en `str`-instans. Därmed är `txt` och `unicode_string` *inte* lika.

Detta är särskilt viktigt om du vill internationalisera ditt program med gettext. Du måste säkerställa att gettext kommer returnera UTF-8-kodade 8-bitars strängar för alla språk. I allmänhet rekommenderas det att inte använda `unicode`-objekt i GTK+-program överhuvudtaget och endast använda UTF-8-kodade `str`-objekt då GTK+ inte är helt integrerat med `unicode`-objekt. I annat fall kommer du behöva avkoda returvärdena till Unicode-strängar varje gång du anropar en GTK+-metod:

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

## 4.3 Python 3

### 4.3.1 Unicode-stöd i Python 3.x

Sedan Python 3.0 lagras alla strängar som Unicode i en instans av typen `str`. *Kodade* strängar representeras å andra sidan som binärdata i form av instanser av typen `bytes`. Konceptuellt refererar `str` till *text*, medan `bytes` refererar till *data*. Använd `str.encode()` för att gå från `str` till `bytes`, och `bytes.decode()` för att gå från `bytes` till `str`.

Vidare är det inte längre möjligt att mixa Unicode-strängar med kodade strängar, eftersom det kommer resultera i ett `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 i GTK+

Som en följd av detta är saker mycket prydligare och mer konsekventa med Python 3.x, för PyGObject kommer automatiskt koda/avkoda till/från UTF-8 om du skickar med en sträng till en metod eller om en metod returnerar en sträng. Strängar, eller *text*, kommer alltid endast att representeras som instanser av `str`:

```
>>> 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 Referenser

Vad är nytt i Python 3.0 beskriver de nya koncept som tydligt skiljer mellan text och data.

Unicode HOWTO diskuterar Unicode-stödet i Python 2.x, och förklarar olika problem som folk vanligen stöter på då de försöker arbeta med Unicode.

Unicode HOWTO för Python 3.x diskuterar Unicode-stöd i Python 3.x.

UTF-8 encoding table and Unicode characters innehåller en lista över Unicode-kodpunkter och deras respektive UTF-8-kodning.

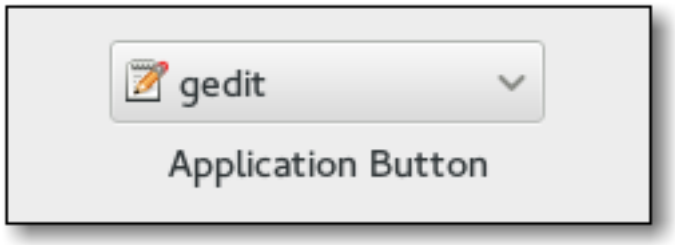
KAPITEL 5

# Komponentgalleri

| | |
|---|---|
| `Gtk.AboutDialog` |   |

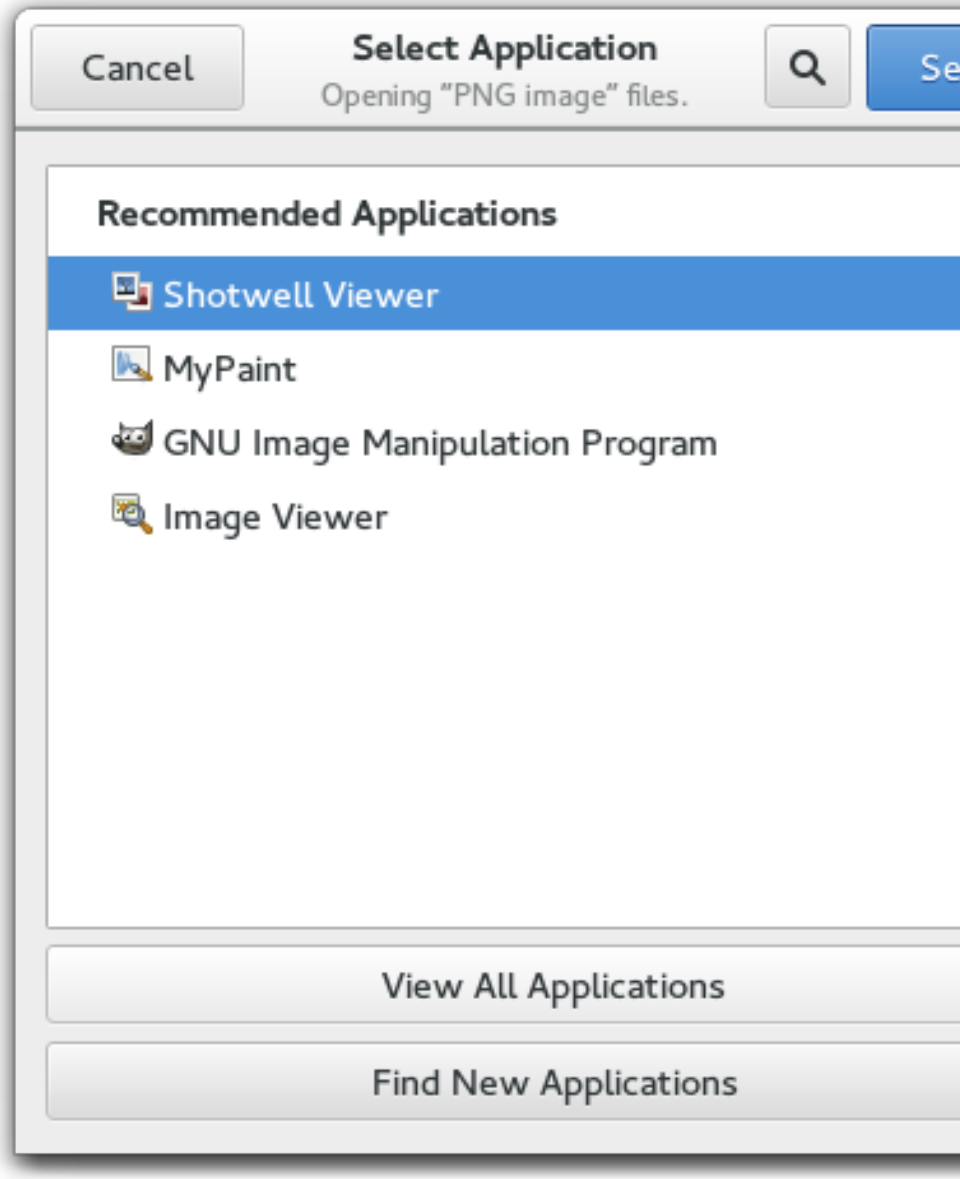
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.AccelLabel |   |
| Gtk.ActionBar |  |
| Gtk.AppChooserButton |   |

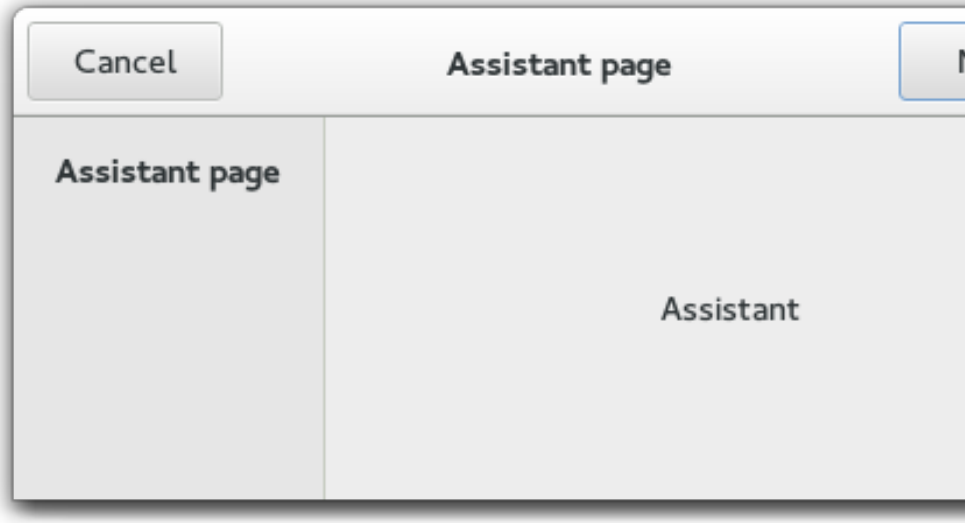
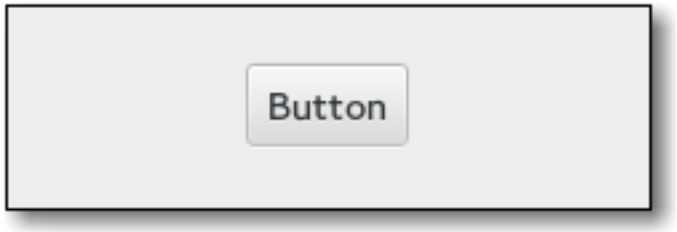
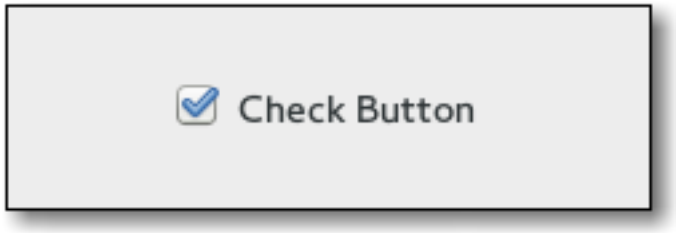
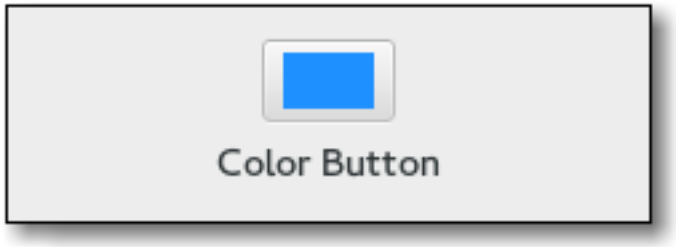
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| `Gtk.AppChooserDialog` |   |

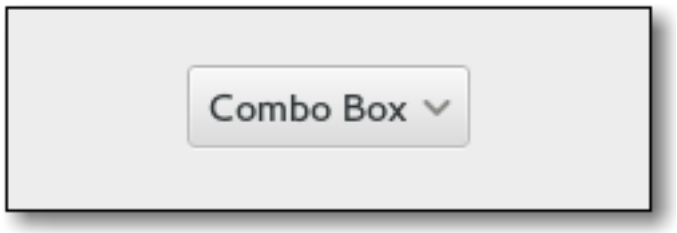
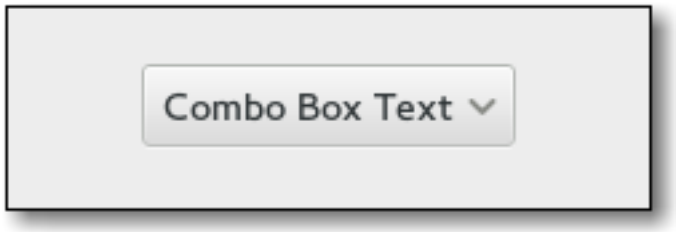
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.Assistant |   |
| Gtk.Button |   |
| Gtk.CheckButton |   |
| Gtk.ColorButton |   |

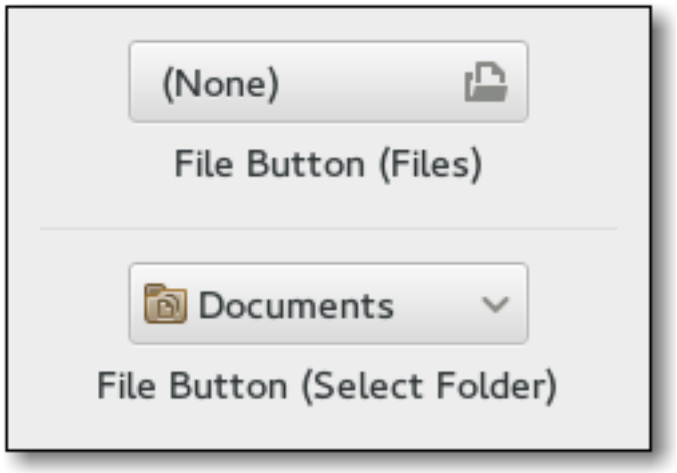
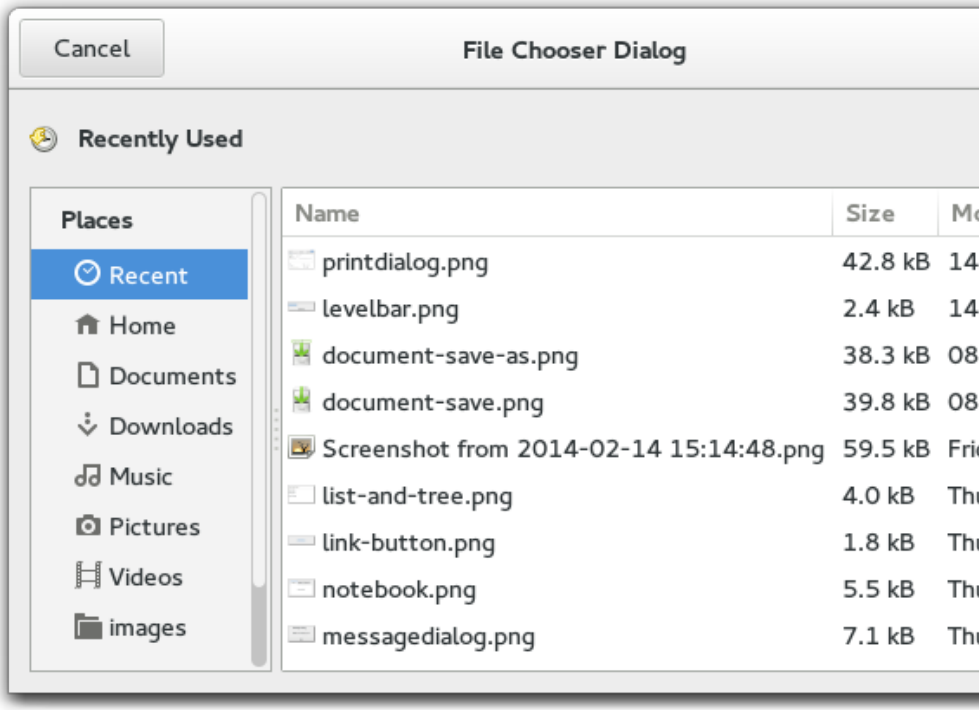
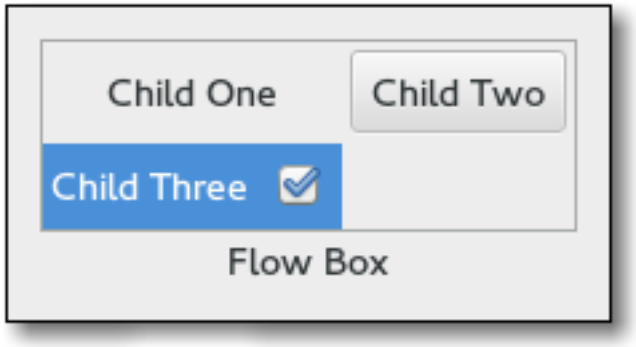
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.ColorChooserDialog |  |
| Gtk.ComboBox |  |
| Gtk.ComboBoxText |  |
| Gtk.Entry |  |

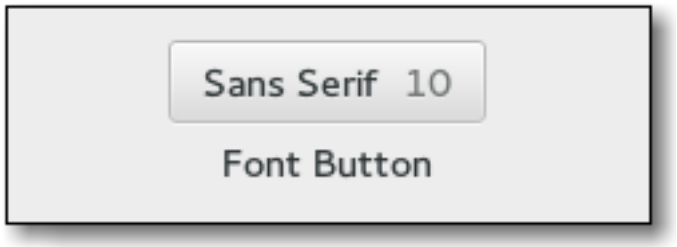
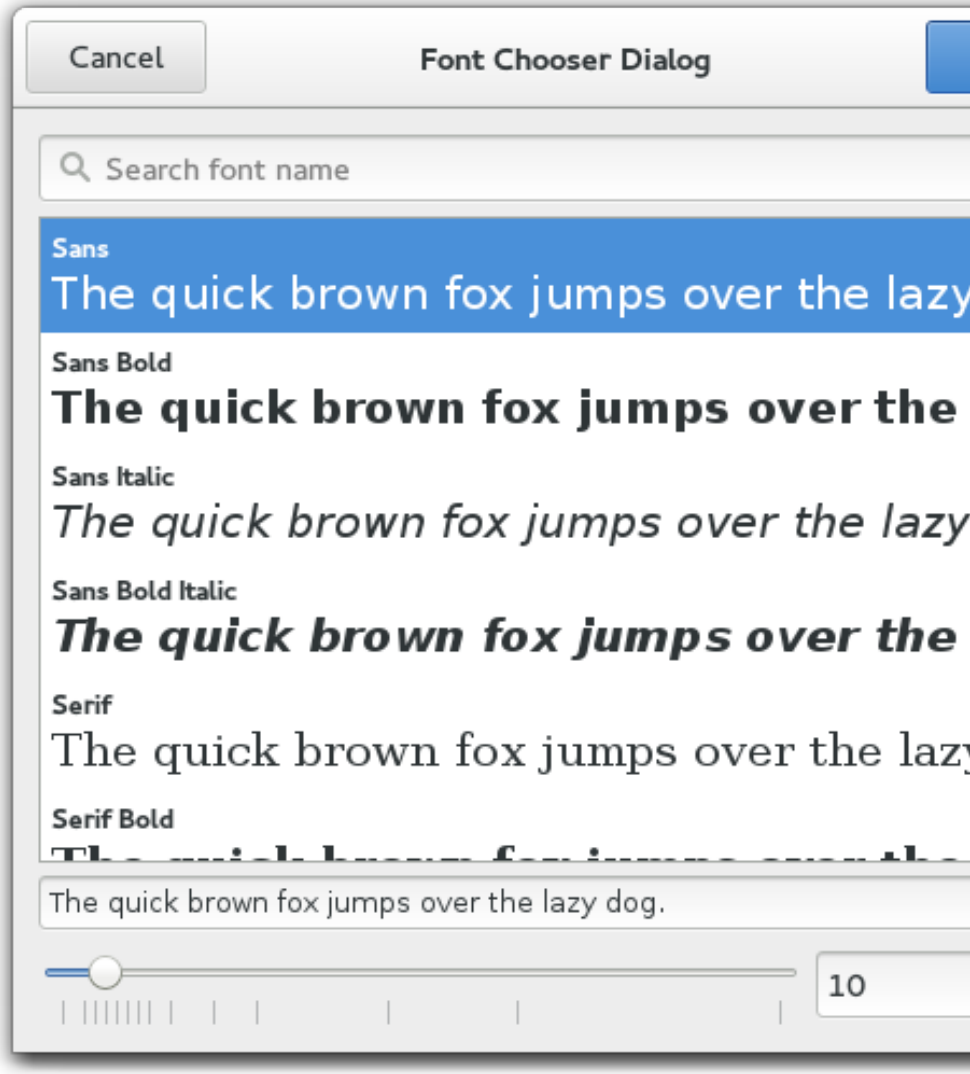
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.FileChooserButton |  |
| Gtk.FileChooserDialog |  |
| Gtk.FlowBox |  |

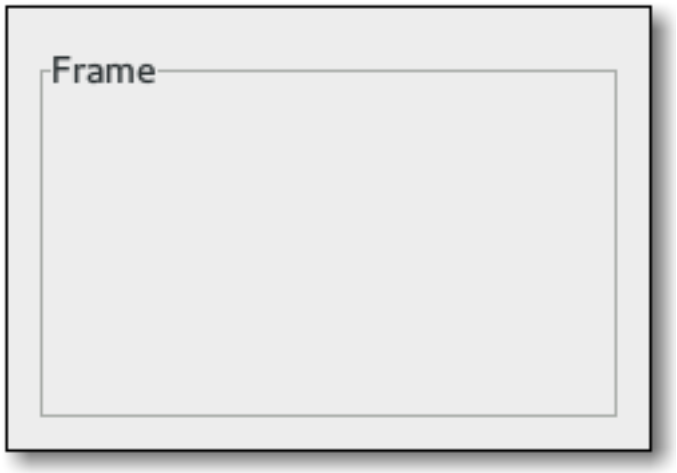
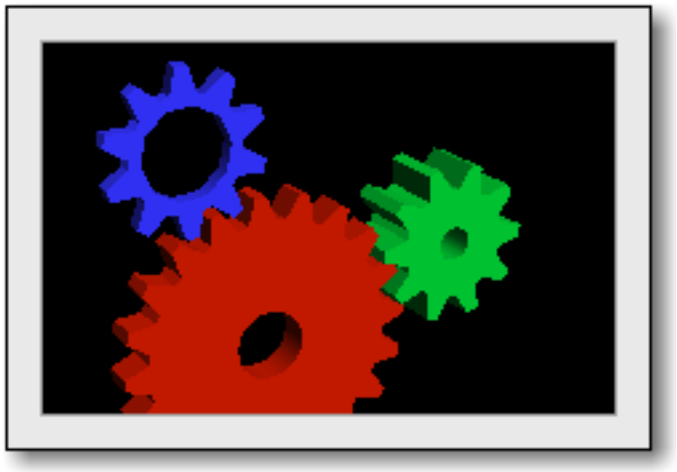
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| `Gtk.FontButton` |  |
| `Gtk.FontChooserDialog` |  |

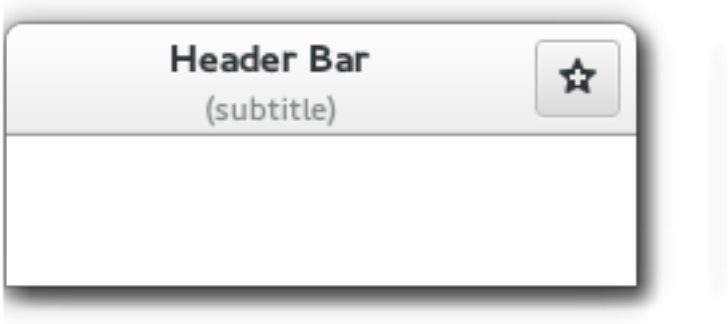
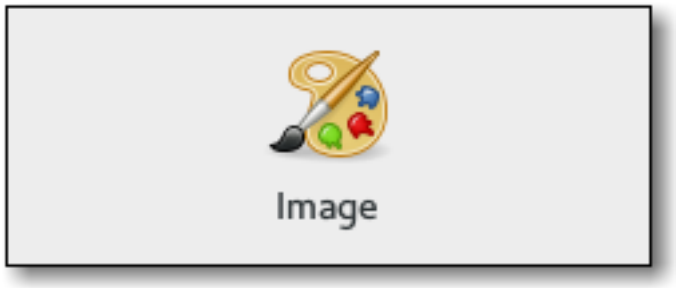
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| `Gtk.Frame` |  |
| `Gtk.GLArea` |  |
| `Gtk.Grid` |  |

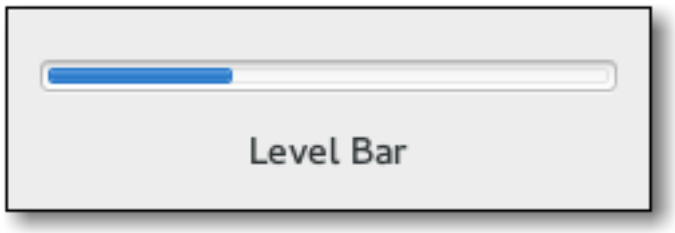
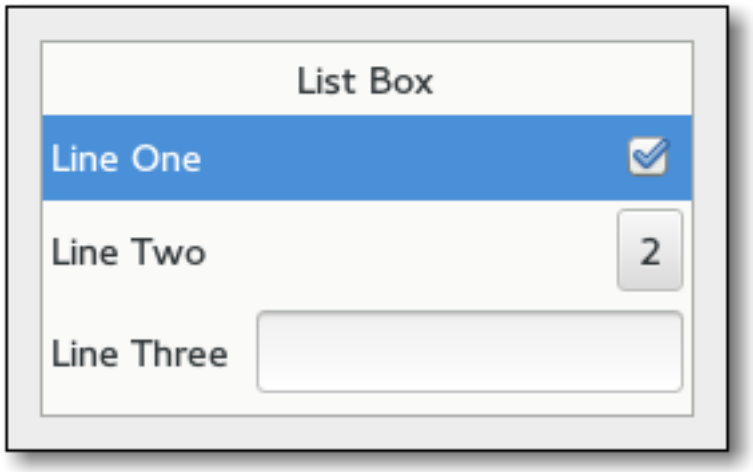
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.HeaderBar |  |
| Gtk.IconView |  |
| Gtk.Image |  |
| Gtk.InfoBar |  |

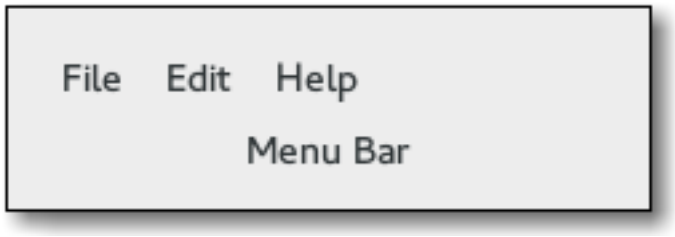
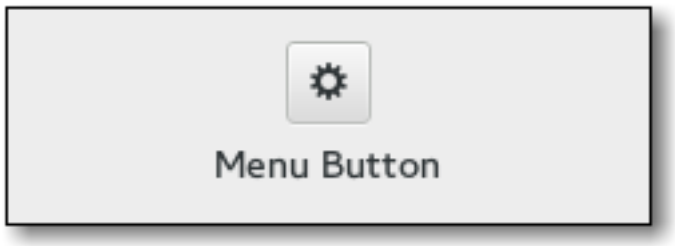
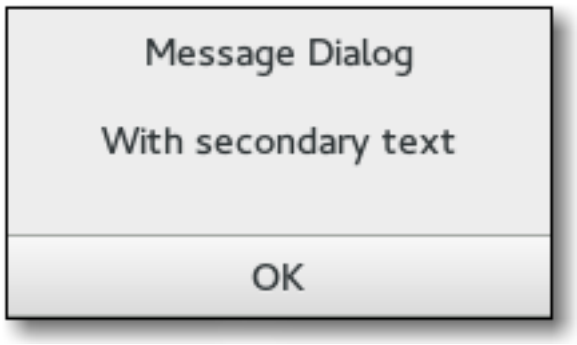
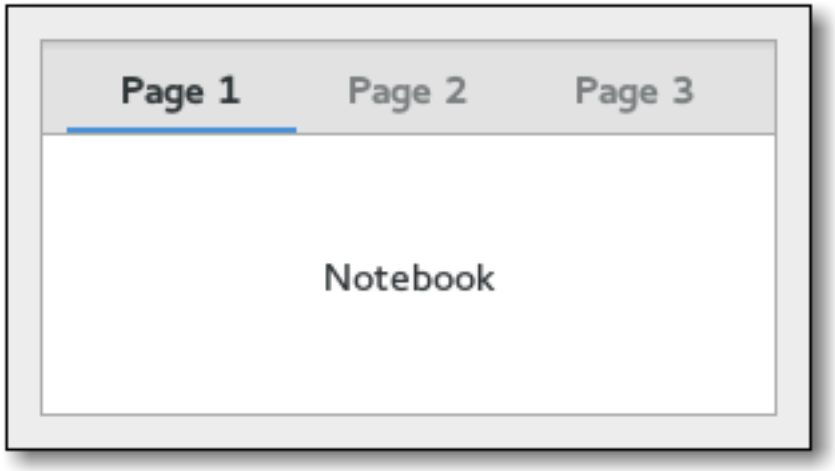
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.Label |   |
