入門GTK+

菅谷 保之 著

2021年1月27日

まえがき

GTK+ は，GUI アプリケーションを作成するためのツールキット（ライブラリ）です．もともとは GIMP という画像編集ソフトを作成するために開発されましたが，現在では GIMP に留まらずさまざまなアプリケーションが GTK+ によって作られています．特に，Windows にひけを取らないデスクトップ環境を実現している GNOME も，そのベースは GTK+ によって作られています．そのため，GTK+ は現在最も注目されているツールキットだと言えるでしょう（GNOME と並んで二大デスクトップ環境と呼ばれる KDE のベースとなる Qt というツールキットも有名です）．

GTK+ が登場する以前にもさまざまなツールキットが存在しましたが，見た目があまり良くなかったり，かなりの手間をかけなければ思ったような外観が作成できないなどの問題がありました．GTK+ はその見た目のクールさもさることながら，リリース当初からテーマ機能が実装されており，ユーザが見た目を自由に変更できるという魅力的な特徴がありました．また，比較的簡単なプログラミングで思ったような外観が作成できるのも大きな特徴です．最近では GUI による GUI 作成支援アプリケーション Glade やプログラム作成統合環境 Anjuta なども開発され，GUI プログラミングの初心者でも簡単に GUI アプリケーションが作れるようになってきています．

しかし，GTK+ によるプログラミングに関する日本語の書籍やドキュメントはそう多くないことから, GUI プログラミングの初心者にもわかりやすく GTK+ によるプログラミングが学べるように, GTK+ のバージョン 2 を対象としたドキュメントを 2009 年に執筆しました．現在では, GTK+ のメジャーバージョンが 3 となり, バージョン 2 から大きく関数の仕様が変更されたものも少なくありません．本書は, 2009 年に執筆したものを GTK+ のバージョン 3 に対応させるとともに, 新たなトピックも多少追加した内容となっています．

2016 年 10 月

菅谷 保之

対象とする読者

本書は C 言語をある程度習得していて，GUI アプリケーションの作成に興味を持っている人を対象に書かれています．GTK+ についてあまり知識がない人でも，チュートリアルを読み進めながら GUI アプリケーションを作れるでしょう．また GTK+ で使用しているさまざまなライブラリや GTK+ が提供するウィジェットの使い方も解説しているので，ある程度 GTK+ を知っているけれどもっと詳しく知りたいという人にも最適です．

本書の構成

本書では，GTK+ のインストールや簡単なプログラム作成から始まり，独自のウィジェット作成やプログラム作成統合環境を使用したプログラム作成など発展的な話題を扱います．本書の構成は次のようになっています．

- **1 章 GTK+ ってなんだろう**
  GTK+ に関する簡単な紹介と，Ubuntu 16.04 日本語 Remix 版での GTK+ の開発環境の設定方法を説明します．
- **2 章 GTK+ で画像ビューワを作ってみよう**
  チュートリアル形式で画像ビューワを作成しながら，GTK+ によるアプリケーション作成を体験します．ここではあくまで GTK+ の体験を目的としているので，詳細な説明は省略して，GTK+ を使うとどんなことができるのかを中心に紹介します．
- **3 章 もっと GTK+**
  GUI アプリケーションを作成する際に最も基本となるウィジェットの配置方法や，ボタンをクリックしたときにどのような仕組みでコールバック関数が呼び出されるかを説明します．
- **4 章 GLib**
  GLib は，C ライブラリの標準関数を拡張したいくつかの関数を提供します．またリストやハッシュなどの拡張データ構造も提供されており，それらのデータ構造を簡単に操作できます．ここでは，GLib が提供する便利な関数やデータ構造と，それらに対する操作関数を具体例を挙げて説明します．
- **5 章 cairo による図形の描画**
  cairo は，GTK+ のベクタグラフィック部分を担当するライブラリです．ここでは cairo による図形描画について解説し，直線や矩形，円弧など具体的な図形の描画方法を説明します．
- **6 章 GdkPixbuf**
  GTK+ のバージョン 2 以降では，画像読み込み用のフレームワークとして GdkPixbuf が正式に採用されました．GdkPixbuf を使うと，PNG や JPEG といった一般的な画像ファイルを読み込み，画像に対して簡単な操作を行うことができます．この章では画像ファイルの読み込みや画像のプロパティへのアクセス方法，読み込んだ画像データを GTK+ のフレームワーク内でどのように表示するかについて紹介します．
- **7 章 ウィジェットリファレンス**
  GTK+ では，操作性が良く，便利な GUI を作成するためにさまざまなウィジェットが用意されています．これらのウィジェットのそれぞれについて，機能や具体的な使用方法を紹介します．
- **8 章 プログラミングの小箱**
  マウスやキープレスの検出やドラッグ&ドロップの実装など，実際にアプリケーションを作成する際に欠かせない処理から，少し発展的な内容まで幅広く紹介します．
- **9 章 独自ウィジェットの作成**
  GTK+ では既存のウィジェットのほかに独自のウィジェットを一から作成することができます．この章では新しいウィジェットの作成方法を具体例を挙げて解説します．
- **10 章 統合開発環境によるソフトウェア開発**
  GTK+/GNOME の統合開発環境である anjuta によるアプリケーション作成の手順を解説します．anjuta によるプロジェクト管理や glade による GUI のレイアウト作成機能によって，GTK+ アプリケーションを簡単に作成できるようになります．
- **11 章 GTK+ に関連するその他のライブラリ**
  本章では, GTK+ に関連するその他のライブラリについて, ごく簡単にですがサンプルを示しながら紹介します.

- **付録 A Visual Studio でのビルド方法**
  Windows 上で Visual Studio を使ってプログラムをビルドする方法を説明します．
- **付録 B アイコンブラウザ**
  GTK+ であらかじめ定義されているアイコン名を確認するためのアプリケーション gtk3-icon-browser について説明します．
- **付録 C サンプルプログラムのソースコードリスト**
  著者の Web サイトで公開している，本書で扱ったプログラムソースコードの構成を説明しています．

必要なソフトウェア

本書は，Ubuntu 16.04 日本語 Remix 版で動作確認を行っています．その他の OS でも，GTK+ の開発環境をインストールすれば，本書で紹介した内容は動作すると思われます．

本書での記述方法

本書では，次に示す形式で解説を行っています．

コマンドの入力

本書ではコマンドラインからのコマンド入力を以下のような網掛け表示で示します．また，$ はプロンプトを，↵ は改行記号を表すものとします．

```
$ sudo apt-get install libgtk-3-dev ↵
```

プログラムのソースコード

本書内で紹介するプログラムのソースコードは，以下のように左端に行番号を表示した形式で示します．解説中に参照する行番号はこの番号と対応しています．また，本書で紹介するソースコードは，基本的に GNU スタイルでコーディングされています．また，空白文字を強調する際には “␣” という記号を使っています．

ソース 2–1 ウィンドウの作成：image-viewer.c

```
1  #include <gtk/gtk.h>
2
3  /*
4    メイン関数
5  */
6  int main (int argc, char *argv[])
7  {
8    GtkWidget* window;
9
10   /* GTK+の初期化およびオプション解析 */
11   gtk_init (&argc, &argv);
```

ご意見とご質問

本書に関するご意見，ご質問は，上記の Web ページで公開している掲示板へ書き込むか，次のメールアドレスまでお願いいたします．

sugaya@iim.cs.tut.ac.jp

謝辞

本書の執筆にあたって貴重なご意見を頂きました大阪大学の向川康博准教授，パナソニック株式会社石井育規氏に感謝の意を

表します．また株式会社クリアコード代表取締役の須藤功平氏には，本書に対するご意見だけでなく，本書で紹介したサンプルプログラムに対する大変貴重なご意見を頂きましたことに感謝の意を表します．今回執筆の機会を与えて下さり，多岐にわたりお世話になりました株式会社オーム社の皆様，執筆に多大なるご協力を頂きました毛利勝久氏に感謝の意を表します．

最後に，この執筆活動を陰ながら支えてくれた妻と息子に感謝します．

目次

1 GTK+ ってなんだろう

1.1 GTK+

GTK+[*1] は **GUI**（*Graphical User Interface*）アプリケーションを作成するためのライブラリです．もともとは **GIMP**[*2] という画像編集アプリケーション（図 1.1）を作成するために開発されましたが，現在では GIMP 以外にもさまざまなアプリケーションに用いられており，**GNOME デスクトップ環境**[*3] のためのツールキットとして利用されています．

GTK+ が開発された当初はウィジェット間に階層構造がありませんでしたが，ウィジェット間にオブジェクト指向的な階層

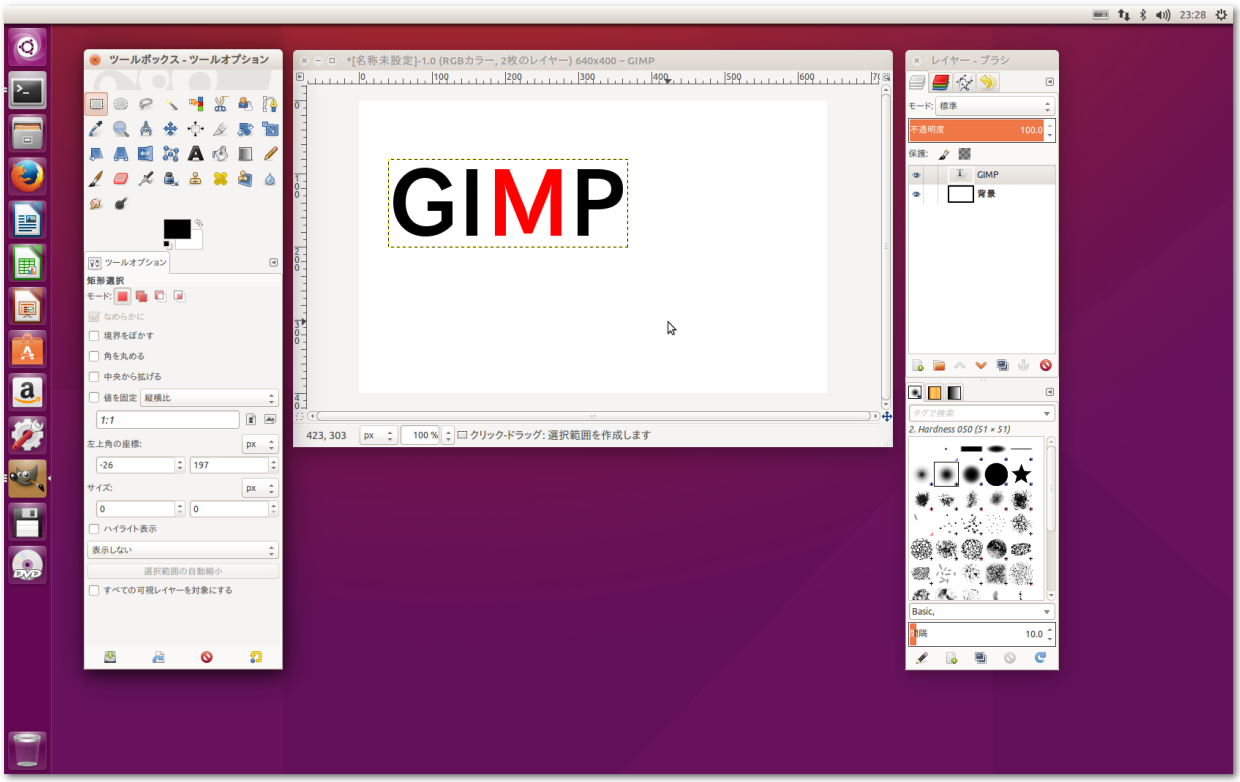


図 1.1　GIMP の実行画面

[*1] GTK+（The GIMP Toolkit）：http://www.gtk.org/
[*2] GIMP—The GNU Image Manipulation Program: http://www.gimp.org/
[*3] GNOME: The Free Software Desktop Project: http://www.gnome.org/

関係を持たせるように拡張されて，その名前に ‘+’ が加わりました．また GTK+ のバージョンが 2 になってからは，**Pango** による多言語表示環境のサポート，**ATK** によるアクセシビリティの向上，**cairo** による 2 次元ベクタグラフィックのサポートなど，多くの拡張がなされました．これらを含めて，GTK+ は以下に挙げるライブラリ群を使用しています．

- GLib ... 基本的なデータ型やリストやハッシュ等の便利な関数を提供します．
- Pango ... 多言語やマークアップされたテキストを表示するインターフェイスを提供します．
- ATK ... アクセシビリティのインターフェイスを提供します．
- GDK ... ウィンドウシステムを抽象化した関数を提供します．
- GdkPixbuf ... 画像ファイルを操作する関数を提供します．
- cairo ... ベクタグラフィックスに関する関数を提供します．

GTK+ の Web ページは http://www.gtk.org/ で公開されており，GTK+ についての情報を得たり，GTK+ の最新バージョンをダウンロードしたりできます．また，この Web ページには GTK+ を使って作成されたアプリケーションのスクリーンショット（図 1.2）を見ることができます．このページで紹介されている画像編集アプリケーション GIMP やベクタ画像編集アプリケーション Inkscape，表計算アプリケーション Gnumeric をはじめとして，そのほかにもテキストエディタ gedit や画像ビューワ eog（Eye of GNOME）など（図 1.3）非常に多くのアプリケーションが GTK+ を用いて作成されています．

1.2 GUI ツールキット

GTK+ をはじめとする GUI ツールキットとは，グラフィカルインターフェイスを作成するための部品（ウィンドウやボタンなど）の集合です．こうした部品を，GTK+ では**ウィジェット**と呼んでいます．