| Gtk.LevelBar |   |
| Gtk.LinkButton |   |
| Gtk.ListBox |   |
| Gtk.LockButton |   |

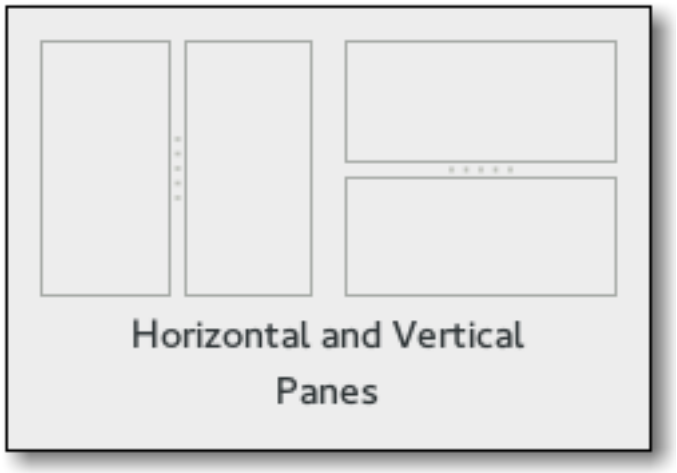
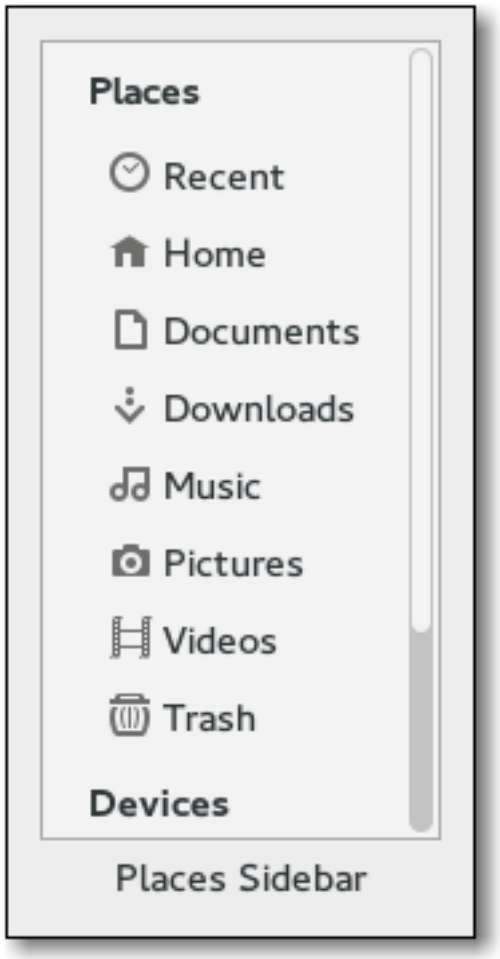
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.MenuBar |  |
| Gtk.MenuButton |  |
| Gtk.MessageDialog |  |
| Gtk.Notebook |  |

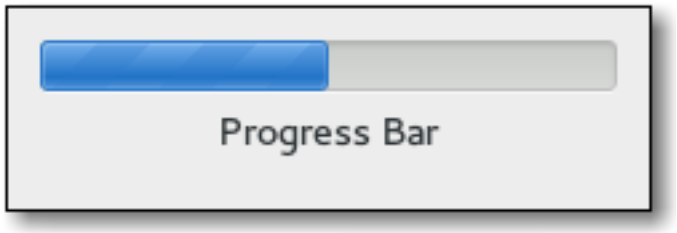
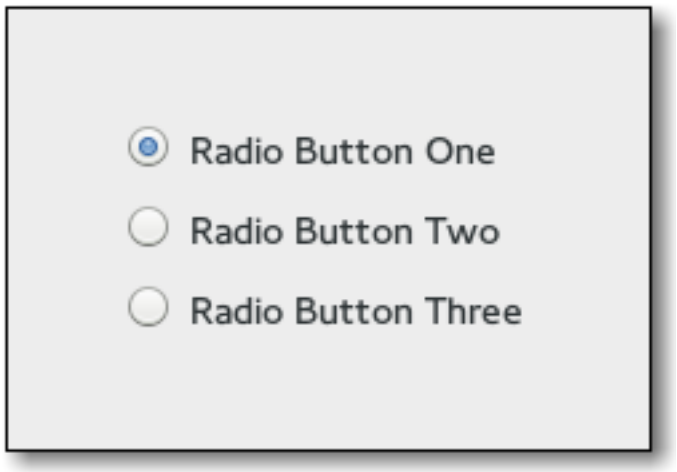
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| `Gtk.Paned` |  |
| `Gtk.PlacesSidebar` |  |

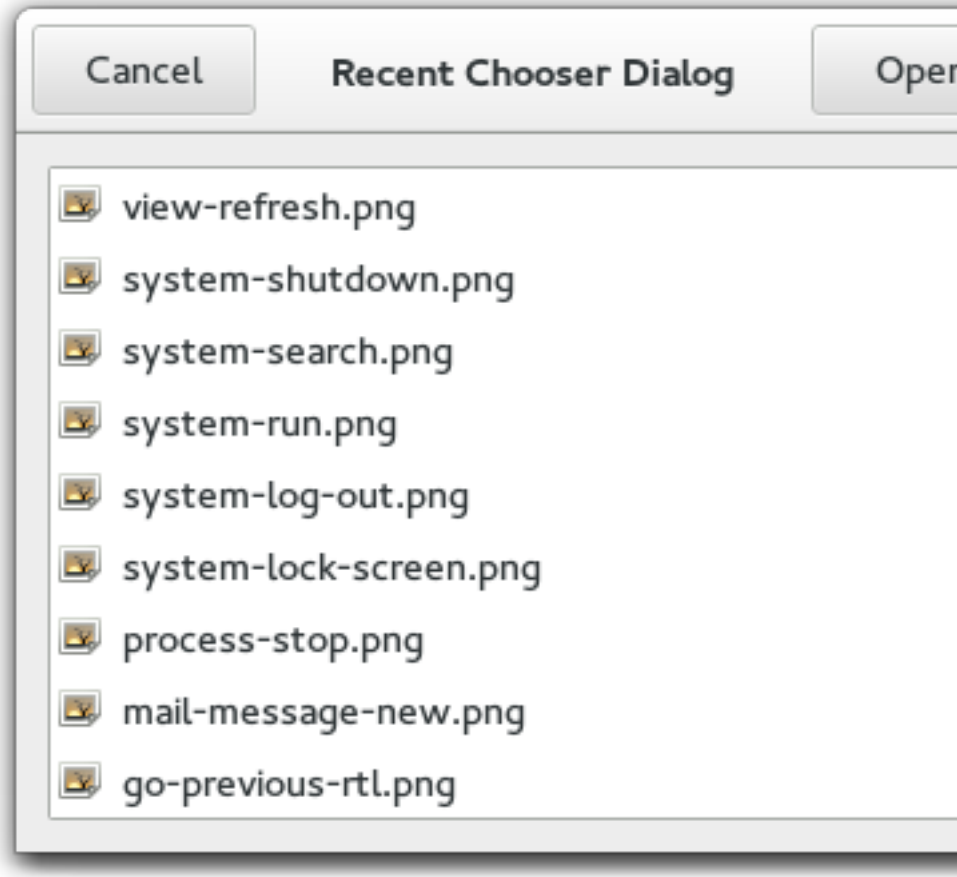
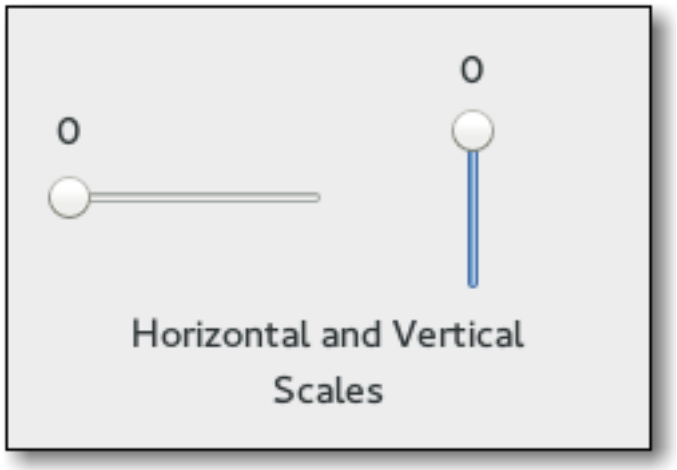
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| `Gtk.ProgressBar` |  |
| `Gtk.RadioButton` |  |

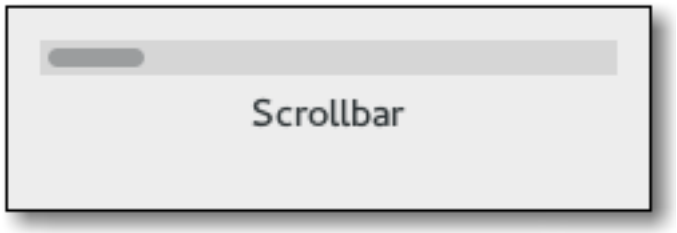
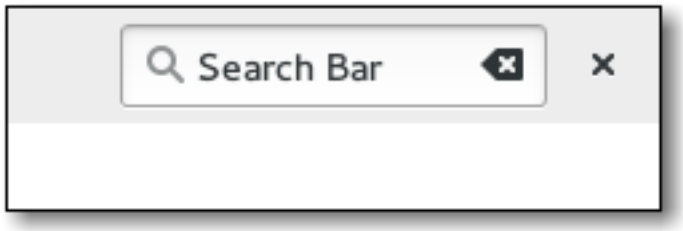
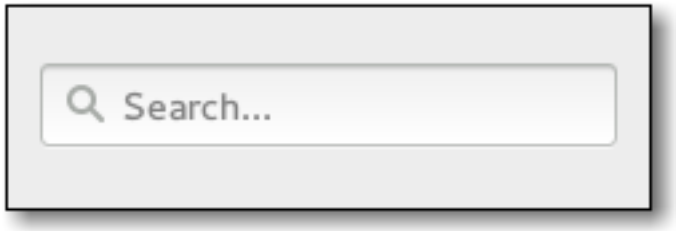
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| `Gtk.RecentChooserDialog` |   |
| `Gtk.Scale` |   |

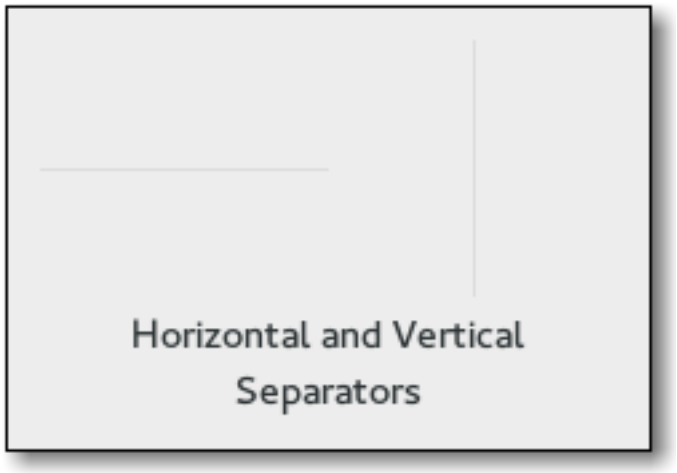
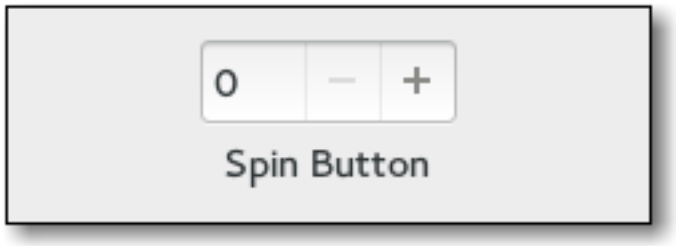
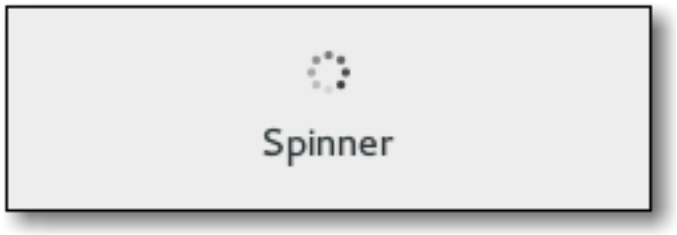
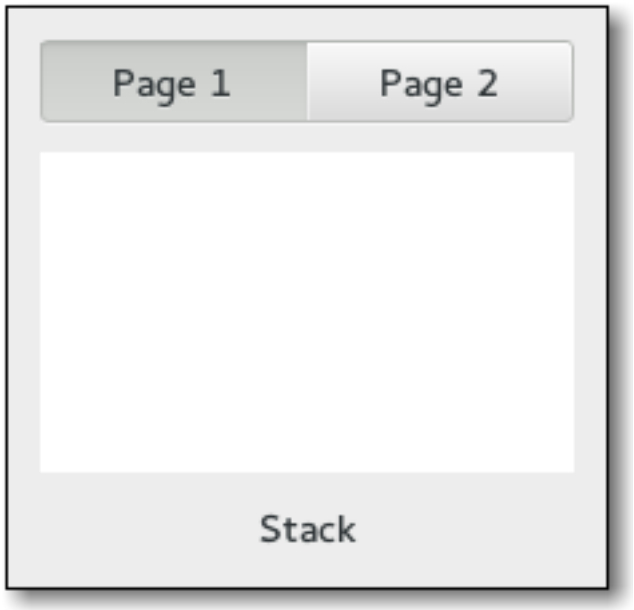
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.Scrollbar |  |
| Gtk.ScrolledWindow |  |
| Gtk.SearchBar |  |
| Gtk.SearchEntry |  |

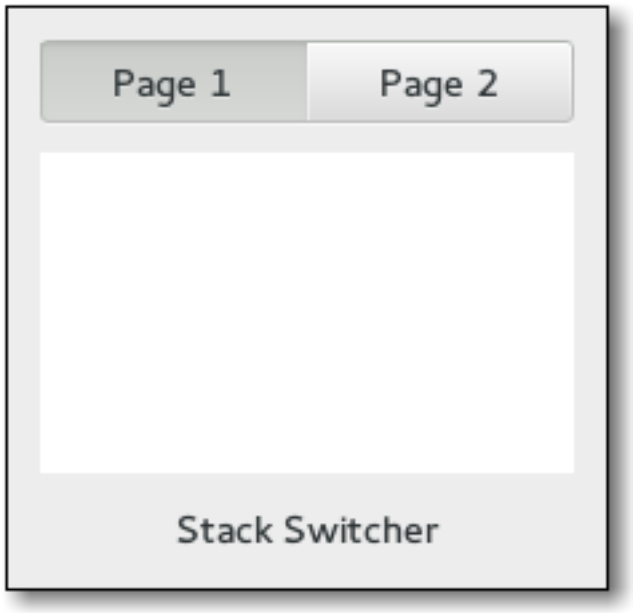
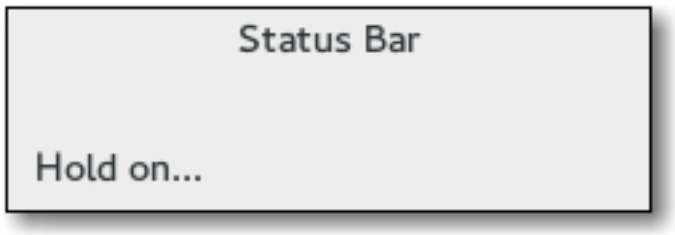
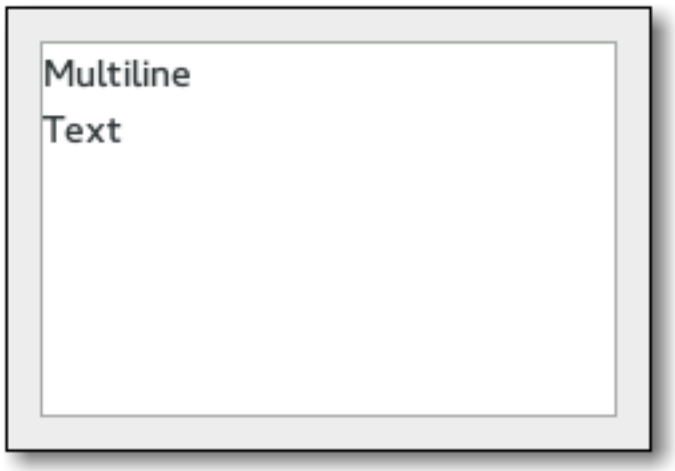
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.Separator |  |
| Gtk.SpinButton |  |
| Gtk.Spinner |  |
| Gtk.Stack |  |

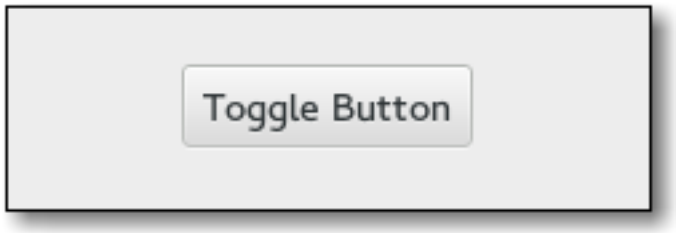
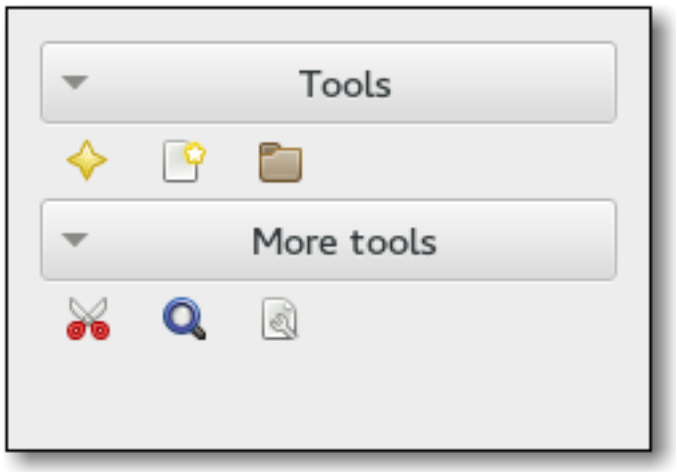
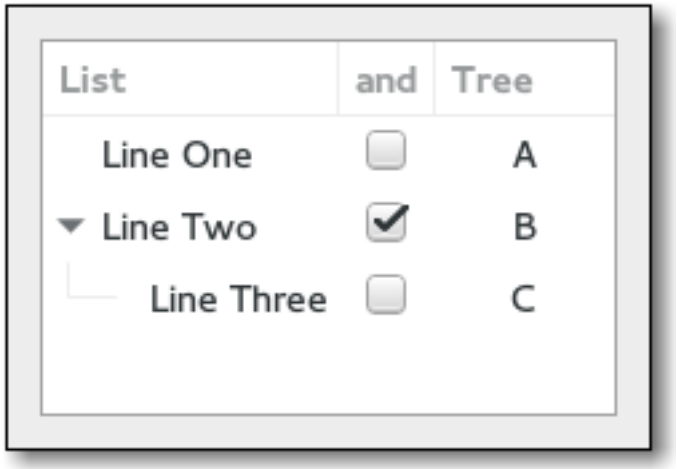
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.StackSwitcher |  |
| Gtk.Statusbar |  |
| Gtk.Switch |  |
| Gtk.TextView |  |

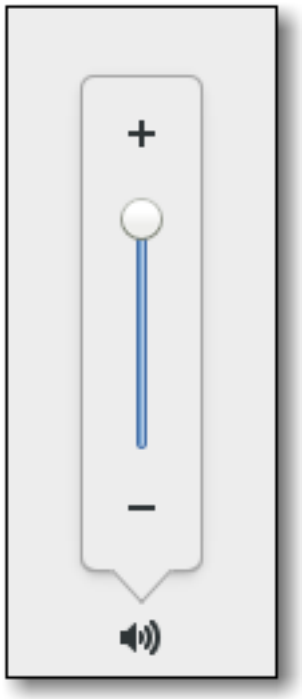
cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| Gtk.ToggleButton |  |
| Gtk.ToolPalette |  |
| Gtk.Toolbar |  |
| Gtk.TreeView |  |

cont

Table 1 – fortsättning från föregående sida

| | |
|---|---|
| `Gtk.VolumeButton` |  |
| `Gtk.Window` |  |

KAPITEL 6

# Layoutbehållare

Medan många verktygslådor för grafiska användargränssnitt tvingar dig att precist placera komponenter i ett fönster med absolut positionering, så använder GTK+ ett annat tillvägagångssätt. Snarare än att ange positionen och storleken för varje komponent i fönstret så kan du arrangera dina komponenter i rader, kolumner och/eller tabeller. Storleken på ditt fönster kan avgöras automatiskt, baserat på storleken för de komponenter som det innehåller. Storlekarna för komponenterna avgörs i sin tur av mängden text som de innehåller, eller minimi- och maximistorlekarna som du anger, och/eller hur du har begärt att det tillgängliga utrymmet ska delas mellan uppsättningar av komponenter. Du kan göra din layout bättre genom att ange utfyllnadsavstånd och centreringsvärden för var och en av dina komponenter. GTK+ använder sedan all denna information för att ändra storlek och position för allt på ett förnuftigt och smidigt sätt när användaren manipulerar fönstret.

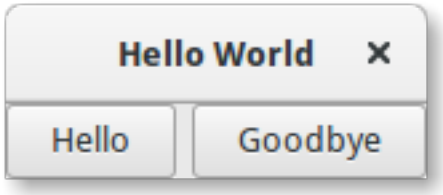
GTK+ arrangerar komponenter hierarkiskt, med *behållare*. De är osynliga för slutanvändaren och infogas i ett fönster, eller placeras i varandra för att skapa en layout för komponenter. Det finns två sorters behållare: behållare med ett barn, vilka alla är ättlingar till `Gtk.Bin`, och behållare med flera barn, vilka är ättlingar till `Gtk.Container`. De mest använda är vertikala eller horisontella rutor (`Gtk.Box`) och rutnät (`Gtk.Grid`).

## 6.1 Rutor

Rutor är osynliga behållare som vi kan packa våra komponenter i. Då komponenter packas i en horisontell ruta infogas objekten horisontellt från vänster till höger eller höger till vänster beroende på huruvida `Gtk.Box.pack_start()` eller `Gtk.Box.pack_end()` används. I en vertikal ruta packas komponenter uppifrån och ner eller omvänt. Du kan använda alla tänkbara kombinationer av rutor i eller bredvid andra rutor för att skapa önskad effekt.

### 6.1.1 Exempel

Låt oss ta en titt på en något ändrad version av det utökade exemplet med två knappar.