GTK+ 以外にもさまざま GUI ツールキットがあり，その一例を以下に挙げます．

- Xaw ... Project Athena という開発プロジェクトにより開発された X Window System 用のツールキットです．

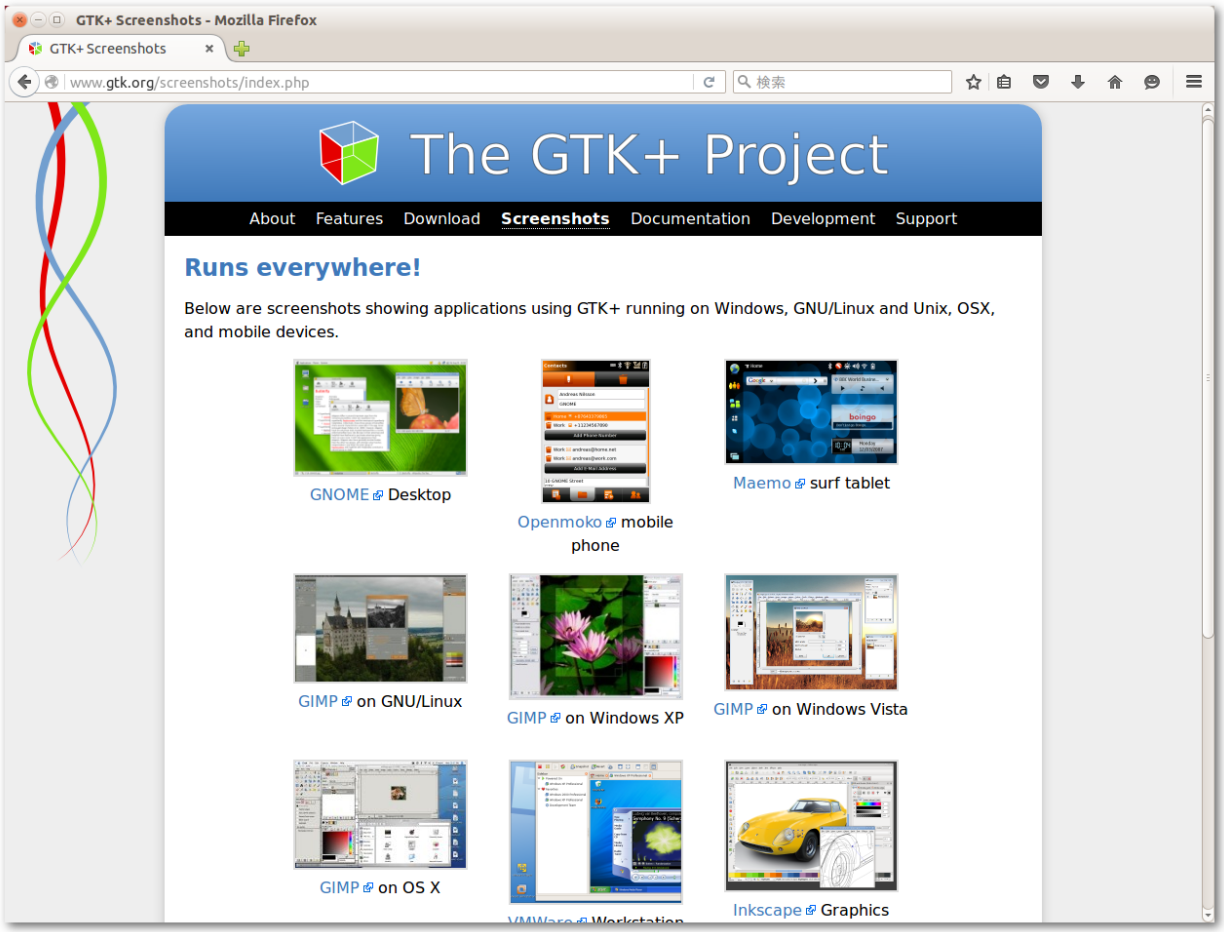
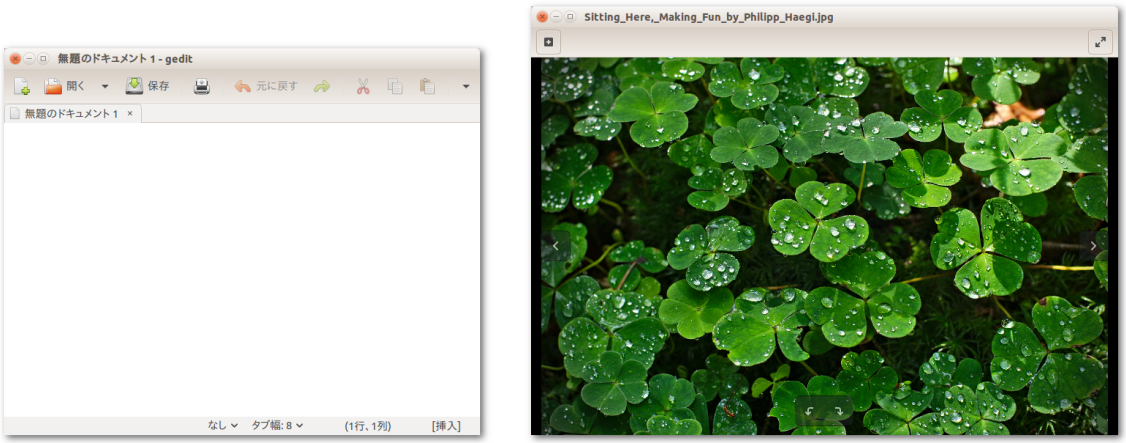


図 1.2　gtk.org では GTK+ を使って作られたアプリケーションがたくさん紹介されている



(a) テキストエディタ gedit (b) 画像ビューワ eog

図 1.3 GTK+ を使用して作成されたアプリケーション

- Motif ... Motif は X Window System の見た目（ルック＆フィール）を統一するために定められた GUI 規格で，この規格に沿って開発されたツールキットを Motif ツールキットと呼びます．単に Motif と言った場合，多くは Motif ツールキットを指します．
- LessTif ... Motif をオープンソースのライブラリとして実装したツールキットです．
- Swing ... Java で実装された GUI ツールキットです．
- Tk ... スクリプト言語 Tcl などで使用されている GUI ツールキットです．
- Qt ... C++ で書かれた GUI ツールキットです．GNOME デスクトップ環境と並ぶ KDE デスクトップ環境に使用されているツールキットとして有名です．

1.3 GTK+ の特徴

GTK+ の特徴を簡単に以下にまとめます．

- オープンソースのライブラリ ... GTK+ は **GNU LGPL** [*4]のもとで開発されています．
- 豊富なウィジェット ... GTK+ には，GUI を構築するのに必要なボタンなどの基本的なウィジェットから，ファイルを選択するダイアログのように特定の目的を達成する応用的なものまで，数多くのウィジェットが存在します．このため，プログラマは少ない手間で高機能なアプリケーションを作成することが可能です．
- C 言語による実装 ... ウィジェットには階層的な関係にあるものが多く，これらは C++ のクラス構造を用いると容易に実現できます（例えば先に紹介した Qt は C++ のクラスで実装されています）．しかし GTK+ では，ウィジェット間の階層構造を C 言語により，あたかもクラスであるように巧妙に実装しています．このため，C++ が苦手なプログラマでも，広く利用されている C 言語でプログラムを作成できます．
- GUI レイアウト ... 古いツールキットでは，ウィジェットのレイアウトを配置する位置や大きさを細かく設定しなければならないものもありました．GTK+ ではこれらの詳細をあまり細かく考えることなく，簡単にそして柔軟に実現できます．
- ルック＆フィール ... GTK+ にはテーマ機能が実装されており，ユーザはテーマを好みに応じて変更することで，アプリケーションの見た目を自由にカスタマイズできます．
- 多くのプログラミング言語に対応 ... GTK+ は C 言語で書かれたライブラリですが，C 言語以外のプログラミング言語でも使用できるように，C++，C#，Python，Ruby をはじめとするさまざまな**バインディング**[*5]が公開されています．
- マルチプラットフォーム ... GTK+ は Linux だけではなく，Windows や Mac OS X 上でも動作する**マルチプラットフォーム**なライブラリです．

[*4] GNU Lesser General Public License: http://www.gnu.org/copyleft/lesser.html
[*5] binding: 他の言語から利用するためのインターフェイス．

1.4 プログラミング環境を整えよう

次章から早速 GTK+ によるプログラミングを紹介していくために，ここでは GTK+ の開発環境を構築する方法を説明します．本書の内容は基本的に OS や Linux の**ディストリビューション**，Unix 互換 OS に依存しませんが，プログラムの実行例などは，Ubuntu 16.04 の日本語 Remix 版がインストールされた環境で紹介します．

本章では，Ubuntu 16.04 の日本語 Remix 版での GTK+ の開発環境の構築方法について説明しますが，Windows で開発される方は付録 A （p. 313）を参考に開発環境を整えてください．

Ubuntu 16.04 の日本語 Remix 版は，Ubuntu Japanese Local Community Team の Web サイト（http://www.ubuntulinux.jp/）で公開されています．ここでは Ubuntu 16.04 日本語 Remix 版が既にインストールされているものとして，GTK+ の開発環境をインストールする部分のみを説明します．

Ubuntu をインストールした直後の状態では，GTK+ の開発環境はインストールされていません．開発に必要なパッケージを，ユーザーがインストールしなければなりません．パッケージのインストール方法には，GUI アプリケーションによるインストールと，コマンドラインでのインストールの 2 通りがあります．次節からそれぞれの方法について解説します．

1.4.1 コマンドラインによるアプリケーションのインストール

Ubuntu では，**apt-get** というコマンドでパッケージの追加や削除を行えます．インストールしたいパッケージ名が既にわかっている場合には，この apt-get コマンドを使うと簡単にインストールできます．

GTK+ のバージョン 3 の開発環境は libgtk-3-dev という名前のパッケージです．このパッケージを apt-get でインストールするには，次のようにします．

```
$ sudo apt-get install libgtk-3-dev ↵
```

コマンドを実行すると，以下のようなメッセージが表示され，処理を続けるかどうか聞いてきます．処理を続けるには “y” を入力します（-y オプションを指定しておくと，この “y” の入力を省くことができます）．

```
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
以下の追加パッケージがインストールされます:
...
アップグレード: 2 個、新規インストール: 89 個、削除: 0 個、保留: 22 個。
15.3 MB のアーカイブを取得する必要があります。
この操作後に追加で 80.8 MB のディスク容量が消費されます。
続行しますか? [Y/n] y
```

同様に Anjuta や glade, gtranslator，devhelp も，以下のようにインストールできます．

```
$ sudo apt-get install anjuta glade devhelp gtranslator ↵
```

1.4.2 GUI によるアプリケーションのインストール

ここでは **Synaptic パッケージマネージャ**を用いて，必要なパッケージをインストールする方法を紹介します．Synaptic では，インストールしたいパッケージの名前がわからなくても，パッケージを検索してインストールできます．

しかし, Ubuntu16.04 をインストールした直後の状態では, Synaptic パッケージマネージャーはインストールされていません．まずは apt-get を使ってインストールしましょう．

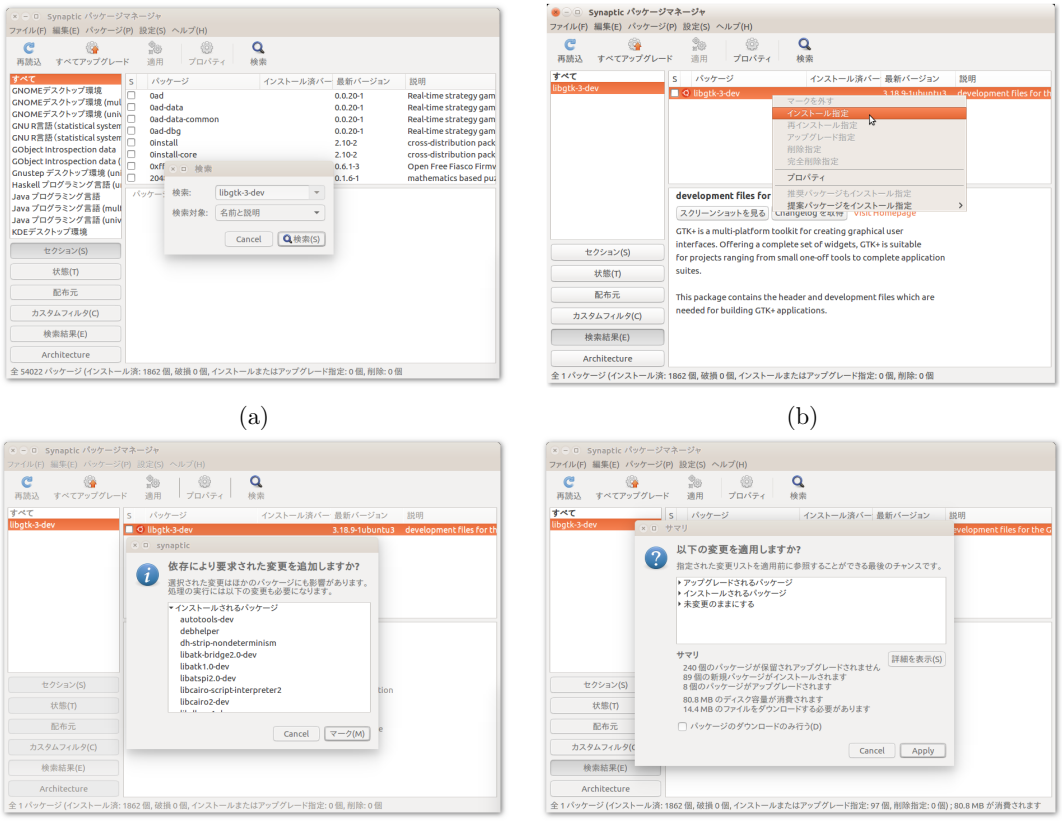
```
$ sudo apt-get install synaptic ↵
```

Synaptic パッケージマネージャが無事インストールできたら, 次に GTK+ の開発環境をインストールします.

1. コマンドラインから Synaptic パッケージマネージャを起動します.

```
$ sudo synaptic ↵
```

2. Synaptic パッケージマネージャの右端の「検索」ボタンをクリックして，検索ウィンドウの検索入力欄に”libgtk-3-dev”と入力し，同ウィンドウの「検索 (S)」ボタンをクリックします（図 1.4(a)）.
3. ”libgtk-3-dev” が表示されるのでパッケージ名にマウスカーソルを合わせて, マウスの右ボタンをクリックするとポップアップメニューが表示されるので, メニュー内の「インストール設定」をクリックします. （図 1.4(b)）.
4. ”libgtk-3-dev” をインストールするめたに依存関係のあるパッケージ一覧が表示されるので「マーク (M)」ボタンをクリックします. （図 1.4(c)）.
5. 確認のために図 1.4(d) のようにダイアログが表示されるので,「適用 (A)」ボタンを押してください.



(a)　(b)

(c)　(d)

図 1.4　Synaptic パッケージマネージャによる開発環境のインストール

これで，GTK+ プログラミングの基本的な開発環境が整いました．10 章で必要となる Anjuta や gtranslator，2.2.5 節で紹介する devhelp なども同じ手順でインストールできます.

2 GTK+ で画像ビューワを作ってみよう

2.1 とにかく作ってみよう

本章ではまず難しい理屈はおいて，とにかく GTK+ を用いたアプリケーション作成を体験してみることにします．作成するアプリケーションは図 2.1 に示す画像ビューワです．画像ビューワなんて初心者に作れるの？ と思う人も，本章のチュートリアルに従って段階的にプログラムを拡張していくことで，それなりのアプリケーションが簡単に作れてしまうことに驚かれることでしょう．ぜひ，このチュートリアルを通して，GTK+ の魅力と可能性を体験してください．

本チュートリアル中の各ステップで GTK+ の機能にもいくつか触れますが，本章の目的はそれぞれの機能の詳細を説明することではありません．詳しい説明は後に続く章で解説していくので，本章ではそんなものかという程度の理解で問題ありません．

早速，次節から画像ビューワの作成を始めていきます．なお，Windows でプログラムを作成する場合は，付録 A （p. 313）も参考にしてください．

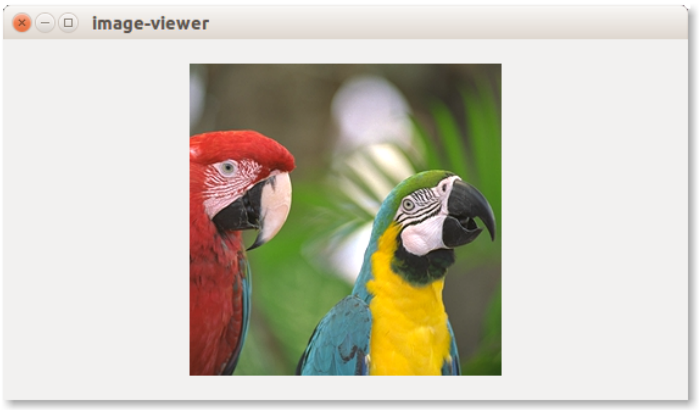


図 2.1　チュートリアルで作成する画像ビューワ

2.2　ウィンドウの作成

ここでは空のウィンドウを作ります．アプリケーションとしては最も単純ですが，ウィンドウの作成を通して GTK+ のプログラミングの流れを学びます．

2.2.1　作業ディレクトリとソースコードの作成

まずは準備として，このチュートリアル用のディレクトリを作成してください．ここでは，ホームディレクトリの下に tutorial というディレクトリを作成することにします．これ以降の処理は，特に説明しない限り，この tutorial ディレクトリ内で行うものとします．

```
$ mkdir tutorial ↵
$ cd tutorial ↵
```

tutorial ディレクトリが作成できたら，次に適当なテキストエディタでソース 2–1 を入力して，tutorial 内に image-viewer.c という名前で保存してください．ここでは Ubuntu に標準でインストールされる gedit を使用します（図 2.2）．

```
$ gedit image-viewer.c ↵
```

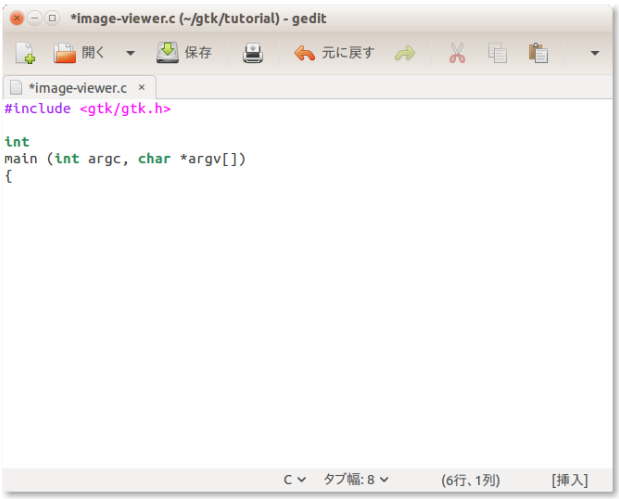


図 2.2　gedit でソースコードを入力する

ソース 2–1　ウィンドウの作成：image-viewer.c

```
1  #include <gtk/gtk.h>
2
3  /*
4    メイン関数
5  */
6  int
7  main (int argc, char *argv[])
8  {
9    GtkWidget *window;
10
11   /* GTK+の初期化およびオプション解析 */
12   gtk_init (&argc, &argv);
13   /* ウィンドウの作成 */
14   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
15   /* ウィンドウの大きさの設定 */
16   gtk_widget_set_size_request (window, 300, 200);
```

```
17   /* ウィンドウの表示 */
18   gtk_widget_show (window);
19   /* メインループ */
20   gtk_main ();
21
22   return 0;
23 }
```

2.2.2 ソースコードのコンパイル

ソースコードが入力できたら，それをコンパイルしてプログラムを作成します．

GTK+ を使用したプログラムをコンパイルするためには，コンパイル時に読み込むヘッダファイルのあるディレクトリと，プログラムを実行するときに使用するライブラリのあるディレクトリを指定する必要があります．これらの情報は **pkg-config** というコマンドを使って調べることができます．

試しに端末上で，次のように pkg-config コマンドを実行してみましょう．

```
$ pkg-config --cflags gtk+-3.0 ↵
-pthread -I/usr/include/gtk-3.0 -I/usr/include/at-spi2-atk/2.0 -I/usr/include/at
-spi-2.0 -I/usr/include/dbus-1.0 -I/usr/lib/x86_64-linux-gnu/dbus-1.0/include -I
/usr/include/gtk-3.0 -I/usr/include/gio-unix-2.0/ -I/usr/include/mirclient -I/us
r/include/mircommon -I/usr/include/cairo -I/usr/include/pango-1.0 -I/usr/include
/harfbuzz -I/usr/include/pango-1.0 -I/usr/include/atk-1.0 -I/usr/include/cairo -
I/usr/include/pixman-1 -I/usr/include/freetype2 -I/usr/include/libpng12 -I/usr/i
nclude/gdk-pixbuf-2.0 -I/usr/include/libpng12 -I/usr/include/glib-2.0 -I/usr/lib
/x86_64-linux-gnu/glib-2.0/include
```

この結果からもわかるように，GTK+ を使用したプログラムをコンパイルする際には，非常に多くのオプションを指定しなければなりません．

しかし，コンパイル時にも pkg-config コマンドを利用すれば，オプションを入力する手間を省くことができます．以下のようにコマンドを実行して，ソースコードをコンパイルしてみてください．

```
$ gcc image-viewer.c -o image-viewer `pkg-config --cflags --libs gtk+-3.0` ↵
```

上記のコマンドを実行してエラーが出る場合には，pkg-config のコマンド部分を’（シングルクォート）ではなく，‘（バッククォート）で囲んでいるかどうか確かめてください．通常の日本語キーボードであれば，‘（バッククォート）は@（アットマーク）と同じ位置にあるはずです（入力するには Shift+@）．

2.2.3 プログラムの実行

コンパイルが無事終了したら，image-viewer という名前のプログラムが作成されているので，実行してみてください．

```
$ ./image-viewer ↵
```

図 2.3 のようなウィンドウが表示されたでしょうか．

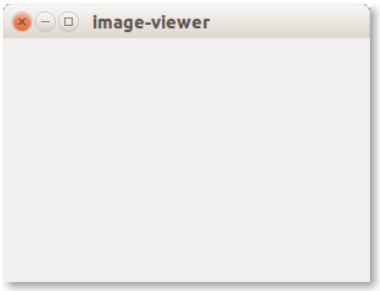


図 2.3　ウィンドウの表示

2.2.4　ソースコードの解説

無事にウィンドウが表示されることを確認したところで，ソースコードの解説を始めます．

ヘッダファイルのインクルード（1 行目）

GTK+ が提供する関数のプロトタイプ宣言が記述されたヘッダファイル gtk.h をインクルードしています．GTK+ の関数を使用する場合，必ずこのヘッダファイルをインクルードしなければなりません．

GTK+ の初期化（12 行目）

関数 gtk_init は GTK+ の初期化を行ったり，GTK+ 共通のオプションを解析する関数です．GTK+ でアプリケーションを作成する場合には，必ずこの関数をはじめに呼び出す必要があります．

```
void gtk_init (int *argc, char *argv[]);
```

第 1 引数	メイン関数の第 1 引数（コマンドライン引数の数）へのポインタ
第 2 引数	メイン関数の第 2 引数（コマンドライン引数）へのポインタ

ウィンドウの作成（14 行目）

関数 gtk_window_new によってウィンドウを作成しています．この関数の引数には作成するウィンドウの種類を指定します．今回のようにアプリケーションのメインになるようなウィンドウの場合には GTK_WINDOW_TOPLEVEL を，マウスをクリックしたときにポップアップで表示されるようなウィンドウでは GTK_WINDOW_POPUP を指定します．

```
GtkWidget* gtk_window_new (GtkWindowType type);
```

第 1 引数	ウィンドウの種類
戻り値	生成したウィンドウウィジェット

ウィンドウをはじめとしてボタンなど GUI を構成する部品を GTK+ では**ウィジェット**と呼びます．ウィジェットを作成する関数を呼び出すと，その関数内でウィジェットが作成され，作成したウィジェットを指すポインタが関数の戻り値として返ってきます．9 行目で宣言したポインタ変数 window に関数 gtk_window_new の戻り値を代入しているので，変数 window は作成したウィンドウを指します．

ウィンドウの大きさ設定（16 行目）

ここでは，関数 gtk_widget_set_size_request を使用してウィンドウの大きさを指定しています．第 1 引数に大きさを設定するウィジェットを，第 2 および第 3 引数にそれぞれ幅と高さを指定します．

```
void gtk_widget_set_size_request (GtkWidget *widget,
                                  gint       width,
                                  gint       height);
```

第 1 引数	ウィジェット
第 2 引数	ウィジェットの幅
第 3 引数	ウィジェットの高さ

ウィンドウの表示（18 行目）

作成したウィジェットを表示するには，関数 gtk_widget_show を使用します．

```
void gtk_widget_show (GtkWidget *widget);
```

第 1 引数　表示するウィジェット

メインループ（20 行目）

アプリケーションはユーザからの何らかの操作を待つ状態となります．

```
void gtk_main (void);
```

2.2.5　devhelp による関数検索

GTK+ では非常に多くの関数が提供されているため，自分が使用したい関数が存在するのか調べたり，一度使用した関数の引数が何だったかを調べるのは非常に大変です．**devhelp** はそんな問題を解決してくれるアプリケーションです．devhelp はライブラリのマニュアルやリファレンスを表示するツールです．

例えば，今回使用した関数 gtk_window_new の使い方を調べるには，devhelp 画面の左側の検索タブで gtk_window_new と入力します．入力するとその下に該当する名前が出てきて（例えば，gtk_window_まで入力すると gtk_window_から始まる候補が表示されます），右側の画面に関数の説明が表示されます（図 2.4）．

また，表示されるテキストはハイパーテキスト形式になっており，関連する項目を簡単に調べることができるので，知っておくと非常に便利です．

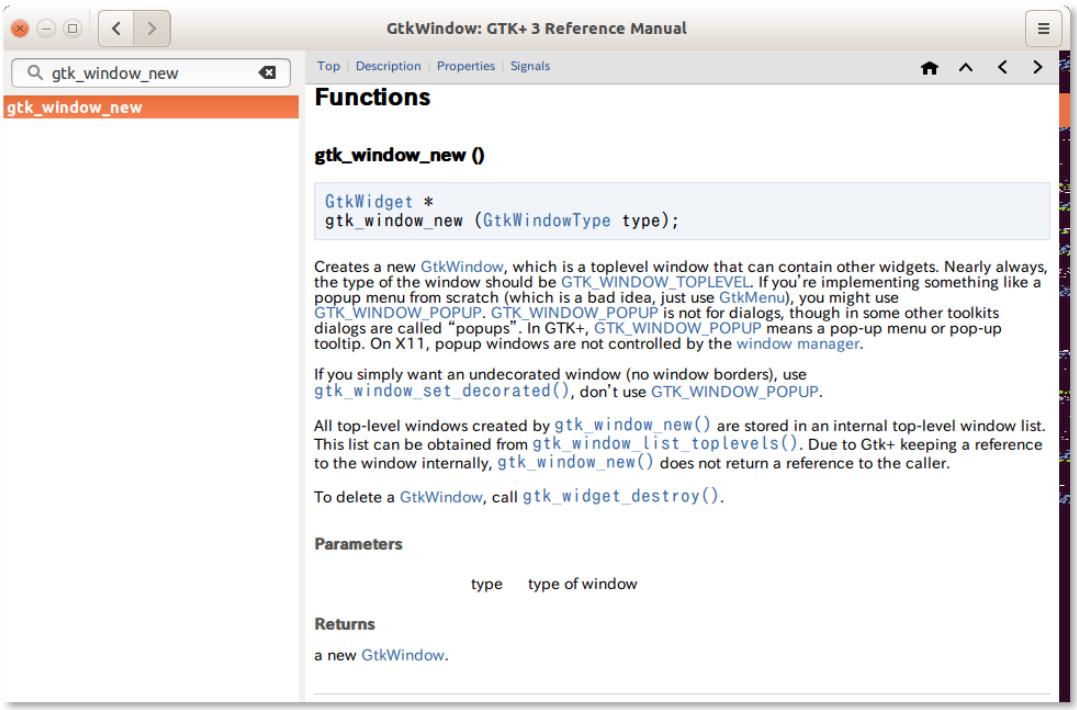


図 2.4　devhelp による関数検索

2.2.6　まとめ

本節では，アプリケーションの基本となるウィンドウの作成を通して，GTK+ のアプリケーションを作成するための基礎について説明しました．GTK+ のアプリケーションを作成する際の基本的な流れを以下にまとめます．

次節からは本節で作成したプログラムを少しずつ拡張していき，最終的に画像ビューワを完成させます．

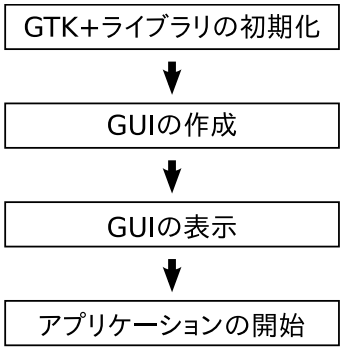


図 2.5　アプリケーション作成の流れ

2.3 ボタンの追加

前節ではウィンドウを作成しましたが，このプログラムには終了する方法が用意されていませんでした．そこで本節では，前節で作成したウィンドウにボタンを追加して，ボタンをクリックするとプログラムが終了するようにしてみます．

2.3.1 ボタンウィジェットの作成

まずは追加するボタンを作成します．今回は関数 gtk_button_new_with_label を使用して，次のように Quit というラベルのついたボタンを作成します．

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

2.3.2 ボタンの配置

ボタンを作成したら，次にそのボタンをウィンドウ上に配置します．ウィンドウ上にボタンを配置するには，次のようにします．

```
gtk_container_add (GTK_CONTAINER (window), button);
```

ウィンドウウィジェットは，自分自身の中に他のウィジェットを 1 つ配置することのできるウィジェットです．このようなウィジェットを**コンテナ**と呼びます．関数 gtk_container_add はコンテナにウィジェットを配置する関数です．
また，GTK_CONTAINER は，コンテナウィジェットから派生したウィジェットをコンテナウィジェットに型変換します．このほかにも，ウィジェットを GObject に型変換する G_OBJECT などがあります．

```
void gtk_container_add (GtkContainer *container,
                        GtkWidget    *widget);
```

第 1 引数　コンテナウィジェット
第 2 引数　コンテナ内に配置するウィジェット

配置したボタンを表示するためには，前節で説明した関数 gtk_widget_show を使うこともできるのですが，ここでは関数 gtk_widget_show_all を使用することにします．この関数では，引数に指定したウィジェット内に配置されたすべてのウィジェットを表示できます．

```
gtk_widget_show_all (window);
```

```
void gtk_widget_show_all (GtkWidget *widget);
```

第 1 引数　表示する親ウィジェット

2.3.3 コールバック関数の登録

次に，ボタンをクリックするとプログラムが終了するように，ボタンがクリックされたときに呼び出される関数を登録します．

ボタンをはじめとして，ウィジェットにはそのウィジェットに応じた操作があります．例えばボタンであれば，ボタンをクリックする（ボタンの立場から言えば「クリックされた」）という操作があります．本書ではウィジェットに対する操作を**イベント**と呼ぶことにします．ウィジェットはイベントが起こると，それに対応した**シグナル**を発生させます．

シグナルが発生したときに呼び出される関数を登録しておくことで，ユーザの操作に応じて決められた処理を行えるようになります．シグナルが発生したときに呼び出される関数を，**コールバック関数**と呼びます．

ボタンには，クリックされたときに発生する **clicked シグナル**があります．このシグナルに対するコールバック関数は，次のように登録します．ここでは cb_button_clicked が，登録するコールバック関数名です．

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

g_signal_connect の引数は，それぞれの次のようになっています．

第 1 引数　コールバック関数を関連付けるオブジェクト
第 2 引数　シグナル名
第 3 引数　コールバック関数
第 4 引数　コールバック関数に渡すデータ

GTK+ のウィジェットは，C++ のクラスのような階層構造を持っており，すべてのウィジェットは **GObject** と呼ばれる構造体から派生しています．第 1 引数に出てくる G_OBJECT() は，GObject への型変換を行うマクロです．

コールバック関数に渡すデータ（第 4 引数）には，文字列やその他のウィジェットを指定できます．

2.3.4 コールバック関数の実装

最後にコールバック関数を実装します．今回はプログラムを終了するだけなので，関数の中身は非常に単純です．

```
static void
cb_button_clicked (GtkWidget *button, gpointer user_data)
{
  gtk_main_quit ();
}
```

コールバック関数の第 1 引数は，コールバック関数を呼び出したウィジェットです．また第 2 引数は，g_signal_connect の第 4 引数で指定したデータです．この関数では，プログラムを終了するために，関数 gtk_main_quit を呼び出しています．この関数は，関数 gtk_main の呼び出しで開始したアプリケーションのメインループを終了するものです．

```
void gtk_main_quit (void);
```

2.3.5　コンパイルと動作確認

この節で追加した内容を反映させたソースコードをソース 2–2 に示します．このソースコードを先ほどと同じようにコンパイルして，正しく動作するか確認してみましょう．コンパイルのコマンドをもう一度示します．

```
$ gcc image-viewer.c -o image-viewer `pkg-config --cflags --libs gtk+-3.0`
```

ソース 2–2 ボタンの追加：image-viewer.c

```
1  #include <gtk/gtk.h>
2
3  /*
4    ボタンがクリックされたときに呼び出される関数
5  */
6  static void
7  cb_button_clicked (GtkWidget *button, gpointer user_data)
8  {
9    /* メインループを終了 */
10   gtk_main_quit ();
11 }
12
13 /*
14   メイン関数
15 */
16 int
17 main (int argc, char **argv)
18 {
19   GtkWidget *window;
20
21   /* GTK+の初期化およびオプション解析 */
22   gtk_init (&argc, &argv);
23   /* ウィンドウの作成 */
24   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
25   /* ウィンドウの大きさの設定 */
26   gtk_widget_set_size_request (window, 300, 200);
27   {
28     GtkWidget *button;
29
30     /* ボタンの作成 */
31     button = gtk_button_new_with_label ("Quit");
32     /* ボタンをウィンドウに配置 */
33     gtk_container_add (GTK_CONTAINER (window), button);
34     /* ボタンがクリックされたときに呼び出される関数の設定 */
35     g_signal_connect (G_OBJECT (button), "clicked",
36                       G_CALLBACK (cb_button_clicked), NULL);
37   }
38   /* ウィンドウの表示 */
39   gtk_widget_show_all (window);
40   /* メインループ */
41   gtk_main ();
42
43   return 0;
44 }
```