```
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()
```

Vi skapar först en horisontellt orienterad rutbehållare där 6 bildpunkter placeras mellan barn. Denna ruta blir barnet till toppnivåfönstret.

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

Därefter lägger vi till två olika knappar till rutbehållaren.

```
        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")
```

(continues on next page)

(fortsättning från föregående sida)

```
        self.button2.connect("clicked", self.on_button2_clicked)
        self.box.pack_start(self.button2, True, True, 0)
```

Medan komponenter positioneras från vänster till höger med `Gtk.Box.pack_start()`, så positioneras de från höger till vänster med `Gtk.Box.pack_end()`.

## 6.2 Grid

`Gtk.Grid` är en behållare som arrangerar sina barnkomponenter i rader och kolumner, men du behöver inte ange dimensionerna i konstruktorn. Barn läggs till med `Gtk.Grid.attach()`. De kan spänna över flera rader eller kolumner. Metoden `Gtk.Grid.attach()` tar fem parametrar:

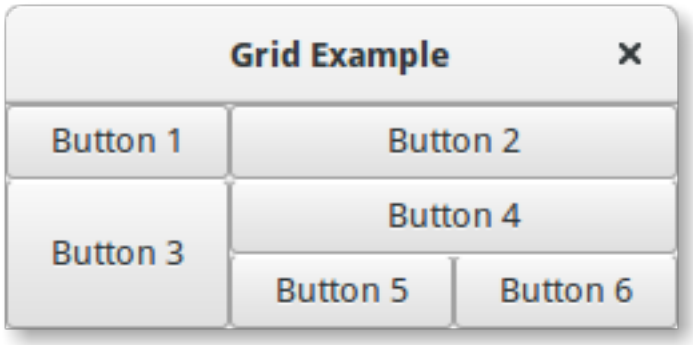
1. Parametern `child` är den `Gtk.Widget` som ska läggas till.
2. `left` är kolumnnumret att fästa vänster sida av `child` till.
3. `top` indikerar radnumret att fästa översidan av `child` till.
4. `width` och `height` indikerar antalet kolumner respektive rader som `child` kommer spänna över.

Det är också möjligt att lägga till ett barn intill ett befintligt barn, med `Gtk.Grid.attach_next_to()`, vilket också tar fem parametrar:

1. `child` är den `Gtk.Widget` som ska läggas till, som ovan.
2. `sibling` är en befintlig barnkomponent till `self` (en `Gtk.Grid`-instans) eller `None`. `child`-komponenten kommer placeras intill `sibling`, eller om `sibling` är `None`, i början eller slutet av rutnätet.
3. `side` är en `Gtk.PositionType` som indikerar vilken sida av `sibling` som `child` positioneras intill.
4. `width` och `height` indikerar antalet kolumner respektive rader som komponenten `child` kommer spänna över.

Slutligen kan `Gtk.Grid` användas som en `Gtk.Box` genom att helt enkelt använda `Gtk.Grid.add()`, vilken kommer placera barn intill varandra i den riktning som avgörs av egenskapen ”orientation” (har standardvärdet `Gtk.Orientation.HORIZONTAL`).

### 6.2.1 Exempel




```
import gi

gi.require_version("Gtk", "3.0")
```

(continues on next page)

(fortsättning från föregående sida)

```
from gi.repository import Gtk


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()
```

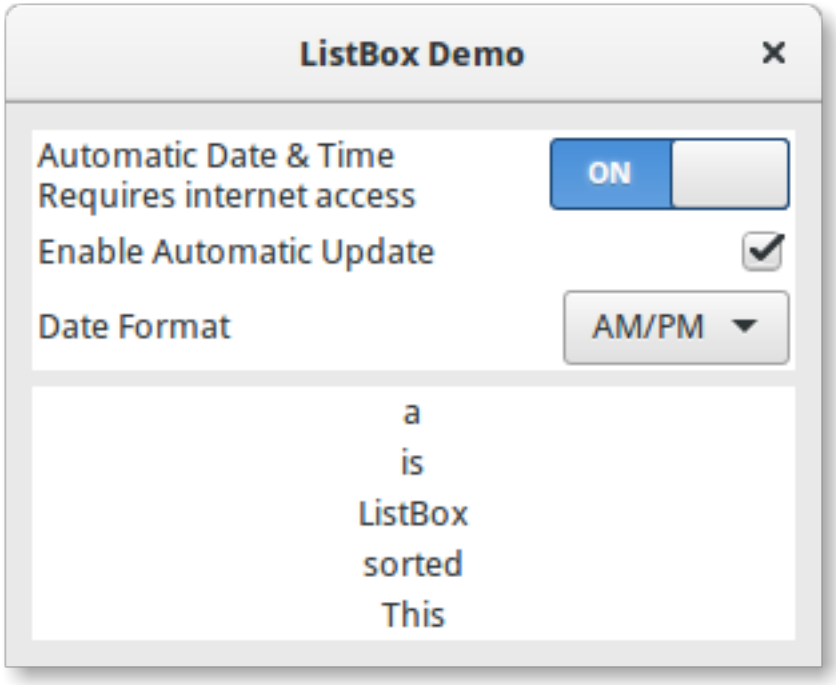
## 6.3 ListBox

En `Gtk.ListBox` är en vertikal behållare som innehåller `Gtk.ListBoxRow`-barn. Dessa rader kan sorteras och filtreras dynamiskt, och rubriker kan läggas till dynamiskt beroende på radinnehållet. Den tillåter också tangentbords- och musnavigering samt markering som en typisk lista.

Att använda `Gtk.ListBox` är ofta ett alternativ till `Gtk.TreeView`, särskilt när listinnehållet har en mer komplicerad layout än vad som tillåts av en `Gtk.CellRenderer`, eller när innehållet är interaktivt (t.ex. har en knapp i sig).

Även om en `Gtk.ListBox` endast får ha barn som är `Gtk.ListBoxRow`, så kan du lägga till valfri komponent till den med `Gtk.Container.add()`, så kommer automatiskt en `Gtk.ListBoxRow`-komponent infogas mellan listan och komponenten.

### 6.3.1 Exempel




```
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)
```

(continues on next page)

(fortsättning från föregående sida)

```
        label1 = Gtk.Label(label="Automatic Date & Time", xalign=0)
        label2 = Gtk.Label(label="Requires internet access", xalign=0)
        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)
        listbox_2.show_all()

```

(continues on next page)

(fortsättning från föregående sida)

```

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

## 6.4 Stack och StackSwitcher

En `Gtk.Stack` är en behållare som bara visar ett av sina barn åt gången. Till skillnad från `Gtk.Notebook` så tillhandahåller `Gtk.Stack` inte något sätt för användare att ändra det synliga barnet. Istället kan komponenten `Gtk.StackSwitcher` användas med `Gtk.Stack` för att tillhandahålla denna funktionalitet.

Övergångar mellan sidor kan animeras som bildspel eller toningar. Detta kan styras med `Gtk.Stack.set_transition_type()`. Dessa animeringar följer inställningen ”gtk-enable-animations”.

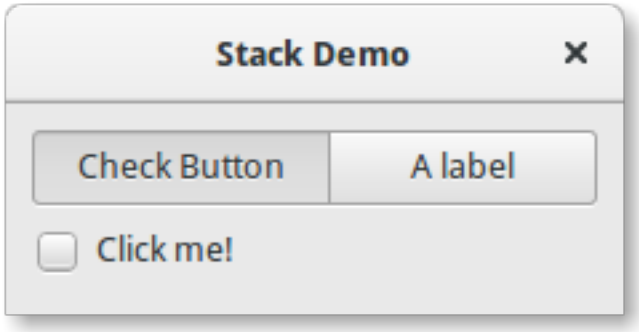
Övergångshastighet kan justeras med `Gtk.Stack.set_transition_duration()`

Komponenten `Gtk.StackSwitcher` agerar som en styrenhet för en `Gtk.Stack`; den visar en knapprad för att växla mellan de olika sidorna för den associerade stackkomponenten.

Allt innehåll för knapparna kommer från barnegenskaperna för `Gtk.Stack`.

Det är möjligt att associera flera `Gtk.StackSwitcher`-komponenter med samma `Gtk.Stack`-komponent.

### 6.4.1 Exempel




```
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)

        stack = Gtk.Stack()
```

(continues on next page)

(fortsättning från föregående sida)

```
        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()
```

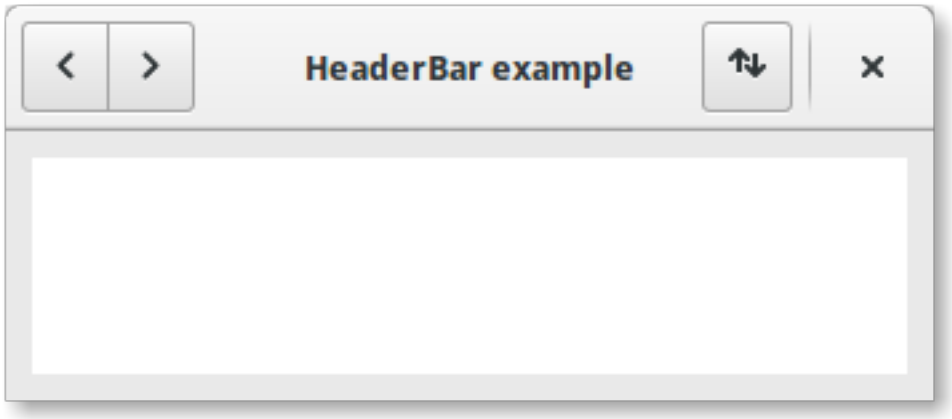
## 6.5 HeaderBar

En `Gtk.HeaderBar` liknar en horisontell `Gtk.Box`, den tillåter att barn placeras i början eller slutet. Dessutom tillåter den att en titel visas. Titeln kommer centreras med avseende på rutans bredd, även om barnen på endera sida tar upp olika mycket utrymme.

Eftersom GTK+ nu stöder dekoration på klientsidan (CSD) så kan en `Gtk.HeaderBar` användas istället för namnlisten (vilken renderas av fönsterhanteraren).

En `Gtk.HeaderBar` placeras vanligen högst upp i ett fönster och bör innehålla vanligen använda kontroller som påverkar innehållet nedan. De tillhandahåller också åtkomst till fönsterstyrkomponenter, vilket inkluderar knappen för att stänga fönstret och fönstermenyn.

### 6.5.1 Exempel



```python
import gi

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

**Observera:** Detta exempel kräver åtminstone GTK+ 3.12.

En `Gtk.FlowBox` är en behållare som positionerar barnkomponenter i ordning enligt sin orientering.

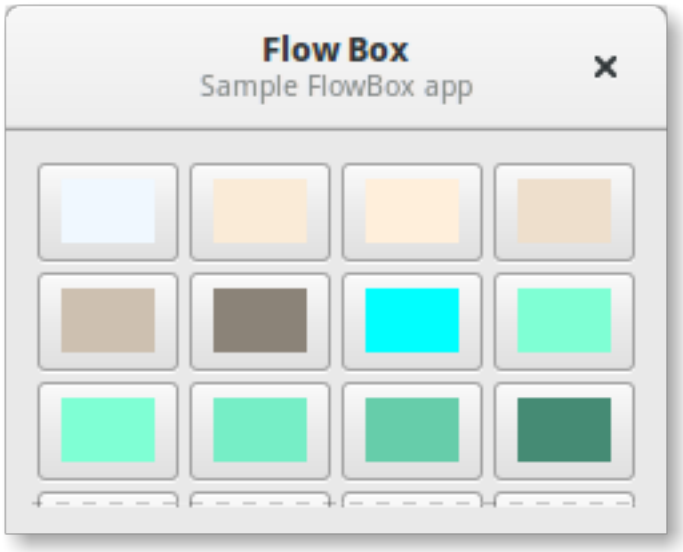
Exempelvis kommer komponenterna vid horisontell orientering att arrangeras från vänster till höger, och starta en ny rad under föregående rad då detta är nödvändigt. Att minska bredden kommer i detta fall kräva fler rader, så en större höjd kommer begäras.

På samma sätt kommer komponenterna vid vertikal orientering att arrangeras uppifrån och ner, och starta en ny kolumn till höger då detta är nödvändigt. Att minska höjden kommer kräva fler kolumner, så en större bredd kommer begäras.

Barnen till en `Gtk.FlowBox` kan sorteras och filtreras dynamiskt.

Även om en `Gtk.FlowBox` endast får ha barn som är `Gtk.FlowBoxChild`, så kan du lägga till valfri komponent till den med `Gtk.Container.add()`, så kommer automatiskt en `Gtk.FlowBoxChild`-komponent infogas mellan rutan och komponenten.

### 6.6.1 Exempel




```
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)

        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)
```

(continues on next page)

(fortsättning från föregående sida)

```
        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",
            "aquamarine1",
            "aquamarine2",
            "aquamarine3",
            "aquamarine4",
            "azure",
            "azure1",
            "azure2",
            "azure3",
            "azure4",
            "beige",
            "bisque",
            "bisque1",
            "bisque2",
```

(continues on next page)

(fortsättning från föregående sida)

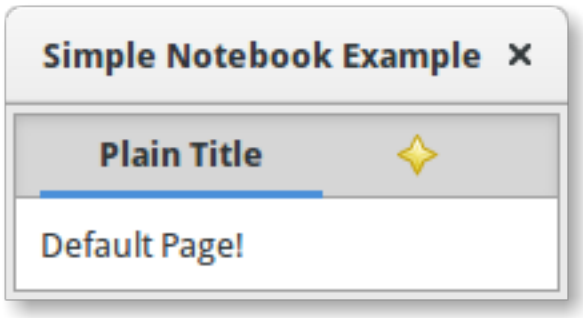
```
            "bisque3",
            "bisque4",
            "black",
            "BlanchedAlmond",
            "blue",
            "blue1",
            "blue2",
            "blue3",
            "blue4",
            "BlueViolet",
            "brown",
            "brown1",
            "brown2",
            "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",
            "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

Komponenten `Gtk.Notebook` är en `Gtk.Container` vars barn är sidor som kan växlas mellan med fliketiketter längs en kant.

Det finns många konfigurationsalternativ för GtkNotebook. Bland annat kan du välja på vilken kant flikarna visas (se `Gtk.Notebook.set_tab_pos()`), huruvida anteckningsboken ska göras större eller om rullningspilar ska läggas till då det finns för många flikar för att rymmas (se `Gtk.Notebook.set_scrollable()`), och huruvida det ska finnas en poppuppmeny som låter användarna växla sidor (se `Gtk.Notebook.popup_enable()`, `Gtk.Notebook.popup_disable()`).

### 6.7.1 Exempel




```
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)

        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)
```

(continues on next page)

(fortsättning från föregående sida)

```
30 win.show_all()
31 Gtk.main()
```

# KAPITEL 7

# Label

Etiketter är huvudmetoden för att placera ej redigerbar text i fönster, exempelvis för att placera en titel intill en `Gtk.Entry`-komponent. Du kan ange texten i konstruktorn, eller senare med metoderna `Gtk.Label.set_text()` eller `Gtk.Label.set_markup()`.

Bredden på etiketten kommer justeras automatiskt. Du kan skapa etiketter med flera rader genom att stoppa nyrader ("\n") i etikettsträngen.

Etiketter kan göras markerbara med `Gtk.Label.set_selectable()`. Markerbara etiketter låter användaren kopiera etikettinnehållet till urklipp. Endast etiketter som innehåller information som är användbar att kopiera — så som felmeddelanden — bör göras markerbara.

Etikettexten kan justeras med metoden `Gtk.Label.set_justify()`. Komponenten kan också radbryta, vilket kan aktiveras med `Gtk.Label.set_line_wrap()`.

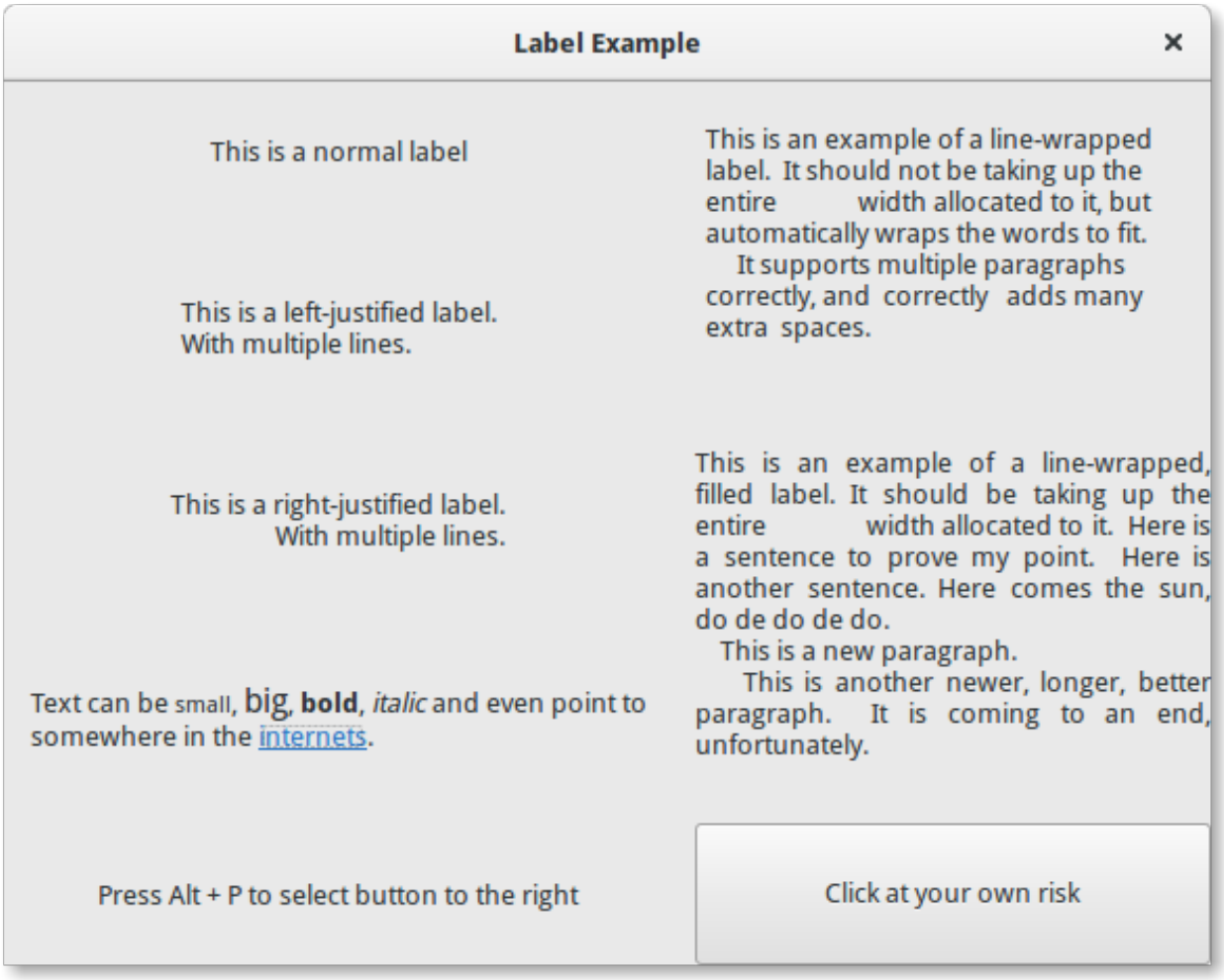
`Gtk.Label` stöder viss enkel formatering, exempelvis låter den dig göra text fet, färgad eller större. Du kan göra detta genom att tillhandahålla en sträng till `Gtk.Label.set_markup()`, med Pango Markup-syntaxen[1]. Exempelvis `<b>fet text</b> och <s>genomstruken text</s>`. Dessutom stöder `Gtk.Label` klickbara hyperlänkar. Markup för länkar har lånats från HTML, och använder "a" med attributen href och title. GTK+ renderar länkar liknande hur de visas i webbläsare, med färgad, understruken text. Attributet title visas som en inforuta över länken.