コンパイルが終了し，プログラムが作成できたことを確認したら，次のようにプログラムを実行してみましょう．

```
$ ./image-viewer
```

図 2.6 のようなウィンドウが表示されたでしょうか．ウィンドウ内に Quit というラベルのついたボタンが表示されていると思います．このボタンをクリックするとプログラムが終了することを確認してください．

ウィンドウが表示されてもボタンが表示されない場合は，ソースコードの 39 行目で，関数 gtk_widget_show ではなく，関数 gtk_widget_show_all を使用しているかどうか確かめてください．

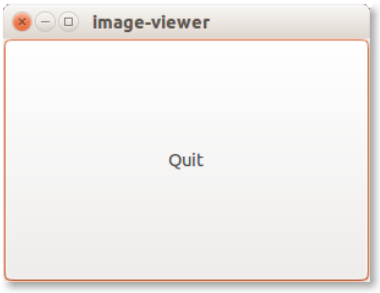


図 2.6 ボタンのついたウィンドウ

2.3.6 まとめ

本節では，ウィンドウ内にボタンを配置して，ボタンをクリックすることでプログラムを終了できるようにしました．その中で以下の項目について扱いました．

- ボタンの作成
- ウィンドウ（コンテナ）へのウィジェットの配置
- コールバック関数の登録

2.4 画像の表示

ボタンでプログラムを終了できるようになったところで，今度はウィンドウ内に画像を表示してみましょう．画像を表示するにはさまざまな実現方法がありますが，ここでは一番簡単な，イメージウィジェットを使用した方法を紹介します．

2.4.1 イメージウィジェットの作成

イメージウィジェットはその名前の通り，画像を表示するウィジェットです．このウィジェットには，ファイルから画像データを読み込んで表示する機能があります（詳細は 6.3 節（p. 107）を参照してください）．今回はこの機能を利用して，次のようにイメージウィジェットを作成します．

```
GtkWidget *image;
image = gtk_image_new_from_file (argv[1]);
```

この関数 gtk_image_new_from_file は，画像ファイル名を引数に取り，画像ウィジェットを作成する関数です．表示できる画像の種類については，表 6.1 （p. 104）を参照してください．

argv[1] は，プログラムを実行する際に，プログラム名に続けて入力する 1 番目の引数です．これによってイメージウィジェット image には，プログラム実行時に指定した画像ファイルが表示されます．

```
GtkWidget* gtk_image_new_from_file (const gchar *filename);
```

第 1 引数	画像ファイル名
戻り値	生成したイメージウィジェット

2.4.2 イメージウィジェットの配置

ボタンと同様に，ウィジェットは作成してもウィンドウ内に配置しなければ表示することができません．では先ほどと同じようにイメージウィジェットをウィンドウに配置したいところですが，1 つ問題があります．それは，ウィンドウのようなコンテナウィジェットは，その中に 1 つのウィジェットしか配置できないということです．

この問題を解決するために，複数のウィジェットを配置できる**パッキングボックス**と呼ばれるウィジェットを使用します．パッキングボックスには，ウィジェットを水平方向に配置する水平パッキングボックスと，垂直方向に配置する垂直パッキングボックスが存在します．今回は，イメージウィジェットの下にボタンを配置するために，垂直パッキングボックスを使用します．

```
GtkWidget *vbox;
vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 2);
gtk_container_add (GTK_CONTAINER (window), vbox);
gtk_box_pack_start (GTK_BOX (vbox), image, TRUE, TRUE, 0);
gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
```

垂直パッキングボックスを作成するには，関数 gtk_box_new を使用し，第 1 引数に GTK_ORIENTATION_VERTICAL を指定します．水平パッキングボックスを作成するには，GTK_ORIENTATION_HORIZONTAL を指定します．

```
GtkWidget* gtk_box_new (GtkOrientation orientation, gint spacing);
```

第 1 引数	配置する子ウィジェットの方向を指定する．
第 2 引数	子ウィジェット間に空けるスペース．
戻り値	生成したボックスウィジェット．

パッキングボックスにウィジェットを配置するには，関数 gtk_box_pack_start を使用します．この関数を実行した順番に，パッキングボックスの上から下へとウィジェットが配置されます．引数については，　3.2.2 節（p. 37）で詳しく説明することにします．

2.4.3　コンパイルと動作確認

本節で変更のあった部分を反映させたソースコードが，ソース 2–3 です．今回の変更で，アプリケーションに表示する画像ファイル名を，プログラムを実行する際に引数で指定する必要があるので，ファイル名が指定されたかどうかのチェックを，ソースコードの 23–27 行目で行っています．

ソース 2–3　画像の表示：image-viewer.c

```
1  #include <gtk/gtk.h>
2  #include <stdlib.h>
3
4  /*
5    ボタンがクリックされたときに呼び出される関数
6  */
7  static void
8  cb_button_clicked (GtkWidget *button, gpointer user_data)
9  {
10   /* メインループを終了 */
11   gtk_main_quit ();
12 }
13
14 /*
15   メイン関数
16 */
17 int
18 main (int argc, char **argv)
19 {
20   GtkWidget *window;
21
22   /* 引数のチェック */
23   if (argc != 2)
24     {
25       g_print ("Usage:␣%s␣image-file\n", argv[0]);
26       exit (1);
27     }
28   /* GTK+の初期化およびオプション解析 */
29   gtk_init (&argc, &argv);
30   /* ウィンドウの作成 */
31   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
32   /* ウィンドウの大きさの設定 */
33   gtk_widget_set_size_request (window, 300, 200);
34   {
35     GtkWidget *vbox;
36
```

```
37      /* 縦にウィジェットを配置するボックスの作成 */
38      vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 2);
39      /* ボックスをウィンドウに配置 */
40      gtk_container_add (GTK_CONTAINER (window), vbox);
41      {
42        GtkWidget *image;
43        GtkWidget *button;
44
45        /* ファイルから画像を読み込んでイメージの作成 */
46        image = gtk_image_new_from_file (argv[1]);
47        /* イメージをボックスに配置 */
48        gtk_box_pack_start (GTK_BOX (vbox), image, TRUE, TRUE, 0);
49
50        /* ボタンの作成 */
51        button = gtk_button_new_with_label ("Quit");
52        /* ボタンをボックスに配置 */
53        gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
54        /* ボタンがクリックされたときに呼び出される関数の設定 */
55        g_signal_connect (G_OBJECT (button), "clicked",
56                          G_CALLBACK (cb_button_clicked), NULL);
57      }
58    }
59    /* ウィンドウの表示 */
60    gtk_widget_show_all (window);
61    /* メインループ */
62    gtk_main ();
63
64    return 0;
65  }
```

コンパイルの方法はもう覚えたでしょうか．覚えていない人は 2.2.2 節（p. 9）に戻って確認して，変更を加えたソースコードをコンパイルしてみましょう．コンパイルが終わったら，いつものように実行して動作を確認してみましょう．プログラムの引数には，表示したい画像ファイルを指定してください．

```
$ ./image-viewer ~/images/Parrots.png
```

図 2.7 のように，指定した画像が表示されたでしょうか．ウィンドウの大きさは表示した画像の大きさに合わせて大きく変化していて，その大きさより小さくすることができない状態です．このままだと，非常に大きな画像を表示する場合に困りますので，次節でウィンドウにスクロールバーを配置することにします．

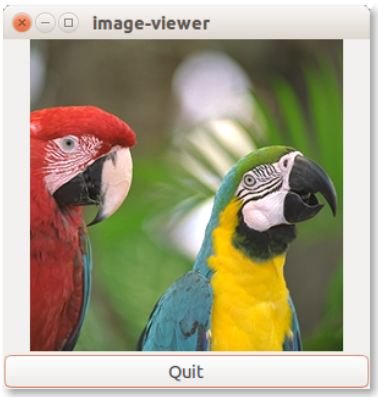


図 2.7 画像の表示

2.4.4 まとめ

本節では，プログラム実行時に画像ファイル名を指定することで，ウィンドウ内に画像を表示するプログラムを作成しました．本節で扱った項目を以下にまとめます．

- イメージウィジェットの作成

- パッキングボックスの作成
- パッキングボックスへのウィジェットの配置

2.5 スクロールバーの追加

設定したウィンドウの大きさより表示したい画像が小さい場合にはいいのですが，前節のように画像がウィンドウより大きい場合はどうしたらよいでしょうか．他のアプリケーションを思い出してみてください．1 つの良い解決方法は，ウィンドウにスクロールバーを付けることです．本節では，スクロールバーの付いたウィンドウ内に，画像を配置することにします．

2.5.1 スクロールバー付きのウィンドウの作成

GTK+ には，スクロールバーの付いたウィンドウが用意されています．前節の問題は，イメージウィジェットを直接パッキングボックスに配置するのではなく，先にイメージウィジェットをスクロールバー付きのウィンドウに配置して，その後にパッキングボックスに配置することで，簡単に解決できます．

スクロールバーの付いたウィンドウは，次のように作成します．

```
GtkWidget *scroll_window;
scroll_window = gtk_scrolled_window_new (NULL, NULL);
```

この関数 gtk_scrolled_window_new の引数には，水平方向と垂直方向のスクロールバーに関する情報を与えます．通常は，両方 NULL を与えておけば OK です．

2.5.2 イメージウィジェットの配置

スクロールバーの付いたウィンドウを作成したら，次に前節で示した方法で作成したイメージウィジェットをこのウィンドウ内に配置します．スクロールバー付きのウィンドウもコンテナの一種です．このウィジェット内にその他のウィジェットを配置するためには，関数 gtk_container_add を使用します．

2.5.3 スクロールバーの表示設定

以上の変更で，スクロールバーの付いたウィンドウ内に画像が表示されます．これで問題解決なのですが，さらにアプリケーションの見た目を良くするために，スクロールバーの表示設定を行います．何も設定しないと，たとえウィンドウ内に配置されたウィジェットがウィンドウより小さくても，スクロールバーが表示されてしまいます．

ここでは，画像のウィンドウよりも大きく，スクロールバーによる操作が必要な場合にだけ，スクロールバーを表示するため，以下のように設定します．

```
gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
                                GTK_POLICY_AUTOMATIC, GTK_POLICY_AUTOMATIC);
```

この関数 gtk_scrolled_window_set_policy で，水平方向および垂直方向に対して GTK_POLICY_AUTOMATIC を指定することにより，必要なときにだけスクロールバーが表示されるようになります．

2.5.4 コンパイルと動作確認

今回の修正を加えたソース 2–4 を入力したら，コンパイルして動作確認をしてみましょう．

ソース 2–4 スクロールバーの追加：image-viewer.c

```
1 #include <gtk/gtk.h>
2 #include <stdlib.h>
3
4 /*
5   ボタンがクリックされたときに呼び出される関数
```

```
6 */
7 static void
8 cb_button_clicked (GtkWidget *button, gpointer user_data)
9 {
10   /* メインループを終了 */
11   gtk_main_quit ();
12 }
13
14 /*
15   メイン関数
16 */
17 int
18 main (int argc, char **argv)
19 {
20   GtkWidget *window;
21
22   /* 引数のチェック */
23   if (argc != 2)
24     {
25       g_print ("Usage: %s image-file\n", argv[0]);
26       exit (1);
27     }
28   /* GTK+の初期化およびオプション解析 */
29   gtk_init (&argc, &argv);
30
31   /* ウィンドウの作成 */
32   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
33   /* ウィンドウの大きさの設定 */
34   gtk_widget_set_size_request (window, 300, 200);
35   {
36     GtkWidget* vbox;
37
38     /* 縦にウィジェットを配置するボックスの作成 */
39     vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 2);
40     /* ボックスをウィンドウに配置 */
41     gtk_container_add (GTK_CONTAINER (window), vbox);
42     {
43       GtkWidget *scroll_window;
44       GtkWidget *button;
45
46       /* スクロールバー付きウィンドウの作成 */
47       scroll_window = gtk_scrolled_window_new (NULL, NULL);
48       /* スクロールバー付きウィンドウをボックスに配置 */
49       gtk_box_pack_start (GTK_BOX (vbox), scroll_window, TRUE, TRUE, 0);
50       /* スクロールバーの表示設定 */
51       gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
52                                       GTK_POLICY_AUTOMATIC,
53                                       GTK_POLICY_AUTOMATIC);
54       {
55         GtkWidget *image;
56
57         /* ファイルから画像を読み込んでイメージの作成 */
58         image = gtk_image_new_from_file (argv[1]);
59         /* イメージをスクロールバー付きウィンドウに配置 */
60         gtk_container_add (GTK_CONTAINER(scroll_window), image);
61       }
62       /* ボタンの作成 */
63       button = gtk_button_new_with_label ("Quit");
64       /* ボタンがクリックされたときに呼び出される関数の設定 */
65       g_signal_connect (G_OBJECT (button), "clicked",
66                         G_CALLBACK (cb_button_clicked), NULL);
67       /* ボタンをボックスに配置 */
68       gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
69     }
70   }
71   /* ウィンドウの表示 */
72   gtk_widget_show_all (window);
73   /* メインループ */
74   gtk_main ();
75
76   return 0;
77 }
```

図 2.8　スクロールバー付きのウィンドウを使用した画像表示

今回も，前回同様に画像ファイル名を指定してプログラムを実行してください．

```
$ ./image-viewer ~/images/Parrots.png
```

図 2.8 に実行結果を示します．ウィンドウより大きな画像を表示させると，スクロールバーが表示されます．このスクロールバーを動かすことで，表示されていなかった部分の画像も見ることができるようになったのがわかるでしょうか．

2.5.5　まとめ

スクロールバーの付いたウィンドウに画像を表示することで，前節よりも画像ビューワらしくなってきました．本節では，以下の項目を扱いました．

- スクロールバー付きウィンドウの作成
- スクロールバー付きウィンドウへのウィジェットの配置
- スクロールバーの表示設定

2.6　GtkApplication フレームワークへの対応

GTK+-3.0 からは，**GtkAppliction** というクラスが追加されました．これは，GTK+ の初期化やセッション管理，デスクトップシステムとの連携などを自動的に行ってくれるものです．また，GtkApplication を使用するとメニューバーを簡単に作成することができます．

次節においてメニューバーを導入する前準備として，本節では前節で作成したソース 2–4 を GtkApplication を用いた形に書き換え，変更点について説明していきます．変更後のソースコードがソース 2–5 になります．


今までの方法でも，2.7 節で紹介する方法によりメニューバーを作成することはできますが，現時点で一部の機能が実現できないようです．

2.6.1　GtkApplication を用いたプログラムの流れ

図 2.9 にこれまで説明したプログラム作成の流れと GtkApplication を用いた場合のプログラム作成の流れの比較を示します．

これまでは，まず第 1 に関数 gtk_init を呼び出して GTK+ の初期化を行っていましたが，その代わりに関数 gtk_application_new を呼び出して，GtkApplication クラスの変数を生成します（ソース 2–5 の 83 行目）．このとき，関数 gtk_init が内部で呼び出されるため，プログラム内部に記述する必要がなくなります．

関数 gtk_application_new の第 1 引数にはアプリケーション ID として固有の文字列を与えます．アプリケーション ID として与える文字列にどのような約束があるのかは詳細は不明ですが，ピリオド (.) が含まれている必要があるようです．第 2 引数のアプリケーションフラグには通常 0 (G_APPLICATION_FLAGS_NONE) を与えればよいでしょう．

```
GtkApplication * gtk_application_new (const gchar       *application_id,
                                      GApplicationFlags flags);
```

第 1 引数	アプリケーション ID.
第 2 引数	アプリケーションフラグ.
戻り値	生成した GtkApplication

アプリケーションのループを開始するためには, これまでの関数 gtk_main の代わりに関数 g_application_run を呼び出します.

```
int g_application_run (GApplication *application, int argc, char **argv);
```

第 1 引数	GApplication クラスの変数.
第 2 引数	コマンドライン引数の数.
第 3 引数	コマンドライン引数.

また, プログラムを終了するための方法が関数 gtk_main_quit から関数 g_application_quit に変わります.

```
void g_application_quit (GApplication *application);
```

第 1 引数	GApplication クラスの変数.

2.6.2 GUI の作成

GUI の作成については, これまでに説明した方法に変わりはありませんが, ソース 2–5 では, GtkApplication クラスの変数が有効になったタイミング, すなわち, activate シグナルが発生したときに呼び出されるコールバック関数内で GUI の作成を行っています.

もう一つの変更点は, ウィンドウウィジェットを作成する方法が関数 gtk_window_new から関数 gtk_application_window_new に変わったことです.

```
GtkWidget * gtk_application_window_new (GtkApplication *application);
```

第 1 引数	GtkApplication クラスの変数.

ソース 2–5 GtkAppliction 版への変更：image-viewer.c

```
1  #include <gtk/gtk.h>
2  #include <stdlib.h>
3
4  /*
5    ボタンがクリックされたときに呼び出される関数
6  */
7  static void
8  cb_button_clicked (GtkWidget *button, gpointer user_data)
9  {
10   /* メインループを終了 */
```

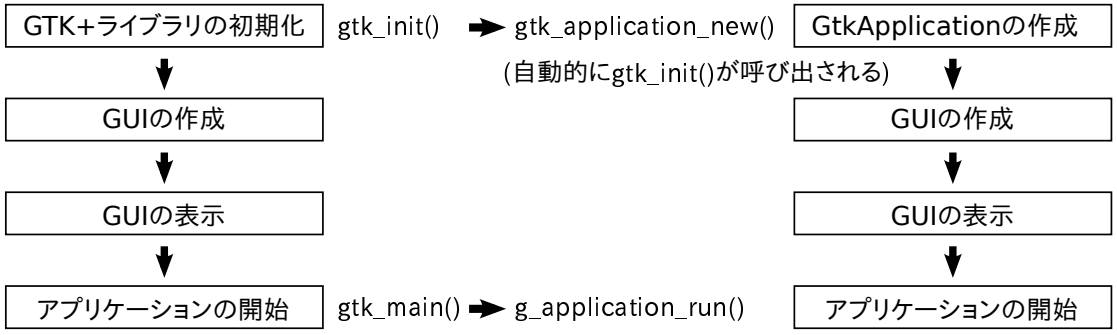


図 2.9 GtkApplication を用いたプログラムの流れ

```
11    GApplication *app = G_APPLICATION (user_data);
12    g_application_quit (app);
13  }
14
15  /*
16   * アプリケーションが有効になったときに呼び出される関数
17   */
18  static void
19  cb_activate (GApplication *app, gpointer user_data)
20  {
21    GtkWidget *window;
22
23    /* ウィンドウの作成 */
24    window = gtk_application_window_new (GTK_APPLICATION (app));
25    /* ウィンドウの大きさの設定 */
26    gtk_widget_set_size_request (window, 300, 200);
27    {
28      GtkWidget* vbox;
29
30      /* 縦にウィジェットを配置するボックスの作成 */
31      vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 2);
32      /* ボックスをウィンドウに配置 */
33      gtk_container_add (GTK_CONTAINER (window), vbox);
34      {
35        GtkWidget *scroll_window;
36        GtkWidget *button;
37
38        /* スクロールバー付きウィンドウの作成 */
39        scroll_window = gtk_scrolled_window_new (NULL, NULL);
40        /* スクロールバー付きウィンドウをボックスに配置 */
41        gtk_box_pack_start (GTK_BOX (vbox), scroll_window, TRUE, TRUE, 0);
42        /* スクロールバーの表示設定 */
43        gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
44                                        GTK_POLICY_AUTOMATIC,
45                                        GTK_POLICY_AUTOMATIC);
46        {
47          GtkWidget *image;
48
49          /* ファイルから画像を読み込んでイメージの作成 */
50          image = gtk_image_new_from_file ((char *) user_data);
51          /* イメージをスクロールバー付きウィンドウに配置 */
52          gtk_container_add (GTK_CONTAINER(scroll_window), image);
53        }
54        /* ボタンの作成 */
55        button = gtk_button_new_with_label ("Quit");
56        /* ボタンがクリックされたときに呼び出される関数の設定 */
57        g_signal_connect (G_OBJECT (button), "clicked",
58                          G_CALLBACK (cb_button_clicked), app);
59        /* ボタンをボックスに配置 */
60        gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
61      }
62    }
63    /* ウィンドウの表示 */
64    gtk_widget_show_all (window);
65  }
66
67  /*
68    メイン関数
69  */
70  int
71  main (int argc, char **argv)
72  {
73    GtkApplication *app;
74
75    /* 引数のチェック */
76    if (argc != 2)
77      {
78        g_print ("Usage:␣%s␣image-file\n", argv[0]);
79        exit (1);
80      }
81    /* アプリケーションの作成 */
82    app = gtk_application_new ("gtk.imageviewer", 0);
83    /* シグナルハンドラの登録 */
84    g_signal_connect (app, "activate", G_CALLBACK(cb_activate), argv[1]);
```

```
85    /* メインループ */
86    g_application_run (G_APPLICATION(app), 0, NULL);
87
88    return 0;
89  }
```

2.7　メニューバーの追加

本節と次節で, 画像ビューワ作成の最後のステップとして，メニューバーの追加について説明します．メニューには，ダイアログから画像ファイルを指定する Open と，アプリケーションを終了する Quit の 2 つを表示することにします．
まず本節では，以下の項目について扱います．

- メニューバーの追加
- メニューアイテム Quit でのアプリケーションの終了

この拡張にともなって，アプリケーションを終了する Quit ボタンを配置する必要がなくなります．
そして次節では，メニューアイテム Open に関して以下の項目を扱い，画像ビューワを完成させます．

- ファイル選択ダイアログの実装
- ダイアログで選択した画像ファイルの表示

2.7.1　メニュー作成の手順

メニューを作成するには, 大きく以下の二つの方法が挙げられます．

1. メニューアイテムの作成やコールバック関数の設定, メニューの階層構造などをすべて対応する関数によって作成する方法
2. **GtkBuiler** を用いてメニュー構成を記述した XML 形式のファイルもしくはテキストから作成する方法

ここでは，メニューを作成する方法として, 2. の方法を使用することにします．そして, この方法でメニューバーを作成する流れは以下のようになります．またメニュー作成の関連図を図 2.10 に示します．

1. XML 形式でメニュー構成を記述
2. **GActionEntry** 構造体でメニューアイテムに対するコールバック関数などを記述
3. 関数 g_action_map_add_action_entries を用いてコールバックをアプリケーションに登録
4. GtkBuilder を用いて XML 形式で記述したメニュー構成からメニューを作成し, そのメニューをアプリケーション上に配置

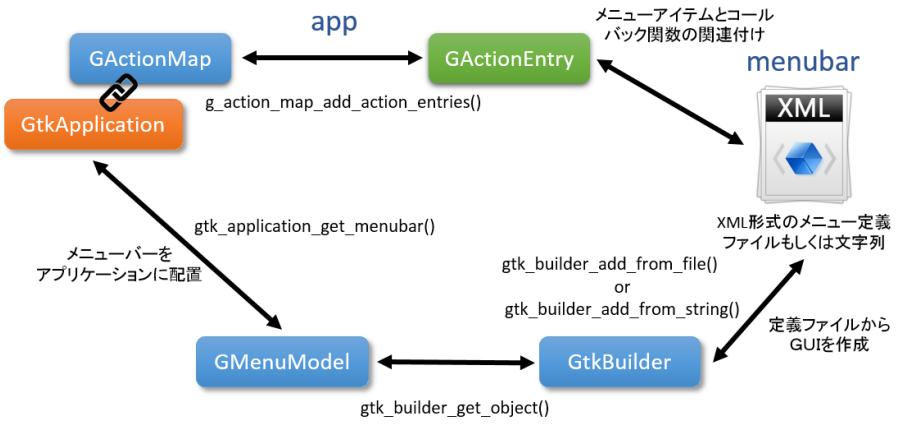


図 2.10　GtkBuilder を用いたメニュー作成の関連図

2.7.2　メニュー構成の作成

メニュー構成は，XML 形式で指定します．XML を外部のテキストファイルに記述するか，その内容をソースコード中に文字列として保持するか，いずれかの方法で作成します．今回はメニュー構成を文字列として作成し，ソースコード中に埋め込むことにします．今回作成するメニューの構成は，次のような文字列で表されます．

```
"<interface>"
"  <menu id='appmenu'>"
"    <submenu>"
"      <attribute name='label'>File</attribute>"
"      <section>"
"        <item>"
"          <attribute name='label'>Open</attribute>"
"          <attribute name='action'>app.open</attribute>"
"          <attribute name='accel'>&lt;Primary&gt;o</attribute>"
"        </item>"
"      </section>"
"      <section>"
"        <item>"
"          <attribute name='label'>Quit</attribute>"
"          <attribute name='action'>app.quit</attribute>"
"          <attribute name='accel'>&lt;Primary&gt;q</attribute>"
"        </item>"
"      </section>"
"    </submenu>"
"  </menu>"
"</interface>";
```

メニューの構成は，<interface> タグから始まり，<menu></menu> タグ内に記述します．<menu> タグ内の ID は GtkBuiler を用いて作成したメニューバーの情報を取り出す際の識別子として用いられます．ここでは，メニューバーの中に File メニューがあり，そのメニューアイテムとして Open と Quit が存在する構成となっています．この場合, File メニューは <submenu> タブで記述し, <attribute name='label'> タブでそのラベルを指定します．また, Open と Quit のメニューアイテムは <item> タブで記述します．メニューアイテムの属性は <attribute> タブで記述し, 属性の種類を name=' ' という形で指定します．メニューアイテムの代表的な属性を以下に挙げます．

- label ... メニューアイテムのラベル
- action ... GActionEntry 構造体の要素と関連付けるための文字列
- accel ... メニューアイテムに対するアクセラレータキー


action 属性の文字列は, GtkApplication（または GtkWidget）に関連付けされた **GActionMap** の識別子と GActionEntry 構造体の第 1 メンバである name 変数の文字列を用いて, "GActionMap 識別子.GActionEntry の name 文字列"の形式で記述する必要があります．

また, Open と Quit の間に仕切り（セパレータ）を配置する場合には, 仕切りの前後のメニューアイテムを <section> タブで挟むようにします．

2.7.3　メニューアイテムに対するコールバック関数の記述

メニューアイテムが選択されたときに呼び出されるコールバック関数の記述は，**GActionEntry** 構造体で行い, 以下の 5 項目を設定します．

1. メニューアイテムと関連付けするための文字列
2. メニューアイテムが選択されたときに呼び出されるコールバック関数

3. パラメータタイプ
4. 状態
5. メニューアイテムの状態が変わったときに呼び出されるコールバック関数

前項で定義したメニューおよびメニューアイテムの場合は，次のように詳細を設定します．

```
static GActionEntry entries[] = {
  {"open", cb_open, NULL, NULL, NULL},
  {"quit", cb_quit, NULL, NULL, NULL},
};
```

特別な処理をしない限りは第 3 から第 5 メンバは設定しなくても問題ありませんので, 今回はすべて NULL としています．

2.7.4 コールバック関数の登録

次に GActionEntry 構造体で記述したコールバック関数をアプリケーションと関連付けした GActionMap に登録します．登録を行うには関数 g_action_map_add_action_entries を使用します．今回のように GtkApplication を使っている場合は, GtkApplication 自身を GActioMap として使用することができます（ソース 2–6 132–134 行目）．

```
void g_action_map_add_action_entries (GActionMap         *action_map,
                                      const GActionEntry *entries,
                                      gint               n_entries,
                                      gpointer           user_data);
```

第 1 引数	GActionMap 型の変数へのポインタ
第 2 引数	GActionEntry で定義されたコールバック関数情報
第 3 引数	GActionEntry のエントリ数
第 4 引数	コールバック関数に渡すデータ

2.7.5 メニューバーウィジェットの作成

最後に，GtkBuiler を用いてメニュー定義情報からメニューバーウィジェットを作成して, アプリケーションに配置します．

メニュー定義情報がファイルの場合は, 関数 gtk_builder_add_from_file を使用します．また, メニュー定義情報が文字列の場合は, 関数 gtk_builder_add_from_string を使用します．以下は, GtkBuilder に文字列で定義したメニュー情報を追加する例になります．

```
  GtkBuilder *builder = gtk_builder_new ();
  gtk_builder_add_from_string (builder, menu_info, -1, NULL);
```

関数 gtk_builder_add_from_string は，文字列からメニュー定義情報を登録する関数です．第 3 引数にはメニュー構成を示す文字列の長さを指定しますが，文字列全体を使用する場合には −1 を指定します．

```
guint gtk_builder_add_from_string (GtkBuilder  *builder,
                                   const gchar *buffer,
                                   gsize       length,
                                   GError      **error);
```

第 1 引数	GtkBuilder 型の変数
第 2 引数	メニュー構成文字列
第 3 引数	メニュー構成文字列の長さ
第 4 引数	エラー情報を格納する変数へのポインタ
戻り値	成功した場合には正の数, エラーの場合には 0

2.7.6　メニューバーウィジェットの取得

すべてのメニュー情報を GtkBuilder に登録したら，関数 gtk_builder_get_object を使用して，メニューバーウィジェットを取得します．

```
GObject * gtk_builder_get_object (GtkBuilder  *builder,
                                  const gchar *name);
```

第 1 引数	GtkBuilder 型の変数
第 2 引数	識別文字列
戻り値	取得したオブジェクト

最後に，以下に示すように関数 gtk_builder_get_object で取得したオブジェクトを **GMenuModel** 型のポインタに変換して，関数 gtk_application_set_menubar を使ってアプリケーションに配置します．

```
GMenuModel *menubar;
menubar = (GMenuModel *) gtk_builder_get_object (builder, "appmenu");
gtk_application_set_menubar (GTK_APPLICATION (app), menubar);
```

2.7.7　コンパイルと動作確認

ソースコードを入力し，コンパイルしたら，いつものように実行してみましょう．

```
$ ./image-viewer ~/images/Parrots.png
```

図 2.11 のようにメニューバーが表示され，メニューやショートカットキーを利用してプログラムを終了できるようになりました．プログラムを終了する際にはコールバック関数（ソース 2–6 18–26 行目）が呼び出されますが，関数の第 3 引数の user_data には GtkApplication 型の変数が与えられています．これは，ソース 2–6 122–124 行目でコールバック関数を GActionMap に登録する際に，関数の第 3 引数に GtkApplication 型の変数 app を指定しているためです．