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

Etiketter kan innehålla *snabbtangenter*. Snabbtangenter är understrukna tecken i etiketten, använda för tangentbordsnavigering. Snabbtangenter skapas genom att tillhandahålla en sträng med ett understreck innan tecknet för snabbtangenten, så som "_Arkiv", till funktionerna `Gtk.Label.new_with_mnemonic()` eller `Gtk.Label.set_text_with_mnemonic()`. Snabbtangenter aktiverar automatiskt alla aktiverbara komponenter som etiketten är i, så som en `Gtk.Button`; om etiketten inte är inuti snabbtangentens målkomponent, så kommer du behöva säga till etiketten om målet med `Gtk.Label.set_mnemonic_widget()`.

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

## 7.1 Exempel




```
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)

```

(continues on next page)

(fortsättning från föregående sida)

```
        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)
        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"
```

(continues on next page)

(fortsättning från föregående sida)

```
        )
        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()
```

# KAPITEL 8

# Entry

Entry-komponenter låter användaren mata in text. Du kan ändra innehållet med metoden `Gtk.Entry.set_text()`, och läsa det aktuella innehållet med metoden `Gtk.Entry.get_text()`. Du kan också begränsa antalet tecken ditt Entry kan innehålla genom att anropa `Gtk.Entry.set_max_length()`.

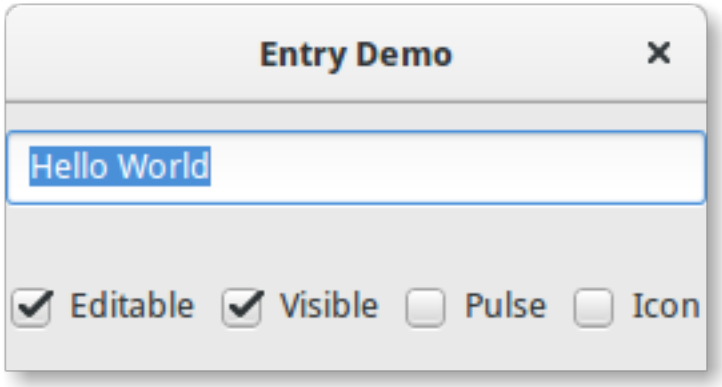
Emellanåt kan du vilja göra en Entry-komponent skrivskyddad. Detta kan göras genom att skicka `False` till metoden `Gtk.Entry.set_editable()`.

Entry-komponenter kan också användas för att ta emot lösenord från användaren. Det är praxis att dölja tecknen som skrivs in i fältet för att undvika att avslöja lösenordet till en tredje part. Att anropa `Gtk.Entry.set_visibility()` med `False` får texten att döljas.

`Gtk.Entry` har förmågan att visa förlopps- eller aktivitetsinformation bakom texten. Detta liknar `Gtk.ProgressBar`-komponenten och hittas vanligen i webbläsare för att indikera hur mycket av en sidhämtning som har slutförts. För att få ett inmatningsfält att visa sådan information, använd `Gtk.Entry.set_progress_fraction()`, `Gtk.Entry.set_progress_pulse_step()` eller `Gtk.Entry.progress_pulse()`.

Vidare kan ett Entry visa ikoner på endera sida av fältet. Dessa ikoner kan vara aktiverbara genom klick, kan ställas in som en dragkälla och kan ha inforutor. För att lägga till en ikon, använd `Gtk.Entry.set_icon_from_icon_name()` eller en av de andra funktionerna som ställer in en ikon från ett ikonnamn, en pixbuf eller ett ikontema. För att ställa in en inforuta på en ikon, använd `Gtk.Entry.set_icon_tooltip_text()` eller motsvarande funktion för markup.

## 8.1 Exempel




```
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)
```

(continues on next page)

(fortsättning från föregående sida)

```
        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()
```

KAPITEL 9

# Knappkomponenter

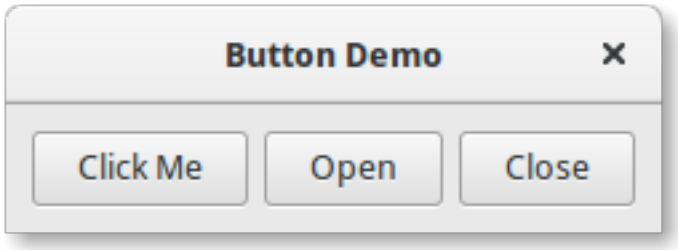
## 9.1 Button

Komponenten Button är en annan ofta använd komponent. Den används vanligen för att ansluta en funktion som anropas när knappen trycks ned.

Komponenten `Gtk.Button` kan innehålla alla giltiga barnkomponenter. Det vill säga att den kan innehålla de flesta andra vanliga `Gtk.Widget`. Det oftast använda barnet är `Gtk.Label`.

Vanligen vill du ansluta till knappens ”clicked”-signal som sänds ut när knappen har tryckts ned och släppts.

### 9.1.1 Exempel




```
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)
```

(continues on next page)

(fortsättning från föregående sida)

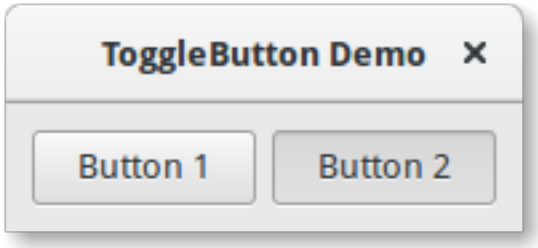
```
        hbox = Gtk.Box(spacing=6)
        self.add(hbox)

        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

En `Gtk.ToggleButton` är väldigt lik en vanlig `Gtk.Button`, men när de klickats så förbli de aktiverade, eller nedtryckta, tills de klickas på igen. När tillståndet för knappen ändras sänder den ut signalen ”toggled”.

För att erhålla tillståndet för `Gtk.ToggleButton` kan du använda metoden `Gtk.ToggleButton.get_active()`. Denna returnerar `True` om knappen är ”nere”. Du kan också ställa in växlingsknappens tillstånd med `Gtk.ToggleButton.set_active()`. Observera att om du gör detta och tillståndet faktiskt ändras så får det signalen ”toggled” att sändas ut.

### 9.2.1 Exempel



```
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` ärver från `Gtk.ToggleButton`. Den enda reella skillnaden mellan dem är utseendet för `Gtk.CheckButton`. En `Gtk.CheckButton` placerar en diskret `Gtk.ToggleButton` intill en komponent (vanligen en `Gtk.Label`). Signalen ”toggled”, `Gtk.ToggleButton.set_active()` och `Gtk.ToggleButton.get_active()` ärvs.

## 9.4 RadioButton

Liksom kryssrutor så ärver även radioknappar från `Gtk.ToggleButton`, men dessa fungerar i grupp, och endast en `Gtk.RadioButton` i en grupp kan väljas vid varje givet tillfälle. Därigenom är en `Gtk.RadioButton` ett sätt att ge användaren ett val bland många alternativ.

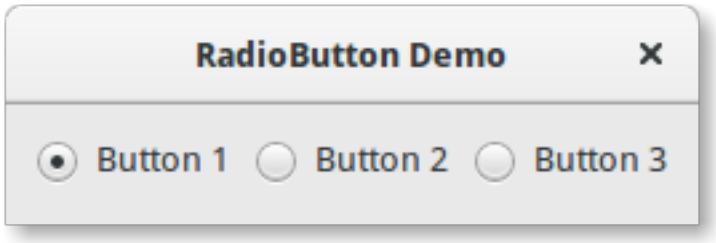
Radioknappar kan skapas med en av de statiska metoderna `Gtk.RadioButton.new_from_widget()`, `Gtk.RadioButton.new_with_label_from_widget()` eller `Gtk.RadioButton.new_with_mnemonic_from_widget()`. Den första radioknappen i en grupp kommer skicka `None` som

*group*-argument då den skapas. I följande anrop ska gruppen som du vill lägga till denna knapp till skickas med som ett argument.

Då den först körs kommer den första radioknappen vara aktiv. Detta kan ändras genom att anropa `Gtk.ToggleButton.set_active()` med `True` som första argument.

Att ändra komponentgrupp för en `Gtk.RadioButton` efter att den skapats kan göras genom att anropa `Gtk.RadioButton.join_group()`.

### 9.4.1 Exempel




```
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)

        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()
```

(continues on next page)

(fortsättning från föregående sida)

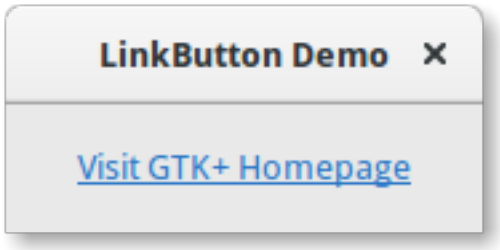
```
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

## 9.5 LinkButton

En `Gtk.LinkButton` är en `Gtk.Button` med en hyperlänk, liknande den som används av webbläsare, vilken utlöser en åtgärd då den klickas på. Den är användbar för att visa snabblänkar till resurser.

Den URI som är bunden till en `Gtk.LinkButton` kan specifikt ställas in med `Gtk.LinkButton.set_uri()`, och erhållas med `Gtk.LinkButton.get_uri()`.

### 9.5.1 Exempel




```
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")
        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

En `Gtk.SpinButton` är ett idealiskt sätt att låta användaren ställa in värdet för något attribut. Snarare än att direkt behöva skriva in ett tal i ett `Gtk.Entry`, så låter `Gtk.SpinButton` användaren klicka på en av två pilar för att öka eller minska det visade värdet. Ett värde kan fortfarande skrivas in, med bonusen att det kan kontrolleras för att säkerställa att det är i ett givet intervall. Huvudegenskaperna för en `Gtk.SpinButton` ställs in genom `Gtk.Adjustment`.

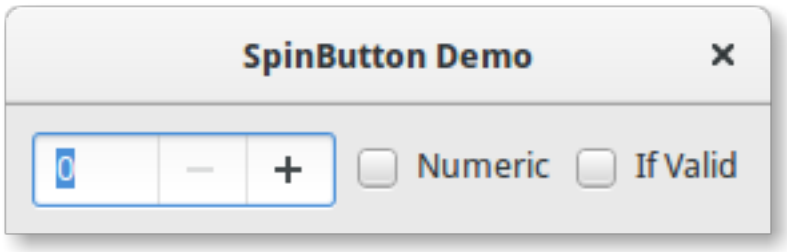
För att ändra värdet som `Gtk.SpinButton` visar, använd `Gtk.SpinButton.set_value()`. Värdet som matas in kan antingen vara ett hel- eller flyttal, beroende på dina krav. Använd `Gtk.SpinButton.get_value_as_int()` respektive `Gtk.SpinButton.get_value()`.

När du tillåter visning av flyttalsvärden i stegningsrutan kan du vilja justera antalet decimaler som visas genom att anropa `Gtk.SpinButton.set_digits()`.

Som standard accepterar `Gtk.SpinButton` textdata. Om du vill begränsa detta till endast numeriska värden anropar du `Gtk.SpinButton.set_numeric()` med `True` som argument.

Vi kan även justera uppdateringspolicyn för `Gtk.SpinButton`. Det finns två alternativ här; som standard uppdaterar stegningsrutan värdet även om de data som matas in är ogiltiga. Alternativt kan vi ställa in policyn så att det bara uppdateras när det inmatade värdet är giltigt genom att anropa `Gtk.SpinButton.set_update_policy()`.

### 9.6.1 Exempel




```
import gi

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


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

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

        adjustment = Gtk.Adjustment(upper=100, step_increment=1, page_increment=10)
        self.spinbutton = Gtk.SpinButton()
        self.spinbutton.set_adjustment(adjustment)
        self.spinbutton.connect("value-changed", self.on_value_changed)
        hbox.pack_start(self.spinbutton, False, False, 0)

        check_numeric = Gtk.CheckButton(label="Numeric")
        check_numeric.connect("toggled", self.on_numeric_toggled)
```

(continues on next page)

(fortsättning från föregående sida)

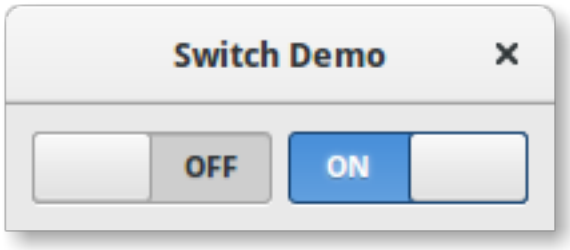
```
        hbox.pack_start(check_numeric, False, False, 0)

        check_ifvalid = Gtk.CheckButton(label="If Valid")
        check_ifvalid.connect("toggled", self.on_ifvalid_toggled)
        hbox.pack_start(check_ifvalid, False, False, 0)

    def on_value_changed(self, scroll):
        print(self.spinbutton.get_value_as_int())

    def on_numeric_toggled(self, button):
        self.spinbutton.set_numeric(button.get_active())

    def on_ifvalid_toggled(self, button):
        if button.get_active():
            policy = Gtk.SpinButtonUpdatePolicy.IF_VALID
        else:
            policy = Gtk.SpinButtonUpdatePolicy.ALWAYS
        self.spinbutton.set_update_policy(policy)


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

## 9.7 Switch

En `Gtk.Switch` är en komponent som har två tillstånd: på eller av. Användaren kan styra vilket tillstånd som ska vara aktivt genom att klicka på den tomma ytan, eller genom att dra handtaget.

Du bör inte använda ”activate”-signalen vilken är en åtgärdssignal på Gtk.Switch, att sända ut den får brytaren att animeras. Program ska aldrig ansluta till denna signal, utan använda signalen ”notify::active”, se exemplet nedan.

### 9.7.1 Exempel




```
import gi

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


class SwitcherWindow(Gtk.Window):
```

(continues on next page)

(fortsättning från föregående sida)

```
8       def __init__(self):
9           super().__init__(title="Switch Demo")
10          self.set_border_width(10)
11
12          hbox = Gtk.Box(spacing=6)
13          self.add(hbox)
14
15          switch = Gtk.Switch()
16          switch.connect("notify::active", self.on_switch_activated)
17          switch.set_active(False)
18          hbox.pack_start(switch, True, True, 0)
19
20          switch = Gtk.Switch()
21          switch.connect("notify::active", self.on_switch_activated)
22          switch.set_active(True)
23          hbox.pack_start(switch, True, True, 0)
24
25      def on_switch_activated(self, switch, gparam):
26          if switch.get_active():
27              state = "on"
28          else:
29              state = "off"
30          print("Switch was turned", state)
31
32
33  win = SwitcherWindow()
34  win.connect("destroy", Gtk.main_quit)
35  win.show_all()
36  Gtk.main()
```

# KAPITEL 10

## Expander

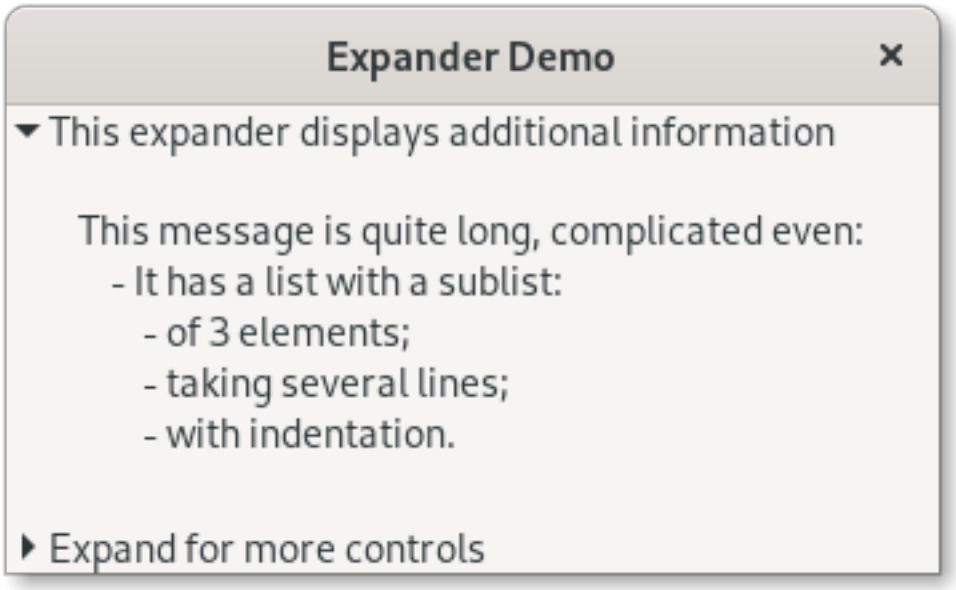
Expanderare låter dig dynamiskt dölja eller visa information i ett fönster eller en dialog. En expanderare kan ta en ensam komponent som kommer att visas när den expanderas.

Expanderare förblir expanderade tills de klickas på igen. När tillståndet för en expanderare ändras sänds signalen ”activate” ut.

En expanderare kan programmatiskt expanderas eller fällas ihop genom skicka *True* eller *False* till `Gtk.Expander.set_expanded()`. Observera att göra så får signalen ”activate” att sändas ut.

Mer än en komponent, så som en `Gtk.Label` och `Gtk.Button`, kan läggas till genom att lägga till dem till en `Gtk.Box`.

## 10.1 Exempel




```
import gi

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)

```

(continues on next page)

(fortsättning från föregående sida)

```
        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()
```

# KAPITEL 11

## ProgressBar

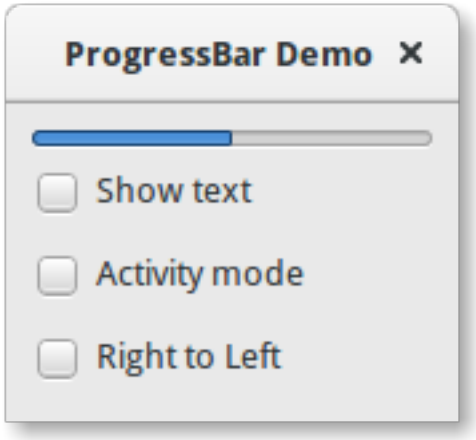
`Gtk.ProgressBar` används typiskt för att visa förloppet för en operation som pågår länge. Den tillhandahåller en visuell ledtråd om att bearbetning pågår. `Gtk.ProgressBar` kan användas i två olika lägen: *procentläge* och *aktivitetsläge*.

Då ett program kan avgöra hur mycket arbete som behöver utföras (t.ex. läsa ett fast antal byte från en fil) och kan övervaka sina framsteg, så kan det använda `Gtk.ProgressBar` i *procentläge*, så ser användaren en växande stapel som indikerar procentdelen av arbetet som har slutförts. I detta läge behöver programmet anropa `Gtk.ProgressBar.set_fraction()` periodiskt för att uppdatera förloppsindikatorn, och skicka med ett flyttal mellan 0 och 1 för att tillhandahålla det nya procentvärdet.

Då ett program inte har något exakt sätt att veta mängden arbete det måste utföra, kan det använda *aktivitetsläge*, som visar aktivitet med ett block som rör sig fram och tillbaka i förloppsområdet. I detta läget behöver programmet anropa `Gtk.ProgressBar.pulse()` periodiskt för att uppdatera förloppsindikatorn. Du kan också välja stegstorleken, med metoden `Gtk.ProgressBar.set_pulse_step()`.

Som standard är `Gtk.ProgressBar` horisontell och vänster-till-höger, men du kan ändra den till en vertikal förloppsindikator genom att använda metoden `Gtk.ProgressBar.set_orientation()`. Att ändra riktningen som förloppsindikatorn växer kan göras med `Gtk.ProgressBar.set_inverted()`. `Gtk.ProgressBar` kan också innehålla text som kan ställas in genom att anropa `Gtk.ProgressBar.set_text()` och `Gtk.ProgressBar.set_show_text()`.

## 11.1 Exempel




```
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
```

(continues on next page)

(fortsättning från föregående sida)

```
39          self.progressbar.set_text(text)
40          self.progressbar.set_show_text(show_text)
41
42      def on_activity_mode_toggled(self, button):
43          self.activity_mode = button.get_active()
44          if self.activity_mode:
45              self.progressbar.pulse()
46          else:
47              self.progressbar.set_fraction(0.0)
48
49      def on_right_to_left_toggled(self, button):
50          value = button.get_active()
51          self.progressbar.set_inverted(value)
52
53      def on_timeout(self, user_data):
54          """
55          Update value on the progress bar
56          """
57          if self.activity_mode:
58              self.progressbar.pulse()
59          else:
60              new_value = self.progressbar.get_fraction() + 0.01
61
62              if new_value > 1:
63                  new_value = 0
64
65              self.progressbar.set_fraction(new_value)
66
67          # As this is a timeout function, return True so that it
68          # continues to get called
69          return True
70
71
72  win = ProgressBarWindow()
73  win.connect("destroy", Gtk.main_quit)
74  win.show_all()
75  Gtk.main()
```

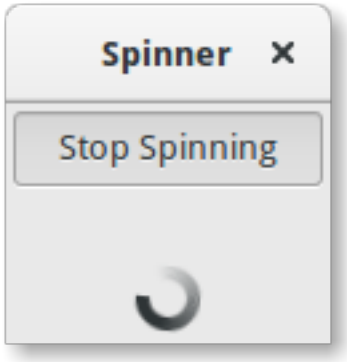
KAPITEL 12

# Spinner

`Gtk.Spinner` visar en snurrande animering av ikonstorlek. Den används ofta som ett alternativ till en `GtkProgressBar` för att visa aktivitet av obestämd längd, istället för det faktiska förloppet.

För att starta animeringen, använd `Gtk.Spinner.start()`. För att stoppa den, använd `Gtk.Spinner.stop()`.

## 12.1 Exempel




```
import gi

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


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

        super().__init__(title="Spinner")
```

(continues on next page)

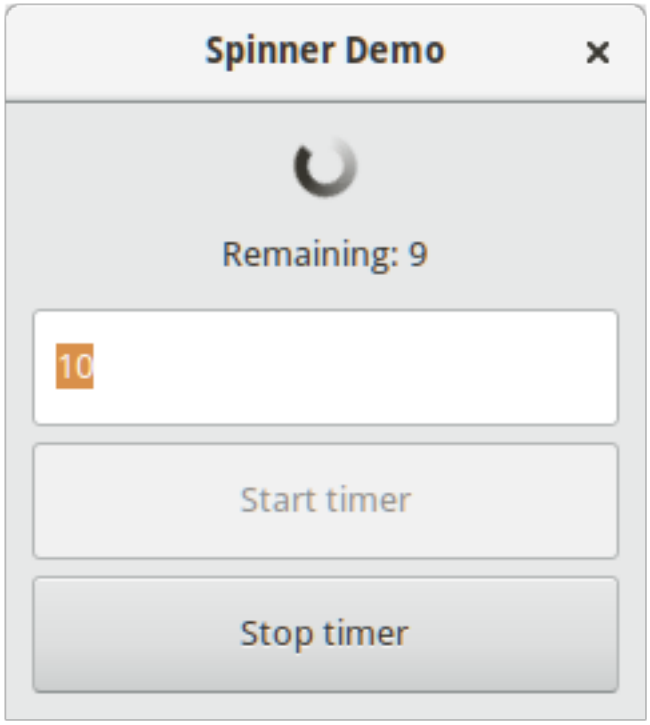
(fortsättning från föregående sida)

```
        self.set_border_width(3)
        self.connect("destroy", Gtk.main_quit)

        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 Utökat exempel

Ett utökat exempel som använder en tidsgränsfunktion för att starta och stoppa den snurrande animeringen. Funktionen `on_timeout()` anropas med regelbundna intervall till den returnerar `False`, och vid denna tidpunkt förstörs tidsgränsen automatiskt, och funktionen kommer inte anropas igen.