図 2.11　メニューバーを追加した画像ビューワ

ソース 2–6 メニューバーの追加：image-viewer.c

```
1 #include <gtk/gtk.h>
2 #include <stdlib.h>
3
4 /*
5   Openメニューが選択されたときに呼び出される関数
6 */
```

```
7  static void
8  cb_open (GSimpleAction  *action,
9           GVariant       *paramter,
10          gpointer       user_data)
11 {
12   g_print ("This␣function␣is␣not␣implemented␣yet.\n");
13 }
14
15 /*
16   ボタンがクリックされたときに呼び出される関数
17 */
18 static void
19 cb_quit (GSimpleAction  *action,
20          GVariant       *parameter,
21          gpointer       user_data)
22 {
23   /* メインループを終了 */
24   GApplication *app = G_APPLICATION (user_data);
25   g_application_quit (app);
26 }
27
28 /*
29  * メニューの構造
30  */
31 static const gchar menu_info[] =
32   "<interface>"
33   "␣␣<menu␣id='appmenu'>"
34   "␣␣␣␣<submenu>"
35   "␣␣␣␣␣␣<attribute␣name='label'>File</attribute>"
36   "␣␣␣␣␣␣<section>"
37   "␣␣␣␣␣␣␣␣<item>"
38   "␣␣␣␣␣␣␣␣␣␣<attribute␣name='label'>Open</attribute>"
39   "␣␣␣␣␣␣␣␣␣␣<attribute␣name='action'>app.open</attribute>"
40   "␣␣␣␣␣␣␣␣␣␣<attribute␣name='accel'>&lt;Primary&gt;o</attribute>"
41   "␣␣␣␣␣␣␣␣</item>"
42   "␣␣␣␣␣␣</section>"
43   "␣␣␣␣␣␣<section>"
44   "␣␣␣␣␣␣␣␣<item>"
45   "␣␣␣␣␣␣␣␣␣␣<attribute␣name='label'>Quit</attribute>"
46   "␣␣␣␣␣␣␣␣␣␣<attribute␣name='action'>app.quit</attribute>"
47   "␣␣␣␣␣␣␣␣␣␣<attribute␣name='accel'>&lt;Primary&gt;q</attribute>"
48   "␣␣␣␣␣␣␣␣</item>"
49   "␣␣␣␣␣␣</section>"
50   "␣␣␣␣</submenu>"
51   "␣␣</menu>"
52   "</interface>";
53
54 /*
55  * メニューアイテムの詳細
56  */
57 static GActionEntry entries[] = {
58   {"open", cb_open, NULL, NULL, NULL},
59   {"quit", cb_quit, NULL, NULL, NULL},
60 };
61
62 /*
63  * アプリケーションが有効になったときに呼び出される関数
64  */
65 static void cb_activate (GApplication   *app,
66                          gpointer       user_data)
67 {
68   GtkWidget *window;
69
70   /* ウィンドウの作成 */
71   window = gtk_application_window_new (GTK_APPLICATION (app));
72   /* ウィンドウの大きさの設定 */
73   gtk_widget_set_size_request (window, 300, 200);
74   /* メニューの作成 */
75   GtkBuilder *builder;
76   builder = gtk_builder_new ();
77   gtk_builder_add_from_string (builder, menu_info, -1, NULL);
78   GMenuModel *menubar;
79   menubar = (GMenuModel *) gtk_builder_get_object (builder, "appmenu");
80   gtk_application_set_menubar (GTK_APPLICATION (app), menubar);
81   {
```

```
82      /* スクロールバー付きウィンドウの作成 */
83      GtkWidget *scroll_window;
84      scroll_window = gtk_scrolled_window_new (NULL, NULL);
85      /* スクロールバー付きウィンドウを配置 */
86      gtk_container_add (GTK_CONTAINER (window), scroll_window);
87      /* スクロールバーの表示設定 */
88      gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
89                                      GTK_POLICY_AUTOMATIC,
90                                      GTK_POLICY_AUTOMATIC);
91      {
92        GtkWidget *image;
93
94        /* ファイルから画像を読み込んでイメージの作成 */
95        image = gtk_image_new_from_file ((char *) user_data);
96        /* イメージをスクロールバー付きウィンドウに配置 */
97        gtk_container_add (GTK_CONTAINER(scroll_window), image);
98      }
99    }
100   /* ウィンドウの表示 */
101   gtk_widget_show_all (window);
102 }
103
104 /*
105   メイン関数
106 */
107 int
108 main (int argc, char **argv)
109 {
110   GtkApplication *app;
111   GtkWidget *window;
112
113   /* 引数のチェック */
114   if (argc != 2)
115     {
116       g_print ("Usage:␣%s␣image-file\n", argv[0]);
117       exit (1);
118     }
119   /* アプリケーションの作成 */
120   app = gtk_application_new ("test.gtk.imageviewer", 0);
121   /* アクションの登録 */
122   g_action_map_add_action_entries (G_ACTION_MAP (app),
123                                    entries, G_N_ELEMENTS (entries),
124                                    app);
125   /* シグナルハンドラの登録 */
126   g_signal_connect (app, "activate", G_CALLBACK(cb_activate), argv[1]);
127   /* メインループ */
128   g_application_run (G_APPLICATION(app), 0, NULL);
129
130   return 0;
131 }
```

2.7.8　まとめ

GtkBuilder を利用することで簡単にメニューが作成できるとはいえ，メニューを作成するまでにさまざまな手順が必要でした．メニュー作成の手順をもう一度簡単にまとめておきます．

1. メニュー構成の記述
   今回は文字列で定義しましたが，テキストファイルに定義する方法もあります．
2. コールバック関数の記述
3. コールバック関数をアプリケーションに登録
4. メニューバーの取得とアプリケーションへの配置

2.8　ファイル選択ダイアログの実装

本チュートリアルの最後のステップとして，**ファイル選択ダイアログ**から画像ファイルを指定して画像を表示できるようにしましょう．大変難しそうに思えますが，GTK+ ではファイル選択用のダイアログが用意されており，選択したファイルを開いたり，指定したファイル名でデータを保存したりといった目的に合わせて，簡単にダイアログを作成することができます．

2.8.1 ファイル選択ダイアログの作成

ファイル選択ダイアログを作成する関数は gtk_file_chooser_dialog_new です．選択した画像ファイルを開いて表示するダイアログは，次のように作成します．

```
GtkWidget *dialog;
dialog = gtk_file_chooser_dialog_new ("Open an image",
                                      GTK_WINDOW (window),
                                      GTK_FILE_CHOOSER_ACTION_OPEN,
                                      "_Cancel", GTK_RESPONSE_CANCEL,
                                      "_Open", GTK_RESPONSE_ACCEPT,
                                      NULL);
```

関数 gtk_file_chooser_dialog_new の引数は，以下の通りです．

```
GtkWidget* gtk_file_chooser_dialog_new (const gchar *title,
                                        GtkWindow *parent,
                                        GtkFileChooserAction action,
                                        const gchar *first_button_text,
                                        ...);
```

第 1 引数	ダイアログのタイトル
第 2 引数	親ウィンドウ
第 3 引数	ダイアログの種類（開くか保存か）
第 4 引数	追加するボタンのアイコン
第 5 引数	ボタンを押したときの応答 ID
戻り値	生成したファイル選択ダイアログ

この関数の第 2 引数には，ダイアログの呼び出し元になる親ウィンドウを指定します．ダイアログを操作している間は，このウィンドウに対する操作ができない状態になります．第 3 引数には，このダイアログの種類を指定します．今回はファイルを開く目的なので，GTK_FILE_CHOOSER_ACTION_OPEN を指定しています．指定したファイル名で保存するダイアログを作成するならば，GTK_FILE_CHOOSER_ACTION_SAVE を指定します．

第 4，5 引数以降は 2 つ 1 組で与え，それぞれ追加するボタンのラベルと，どのボタンが押されたかを知るための応答 ID を指定します．この例では，「キャンセル」ボタンと「開く」ボタンをダイアログに追加して，「キャンセル」が押されたときに GTK_RESPONSE_CANCEL という ID が，「開く」では GTK_RESPONSE_ACCEPT が返ってくるように設定しています．

引数の終わりには必ず NULL を与えます．

2.8.2 ファイル名の取得と画像の表示

ダイアログを作成したら，次にそのダイアログを表示してファイルを選択する処理を行います．ファイル選択処理を開始するには，関数 gtk_dialog_run を呼び出します．

```
gint response;
response = gtk_dialog_run (GTK_DIALOG (dialog));
```

ファイルの選択処理が終わると（ファイルを選択して「開く」ボタンを押すか，「キャンセル」ボタンを押した場合に処理が終了する），この関数の戻り値として，ダイアログ作成時に設定した応答 ID が返ってきます．

```
gint gtk_dialog_run (GtkDialog *dialog);
```

第 1 引数	ダイアログ
戻り値	応答 ID

ダイアログでファイルを選択した場合（関数 gtk_dialog_run の戻り値として GTK_RESPONSE_ACCEPT が返ってきた場合），選択したファイル名を取得し，イメージウィジェットを作成して表示します．
まず次のように，選択したファイル名を取得します．

```
gchar *filename;
filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
```

そして，イメージウィジェットの機能を使用して，指定した画像ファイルから画像を表示します．

```
gtk_image_set_from_file (GTK_IMAGE (image), filename);
```

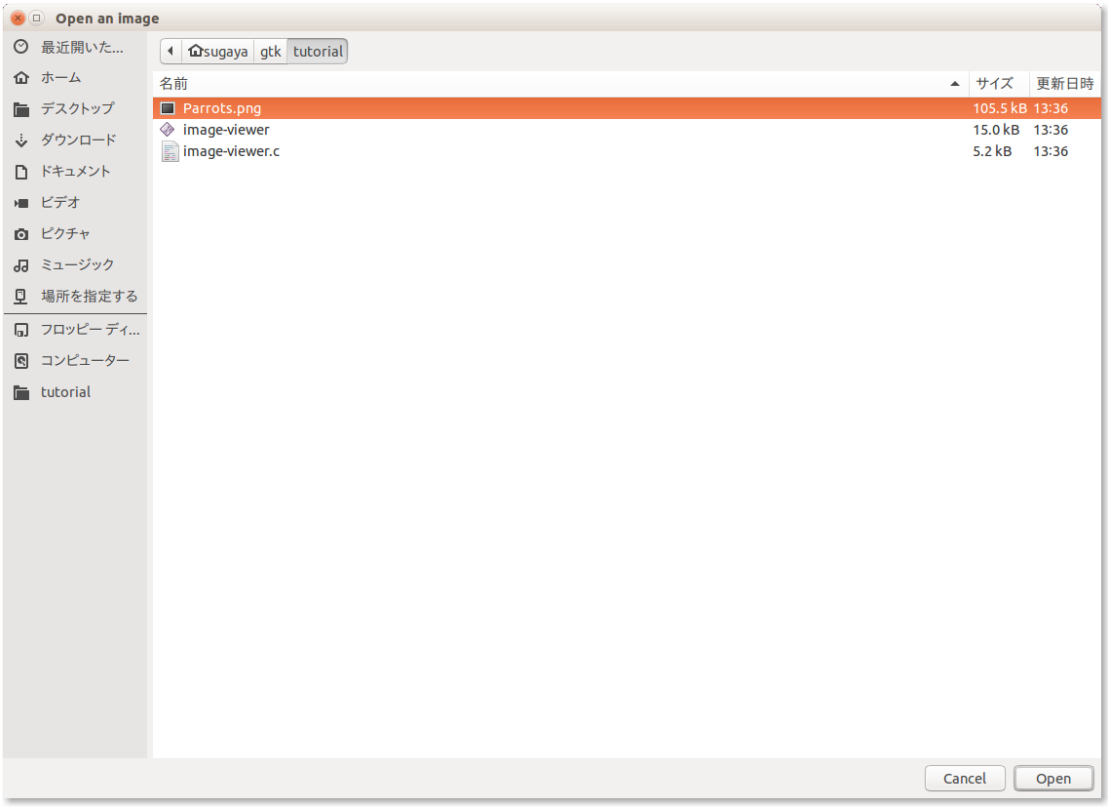
最後に，関数 gtk_file_chooser_get_filename によって取得した文字列のためのメモリ領域を，関数 g_free で解放します．

2.8.3　コンパイルと動作確認

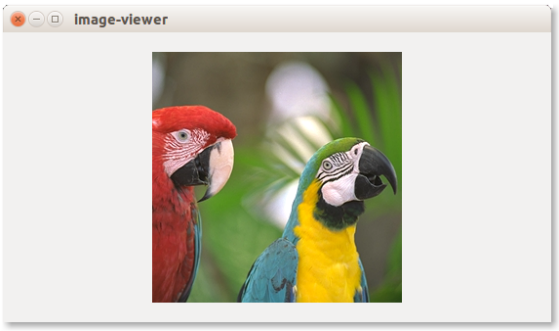
ソース 2–7 を入力して，コンパイルしてプログラムを完成させてみましょう．
今回の実装でダイアログからファイルを指定して画像を表示させられるようになったので，プログラムの実行時にファイル名を指定する必要がなくなりました．それにともない，ソースコードの 131 行目で，関数 gtk_image_new を使用して，空のイメージウィジェットを作成するようにしています．
プログラムを実行したら，早速ファイル選択ダイアログからファイルを選択して，画像を表示してみましょう．図 2.12 のように，ダイアログから選択した画像を表示できたでしょうか．

ソース 2–7　画像ビューワ完成版：image-viewer.c

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

/*
  Openメニューが選択されたときに呼び出される関数
*/
static void
cb_open (GSimpleAction  *action,
         GVariant       *paramter,
         gpointer       user_data)
{
  GtkApplication *app;
  GtkWindow      *window;
  GtkWidget      *dialog;
  gint           response;

  /* ウィンドウの取得 */
  app = GTK_APPLICATION (user_data);
  window = gtk_application_get_active_window (app);
  /* ファイル選択ダイアログの作成 */
  dialog = gtk_file_chooser_dialog_new ("Open␣an␣image",
                                        GTK_WINDOW (window),
                                        GTK_FILE_CHOOSER_ACTION_OPEN,
                                        "_Cancel",
                                        GTK_RESPONSE_CANCEL,
                                        "_Open",
                                        GTK_RESPONSE_ACCEPT,
                                        NULL);
  /* ダイアログの表示 */
  gtk_widget_show_all (dialog);
  /* ダイアログによるファイル選択処理 */
  response = gtk_dialog_run (GTK_DIALOG (dialog));
  if (response == GTK_RESPONSE_ACCEPT)
    {
      gchar     *filename;
      GtkWidget *image;

      /* 選択したファイル名の取得 */
      filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
```



(a) ファイル選択ダイアログ



(b) 選択した画像の表示

図 2.12　完成した画像ビューワ

```
40        /* イメージの取得 */
41        image = GTK_WIDGET (g_object_get_data (G_OBJECT (window), "image"));
42        /* ファイルから画像を読み込んでイメージにセット */
43        gtk_image_set_from_file (GTK_IMAGE (image), filename);
44        /* 文字列領域の解放 */
45        g_free (filename);
46      }
47    /* ダイアログの破棄 */
48    gtk_widget_destroy (dialog);
49  }
50
51  /*
52    ボタンがクリックされたときに呼び出される関数
53  */
54  static void
55  cb_quit (GSimpleAction  *action,
56           GVariant       *parameter,
57           gpointer       user_data)
```

```
58 {
59   /* メインループを終了 */
60   GApplication *app = G_APPLICATION (user_data);
61   g_application_quit (app);
62 }
63 
64 /*
65  * メニューの構造
66  */
67 static const gchar menu_info[] =
68   "<interface>"
69   "␣␣<menu␣id='appmenu'>"
70   "␣␣␣␣<submenu>"
71   "␣␣␣␣␣␣␣<attribute␣name='label'>File</attribute>"
72   "␣␣␣␣␣␣␣<section>"
73   "␣␣␣␣␣␣␣␣␣␣<item>"
74   "␣␣␣␣␣␣␣␣␣␣␣␣<attribute␣name='label'>Open</attribute>"
75   "␣␣␣␣␣␣␣␣␣␣␣␣<attribute␣name='action'>app.open</attribute>"
76   "␣␣␣␣␣␣␣␣␣␣␣␣<attribute␣name='accel'>&lt;Primary&gt;o</attribute>"
77   "␣␣␣␣␣␣␣␣␣␣</item>"
78   "␣␣␣␣␣␣␣</section>"
79   "␣␣␣␣␣␣␣<section>"
80   "␣␣␣␣␣␣␣␣␣␣<item>"
81   "␣␣␣␣␣␣␣␣␣␣␣␣<attribute␣name='label'>Quit</attribute>"
82   "␣␣␣␣␣␣␣␣␣␣␣␣<attribute␣name='action'>app.quit</attribute>"
83   "␣␣␣␣␣␣␣␣␣␣␣␣<attribute␣name='accel'>&lt;Primary&gt;q</attribute>"
84   "␣␣␣␣␣␣␣␣␣␣</item>"
85   "␣␣␣␣␣␣␣</section>"
86   "␣␣␣␣</submenu>"
87   "␣␣</menu>"
88   "</interface>";
89 
90 /*
91  * メニューアイテムの詳細
92  */
93 static GActionEntry entries[] = {
94   {"open", cb_open, NULL, NULL, NULL},
95   {"quit", cb_quit, NULL, NULL, NULL},
96 };
97 
98 /*
99  * アプリケーションが有効になったときに呼び出される関数
100  */
101 static void cb_activate (GApplication   *app,
102                          gpointer       user_data)
103 {
104   GtkWidget *window;
105 
106   /* ウィンドウの作成 */
107   window = gtk_application_window_new (GTK_APPLICATION (app));
108   /* ウィンドウの大きさの設定 */
109   gtk_widget_set_size_request (window, 300, 200);
110   /* メニューの作成 */
111   GtkBuilder *builder;
112   builder = gtk_builder_new ();
113   gtk_builder_add_from_string (builder, menu_info, -1, NULL);
114   GMenuModel *menubar;
115   menubar = (GMenuModel *) gtk_builder_get_object (builder, "appmenu");
116   gtk_application_set_menubar (GTK_APPLICATION (app), menubar);
117   {
118     /* スクロールバー付きウィンドウの作成 */
119     GtkWidget *scroll_window;
120     scroll_window = gtk_scrolled_window_new (NULL, NULL);
121     /* スクロールバー付きウィンドウを配置 */
122     gtk_container_add (GTK_CONTAINER (window), scroll_window);
123     /* スクロールバーの表示設定 */
124     gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
125                                     GTK_POLICY_AUTOMATIC,
126                                     GTK_POLICY_AUTOMATIC);
127     {
128       GtkWidget *image;
129 
130       /* ファイルから画像を読み込んでイメージの作成 */
131       image = gtk_image_new ();
132       /* ウィンドウにイメージをデータとしてセット */
```

```
133         g_object_set_data (G_OBJECT (window), "image", (gpointer) image);
134         /* イメージをスクロールバー付きウィンドウに配置 */
135         gtk_container_add (GTK_CONTAINER(scroll_window), image);
136       }
137     }
138     /* ウィンドウの表示 */
139     gtk_widget_show_all (window);
140 }
141
142 /*
143     メイン関数
144 */
145 int
146 main (int argc, char **argv)
147 {
148     GtkApplication *app;
149
150     /* アプリケーションの作成 */
151     app = gtk_application_new ("test.gtk.imageviewer", 0);
152     /* アクションの登録 */
153     g_action_map_add_action_entries (G_ACTION_MAP (app),
154                                      entries, G_N_ELEMENTS (entries),
155                                      app);
156     /* シグナルハンドラの登録 */
157     g_signal_connect (app, "activate", G_CALLBACK(cb_activate), NULL);
158     /* メインループ */
159     g_application_run (G_APPLICATION(app), 0, NULL);
160
161     return 0;
162 }
```

2.8.4 まとめ

チュートリアル最後となる本節では，ファイル選択ダイアログを実装して，ダイアログから指定した画像を表示できるようにしました．今回解説した内容は以下の通りです．

- ファイル選択ダイアログの作成
- ダイアログからのファイル名の取得

このチュートリアルもこれで終了です．単純なウィンドウの作成に始まり，最終的にはメニューからダイアログを表示させ，指定した画像を表示する画像ビューワを完成させました．それぞれの節で説明を省略しているので，完全には内容を理解できなかったかもしれませんが，このようなアプリケーションを比較的簡単に作成できることがわかってもらえたのではないでしょうか．

次の章からは，GTK+ や関連する内容について詳しく説明していきます．チュートリアルでは語りきれなかった GTK+ の魅力がたっぷりと詰まっています．是非読み進めてください．

3 もっと GTK+

3.1 ウィジェットの階層構造

前章でも簡単に説明したように，ウィジェットは C++ のクラスのような構造を持ちます．すべてのウィジェットは GObject 型を基本型として，階層構造を持っています．そのため，あるウィジェットから派生したウィジェットは，その元となったウィジェットのメンバやシグナルを持ちます．

ウィジェットの階層構造の一部を図 3.1 に示します．すべてのウィジェットの階層構造を知るには GTK Reference Manual の Object Hierarchy （https://developer.gnome.org/gtk3/3.10/ch02.html）を参照してください．

また，派生したウィジェットには，元となったウィジェットに対する関数を適用できます．このときには，階層関係にあるウィジェット間で，型変換を行って関数を適用することになります．この型変換を行うために，マクロが用意されています．

例えば前章では，GtkWindow 型の変数をその上位の GtkContainer 型に変換するために，マクロ GTK_CONTAINER を使用しました．このように型変換のマクロは，変換後のウィジェット型名を大文字にして '_'（アンダースコア）で区切った形で定義されています．

3.2 ウィジェットの配置

GUI アプリケーションを作成するとき，ボタンやスライダなどのウィジェットをウィンドウ内にどう配置するかが重要になります．GTK+ ではウィジェットを配置するためのウィジェットに他のウィジェットを配置することで，簡単に，そして，見た目のよい GUI を作れます．

本節では，ウィジェットの配置について説明します．ここではウィジェットの配置方法として代表的な 3 つの方法（コンテナ，パッキングボックス，テーブル）を挙げて，具体例とともに解説します．

3.2.1 コンテナ

コンテナとは，ウィジェットを 1 つだけ配置できるウィジェットを表します．GtkWindow などがコンテナウィジェットです．コンテナにウィジェットを配置するには，関数 gtk_container_add を使います．第 1 引数はコンテナウィジェット，第 2 引数は配置するウィジェットを指定します．

コンテナのボーダー幅

コンテナにウィジェットを配置するときに，コンテナとウィジェットの間にどれだけのすき間（**ボーダー幅**）を空けるかを指定できます．図 3.2 の矢印で示した空白がボーダー幅になります．ボーダー幅の設定には関数 gtk_container_set_border_width を使用します．

```
GObject
 + -- GInitiallyUnowned
       + -- GtkObject
             + -- GtkWidget
                   + -- GtkContainer
                         + -- GtkBin
                               + -- GtkWindow
                               |     + -- GtkDialog
                               |     |     + -- GtkAboutDialog
                               |     |     + -- GtkAppChooserDialog
                               |     |     + -- GtkColorChooserDialog
                               |     |     + -- GtkColorSelectionDialog
                               |     |     + -- GtkFileChooserDialog
                               |     |     + -- GtkFileSelection
                               |     |     + -- GtkFontChooserDialog
                               |     |     + -- GtkFontSelectionDialog
                               |     |     + -- GtkMessageDialog
                               |     |     + -- GtkPageSetupUnixDialog
                               |     |     + -- GtkPrintUnixDialog
                               |     |     + -- GtkRecentChooserDialog
                               |     + -- GtkApplicationWindow
                               |     + -- GtkAssistant
                               |     + -- GtkOffscreenWindow
                               + -- GtkAlignment
                               + -- GtkComboBox
                               |     + -- GtkAppChooserButton
                               |     + -- GtkComboBoxText
                               + -- GtkFrame
                               |     + -- GtkAspectFrame
                               + -- GtkButton
                               |     + -- GtkToggleButton
                               |     |     + -- GtkCheckButton
                               |     |           + -- GtkRadioButton
                               |     + -- GtkColorButton
                               |     + -- GtkFontButton
                               |     + -- GtkLinkButton
                               |     + -- GtkLockButton
                               |     + -- GtkScaleButton
                               |           + -- GtkVolumeButton
                               + -- GtkMenuItem
                               |     + -- GtkCheckMenuItem
                               |     |     + -- GtkRadioMenuItem
                               |     + -- GtkImageMenuItem
                               |     + -- GtkSeparatorMenuItem
                   ...
```

図 3.1　ウィジェットの階層構造

```
void gtk_container_set_border_width (GtkContainer *container,
                                     gint         border_width);
```

第 1 引数　コンテナウィジェット
第 2 引数　ボーダー幅

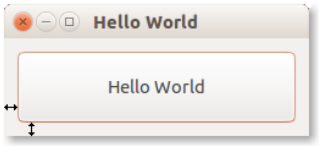


図 3.2　コンテナウィジェットのボーダー幅

3.2.2 パッキングボックス

コンテナはウィジェットを 1 つしか配置できないので，コンテナだけでは複雑な GUI を作れません．次に紹介するパッキングボックスはウィジェットを複数並べて配置できるウィジェットです．パッキングボックスにはウィジェットを水平に配置する**水平ボックス**とウィジェットを垂直に配置する**垂直ボックス**が存在します．

パッキングボックスの作成

パッキングボックスを作成するには，関数 gtk_box_new を使用します．第 1 引数にウィジェットを並べる方向を，第 2 引数には配置されるウィジェット間にどれだけのすき間を空けるかを指定します．

```
GtkWidget* gtk_box_new (GtkOrientation orientation, gint spacing);
```

第 1 引数	配置する子ウィジェットの方向を指定する
第 2 引数	子ウィジェット間に空けるスペース
戻り値	生成したボックスウィジェット

ウィジェットの配置

パッキングボックスにウィジェットを配置するには，関数 gtk_box_pack_start を使用します．この関数では，ウィジェットを左から右（もしくは上から下）に配置します．引数は次のようになっています．

```
void gtk_box_pack_start (GtkBox     *box,
                         GtkWidget  *child,
                         gboolean   expand,
                         gboolean   fill,
                         guint      padding);
```

第 1 引数	パッキングボックス
第 2 引数	配置するウィジェット
第 3 引数	ボックスを与えられた領域いっぱいに広げるかどうか．TRUE または FALSE を指定する．
第 4 引数	ウィジェットを与えられた領域いっぱいに広げるかどうか．TRUE または FALSE を指定する．
第 5 引数	ボックスの両端にどれだけすき間を空けるか

同じような関数に gtk_box_pack_end があります．こちらは関数 gtk_box_pack_start と反対に，ウィジェットを右から左（もしくは下から上）へ配置していきます．

これらの関数では，第 3 と第 4 引数の与え方によって，GUI の見た目が変化します．次に具体例を挙げて，引数の与え方と見た目の変化を見てみましょう．

配置したウィジェットの見え方

関数 gtk_box_pack_start に与えるパラメータ expand と fill の値の組み合わせを，表 3.1 にまとめました．fill パラメータは，expand パラメータが TRUE のときに有効だということに注意してください．このため値の組み合わせは，3 パターンになります．

実際にこの 3 パターンを配置したのが，図 3.3 です．これは，ソース 3–1 のソースコードをコンパイルして実行した結果です．垂直ボックス中に水平ボックスを 3 つ配置して，それぞれの水平ボックスが表 3.1 の各行に対応するように expand と fill の値を変え，同じ値のボタンを 2 つ左右に並べて配置しました．ボタンの大きさや配置が，各行で異なるのがわかるでしょう．

expand と fill をともに TRUE とした場合には，図 3.3 の 1 行目を見てわかるとおり，パッキングボックスもボタンウィジェットも領域全体に広がって配置されます．

2 行目は expand が TRUE ですが，fill は FALSE です．パッキングボックスは広がっていますが，ボタンウィジェットは広がっていません．最後に 3 行目では，両方が FALSE なので，2 つのウィジェットは最小限の大きさに留まっています．

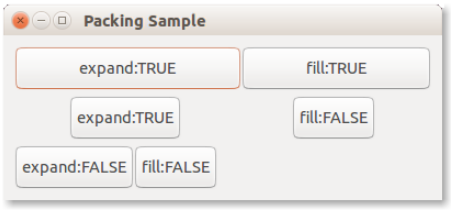


図 3.3　ウィジェットの配置方法

表 3.1　gtk_box_pack_start に与えるパラメータの組み合わせ

	expand	fill
第 1 行	TRUE	TRUE
第 2 行	TRUE	FALSE
第 3 行	FALSE	FALSE

ソース 3–1 ウィジェットの配置：packing-sample.c

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

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

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "Packing␣Sample");
  gtk_container_set_border_width(GTK_CONTAINER (window), 10);
  gtk_widget_set_size_request (window, 400, 150);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 5);
  gtk_container_add (GTK_CONTAINER (window), vbox);

  hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
  gtk_box_pack_start (GTK_BOX (vbox), hbox, TRUE, TRUE, 0);
  button = gtk_button_new_with_label ("expand:TRUE");
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
  button = gtk_button_new_with_label ("fill:TRUE");
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);

  hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
  gtk_box_pack_start (GTK_BOX (vbox), hbox, TRUE, TRUE, 0);
  button = gtk_button_new_with_label ("expand:TRUE");
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, FALSE, 0);
  button = gtk_button_new_with_label ("fill:FALSE");
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, FALSE, 0);

  hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
  gtk_box_pack_start (GTK_BOX (vbox), hbox, TRUE, TRUE, 0);
  button = gtk_button_new_with_label ("expand:FALSE");
  gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
  button = gtk_button_new_with_label ("fill:FALSE");
  gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);

  gtk_widget_show_all (window);
  gtk_main ();
  return 0;
}
```

3.2.3 グリッド

ウィジェットを規則的にマス目状に配置したい場合には，**グリッドウィジェット（GtkGrid）**を用いると便利です．以前は GtkTable というウィジェットがありましたが，GTK+ のバージョン 3.4 からは GtkTable ウィジェットの代わりに GtkGrid ウィジェットが使われるようになり，GtkTable ウィジェットよりも扱いが簡単になりました．

グリッドウィジェットを作成するには，関数 gtk_grid_new を使用します．

```
GtkWidget* gtk_grid_new (void);
```

戻り値　生成したグリッドウィジェット

グリッドへの配置

グリッドウィジェットにウィジェットを配置するには関数 gtk_grid_attach を使用します．関数の引数の説明を次にまとめます．

```
void gtk_grid_attach (GtkGrid   *grid,
                      GtkWidget *child,
                      gint      left,
                      gint      top,
                      gint      width,
                      gint      height);
```

第 1 引数	グリッドウィジェット
第 2 引数	グリッドに配置するウィジェット
第 3 引数	ウィジェットを配置する列番号
第 4 引数	ウィジェットを配置する行番号
第 5 引数	配置するウィジェットの幅
第 6 引数	配置するウィジェットの高さ

関数の第 3 引数から第 6 引数の設定の仕方を図 3.4 を参照しながら説明します．図中の各ボタンのラベルとして表示されている数値がそのボタンウィジェットを配置するときに関数 gtk_grid_attach の第 3 引数から第 6 引数に与えた値になります．

1 行目はそれぞれ幅 1，高さ 1 でウィジェットを配置した結果です．2 行目は右側のウィジェットを幅 2 で配置したもので，1 行目のウィジェット配置と比較すると 2 ウィジェット分の領域を占めていることがわかります．左下に配置したウィジェットは幅 2，高さ 2 で配置しているため，横方向も縦方向も 2 ウィジェット分の領域を占めています．

以下の記述は，2 行目のウィジェットを配置するときのものになります．

```
button = gtk_button_new_with_label ("(0, 1, 1, 1)");
gtk_grid_attach (GTK_GRID (grid), button, 0, 1, 1, 1);
button = gtk_button_new_with_label ("(1, 1, 2, 1)");
gtk_grid_attach (GTK_GRID (grid), button, 1, 1, 2, 1);
```

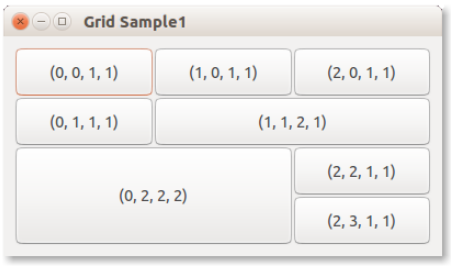


図 3.4　グリッドウィジェットへの配置 その 1

次に, ウィジェットの均等配置と配置するウィジェット間のスペースについて説明します. 配置するウィジェットの幅を均等にするには関数 gtk_grid_set_column_homogeneous を使用します. また 配置するウィジェットの高さを均等にするには関数 gtk_grid_set_rpw_homogeneous を使用します. それぞれの関数の第 2 引数に TRUE を指定すると均等配置, FALSE にすると異なる幅もしくは高さになります.

```
void gtk_grid_set_column_homogeneous (GtkGrid  *grid,
                                      gboolean homogeneous);
```

第 1 引数　グリッドウィジェット
第 2 引数　均等配置なら TRUE, そうでなければ FALSE

```
void gtk_grid_set_row_homogeneous (GtkGrid  *grid,
                                   gboolean homogeneous);
```

第 1 引数　グリッドウィジェット
第 2 引数　均等配置なら TRUE, そうでなければ FALSE

配置するウィジェット間のスペースを設定するには, 関数 gtk_grid_set_column_spacing と関数 gtk_grid_set_row_spacing を使用します.