### 12.2.1 Exempel




```
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")
```

(continues on next page)

(fortsättning från föregående sida)

```
        self.buttonStop.set_sensitive(False)
        self.buttonStop.connect("clicked", self.on_buttonStop_clicked)
        mainBox.pack_start(self.buttonStop, True, True, 0)

        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)


```

(continues on next page)

(fortsättning från föregående sida)

```
win = SpinnerWindow()
win.show_all()
Gtk.main()
```

KAPITEL 13

# Träd- och listkomponenter

En `Gtk.TreeView` och dess associerade komponenter är ett extremt kraftfullt sätt att visa data. De används tillsammans med en `Gtk.ListStore` eller `Gtk.TreeStore` och tillhandahåller ett sätt att visa och manipulera data på många sätt, inklusive:

- Automatiska uppdateringar då data läggs till, tas bort eller redigeras
- Stöd för dra-och-släpp
- Sortering av data
- Inbäddning av komponenter så som kryssrutor, förloppsindikatorer o.s.v.
- Kolumner som går att ordna om och ändra storlek på
- Filtrering av data

Med kraften och flexibiliteten hos en `Gtk.TreeView` kommer komplexitet. Det är ofta svårt för nybörjarutvecklare att kunna använda den korrekt på grund av de antal metoder som krävs.

## 13.1 Modellen

Varje `Gtk.TreeView` har en associerad `Gtk.TreeModel` som innehåller de data som visas av denna TreeView. Varje `Gtk.TreeModel` kan användas av mer än en `Gtk.TreeView`. Exempelvis låter detta samma underliggande data visas och redigeras på två olika sätt på samma gång. Eller så kan de två vyerna visa olika kolumner från samma Model-data, på samma sätt som två SQL-frågor (eller ”vyer”) kan visa olika fält från samma databastabell.

Även om du teoretiskt kan implementera din egen Model, så kommer du vanligen använda antingen modellklasserna `Gtk.ListStore` eller `Gtk.TreeStore`. `Gtk.ListStore` innehåller enkla rader med data, och varje rad har inget barn, medan `Gtk.TreeStore` innehåller rader av data, och varje rad kan ha barnrader.

Då du konstruerar en modell måste du ange datatyperna för varje kolumn som modellen innehåller.

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

Detta skapar en listlagring med tre kolumner, två strängkolumner och en flyttalskolumn.

Att lägga till data till modellen görs med `Gtk.ListStore.append()` eller `Gtk.TreeStore.append()`, beroende på vilken sorts modell som skapades.

För en `Gtk.ListStore`:

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

För en `Gtk.TreeStore` måste du ange en befintlig rad att lägga till den nya raden till, med en `Gtk.TreeIter`, eller None för toppnivån på trädet:

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

Båda metoderna returnerar en `Gtk.TreeIter`-instans, vilken pekar på platsen för den nyligen infogade raden. Du kan erhålla en `Gtk.TreeIter` genom att anropa `Gtk.TreeModel.get_iter()`.

När data har infogats kan du erhålla eller ändra data med träditeratorn och kolumnindexet.

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

Som med Pythons inbyggda `list`-objekt kan du använda `len()` för att få antalet rader och använda slicing för att få eller ställa in värden.

```
# 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]
```

Att iterera över alla rader i en trädmodell är också väldigt enkelt.

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

Tänk på att om du använder `Gtk.TreeStore` kommer koden ovan endast iterera över raderna på toppnivån, inte nodernas barn. För att iterera över alla rader, använd `Gtk.TreeModel.foreach()`.

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

store.foreach(print_row)
```

Förutom att komma åt värden lagrade i en `Gtk.TreeModel` med den listliknande metoden nämnd ovan, så är det också möjligt att använda instanser av antingen `Gtk.TreeIter` eller `Gtk.TreePath`. Båda hänvisar till en specifik rad i en trädmodell. Man kan konvertera en stig till en iterator genom att anropa `Gtk.TreeModel.get_iter()`. Då `Gtk.ListStore` innehåller endast en nivå, d.v.s. att noder inte har några barnnoder, så är en stig helt enkelt indexet för raden som du vill komma åt.

```
# Get path pointing to 6th row in list store
path = Gtk.TreePath(5)
```

(continues on next page)

(fortsättning från föregående sida)

```
treeiter = liststore.get_iter(path)
# Get value at 2nd column
value = liststore.get_value(treeiter, 1)
```

I fallet för `Gtk.TreeStore` är en stig en lista över index eller en sträng. Strängformen är en lista över tal som skiljs åt av ett kolon. Varje tal hänvisar till positionen på den nivån. Därmed hänvisar stigen ”0” till rotnoden och stigen ”2:4” till det femte barnet till den tredje noden.

```
# 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)
```

Instanser av `Gtk.TreePath` kan kommas åt som listor, d.v.s. `len(treepath)` returnerar djupet på objektet som `treepath` pekar på, och `treepath[i]` returnerar barnets index på nivå *i*.

## 13.2 Vyn

Medan det finns flera olika modeller att välja på, så finns det endast en vykomponent att hantera, Den fungerar med antingen listan eller trädlagringen. Att konfigurera en `Gtk.TreeView` är inte svårt. Den behöver en `Gtk.TreeModel` för att veta var den ska erhålla sina data från, antingen genom att skicka det till `Gtk.TreeView`-konstruktorn, eller genom att anropa `Gtk.TreeView.set_model()`.

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

Då komponenten `Gtk.TreeView` har en modell kommer den behöva veta hur den ska visa modellen. Den gör detta med kolumner och cellrenderare. `headers_visible` styr huruvida den visar kolumnrubriker.

Cellrenderare används för att rita data i trädmodellen på ett specifikt sätt. Det finns ett antal cellrenderare som kommer med GTK+, exempelvis `Gtk.CellRendererText`, `Gtk.CellRendererPixbuf` och `Gtk.CellRendererToggle`. Vidare är det relativt enkelt att själv skriva en anpassad renderare genom att skapa en underklass till `Gtk.CellRenderer` och lägga till egenskaper med `GObject.Property()`.

En `Gtk.TreeViewColumn` är objektet som `Gtk.TreeView` använder för att organisera de vertikala kolumnerna i trädvyn och hålla en eller flera cellrenderare. Varje kolumn kan ha en `title` som kommer vara synlig om `Gtk.TreeView` visar kolumnrubriker. Modellen mappas till kolumnen genom att använda nyckelordsargument med egenskaper för renderaren som identifierare och index för modellkolumnerna som argument.

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

Positionsargument kan användas för kolumntiteln och renderaren.

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

För att rendera mer än en modellkolumn i en vykolumn behöver du skapa en `Gtk.TreeViewColumn`-instans och använda `Gtk.TreeViewColumn.pack_start()` för att lägga till modellkolumnerna till den.

```
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 Valet

De flesta program kommer behöva inte bara arbeta med att visa data, utan också ta emot inmatningshändelser från användare. För att göra detta, ta helt enkelt en referens till ett valobjekt och anslut till ”changed”-signalen.

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

För att sedan erhålla data för den valda raden:

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

Du kan styra vilka val som tillåts genom att anropa `Gtk.TreeSelection.set_mode()`. `Gtk.TreeSelection.get_selected()` fungerar inte om valläget är inställt till `Gtk.SelectionMode.MULTIPLE`, använd `Gtk.TreeSelection.get_selected_rows()` istället.

## 13.4 Sortering

Sortering är en viktig funktion för trädvyer och stöds av standardträdmodellerna (`Gtk.TreeStore` och `Gtk.ListStore`), vilka implementerar gränssnittet `Gtk.TreeSortable`.

### 13.4.1 Sortering genom att klicka på kolumner

En kolumn för en `Gtk.TreeView` kan lätt göras sorterbar med ett anrop till `Gtk.TreeViewColumn.set_sort_column_id()`. Efter det kan kolumnen sorteras genom att klicka på dess rubrik.

Först behöver vi en enkel `Gtk.TreeView` och en `Gtk.ListStore` som en modell.

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

(continues on next page)

(fortsättning från föregående sida)

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

treeView = Gtk.TreeView(model=model)

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

Nästa steg är att aktivera sortering. Observera att *column_id* (`0` i exemplet) hänvisar till modellens kolumn och **inte** till kolumnen för vår TreeView.

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

### 13.4.2 Ställa in en anpassad sorteringsfunktion

Det är också möjligt att ställa in en anpassad jämförelsefunktion för att ändra sorteringsbeteendet. Som ett exempel kommer vi skapa en jämförelsefunktion som sorterar skiftlägeskänsligt. I exemplet ovan såg listan ut som:

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

Den skiftlägeskänsliga sorterade listan kommer se ut som:

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

Först av allt behövs en jämförelsefunktion. Denna funktion får två rader och ska returnera ett negativt heltal om den första skulle komma före den andra, noll om de är lika, och ett positivt heltal om den andra skulle komma före den första.

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

Sedan måste sorteringsfunktionen ställas in med `Gtk.TreeSortable.set_sort_func()`.

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

## 13.5 Filtrering

Till skillnad från sortering hanteras filtrering inte av de två modeller vi tidigare såg, utan av klassen `Gtk.TreeModelFilter`. Denna klass är liksom `Gtk.TreeStore` och `Gtk.ListStore` en `Gtk.TreeModel`. Den agerar som ett lager mellan den ”riktiga” modellen (en `Gtk.TreeStore` eller en `Gtk.ListStore`) och döljer några element för vyn. I praktiken tillhandahåller den en delmängd av den underliggande modellen åt `Gtk.TreeView`. Instanser av `Gtk.TreeModelFilter` kan staplas på varandra, för att använda flera filter på samma modell (på samma sätt som du skulle använda ”AND”-klausuler i en SQL-förfrågan). De kan också sättas i en kedja med instanser av `Gtk.TreeModelSort`.

Du kan skapa en ny instans av ett `Gtk.TreeModelFilter` och ge det en modell att filtrera, men det lättaste sättet är att starta det direkt från den filtrerade modellen, med metoden `Gtk.TreeModel.filter_new()`.

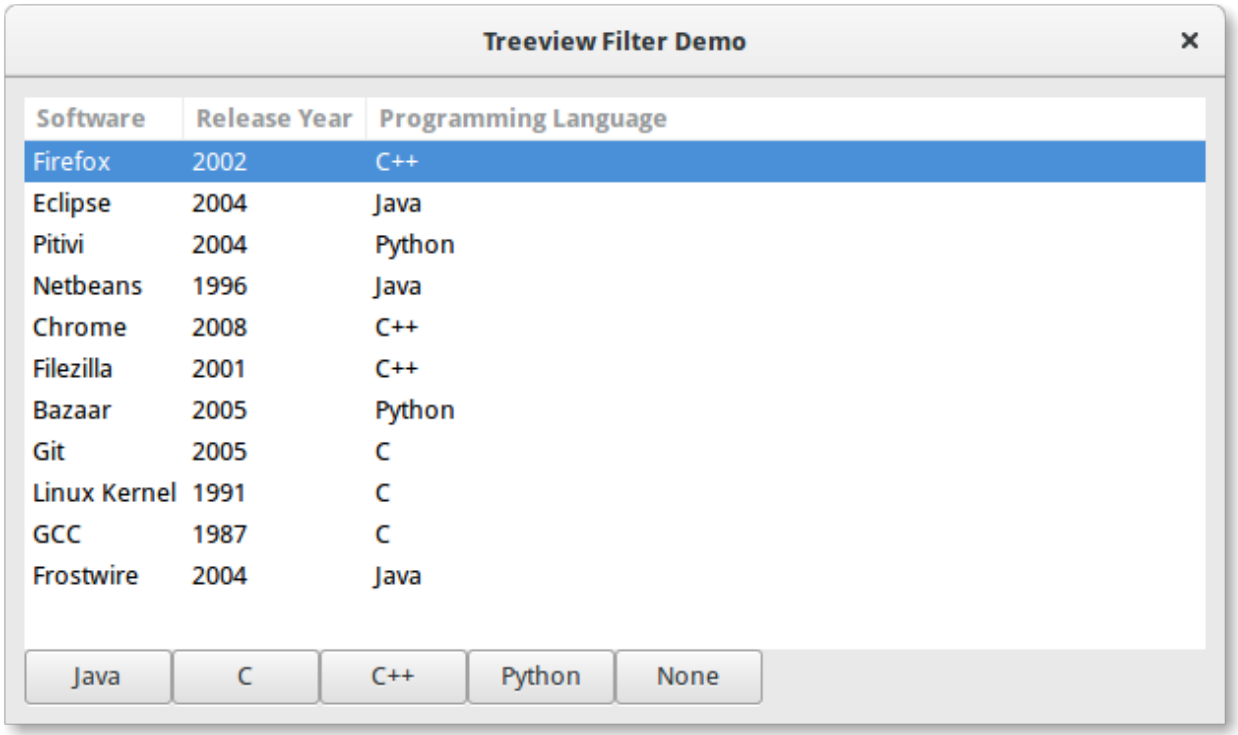
```
filter = model.filter_new()
```

På samma sätt som sorteringsfunktionen fungerar, så använder `Gtk.TreeModelFilter` en ”synlighets”-funktion, som givet en rad från den underliggande modellen, kommer returnera ett booleskt värde som indikerar om denna rad ska filtreras ut eller inte. Den ställs in av `Gtk.TreeModelFilter.set_visible_func()`:

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

Alternativet till en ”synlighets”-funktion är att använda en boolesk kolumn i modellen för att ange vilka rader som ska filtreras. Välj vilken kolumn med `Gtk.TreeModelFilter.set_visible_column()`.

Låt oss se ett fullständigt exempel som använder hela stacken `Gtk.ListStore` - `Gtk.TreeModelFilter` - `Gtk.TreeModelFilter` - `Gtk.TreeView`.



```python
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()
        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()
```

(continues on next page)

(fortsättning från föregående sida)

```
        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
            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()
```

# KAPITEL 14

# CellRenderer-komponenter

`Gtk.CellRenderer`-komponenter används för att visa information i komponenter så som `Gtk.TreeView` eller `Gtk.ComboBox`. De arbetar nära med de associerade komponenterna och är väldigt kraftfulla, med många konfigurationsalternativ för att visa en stor mängd data på olika sätt. Det finns sju `Gtk.CellRenderer`-komponenter som kan användas för olika syften:

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

## 14.1 CellRendererText

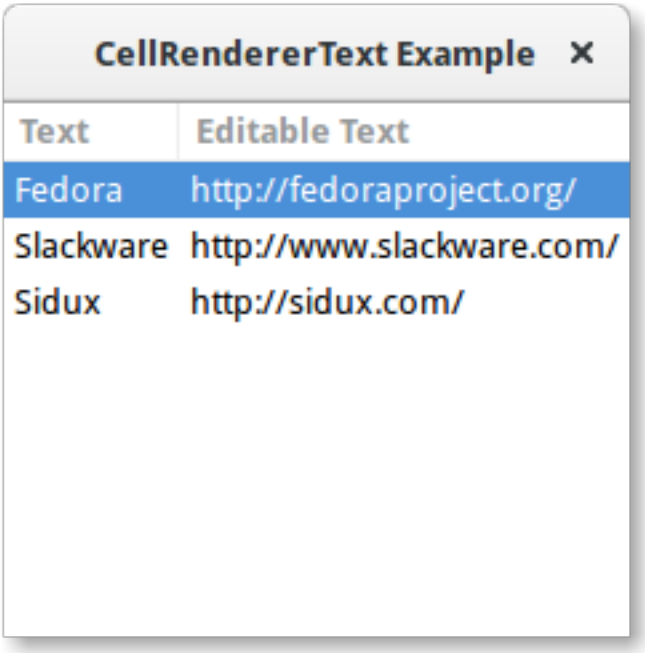
En `Gtk.CellRendererText` renderar en given text i en cell, med det typsnitt, färg och stilinformation som ges av dess egenskaper. Texten kommer elliptiseras om den är för lång och egenskapen ”ellipsize” tillåter det.

Som standard är text i `Gtk.CellRendererText`-komponenter inte redigerbar. Detta kan ändras genom att ställa in värdet för egenskapen ”editable” till `True`:

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

Du kan sedan ansluta till ”edited”-signalen och uppdatera din `Gtk.TreeModel` enligt det.

### 14.1.1 Exempel




```
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)

```

(continues on next page)

(fortsättning från föregående sida)

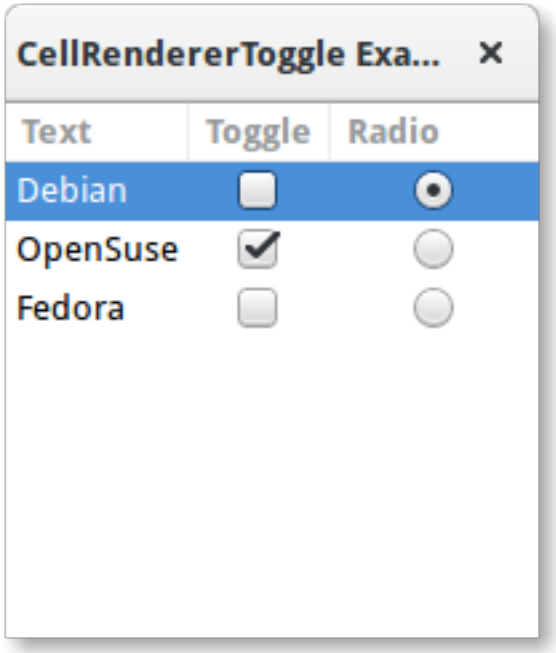
```
        renderer_editabletext.connect("edited", self.text_edited)

        self.add(treeview)

    def text_edited(self, widget, path, text):
        self.liststore[path][1] = text


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

## 14.2 CellRendererToggle

`Gtk.CellRendererToggle` renderar en växlingsknapp i en cell. Knappen ritas som en radioknapp eller en kryssruta, beroende på egenskapen ”radio”. Då den aktiveras sänder den signalen ”toggled”.

Då en `Gtk.CellRendererToggle` kan ha två tillstånd, aktiv och inte aktiv, så kommer du mest troligt vilja binda egenskapen ”active” på cellrenderaren till ett booleskt värde i modellen, och därigenom få kryssrutan att motsvara tillståndet för modellen.

### 14.2.1 Exempel




```
import gi

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

(continues on next page)

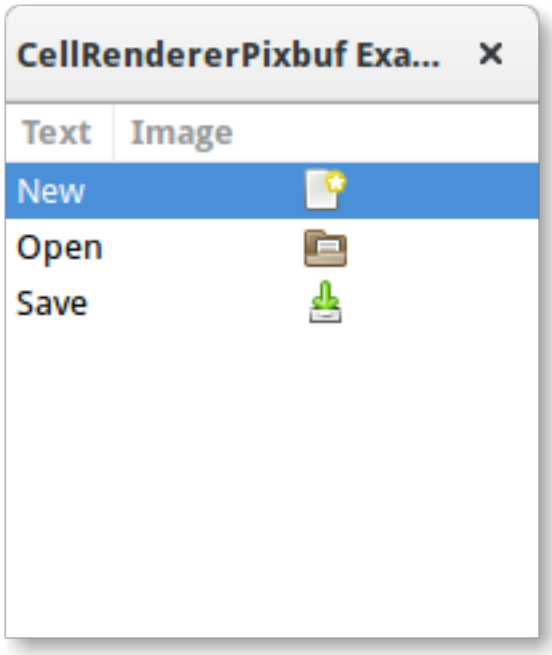
(fortsättning från föregående sida)

```
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)
12
13         self.liststore = Gtk.ListStore(str, bool, bool)
14         self.liststore.append(["Debian", False, True])
15         self.liststore.append(["OpenSuse", True, False])
16         self.liststore.append(["Fedora", False, False])
17
18         treeview = Gtk.TreeView(model=self.liststore)
19
20         renderer_text = Gtk.CellRendererText()
21         column_text = Gtk.TreeViewColumn("Text", renderer_text, text=0)
22         treeview.append_column(column_text)
23
24         renderer_toggle = Gtk.CellRendererToggle()
25         renderer_toggle.connect("toggled", self.on_cell_toggled)
26
27         column_toggle = Gtk.TreeViewColumn("Toggle", renderer_toggle, active=1)
28         treeview.append_column(column_toggle)
29
30         renderer_radio = Gtk.CellRendererToggle()
31         renderer_radio.set_radio(True)
32         renderer_radio.connect("toggled", self.on_cell_radio_toggled)
33
34         column_radio = Gtk.TreeViewColumn("Radio", renderer_radio, active=2)
35         treeview.append_column(column_radio)
36
37         self.add(treeview)
38
39     def on_cell_toggled(self, widget, path):
40         self.liststore[path][1] = not self.liststore[path][1]
41
42     def on_cell_radio_toggled(self, widget, path):
43         selected_path = Gtk.TreePath(path)
44         for row in self.liststore:
45             row[2] = row.path == selected_path
46
47
48 win = CellRendererToggleWindow()
49 win.connect("destroy", Gtk.main_quit)
50 win.show_all()
51 Gtk.main()
```

## 14.3 CellRendererPixbuf

En `Gtk.CellRendererPixbuf` kan användas för att rendera en bild i en cell. Den tillåter att rendera antingen en given `Gdk.Pixbuf` (satt via egenskapen ”pixbuf”) eller en namngiven ikon (satt via egenskapen ”icon-name”).

### 14.3.1 Exempel




```
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()
```

(continues on next page)

(fortsättning från föregående sida)

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

        self.add(treeview)


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

## 14.4 CellRendererCombo

`Gtk.CellRendererCombo` renderar text i en cell som `Gtk.CellRendererText` från vilken den härletts. Men medan den senare erbjuder ett enkelt inmatningsfält för att redigera texten så erbjuder `Gtk.CellRendererCombo` en `Gtk.ComboBox`-komponent för att redigera texten. Värdena att visa i kombinationsrutan tas från den `Gtk.TreeModel` som anges i egenskapen ”model”.

Kombinationscellrenderaren tar hand om att lägga till en textcellrenderare till kombinationsrutan och ställer in den att visa kolumnen som anges av dess egenskap ”text-column”.

En `Gtk.CellRendererCombo` kan arbeta i två lägen. Den kan användas med eller utan en associerad `Gtk.Entry`-komponent, beroende på värdet på egenskapen ”has-entry”.

### 14.4.1 Exempel


```
import gi

```

(continues on next page)

(fortsättning från föregående sida)

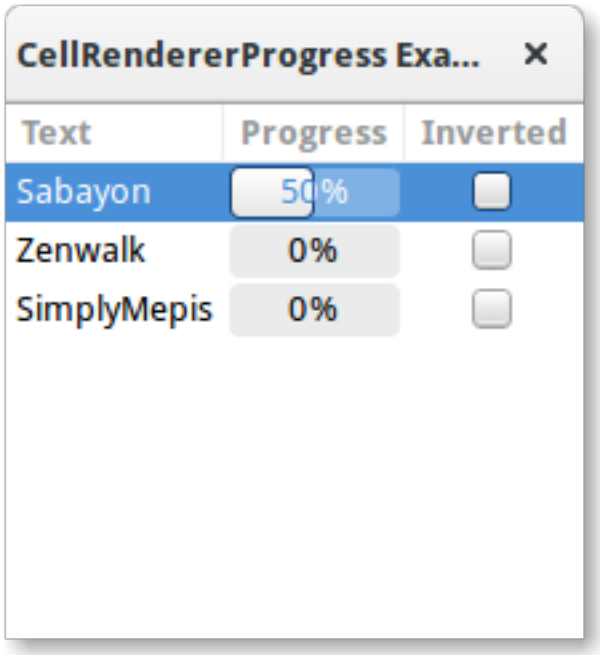
```
gi.require_version("Gtk", "3.0")
from gi.repository import Gtk


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

        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` renderar ett numeriskt värde som en förloppsindikator i en cell. Vidare kan den visa en text ovanpå förloppsindikatorn.

Procentvärdet för förloppsindikatorn kan ändras genom att ändra egenskapen ”value”. Liknande `Gtk.ProgressBar` så kan du aktivera *aktivitetsläget* genom att öka ”pulse”-egenskapen istället för ”value”-egenskapen.

### 14.5.1 Exempel




```
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)
```

(continues on next page)

(fortsättning från föregående sida)

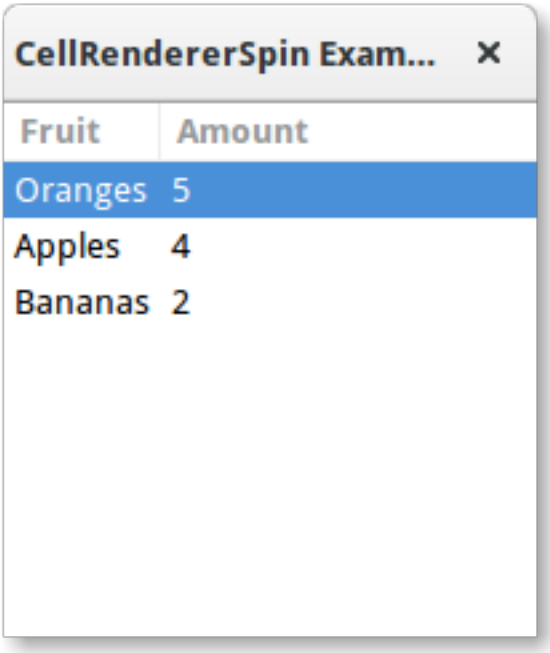
```
        renderer_progress = Gtk.CellRendererProgress()
        column_progress = Gtk.TreeViewColumn(
            "Progress", renderer_progress, value=1, inverted=2
        )
        treeview.append_column(column_progress)

        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)

        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` renderar text i en cell som `Gtk.CellRendererText` från vilken den härletts. Men medan den senare erbjuder ett enkelt inmatningsfält för att redigera texten så erbjuder `Gtk.CellRendererSpin` en `Gtk.SpinButton`-komponent. Detta betyder förstås att texten måste gå att tolka som ett flyttal.

Intervallet för stegningsrutan tas från justeringsegenskapen för cellrenderaren, vilken kan ställas in explicit eller mappas till en kolumn i trädmodellen, som alla egenskaper för cellrenderare. `Gtk.CellRendererSpin` har också egenskaper för steghöjden och antalet siffror att visa.

### 14.6.1 Exempel




```
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,
```

(continues on next page)

(fortsättning från föregående sida)

```
            step_increment=1,
            page_increment=10,
            page_size=0,
        )
        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()
```

# KAPITEL 15

## ComboBox

En `Gtk.ComboBox` möjliggör val av ett objekt från en rullgardinsmeny. De föredras över att ha många radioknappar på skärmen då de tar upp mindre utrymme. Om lämpligt kan den visa extra information om varje objekt, så som text, en bild, en kryssruta eller en förloppsindikator.

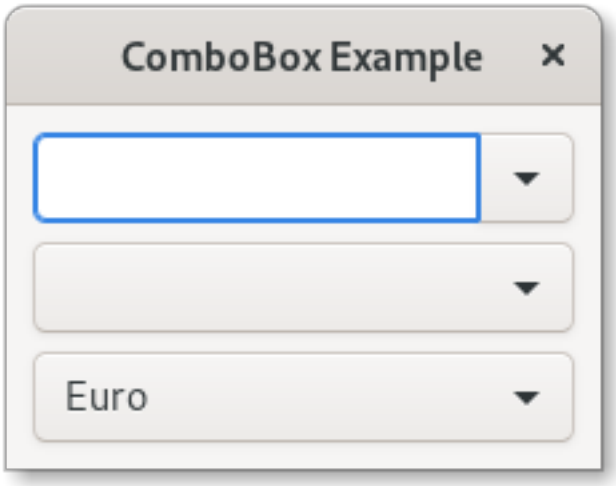
`Gtk.ComboBox` är väldigt lik `Gtk.TreeView`, då de båda använder model-view-mönstret; listan över giltiga val anges i formen av en trädmodell, och visningen av valen kan anpassas till data i modellen genom att använda *cellrenderare*. Om kombinationsrutan innehåller ett stort antal objekt kan det vara bättre att visa dem i ett rutnät snarare än i en lista. Detta kan göras genom att anropa `Gtk.ComboBox.set_wrap_width()`.

Ett standardvärde kan ställas in genom att anropa `Gtk.ComboBox.set_active()` med index för det önskade värdet.

`Gtk.ComboBox`-komponenten begränsar vanligen användaren till de tillgängliga valen, men den kan valfritt ha ett `Gtk.Entry`, vilket låter användaren ange godtycklig text om inget av de tillgängliga valen är lämpligt. För att göra detta, använd en av de statiska metoderna `Gtk.ComboBox.new_with_entry()` eller `Gtk.ComboBox.new_with_model_and_entry()` för att skapa en `Gtk.ComboBox`-instans.

För en enkel lista över textval kan model-view-API:t för `Gtk.ComboBox` vara lite överväldigande. I detta fall är `Gtk.ComboBoxText` ett enkelt alternativ. Både `Gtk.ComboBox` och `Gtk.ComboBoxText` kan innehålla ett inmatningsfält.

## 15.1 Exempel




```
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",
```

(continues on next page)

(fortsättning från föregående sida)

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


```

(continues on next page)

(fortsättning från föregående sida)

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

# KAPITEL 16

# IconView

En `Gtk.IconView` är en komponent som visar en samling ikoner i en rutnätsvy. Den stöder funktioner så som dra-och-släpp, flerval och att ändra ordning på objekt.

Liknande `Gtk.TreeView` använder `Gtk.IconView` en `Gtk.ListStore` för sin modell. Istället för att använda *cellrenderare*, kräver `Gtk.IconView` att en av kolumnerna i dess `Gtk.ListStore` innehåller `GdkPixbuf.Pixbuf`-objekt.

`Gtk.IconView` stöder flera markeringslägen för att tillåta antingen markering av flera ikoner samtidigt, begränsa markeringar till bara ett objekt eller helt förbjuda markering av objekt. För att ange ett markeringsläge används metoden `Gtk.IconView.set_selection_mode()` med ett av markeringslägena i `Gtk.SelectionMode`.

## 16.1 Exempel


```
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)
```

(continues on next page)

(fortsättning från föregående sida)

```


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

KAPITEL 17

# Textredigerare med flera rader

Komponenten `Gtk.TextView` kan användas för att visa och redigera stora mängder av formaterad text. Som `Gtk.TreeView` har den en model/view-design. I detta fall är `Gtk.TextBuffer` modellen som representerar texten som redigeras. Detta låter två eller fler `Gtk.TextView`-komponenter dela på samma `Gtk.TextBuffer`, och låter dessa textbuffrar visas något annorlunda. Eller så kan du underhålla flera textbuffrar och välja att visa var och en vid olika tider i samma `Gtk.TextView`-komponent.

## 17.1 Vyn

`Gtk.TextView` är framänden med vilken användaren kan lägga till, redigera och ta bort textdata. De används vanligen för att redigera flera rader av text. Då du skapar en `Gtk.TextView` innehåller den sin egen standard-`Gtk.TextBuffer`, vilken du kan komma åt via metoden `Gtk.TextView.get_buffer()`.

Som standard kan text läggas till, redigeras och tas bort från `Gtk.TextView`. Du kan inaktivera detta genom att anropa `Gtk.TextView.set_editable()`. Om texten inte är redigerbar vill du vanligen även dölja textmarkören med `Gtk.TextView.set_cursor_visible()`. I vissa fall kan det vara användbart att ställa in justeringen av texten med `Gtk.TextView.set_justification()`. Texten kan visas i vänstra kanten, (`Gtk.Justification.LEFT`), i högra kanten (`Gtk.Justification.RIGHT`), centrerad (`Gtk.Justification.CENTER`) eller jämnt fördelad över hela bredden (`Gtk.Justification.FILL`).

En annan standardinställning för `Gtk.TextView`-komponenten är att långa textrader kommer fortsätta horisontellt till ett avbrott matas in. För att radbryta texten och förhindra den från att försvinna utanför skärmens kanter, anropa `Gtk.TextView.set_wrap_mode()`.

## 17.2 Modellen

`Gtk.TextBuffer` är kärnan av `Gtk.TextView`-komponenten, och används för att hålla vadhelst för text som visas i `Gtk.TextView`. Att ställa in och erhålla innehållet är möjligt med `Gtk.TextBuffer.set_text()` och `Gtk.TextBuffer.get_text()`. Den mesta textmanipuleringen utförs dock med *iteratorer*, representerade av en `Gtk.TextIter`. En iterator representerar en position mellan två tecken i textbufferten. Iteratorer är inte giltiga för alltid, närhelst bufferten ändras på ett sätt som påverkar buffertens innehåll så blir alla tidigare iteratorer ogiltiga.

På grund av detta kan iteratorer inte användas för att bevara positioner över buffertändringar. För att bevara en position, använd `Gtk.TextMark`. En textbuffert innehåller två inbyggda märken; ett ”insert”-märke (som är markörens position) och märket ”selection_bound”. Båda av dem kan erhållas med `Gtk.TextBuffer.get_insert()` respektive `Gtk.TextBuffer.get_selection_bound()`. Som standard visas inte platsen för en `Gtk.TextMark`. Detta kan ändras genom att anropa `Gtk.TextMark.set_visible()`.

Många metoder finns för att erhålla en `Gtk.TextIter`. Exempelvis returnerar `Gtk.TextBuffer.get_start_iter()` en iterator som pekar på den första positionen i textbufferten, medan `Gtk.TextBuffer.get_end_iter()` returnerar en iterator som pekar bortom det sista giltiga tecknet. Att erhålla gränserna för den markerade texten kan åstadkommas genom att anropa `Gtk.TextBuffer.get_selection_bounds()`.

För att infoga text på en specifik position, använd `Gtk.TextBuffer.insert()`. En annan användbar metod är `Gtk.TextBuffer.insert_at_cursor()` som infogar text varhelst markören är placerad för tillfället. Använd `Gtk.TextBuffer.delete()` för att ta bort delar av textbufferten.

Dessutom kan `Gtk.TextIter` användas för att hitta sökträffar på text i bufferten med `Gtk.TextIter.forward_search()` och `Gtk.TextIter.backward_search()`. Start- och slutiteratorerna används som startpunkten för sökningen och går framåt/bakåt beroende på kraven.

## 17.3 Taggar

Text i en buffert kan markeras med taggar. En tagg är ett attribut som kan tillämpas på något textintervall. Exempelvis kan en tagg vara kallad ”bold” och få texten i taggen att vara i fetstil. Taggkonceptet är dock mer allmänt än så; taggar behöver inte påverka utseende. De kan istället påverka beteendet för mus- och tangentryck, ”låsa” ett textintervall så att användaren inte kan redigera det samt många andra saker. En tagg representeras av ett `Gtk.TextTag`-objekt. En `Gtk.TextTag` kan tillämpas på ett valfritt antal textintervall i ett valfritt antal buffertar.

Varje tagg lagras i en `Gtk.TextTagTable`. En taggtabell definierar en uppsättning taggar som kan användas tillsammans. Varje buffert har en taggtabell associerad med sig; endast taggar från den taggtabellen kan användas med bufferten. En enda taggtabell kan dock delas mellan flera buffertar.

För att ange att någon text i bufferten ska ha specifik formatering så måste du definiera en tagg för att hålla den formateringsinformationen, och sedan tillämpa den taggen på textregionen med `Gtk.TextBuffer.create_tag()` och `Gtk.TextBuffer.apply_tag()`:

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

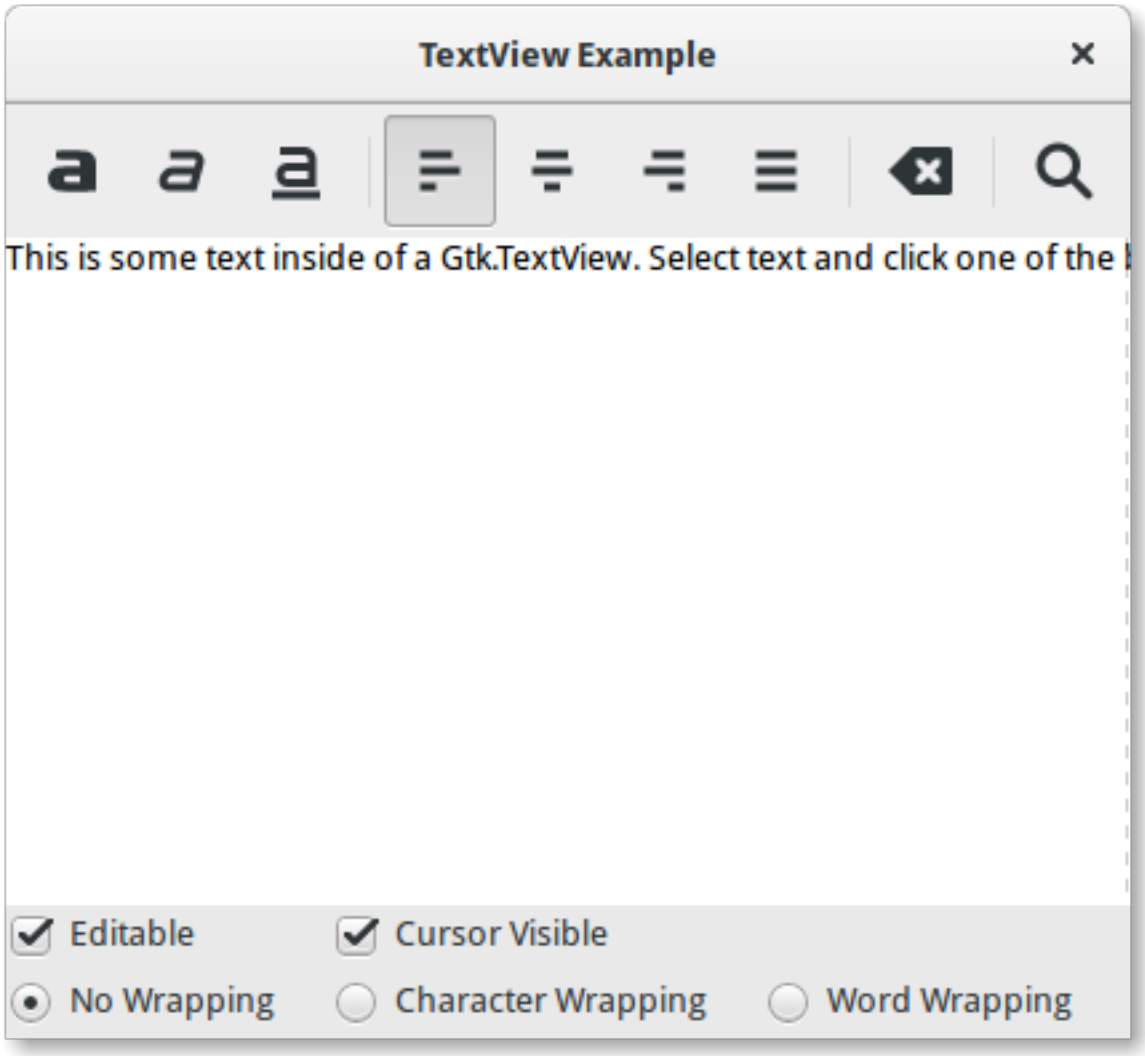
Följande är några av de vanliga stilarna som tillämpas på text:

- Bakgrundsfärg (egenskapen ”background”)
- Förgrundsfärg (egenskapen ”foreground”)
- Understruken (egenskapen ”underline”)
- Fet (egenskapen ”weight”)
- Kursiv (egenskapen ”style”)

- Genomstruken (egenskapen ”strikethrough”)
- Justering (egenskapen ”justification”)
- Storlek (egenskaperna ”size” och ”size-points”)
- Radbrytning av text (egenskapen ”wrap-mode”)

Du kan också ta bort specifika taggar senare med `Gtk.TextBuffer.remove_tag()` eller ta bort alla taggar i en angiven region genom att anropa `Gtk.TextBuffer.remove_all_tags()`.

## 17.4 Exempel




```
import gi

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


```

(continues on next page)

(fortsättning från föregående sida)

```
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,
            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)

```

(continues on next page)

(fortsättning från föregående sida)

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

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

(continues on next page)

(fortsättning från föregående sida)

```
        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()
        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())
```

(continues on next page)

(fortsättning från föregående sida)

```
    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()
```

KAPITEL 18

# Dialoger

Dialogfönster liknar väldigt mycket standardfönster, och används för att tillhandahålla eller erhålla information från användaren. De används ofta för att exempelvis tillhandahålla ett inställningsfönster. Den stora skillnaden som en dialog har är några i förväg packade komponenter som automatiskt gör en layout för dialogen. Därifrån kan vi enkelt lägga till etiketter, knappar, kryssrutor o.s.v. En annan stor skillnad är hanteringen av svar för att styra hur programmet ska bete sig efter interaktion med dialogen.

Det finns flera härledda Dialog-klasser som du kan finna användbara. `Gtk.MessageDialog` används för de flesta enkla aviseringar. Men vid andra tillfällen kan du behöva härleda din egen dialogklass för att tillhandahålla mer komplex funktionalitet.

## 18.1 Anpassade dialoger

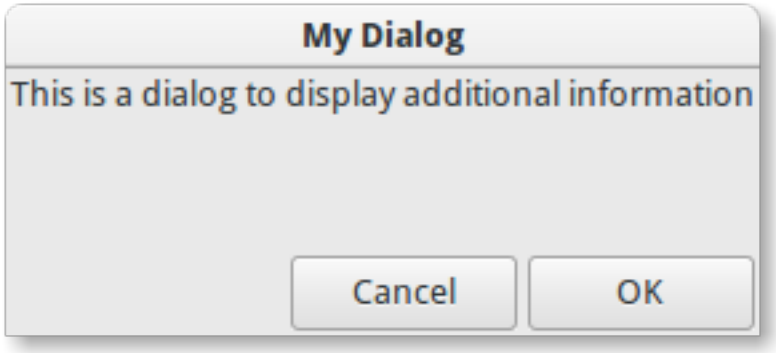
För att packa komponenter i en anpassad dialog bör du packa dem i din `Gtk.Box`, tillgänglig via `Gtk.Dialog.get_content_area()`. För att bara lägga till en `Gtk.Button` längst ner i dialogen kan du använda metoden `Gtk.Dialog.add_button()`.

En ”modal” dialog (det vill säga en som fryser resten av programmet från användarinmatning) kan skapas genom att anropa `Gtk.Dialog.set_modal` på dialogen eller ställa in argumentet `flags` för konstruktorn `Gtk.Dialog` att inkludera flaggan `Gtk.DialogFlags.MODAL`.

Att klicka på en knapp kommer sända ut en signal kallad ”response”. Om du vill blockera och vänta på att en dialog ska returnera innan du returnerar kontrollflödet till din kod kan du anropa `Gtk.Dialog.run()`. Denna metod returnerar en int som kan vara ett värde från `Gtk.ResponseType` eller så kan den vara det anpassade svarsvärdet som du angav i `Gtk.Dialog`-konstruktorn eller `Gtk.Dialog.add_button()`.

Slutligen finns det två sätt att ta bort en dialog. Metoden `Gtk.Widget.hide()` får dialogen att sluta visas, men behåller den i minnet. Detta är användbart för att slippa behöva konstruera dialogen igen om den behöver kommas åt vid ett senare tillfälle. Alternativt kan metoden `Gtk.Widget.destroy()` användas för att ta bort dialogen från minnet då den inte längre behövs. Det bör noteras att om dialogen behöver kommas åt efter att den förstörts så kommer den behöva konstrueras igen, annars kommer dialogfönstret vara tomt.

### 18.1.1 Exempel




```
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")
```

(continues on next page)

(fortsättning från föregående sida)

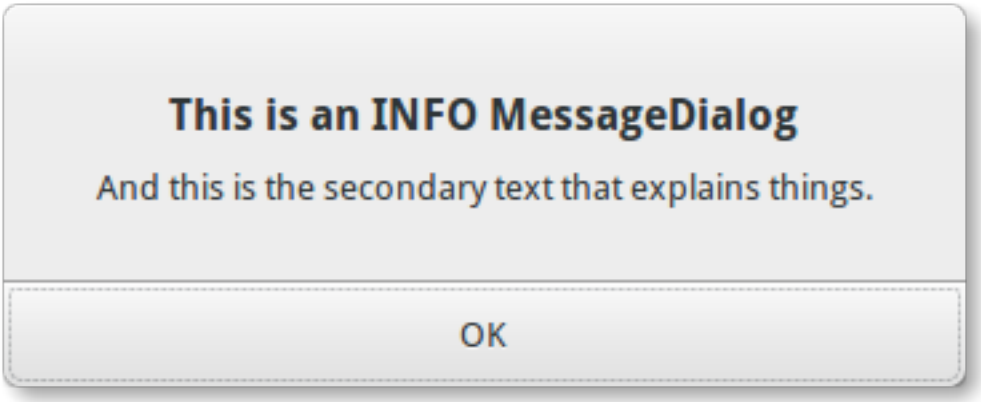
```
        dialog.destroy()


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

## 18.2 MessageDialog

`Gtk.MessageDialog` är en bekvämlighetsklass, använd för att skapa enkla standardmeddelandedialoger, med ett meddelande, en ikon och knappar för användarsvar. Du kan ange meddelandetypen och texten i `Gtk.MessageDialog`-konstruktorn, såväl som att ange standardknappar.

I några dialoger som kräver vidare förklaring av vad som har hänt kan en sekundär text läggas till. I detta fall görs det primära meddelandet som matats in då meddelandedialogen skapades större och ställs in till fetstil. Det sekundära meddelandet kan ställas in genom att anropa `Gtk.MessageDialog.format_secondary_text()`.

### 18.2.1 Exempel




```
import gi

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


class MessageDialogWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="MessageDialog Example")

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

        button1 = Gtk.Button(label="Information")
        button1.connect("clicked", self.on_info_clicked)
        box.add(button1)
```

(continues on next page)

(fortsättning från föregående sida)

```
        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",
        )
        dialog.format_secondary_text(
            "And this is the secondary text that explains things."
        )
        response = dialog.run()
```

(continues on next page)

(fortsättning från föregående sida)

```
        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

`Gtk.FileChooserDialog` är lämplig för användning med menyobjekten ”Arkiv/Öppna” eller ”Arkiv/Spara”. Du kan använda alla metoder för `Gtk.FileChooser` på filväljardialogen såväl som de för `Gtk.Dialog`.

När du skapar en `Gtk.FileChooserDialog` måste du definiera dialogens syfte:

- För att välja en fil att öppna, som för ett Arkiv/Öppna-kommando, använd `Gtk.FileChooserAction.OPEN`
- För att spara en fil för första gången, som för ett Arkiv/Spara-kommando, använd `Gtk.FileChooserAction.SAVE`, och föreslå ett namn som ”Namnlös” med `Gtk.FileChooser.set_current_name()`.
- För att spara en fil under ett annat namn, som för ett Arkiv/Spara som-kommando, använd `Gtk.FileChooserAction.SAVE`, och ställ in det befintliga filnamnet med `Gtk.FileChooser.set_filename()`.
- För att välja en mapp istället för en fil, använd `Gtk.FileChooserAction.SELECT_FOLDER`.

`Gtk.FileChooserDialog` ärver från `Gtk.Dialog`, så knappar har svars-ID:n så som `Gtk.ResponseType.ACCEPT` och `Gtk.ResponseType.CANCEL` vilka kan anges i konstruktorn för `Gtk.FileChooserDialog`. I kontrast till `Gtk.Dialog` så kan du inte använda anpassade svarskoder med `Gtk.FileChooserDialog`. Den förväntar sig att åtminstone en knapp kommer ha följande svars-ID:n:

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

När användaren är klar med att välja filer kan ditt program få de valda namnen antingen som filnamn (`Gtk.FileChooser.get_filename()`) eller som URI:er (`Gtk.FileChooser.get_uri()`).

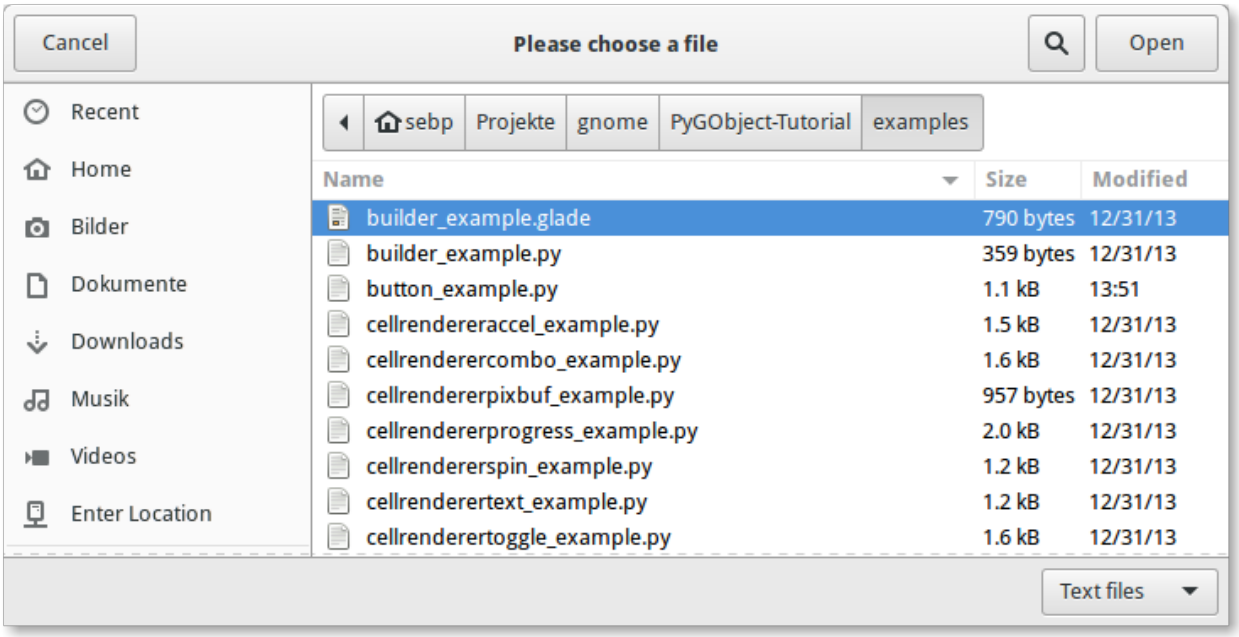
Som standard tillåter `Gtk.FileChooser` endast att en fil väljs åt gången. För att aktivera val av flera filer, använd `Gtk.FileChooser.set_select_multiple()`. Att erhålla en lista över valda filer är möjligt med antingen `Gtk.FileChooser.get_filenames()` eller `Gtk.FileChooser.get_uris()`.

`Gtk.FileChooser` stöder också ett antal alternativ som gör filerna och mapparna mer konfigurerbara och åtkomliga.

- `Gtk.FileChooser.set_local_only()`: Endast lokala filer kan väljas.
- `Gtk.FileChooser.show_hidden()`: Dolda filer och mappar visas.
- `Gtk.FileChooser.set_do_overwrite_confirmation()`: Om filväljaren konfigurerades i läget `Gtk.FileChooserAction.SAVE`, kommer den presentera en bekräftelsedialog om användaren skriver in ett filnamn som redan finns.

Vidare kan du ange vilka typer av filer som visas genom att skapa `Gtk.FileFilter`-objekt och anropa `Gtk.FileChooser.add_filter()`. Användaren kan sedan välja ett av de tillagda filtren från en kombinationsruta längst ner i filväljaren.

### 18.3.1 Exempel




```
import gi

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

(continues on next page)

(fortsättning från föregående sida)

```
5
6
7  class FileChooserWindow(Gtk.Window):
8      def __init__(self):
9          super().__init__(title="FileChooser Example")
10
11         box = Gtk.Box(spacing=6)
12         self.add(box)
13
14         button1 = Gtk.Button(label="Choose File")
15         button1.connect("clicked", self.on_file_clicked)
16         box.add(button1)
17
18         button2 = Gtk.Button(label="Choose Folder")
19         button2.connect("clicked", self.on_folder_clicked)
20         box.add(button2)
21
22     def on_file_clicked(self, widget):
23         dialog = Gtk.FileChooserDialog(
24             title="Please choose a file", parent=self, action=Gtk.FileChooserAction.
↪OPEN
25         )
26         dialog.add_buttons(
27             Gtk.STOCK_CANCEL,
28             Gtk.ResponseType.CANCEL,
29             Gtk.STOCK_OPEN,
30             Gtk.ResponseType.OK,
31         )
32
33         self.add_filters(dialog)
34
35         response = dialog.run()
36         if response == Gtk.ResponseType.OK:
37             print("Open clicked")
38             print("File selected: " + dialog.get_filename())
39         elif response == Gtk.ResponseType.CANCEL:
40             print("Cancel clicked")
41
42         dialog.destroy()
43
44     def add_filters(self, dialog):
45         filter_text = Gtk.FileFilter()
46         filter_text.set_name("Text files")
47         filter_text.add_mime_type("text/plain")
48         dialog.add_filter(filter_text)
49
50         filter_py = Gtk.FileFilter()
51         filter_py.set_name("Python files")
52         filter_py.add_mime_type("text/x-python")
53         dialog.add_filter(filter_py)
54
55         filter_any = Gtk.FileFilter()
56         filter_any.set_name("Any files")
57         filter_any.add_pattern("*")
58         dialog.add_filter(filter_any)
59
60     def on_folder_clicked(self, widget):
```

(continues on next page)

(fortsättning från föregående sida)

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

KAPITEL 19

# Kontextfönster

`Gtk.Popover` är ett separat fönster som används för att visa ytterligare information och ofta används som en del av knappmenyer och snabbvalsmenyer. Kontextfönster är visuellt anslutna till en relaterad komponent med en liten triangel. Deras användning liknar de hos dialogfönster med fördelen att de är mindre störande och har en anslutning till komponenten som kontextfönstret pekar på.

Ett kontextfönster kan skapas med `Gtk.Popover`; använd `Gtk.Popover.popup()` för att öppna kontextfönstret.

## 19.1 Anpassat kontextfönster

En komponent kan läggas till ett kontextfönster med `Gtk.Container.add()`.

### 19.1.1 Exempel

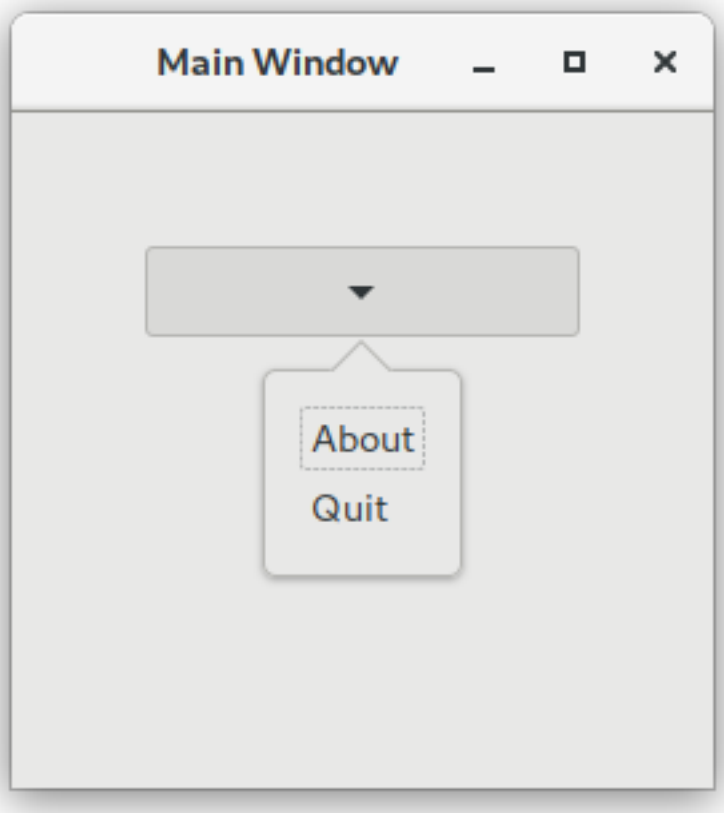

```
import gi

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 Menykontextfönster

Ett kontextfönster kan skapas från en `Gio.MenuModel` med `Gtk.Popover.new_from_model()` och kan ändras efter att det skapats med `Gtk.Popover.bind_model()`.

Att skicka en `Gio.MenuModel` som ett `menu_model`-argument till `Gtk.MenuButton`-konstruktorn skapar implicit ett kontextfönster.

### 19.2.1 Exempel




```
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>
        <item>
```

(continues on next page)

(fortsättning från föregående sida)

```
            <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:
            # 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()
```

(continues on next page)

(fortsättning från föregående sida)

```
    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 Se även

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

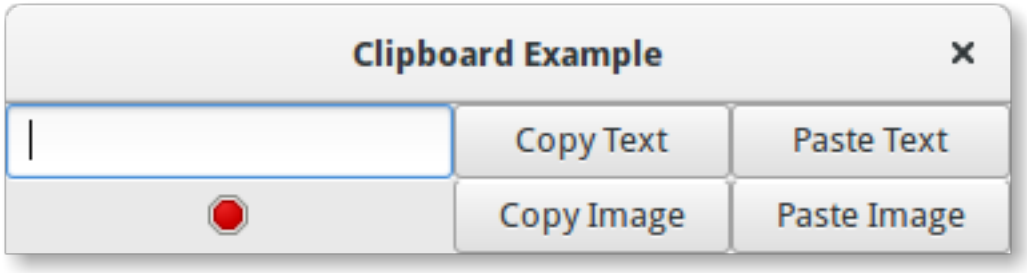
KAPITEL 20

# Clipboard

Gtk.Clipboard tillhandahåller ett lagringsutrymme för en mängd data, inklusive text och bilder. Att använda urklipp låter dessa data delas mellan program genom åtgärder så som kopiering, utklippning och inklistring. Dessa åtgärder utförs vanligen på tre sätt: med tangentbordsgenvägar, med ett Gtk.MenuItem och att ansluta funktionerna till Gtk.Button-komponenter.

Det finns flera urklippsmarkeringar för olika syften. I de flesta omständigheter används markeringen med namn CLIPBOARD för vardaglig kopiering och inklistring. PRIMARY är en annan vanlig markering som lagrar text som markerats av användaren med markören.

## 20.1 Exempel




```
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")

```

(continues on next page)

(fortsättning från föregående sida)

```
        grid = Gtk.Grid()

        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()
```

KAPITEL 21

# Dra-och-släpp

**Observera:** Versioner av PyGObject < 3.0.3 innehåller ett fel som inte låter dra-och-släpp fungera korrekt. Därför krävs en version av PyGObject >= 3.0.3 för att följande exempel ska fungera.

Att konfigurera dra-och-släpp mellan komponenter består i att välja en dragkälla (komponenten som användaren startar dragningen från) med metoden `Gtk.Widget.drag_source_set()`, välja ett dragmål (komponenten som användaren släpper på) med metoden `Gtk.Widget.drag_dest_set()` och sedan hantera de relevanta signalerna för båda komponenterna.

Istället för att använda `Gtk.Widget.drag_source_set()` och `Gtk.Widget.drag_dest_set()` så kräver vissa specialiserade komponenter användningen av specifika funktioner (så som `Gtk.TreeView` och `Gtk.IconView`).

Ett enkelt dra-och-släpp kräver bara att källan ansluter till signalen ”drag-data-get” och att målet ansluter till signalen ”drag-data-received”. Mer komplexa saker som specifika släppområden och anpassade dragikoner kommer kräva att du ansluter till *ytterligare signaler* och interagerar med `Gdk.DragContext`-objektet det tillhandahåller.

För att överföra data mellan källan och målet behöver du interagera med variabeln `Gtk.SelectionData` som tillhandahålls i signalerna *”drag-data-get”* och *”drag-data-received”* med get- och set-metoderna för `Gtk.SelectionData`.

## 21.1 Målfält

För att dragningens källa och mål ska få veta vilka data de tar emot och skickar krävs en gemensam lista över `Gtk.TargetEntry`. Ett `Gtk.TargetEntry` beskriver ett datastycke som kommer sändas av dragkällan och tas emot av dragmålet.

Det finns två sätt att lägga till `Gtk.TargetEntry` till en källa och ett mål. Om dra-och-släppet är enkelt och varje målfält är av olika typ så kan du använda gruppen av metoder `omnämnda här`.

Om du behöver mer än en typ av data eller vill göra mer komplexa saker med dessa data kommer du behöva skapa ditt `Gtk.TargetEntry` med metoden `Gtk.TargetEntry.new()`.

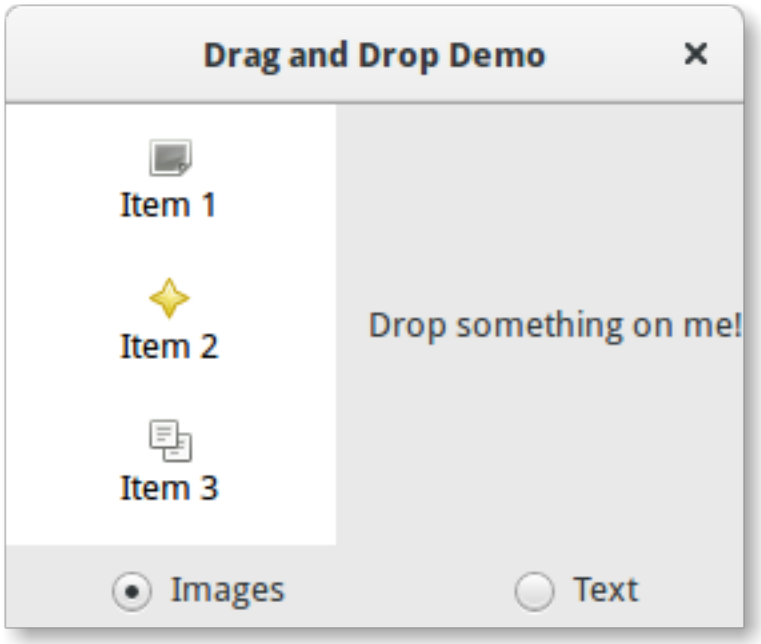
## 21.2 Dragkällsignaler

| Namn | När den sänds ut | Vanligt syfte |
|---|---|---|
| drag-begin | Användaren startar en dragning | Konfigurera dragikon |
| drag-data-get | Då dragningsdata begärs av målet | Överför dragningsdata från källan till målet |
| drag-data-delete | Då en dragning med åtgärden Gdk.DragAction.MOVE slutförts | Ta bort data från källan för att slutföra ”flytten” |
| drag-end | Då dragningen slutförts | Gör allt som gjorts i drag-begin ogjort |

## 21.3 Dragmålsignaler

| Namn | När den sänds ut | Vanligt syfte |
|---|---|---|
| drag-motion | Dragikon flyttar över ett släppområde | Tillåt endast vissa områden att släppas till |
| drag-drop | Ikon släpps på ett dragområde | Tillåt endast vissa områden att släppas till |
| drag-data-received | När dragningsdata tas emot av målet | Överför dragningsdata från källan till målet |

## 21.4 Exempel




```
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)
```

(continues on next page)

(fortsättning från föregående sida)

```
DRAG_ACTION = Gdk.DragAction.COPY


class DragDropWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="Drag and Drop Demo")

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

        hbox = Gtk.Box(spacing=12)
        vbox.pack_start(hbox, True, True, 0)

        self.iconview = DragSourceIconView()
        self.drop_area = DropArea()

        hbox.pack_start(self.iconview, True, True, 0)
        hbox.pack_start(self.drop_area, True, True, 0)

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

        image_button = Gtk.RadioButton.new_with_label_from_widget(None, "Images")
        image_button.connect("toggled", self.add_image_targets)
        button_box.pack_start(image_button, True, False, 0)

        text_button = Gtk.RadioButton.new_with_label_from_widget(image_button, "Text")
        text_button.connect("toggled", self.add_text_targets)
        button_box.pack_start(text_button, True, False, 0)

        self.add_image_targets()

    def add_image_targets(self, button=None):
        targets = Gtk.TargetList.new([])
        targets.add_image_targets(TARGET_ENTRY_PIXBUF, True)

        self.drop_area.drag_dest_set_target_list(targets)
        self.iconview.drag_source_set_target_list(targets)

    def add_text_targets(self, button=None):
        self.drop_area.drag_dest_set_target_list(None)
        self.iconview.drag_source_set_target_list(None)

        self.drop_area.drag_dest_add_text_targets()
        self.iconview.drag_source_add_text_targets()


class DragSourceIconView(Gtk.IconView):
    def __init__(self):
        Gtk.IconView.__init__(self)
        self.set_text_column(COLUMN_TEXT)
        self.set_pixbuf_column(COLUMN_PIXBUF)

        model = Gtk.ListStore(str, GdkPixbuf.Pixbuf)
        self.set_model(model)
        self.add_item("Item 1", "image-missing")
```

(continues on next page)

(fortsättning från föregående sida)

```
        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()
```

KAPITEL 22

# Glade och Gtk.Builder

Klassen `Gtk.Builder` ger dig möjligheten att designa användargränssnitt utan att skriva en enda rad kod. Detta är möjligt genom att beskriva gränssnittet i en XML-fil och sedan läsa in denna XML UI-definition under körtid med Builder-klassen vilken skapar objekten automatiskt. Använd programmet Glade för att inte behöva skriva denna XML manuellt, det låter dig skapa användargränssnittet på ett WYSIWYG-sätt (What You See Is What You Get, vad du ser är vad du får)

Denna metod har flera fördelar:

- Mindre kod behöver skrivas.
- Ändringar i användargränssnitten kan ses snabbare, så de kan förbättras.
- Designers utan programmeringskunskap kan skapa och redigera användargränssnitt.
- Beskrivningen av användargränssnittet är oberoende av programmeringsspråket som används.

Det finns fortfarande kod som krävs för att hantera gränssnittsändringar som utlöses av användaren, men `Gtk.Builder` låter dig fokusera på att implementera den funktionaliteten.

## 22.1 Skapa och läsa in .glade-filen

Först måste du hämta och installera Glade. Det finns flera handledningar om Glade, så det förklaras här inte i detalj. Låt oss starta med att skapa ett fönster med en knapp i, och spara till en fil med namnet *example.glade*. Den resulterande XML-filen bör se ut så här.

```
<?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">
        <property name="label" translatable="yes">button</property>
        <property name="use-action-appearance">False</property>
```

(continues on next page)

(fortsättning från föregående sida)

```
        <property name="visible">True</property>
        <property name="can-focus">True</property>
        <property name="receives-default">True</property>
      </object>
    </child>
  </object>
</interface>
```

För att läsa in denna fil i Python behöver vi ett `Gtk.Builder`-objekt.

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

Den andra raden läser in alla objekt som definieras i *example.glade* till Builder-objektet.

Det är också möjligt att bara läsa in några av objekten. Följande rad skulle bara lägga till de objekt (och deras barnobjekt) som anges i tupeln.

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

Dessa två metoder finns också för att läsa från en sträng snarare än en fil. Deras motsvarande namn är `Gtk.Builder.add_from_string()` och `Gtk.Builder.add_objects_from_string()` och de tar helt enkelt en XML-sträng istället för ett filnamn.

## 22.2 Komma åt komponenter

Nu när fönstret och knappen har lästs in vill vi också visa dem. Därför behöver metoden `Gtk.Window.show_all()` anropas på fönstret. Men hur kommer vi åt det associerade objektet?

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

Varje komponent kan erhållas från byggaren genom metoden `Gtk.Builder.get_object()` och komponentens *id*. Det är precis *så* lätt.

Det är också möjligt att få en lista över alla objekt med

```
builder.get_objects()
```

## 22.3 Ansluta signaler

Glade möjliggör också att definiera signaler som du kan ansluta till hanterare i din kod utan att extrahera varje objekt från byggaren och ansluta till signalerna manuellt. Den första saken att göra är att deklarera signalnamnen i Glade. För detta exempel kommer vi agera när fönstret stängs och när knappen trycktes ned, så vi ger namnet ”onDestroy” till återanropet som hanterar fönstrets ”destroy”-signal och ”onButtonPressed” till återanropet som hanterar knappens ”pressed”-signal. Nu bör XML-filen se ut så här.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <!-- interface-requires gtk+ 3.0 -->
```

(continues on next page)

(fortsättning från föregående sida)

```
  <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>
```

Nu behöver vi definiera hanterarfunktionerna i vår kod. *onDestroy* bör helt enkelt resultera i ett anrop till `Gtk.main_quit()`. När knappen trycks ned skulle vi vilja skriva ut strängen ”Hello World!”, så vi definierar hanteraren enligt följande

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

Härnäst behöver vi ansluta signalerna och hanterarfunktionerna. Det lättaste sättet att göra det är att definiera en *dict* med en mappning från namnen till hanterarna och sedan skicka den till metoden `Gtk.Builder.connect_signals()`.

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

Ett alternativt tillvägagångssätt är att skapa en klass som har metoder som anropas som signalerna. I vårt exempel skulle den sista kodsnutten kunna skrivas om som:

```
from gi.repository import Gtk


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

    def onButtonPressed(self, button):
        print("Hello World!")
builder.connect_signals(Handler())
```

## 22.4 Exempel

Exemplets slutgiltiga kod

```
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` låter UI-definitionsfiler användas för att utöka en komponent, PyGObject tillhandahåller Gtk.Template som ett sätt att komma åt detta från Python.

UI-definitionsfilen som används i exemplet behöver en liten ändring för att inkludera ett <*template*>-element:

```
<?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>
```

Sedan kan den användas för att implementera exemplet med en `Gtk.Window`-underklass:

```python
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()
```

Mer information kan hittas på webbplatsen PyGObject.

KAPITEL 23

# Objekt

GObject är den grundläggande typen som tillhandahåller de gemensamma attributen och metoderna för alla objekttyper i GTK+, Pango och andra bibliotek baserade på GObject. Klassen *GObject.GObject* tillhandahåller metoder för skapande och förstörande av objekt, åtkomstmetoder för egenskaper samt stöd för signaler.

Detta avsnitt kommer introducera några viktiga aspekter om GObject-implementationen i Python.

## 23.1 Ärv från GObject.GObject

Ett inhemskt GObject är åtkomligt via *GObject.GObject*. Det instansieras sällan direkt, vi använder i allmänhet ärvda klasser. En `Gtk.Widget` är en ärvd klass av ett *GObject.GObject*. Det kan vara intressant att göra en ärvd klass för att skapa en ny komponent, exempelvis en inställningsdialog.

För att ärva från *GObject.GObject* måste du anropa `GObject.GObject.__init__()` i din konstruktor (om klassen exempelvis ärver från `Gtk.Button` så måste den anropa `Gtk.Button.__init__()`), som i exemplet nedan:

```
from gi.repository import GObject

class MyObject(GObject.GObject):

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

## 23.2 Signaler

Signaler ansluter godtyckliga programspecifika händelser till ett valfritt antal lyssnare. Till exempel så tas i GTK+ varje användarhändelse (tangenttryckning eller musrörelse) emot från X-servern och genererar en GTK+-händelse i form av en signalutsändning på en given objektinstans.

Varje signal registreras i typsystemet tillsammans med typen på vilken den kan sändas ut: användare av typen sägs ansluta till signalen på en given typinstans när de registrerar en funktion som ska anropas vid signalutsändning. Användare kan också sända ut signalen själva eller stoppa utsändningen av signalen inifrån en av funktionerna som är anslutna till signalen.

### 23.2.1 Ta emot signaler

Se *Huvudslinga och signaler*

### 23.2.2 Skapa nya signaler

Nya signaler kan skapas genom att lägga till dem till `GObject.GObject.__gsignals__`, en ordbok:

Då en ny signal skapas kan en metodhanterare också definieras, den kommer anropas varje gång signalen sänds ut. Den kallas do_signalnamn.

```
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` indikerar att denna signal kommer anropa objektets metodhanterare (här `do_my_signal()`) i det första utsändningssteget. Alternativ är `GObject.SIGNAL_RUN_LAST` (metodhanteraren kommer anropas i det tredje utsändningssteget) och `GObject.SIGNAL_RUN_CLEANUP` (anropa metodhanteraren i det sista utsändningssteget).

Den andra delen, `None`, indikerar returtypen för signalen, vanligen `None`.

`(int,)` indikerar signalargumenten, här kommer signalen bara ta ett argument, vars typ är int. Typer av argument som krävs av signalen deklareras som en sekvens, här är det en tupel med ett argument.

Signaler kan sändas ut med `GObject.GObject.emit()`:

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

## 23.3 Egenskaper

En av de trevliga funktionerna med GObject är dess allmäna get/set-mekanism för objektegenskaper. Varje klass som ärver från *GObject.GObject* kan definiera nya egenskaper. Varje egenskap har en typ som aldrig ändras (t.ex. str, float, int...). De används exempelvis för `Gtk.Button` där det finns en egenskap "label" som innehåller knappens text.

### 23.3.1 Använd befintliga egenskaper

Klassen *GObject.GObject* tillhandahåller flera användbara funktioner för att hantera befintliga egenskaper, *GObject.GObject.get_property()* och *GObject.GObject.set_property()*.

Vissa egenskaper har också avsedda get- och set-funktioner. För egenskapen "label" för en knapp finns det två funktioner för att erhålla och ställa in den, `Gtk.Button.get_label()` och `Gtk.Button.set_label()`.

### 23.3.2 Skapa nya egenskaper

En egenskap definieras med ett namn och en typ. Även om Python i sig är dynamiskt typat så kan du inte ändra typen för en egenskap efter att den definierats. En egenskap kan skapas med `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)
```

Egenskaper kan också vara skrivskyddade om du vill att några egenskaper ska vara läsbara men inte skrivbara. För att göra detta kan du lägga till några flaggor till egenskapsdefinitionen för att styra läs/skriv-åtkomst. Flaggor är *GObject.ParamFlags.READABLE* (endast läsåtkomst för extern kod), *GObject.ParamFlags.WRITABLE* (endast skrivåtkomst), *GObject.ParamFlags.READWRITE* (öppen):

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

Du kan också definiera nya skrivskyddade egenskaper med en ny metod dekorerad med `GObject.Property`:

```
from gi.repository import GObject

class MyObject(GObject.GObject):

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

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

Du kan få denna egenskap med:

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

API:t för `GObject.Property` liknar den inbyggda `property()`. Du kan skapa egenskapsinställare på ett sätt liknande Python-egenskaper:

```
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
```

Det finns också ett sätt att definiera minsta och största värden för tal, med en mer utförlig form:

```
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
        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)
```

Egenskaper måste definieras i *`GObject.GObject.__gproperties__`*, en ordbok, och hanteras i

do_get_property och do_set_property.

### 23.3.3 Bevaka egenskaper

När en egenskap ändras sänds en signal ut, vars namn är ”notify::egenskapsnamn”:

```
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
```

Observera att du måste använda det kanoniska egenskapsnamnet då du ansluter till notify-signalerna, som förklaras i `GObject.Object.signals.notify()`. För en Python-egenskap *foo_bar_baz* skulle du exempelvis ansluta till signalen *notify::foo-bar-baz* med

```
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*)
Erhåller ett egenskapsvärde.

**set_property**(*property_name*, *value*)
Ställer in egenskapen *property_name* till *value*.

**emit**(*signal_name*, ...)
Sänd ut signalen *signal_name*. Signalargument måste följa, t.ex. om din signal är av typen `(int,)` så måste den sändas ut med:

```
self.emit(signal_name, 42)
```

**freeze_notify**()
Denna metod fryser alla ”notify::”-signalerna (som sänds ut då någon egenskap ändras) till metoden *`thaw_notify()`* anropas.

Det rekommenderas att använda *with*-satsen vid anrop av *`freeze_notify()`*, på det viset säkerställs att *`thaw_notify()`* anropas implicit i slutet på blocket:

```
with an_object.freeze_notify():
    # Do your work here
    ...
```

**thaw_notify**()
Tina alla "notify::"-signalerna som frystes av *freeze_notify()*.

Det rekommenderas att inte anropa *thaw_notify()* explicit utan anropa *freeze_notify()* tillsammans med *with*-satsen.

**handler_block**(*handler_id*)
Blockerar en hanterare av en instans så att den inte kommer att anropas under några signalutsändningar om inte *handler_unblock()* anropas för detta *handler_id*. Att "blockera" en signalhanterare betyder därmed att tillfälligt inaktivera den, en signalhanterare behöver avblockeras exakt det antal gånger som den har blockerats innan den blir aktiv igen.

Det rekommenderas att använda *handler_block()* tillsammans med *with*-satsen som kommer anropa *handler_unblock()* implicit i slutet på blocket:

```
with an_object.handler_block(handler_id):
    # Do your work here
    ...
```

**handler_unblock**(*handler_id*)
Gör effekten av *handler_block()* ogjord. En blockerad hanterare hoppas över under signalutsändningar och kommer inte anropas förrän den har avblockerats exakt det antal gånger som den tidigare blockerats.

Det rekommenderas att inte anropa *handler_unblock()* explicit utan anropa *handler_block()* tillsammans med *with*-satsen.

**__gsignals__**
En ordbok där en ärvd klass kan definiera nya signaler.

Varje element i ordboken är en ny signal. Nyckeln är signalnamnet. Värdet är en tupel med formen:

```
(GObject.SIGNAL_RUN_FIRST, None, (int,))
```

*GObject.SIGNAL_RUN_FIRST* kan ersättas med *GObject.SIGNAL_RUN_LAST* eller *GObject.SIGNAL_RUN_CLEANUP*. `None` är returtypen för signalen. `(int,)` är parametertupeln för signalen.

**__gproperties__**
Ordboken *__gproperties__* är en klassegenskap där du definierar ditt objekts egenskaper. Detta är inte det rekommenderade sättet att skapa nya egenskaper, metoden ovan är mycket kortare. Fördelarna med denna metod är att en egenskap kan definieras med fler inställningar, som minsta eller största värde för tal.

Nyckeln är namnet på egenskapen

Värdet är en tupel som beskriver egenskapen. Antalet element i denna tupel beror på dess första element, men tupeln kommer alltid innehålla åtminstone följande objekt:

Det första elementet är egenskapens typ (t.ex. `int`, `float`...).

Det andra elementet är egenskapens smeknamn, vilket är en sträng med en kort beskrivning av egenskapen. Detta används vanligen av program med starka introspektionsförmågor, som den grafiska användargränssnittsbyggaren Glade.

Det tredje är egenskapens beskrivning eller blurb, vilket är en annan sträng med en längre beskrivning av egenskapen. Används också av Glade och liknande program.

Det sista (vilket inte nödvändigtvis är det fjärde som vi kommer se senare) är egenskapens flaggor: `GObject.PARAM_READABLE`, `GObject.PARAM_WRITABLE`, `GObject.PARAM_READWRITE`.

Den absoluta längden på tupeln beror på egenskapstypen (tupelns första element). Därmed har vi följande situationer:

Om typen är `bool` eller `str` så är det fjärde elementet standardvärdet för egenskapen.

Om typen är `int` eller `float` så är det fjärde elementet det minsta accepterade värdet, det femte elementet det största accepterade värdet, och det sjätte elementet är standardvärdet.

Om typen inte är någon av dessa så finns det inget extra element.

`GObject.`**`SIGNAL_RUN_FIRST`**
Anropa objektets metodhanterare i det första utsändningssteget.

`GObject.`**`SIGNAL_RUN_LAST`**
Anropa objektets metodhanterare i det tredje utsändningssteget.

`GObject.`**`SIGNAL_RUN_CLEANUP`**
Anropa objektets metodhanterare i det sista utsändningssteget.

`GObject.ParamFlags.`**`READABLE`**
Egenskapen är läsbar.

`GObject.ParamFlags.`**`WRITABLE`**
Egenskapen är skrivbar.

`GObject.ParamFlags.`**`READWRITE`**
Egenskapen är läsbar och skrivbar.

KAPITEL 24

# Application

`Gtk.Application` täcker flera upprepande uppgifter som ett modernt program behöver så som att hantera flera instanser, D-Bus-aktivering, öppna filer, kommandoradstolkning, uppstart/nedstängning, menyhantering, fönsterhantering med mera.

## 24.1 Åtgärder

`Gio.Action` är ett sätt att exponera varje enskild uppgift ditt program eller din komponent gör med ett namn. Dessa åtgärder kan inaktiveras/aktiveras vid körtid och de kan antingen aktiveras eller få ett tillstånd ändrat (om de innehåller tillstånd).

Orsaken för att använda åtgärder är att separera logiken från användargränssnittet. Till exempel så tillåter detta användning av en menyrad i OSX och en kugghjulsmeny i GNOME genom att helt enkelt referera namnet på en åtgärd. Den huvudsakliga implementationen av detta som du kommer att använda är `Gio.SimpleAction` som kommer demonstreras senare.

Många klasser så som `Gio.MenuItem` och `Gtk.ModelButton` stöder egenskaper för att ställa in ett åtgärdsnamn.

Dessa åtgärder kan grupperas tillsammans i en `Gio.ActionGroup` och då dessa grupper läggs till i en komponent med `Gtk.Widget.insert_action_group()` kommer de få ett prefix. Exempelvis ”win” då de läggs till ett `Gtk.ApplicationWindow`. Du kommer använda det fullständiga åtgärdsnamnet då du refererar till det, som ”app.about”, men då du skapar åtgärden kommer den bara vara ”about” tills den läggs till i programmet.

Du kan också väldigt lätt göra tangentbindningar för åtgärder genom att ställa in egenskapen *accel* i `Gio.Menu`-filen eller genom att använda `Gtk.Application.set_accels_for_action()`.

## 24.2 Menyer

Dina menyer bör definieras i XML med `Gio.Menu` och skulle hänvisa till de tidigare nämnda åtgärderna som du definierade. `Gtk.Application` låter dig konfigurera en meny antingen via `Gtk.Application.set_app_menu()` eller `Gtk.Application.set_menubar()`. Om du använder `Gio.Resource` så kan det automatiskt använda rätt meny baserat på plattform, annars kan du ställa in dem manuellt. Ett detaljerat exempel visas nedan.

## 24.3 Kommandorad

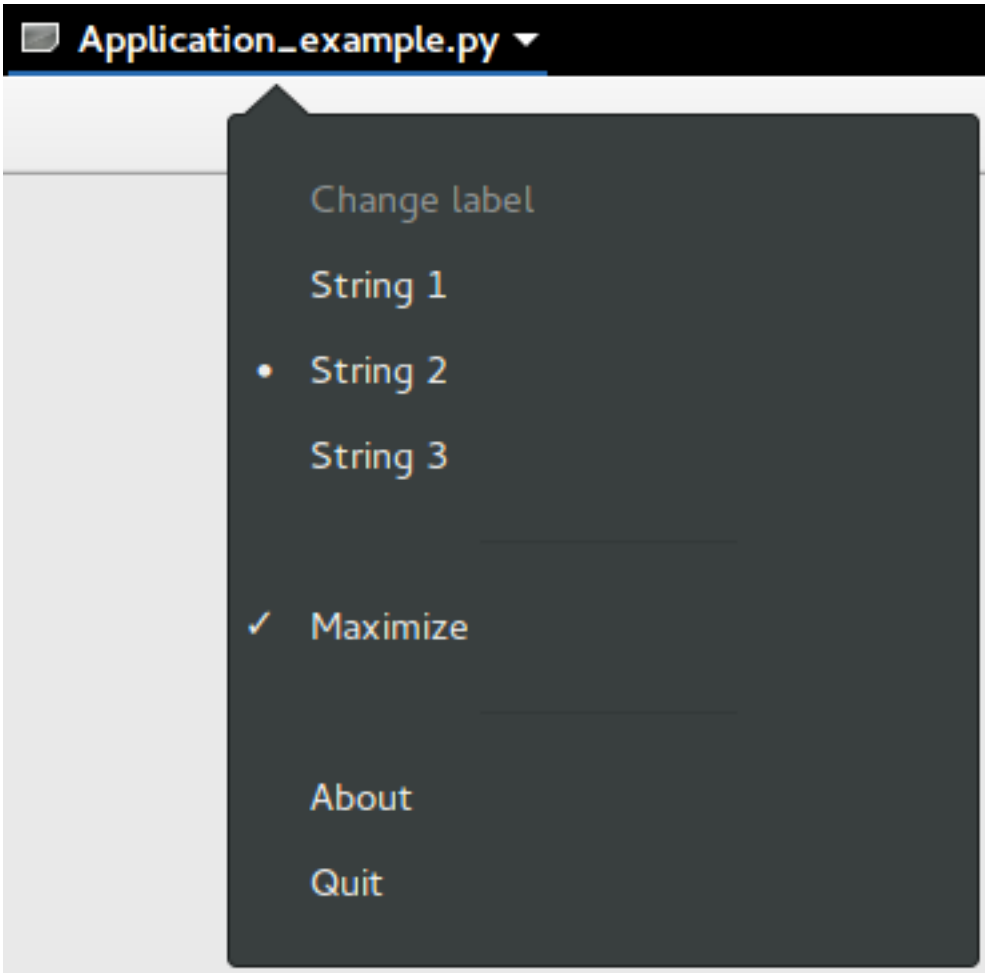
Då du skapar ditt program tar det en flaggegenskap från `Gio.ApplicationFlags`. Med denna kan du låta det hantera allting själv eller ha mer anpassat beteende.

Du kan använda *HANDLES_COMMAND_LINE* för att tillåta anpassat beteende i `Gio.Application.do_command_line()`. Kombinera med `Gio.Application.add_main_option()` för att lägga till anpassade alternativ.

Att använda *HANDLES_OPEN* kommer göra jobbet med att helt enkelt ta filargument åt dig och låta dig hantera det i `Gio.Application.do_open()`.

Om ditt program redan är öppet kommer alla dessa sändas till den befintliga instansen om du inte använder *NON_UNIQUE* för att tillåta flera instanser.

## 24.4 Exempel




```
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>
      <item>
        <attribute name="action">win.change_label</attribute>
```

(continues on next page)

(fortsättning från föregående sida)

```
        <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</antribute>
        <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)
            ),
        )

        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)

```

(continues on next page)

(fortsättning från föregående sida)

```
        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)

        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()
```

(continues on next page)

(fortsättning från föregående sida)

```
    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 Se även

- https://wiki.gnome.org/HowDoI/GtkApplication
- https://wiki.gnome.org/HowDoI/GAction
- https://wiki.gnome.org/HowDoI/ApplicationMenu
- https://wiki.gnome.org/HowDoI/GMenu

KAPITEL 25

# Menyer

**Observera:** `Gtk.UIManager`, `Gtk.Action` och `Gtk.ActionGroup` är föråldrade sedan GTK+ version 3.4 och bör inte användas i nyskriven kod. Använd ramverket *Application* istället.

GTK+ kommer med två olika sorters menyer, `Gtk.MenuBar` och `Gtk.Toolbar`. `Gtk.MenuBar` är en standardmenyrad vilken innehåller en eller flera instanser av `Gtk.MenuItem` eller en av dess underklasser. `Gtk.Toolbar`-komponenter används för snabb åtkomst till vanligen använda funktioner för ett program. Exempel inkluderar att skapa ett nytt dokument, skriva ut en sida eller ångra en operation. Den innehåller en eller flera instanser av `Gtk.ToolItem` eller en av dess underklasser.

## 25.1 Åtgärder

Även om det finns specifika API:er för att skapa menyer och verktygsfält så bör du använda `Gtk.UIManager` och skapa `Gtk.Action`-instanser. Åtgärder organiseras i grupper. En `Gtk.ActionGroup` är i huvudsak en avbildning från namn till `Gtk.Action`-objekt. Alla åtgärder som skulle vara rimliga att använda i ett visst sammanhang bör vara i en enskild grupp. Flera åtgärdsgrupper kan användas för ett specifikt användargränssnitt. Det förväntas faktiskt att de flesta icke-triviala programmen använder flera grupper. Exempelvis ett program som kan redigera flera dokument, med en grupp som innehåller globala åtgärder (t.ex. avsluta, om, nytt), och en grupp per dokument som innehåller åtgärder som agerar på det dokumentet (t.ex. spara, klipp ut/kopiera/klistra in o.s.v.). Varje fönsters menyer skulle då konstrueras från en kombination av två åtgärdsgrupper.

Det finns olika klasser som representerar olika typer av åtgärd:

- `Gtk.Action`: En åtgärd som kan utlösas genom ett meny- eller verktygsfältsobjekt
- `Gtk.ToggleAction`: En åtgärd som kan växlas mellan två tillstånd
- `Gtk.RadioAction`: En åtgärd för vilken endast en i en grupp kan vara aktiv
- `Gtk.RecentAction`: En åtgärd som representerar en lista över senast använda filer

Åtgärder representerar operationer som användaren kan utföra, tillsammans med lite information om hur det ska visas i gränssnittet, inklusive dess namn (ej för visning), dess etikett (för visning), en snabbtangent, huruvida en etikett indikerar en inforuta såväl som återanropet som anropas när åtgärden aktiveras.

Du kan skapa åtgärder antingen genom att anropa en av konstruktorerna direkt och lägga till dem till en `Gtk.ActionGroup` genom att anropa `Gtk.ActionGroup.add_action()` eller `Gtk.ActionGroup.add_action_with_accel()`, eller genom att anropa en av bekvämlighetsfunktionerna:

- `Gtk.ActionGroup.add_actions()`,
- `Gtk.ActionGroup.add_toggle_actions()`
- `Gtk.ActionGroup.add_radio_actions()`.

Observera att du måste ange åtgärder för undermenyer såväl som för menyobjekt.

## 25.2 Användargränssnittshanterare

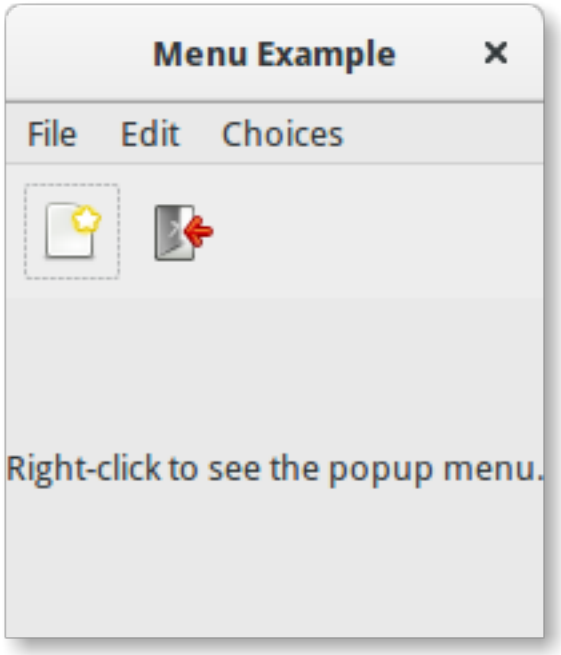
`Gtk.UIManager` tillhandahåller ett enkelt sätt att skapa menyer och verktygsfält med en XML-liknande beskrivning.

Först bör du lägga till `Gtk.ActionGroup` till användargränssnittshanteraren med `Gtk.UIManager.insert_action_group()`. Vid denna punkt är det också god idé att säga till föräldrafönstret att svara på de angivna tangentbordsgenvägarna genom att använda `Gtk.UIManager.get_accel_group()` och `Gtk.Window.add_accel_group()`.

Sedan kan du definiera den faktiska synliga layouten för menyerna och verktygsfälten, och lägga till användargränssnittslayouten. Denna ”ui-sträng” använder ett XML-format, i vilket du bör nämna namnen på åtgärderna som du redan skapat. Kom ihåg att dessa namn bara är identifierarna som vi använde då vi skapade åtgärderna. De är inte texten som användaren kommer se i menyerna och verktygsfälten. Vi tillhandahöll dessa mänskligt läsbara namn då vi skapade åtgärderna.

Slutligen erhåller du rotkomponenten med `Gtk.UIManager.get_widget()` och lägger till komponenten till en behållare så som `Gtk.Box`.

## 25.3 Exempel




```
import gi

gi.require_version("Gtk", "3.0")
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' />
```

(continues on next page)

(fortsättning från föregående sida)

```
    <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()
        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,
```

(continues on next page)

(fortsättning från föregående sida)

```
        )
        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,
                    "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,
        )
```

(continues on next page)

(fortsättning från föregående sida)

```
        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)
        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()
```

# KAPITEL 26

# Table

**Observera:** `Gtk.Table` är föråldrat sedan GTK+ version 3.4 och bör inte användas i nyskriven kod. Använd klassen *Grid* istället.

Tabeller låter oss placera komponenter i ett rutnät liknande `Gtk.Grid`.

Rutnätets dimensioner behöver anges i `Gtk.Table`-konstruktorn. För att placera en komponent i en ruta, använd `Gtk.Table.attach()`.

`Gtk.Table.set_row_spacing()` och `Gtk.Table.set_col_spacing()` ställer in utrymmet mellan raderna vid den angivna raden eller kolumnen. Observera att för kolumner hamnar utrymmet till höger om kolumnen, och för rader hamnar utrymmet under raden.

Du kan också ställa in ett enhetligt avstånd för alla rader och/eller kolumner med `Gtk.Table.set_row_spacings()` och `Gtk.Table.set_col_spacings()`. Observera att med dessa anrop placeras inget utrymme efter den sista raden och sista kolumnen.

Ersatt sedan version 3.4: Du rekommenderas använda `Gtk.Grid` för ny kod.

## 26.1 Exempel


```
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()
```

KAPITEL 27

# Index och tabeller

- search

# Index

## Symbols



## E



## F



## G



## H



## R



## S



## T



## W