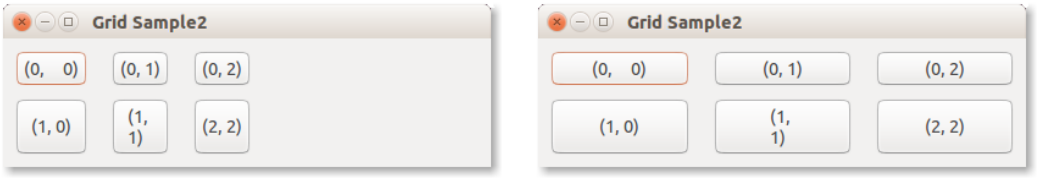
```
void gtk_grid_set_column_spacing (GtkGrid  *grid,
                                  guint    spacing);
```

第 1 引数　グリッドウィジェット
第 2 引数　横方向のウィジェット間のスペース

```
void gtk_grid_set_row_spacing (GtkGrid  *grid,
                               guint    spacing);
```

第 1 引数　グリッドウィジェット
第 2 引数　縦方向のウィジェット間のスペース

図 3.5 は, 水平, 垂直方向のウィジェット間スペースをそれぞれ 20, 10 に設定し, ウィジェットの均等配置の設定をそれぞれ変えたときの違いを示したものです. 図 3.5(b), (d) は水平方向のウィジェット配置を均等に, 図 3.5(c), (d) は垂直方向のウィジェット配置を均等にしています.



(a) (水平, 垂直) = (FALSE, FALSE)　(b) (水平, 垂直) = (TRUE, FALSE)

(c) (水平, 垂直) = (FALSE, TRUE)　(d) (水平, 垂直) = (TRUE, TRUE)

図 3.5　グリッドウィジェットへの配置 その 2

3.3 シグナルとコールバック関数の詳細

本節では具体例を示しながら，シグナルとコールバック関数についてもう少し詳しく説明します．

3.3.1 シグナルとコールバック関数

ウィジェット配置の問題は，見た目の良い GUI を作成するのに非常に重要な要素ですが，イベントとコールバック関数は，実際にアプリケーションを動作させる上で重要な要素です．チュートリアルでも簡単に説明しましたが，ウィジェットに対する操作を**イベント**と呼び，そのイベントが発生したときに実行する関数を**コールバック関数**と呼びます．

GTK+ では，ウィジェットに対してそれぞれイベントが定義されています．イベントが起こった場合には，それに割り当てられた**シグナル**が発生します．プログラマが各シグナルが発生したときに呼び出すコールバック関数を登録しておくことで，ユーザの操作に応じて，決められた処理を行うことができます．


第 10 章で説明する Glade を使うとシグナルを確認できます．

シグナルとコールバック関数を関連付ける

具体的にシグナルとコールバック関数を関連付けるには，関数 g_signal_connect を使用します．チュートリアルのソース 2–2（p. 14）の 35–36 行目では，次のようにボタンウィジェットのコールバック関数を登録しています．

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

それぞれの引数は次のようになっています．

第 1 引数	コールバック関数を関連付けるオブジェクト
第 2 引数	シグナル名
第 3 引数	コールバック関数
第 4 引数	コールバック関数に渡すデータ

コールバック関数の書式

ここで関連付けられるコールバック関数は，一般に次のような書式をしています．

```
void function_name (GtkWidget *widget, gpointer data);
```

コールバック関数の第 1 引数は，シグナルを発生したウィジェットを指します．

第 2 引数は，関数 g_signal_connect の第 4 引数に指定したデータになります．関数 g_signal_connect の第 4 引数には int 型や char 型などのさまざまなデータを与えることができますが，その際にデータを **gpointer** 型にキャストするのを忘れないようにしてください．

4 章の表 4.1 （p. 51）で説明していますが，gpointer 型は gtypes.h で次のように定義されています．

```
typedef void* gpointer;
```

なお，コールバック関数は上記の書式だけではなく，シグナルの種類によってさまざまな書式があります．


シグナルに対するコールバック関数の書式や，それぞれのウィジェットにどんなシグナルが定義されているかを確認するには，devhelp を使用するといいでしょう．

3.3.2 コールバック関数の設定

コールバック関数を設定する中心的な関数は g_signal_connect_data です．この関数のプロトタイプ宣言は次のようになっています．

```
gulong g_signal_connect_data (gpointer       instance,
                              const gchar    *detailed_signal,
                              GCallback      c_handler,
                              gpointer       data,
```

```
                                GClosureNotify destroy_data,
                                GConnectFlags  connect_flags);
```

第 1 引数	コールバック関数を関連付けるオブジェクト
第 2 引数	シグナル名
第 3 引数	コールバック関数
第 4 引数	コールバック関数に渡すデータ
第 5 引数	コールバック関数が呼び出されなくなったときに呼び出される関数
第 6 引数	コールバック関数のフラグ
戻り値	コールバック関数を特定する ID

この関数の戻り値は，設定したコールバック関数を特定する ID として使用され，一度設定したコールバック関数を解除する際などに使用されます．

第 6 引数のフラグ GConnectFlags は，表 3.2 のように定義されています．このフラグの値によってコールバック関数の振る舞いが変わってきます．これについては後で説明します．

コールバック関数が呼び出されなくなったときに呼び出される関数

関数 g_signal_connect_data の第 5 引数に与える関数は，次のように定義されています．

```
void (*GClosureNotify) (gpointer data, GClosure *closure);
```

この関数の第 1 引数に，関数 g_signal_connect_data の第 4 引数で指定したデータが渡されます．第 2 引数に関する解説はここでは省略します．

コールバック関数のプログラム例

関数 g_signal_connect_data を使ったプログラムの例を，ソース 3–2 に示します．コンパイルして実行すると，図 3.6 のウィンドウが表示されます．

“Click me!” ボタンをクリックすると，設定したコールバック関数が呼び出され，関数 g_signal_connect_data の第 4 引数に指定した文字列 “Hello World” がターミナルに表示されます．

この文字列は，関数 g_strdup で領域確保しているので，プログラムの終了時にこの領域を解放する必要があります．関数 g_signal_connect_data の第 5 引数に指定する関数は，設定したコールバック関数が使用されなくなったときに呼び出される関数なので，この関数内でデータの領域を解放します．

この関数を呼び出すために，図 3.6 のウィンドウの閉じるボタンをクリックして，プログラムを終了してみてください．以下に示すように “Destroy the callback function data.” と表示された後，関数に渡されたデータ（この場合は文字列 “Hello World”）が表示され，プログラムが終了します．

```
$ ./signal-sample1 ↵
Hello World
Destroy the callback function data.
The value of the destroyed data = 'Hello World'
```

表 3.2　コールバック関数の接続設定

値	説明
G_CONNECT_AFTER	ユーザが設定するコールバック関数を，標準で設定されているコールバック関数が呼び出された後で実行する
G_CONNECT_SWAPPED	別のウィジェットのコールバック関数として実行する

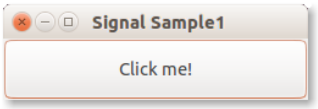


図 3.6　コールバック関数の例 1

ソース 3–2 コールバック関数の例 1：signal-sample1.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_button (GtkWidget *widget, gpointer user_data)
5  {
6    g_print ("%s\n", (gchar *) user_data);
7  }
8
9  static void
10 destroy_data (gpointer user_data, GClosure *closure)
11 {
12   g_print ("Destroy␣the␣callback␣function␣data.\n");
13   g_print ("The␣value␣of␣the␣destroyed␣data␣=␣'%s'\n",
14            (gchar *) user_data);
15   g_free ((gchar *) user_data);
16 }
17
18 int
19 main (int argc, char *argv[])
20 {
21   GtkWidget *window;
22   GtkWidget *button;
23   gulong    handle;
24
25   gtk_init (&argc, &argv);
26
27   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
28   gtk_window_set_title (GTK_WINDOW (window), "Signal␣Sample1");
29   gtk_widget_set_size_request (window, 250, 50);
30   g_signal_connect (G_OBJECT (window), "destroy",
31                     G_CALLBACK (gtk_main_quit), NULL);
32
33   button = gtk_button_new_with_label ("Click␣me!");
34   gtk_container_add (GTK_CONTAINER (window), button);
35
36   g_signal_connect_data (G_OBJECT (button), "clicked",
37                          G_CALLBACK (cb_button),
38                          (gpointer *) g_strdup ("Hello␣World"),
39                          destroy_data, 0);
40   gtk_widget_show_all (window);
41
42   gtk_main ();
43
44   return 0;
45 }
```

g_signal_connect_data の代わりになる関数

関数 g_signal_connect_data の使い方は理解してもらえたでしょうか．コールバック関数のデータをプログラムの終了と同時に解放したいような場合には，この関数は大変便利です．

しかし，実際の場面では GClosureNotify を必要としないことが多いため，以下に示す 3 つの関数が用意されています．

- g_signal_connect
  一番よく使用されるのがこの関数です．設定したコールバック関数は，ウィジェットに標準で設定されているコールバック関数が呼び出される前に実行されます．

```
gulong g_signal_connect (gpointer    instance,
                         const gchar *detailed_signal,
                         GCallback   c_handler,
                         gpointer    data);
```

- g_signal_connect_after
  ウィジェットにコールバック関数を設定するという意味では上の関数と同様です．上の関数との違いは，ウィジェットに標準で設定されているコールバック関数が呼び出された後で実行されることです．

```
gulong g_signal_connect_after (gpointer    instance,
                               const gchar *detailed_signal,
                               GCallback   c_handler,
```

```
                                    gpointer     data);
```

- g_signal_connect_swapped
  この関数は関連付けられたシグナルが発生したときに c_hander に指定したコールバック関数を呼び出しますが，コールバック関数の第 1 引数にはこの関数を関連付けたウィジェットではなく，data に指定したウィジェットが渡されます．

```
gulong g_signal_connect_swapped (gpointer     instance,
                                 const gchar *detailed_signal,
                                 GCallback    c_handler,
                                 gpointer     data);
```

g_signal_connect と g_signal_connect_after のプログラム例

これらの関数について具体例を見てみましょう．まずは関数 g_signal_connect と関数 g_signal_connect_after の動作を見てみます．ソースコードはソース 3–3 になります．

先ほどの例と同じウィンドウにボタンが配置されただけのアプリケーションを作成して，ボタンに対して 4 つのコールバック関数を関連付けます（44–51 行目）．このように 1 つのウィジェットに対して複数のコールバック関数を設定することが可能です．

設定したコールバック関数が，どのような順番で実行されるかを確認します．プログラムを実行してボタンをクリックすると，ターミナルに以下のようなメッセージが表示されます．

```
$ ./signal-sample2 ↵
function 1.
function 2.
function 3.
function 4.
```

この結果から次のことがわかります．

- 設定する順番にかかわらず，関数 g_signal_connect で関連付けたコールバック関数は先に，関数 g_signal_connect_after で関連付けたコールバック関数は最後に呼び出される．
- 同じ関数で関連付けられたコールバック関数は，関連付けられた順番に呼び出される．

ソース 3–3 コールバック関数の動作：signal-sample2.c

```
 1 #include <gtk/gtk.h>
 2
 3 static void
 4 cb_button1 (GtkWidget *widget, gpointer user_data)
 5 {
 6   g_print ("function␣1.\n");
 7 }
 8
 9 static void
10 cb_button2 (GtkWidget *widget, gpointer user_data)
11 {
12   g_print ("function␣2.\n");
13 }
14
15 static void
16 cb_button3 (GtkWidget *widget, gpointer user_data)
17 {
18   g_print ("function␣3.\n");
19 }
20
21 static void
22 cb_button4 (GtkWidget *widget, gpointer user_data)
23 {
24   g_print ("function␣4.\n");
25 }
26
27 int
28 main (int argc, char *argv[])
29 {
```

```
30   GtkWidget *window;
31   GtkWidget *button;
32
33   gtk_init (&argc, &argv);
34
35   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
36   gtk_window_set_title (GTK_WINDOW (window), "Signal␣Sample2");
37   gtk_widget_set_size_request (window, 250, 50);
38   g_signal_connect (G_OBJECT (window), "destroy",
39                     G_CALLBACK(gtk_main_quit), NULL);
40
41   button = gtk_button_new_with_label ("Click␣me!");
42   gtk_container_add (GTK_CONTAINER (window), button);
43
44   g_signal_connect_after (G_OBJECT (button), "clicked",
45                           G_CALLBACK (cb_button3), NULL);
46   g_signal_connect_after (G_OBJECT (button), "clicked",
47                           G_CALLBACK (cb_button4), NULL);
48   g_signal_connect (G_OBJECT (button), "clicked",
49                     G_CALLBACK(cb_button1), NULL);
50   g_signal_connect (G_OBJECT (button), "clicked",
51                     G_CALLBACK(cb_button2), NULL);
52
53   gtk_widget_show_all (window);
54
55   gtk_main ();
56
57   return 0;
58 }
```

g_signal_connect_swapped のプログラム例

次に関数 g_signal_connect_swapped の動作について解説します．

これまで説明した関数では，コールバック関数の第 1 引数にはコールバック関数を関連付けたウィジェットが渡されました．しかし，関数 g_signal_connect_swapped で設定したコールバック関数の第 1 引数には，関数 g_signal_connect_swapped の第 4 引数に指定したウィジェットが渡されます．実際にどのように動作するのか次の例を見てみましょう．ソースコードはソース 3–4 です．

ソースコードの 34–35 行目では，1 つ目のボタンを作成してコールバック関数を設定しています．このコールバック関数では，ボタンをクリックするたびにボタンが何回クリックされたかをボタンのラベルに表示します．

次に 39–40 行目で 2 つ目のボタンを作成して，関数 g_signal_connect_swapped でコールバック関数を設定しています．コールバック関数は 1 つ目のボタンと共通です．

このコールバック関数は，第 1 引数に渡されたウィジェットのラベルを書き換えるようになっているので，2 つ目のボタンのコールバック関数を関数 g_signal_connect 等で設定すると，2 つ目のボタンのラベルが更新されるはずです．しかし，関数 g_signal_connect_swapped の第 4 引数に 1 つ目のウィジェットを指定しているので，2 つ目のボタンがクリックされても，コールバック関数の第 1 引数に渡されるのは 1 つ目のボタンウィジェットということになります．

つまり，どちらのボタンがクリックされても，ラベルが更新されるのは 1 つ目のボタンになります．

プログラムを実行すると，図 3.7 のウィンドウが表示されます．実際に両方のボタンをクリックして動作を確認してみてください．

ソース 3–4 関数 g_signal_connect_swapped の例：signal-sample3.c

```
1 #include <gtk/gtk.h>
2
3 static void
4 cb_button (GtkWidget *widget, gpointer user_data)
5 {
6   static gint count = 0;
7   gchar       buf[64];
8
```

図 3.7 関数 g_signal_connect_swapped の例

```
 9   sprint (buf, "%d␣time(s)␣clicked.", ++count);
10   gtk_button_set_label (GTK_BUTTON (widget), buf);
11 }
12
13 int
14 main (int argc, char *argv[])
15 {
16   GtkWidget *window;
17   GtkWidget *hbox;
18   GtkWidget *button1;
19   GtkWidget *button2;
20
21   gtk_init (&argc, &argv);
22
23   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
24   gtk_window_set_title (GTK_WINDOW (window), "Signal␣Sample3");
25   gtk_widget_set_size_request (window, 500, 50);
26   g_signal_connect (G_OBJECT (window), "destroy",
27                     G_CALLBACK(gtk_main_quit), NULL);
28
29   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
30   gtk_container_add (GTK_CONTAINER (window), hbox);
31
32   button1 = gtk_button_new_with_label ("0␣time(s)␣clicked.");
33   gtk_box_pack_start (GTK_BOX (hbox), button1, TRUE, TRUE, 0);
34   g_signal_connect (G_OBJECT (button1), "clicked",
35                     G_CALLBACK (cb_button), NULL);
36
37   button2 = gtk_button_new_with_label ("Click␣me!");
38   gtk_box_pack_start (GTK_BOX (hbox), button2, TRUE, TRUE, 0);
39   g_signal_connect_swapped (G_OBJECT (button2), "clicked",
40                             G_CALLBACK (cb_button), (gpointer) button1);
41
42   gtk_widget_show_all (window);
43
44   gtk_main ();
45
46   return 0;
47 }
```

3.3.3　コールバック関数の解除

ここまではコールバック関数の設定について説明してきました．一度設定したコールバック関数は再び解除，すなわちコールバック関数を設定していない状態にすることもできます．その代表的な関数は関数 g_signal_handler_disconnect です．この関数は，関数 g_signal_connect などの戻り値を ID としてコールバック関数を解除します．

```
void g_signal_handler_disconnect (gpointer instance,
                                  gulong   handler_id);
```

第 1 引数　コールバック関数が登録されたウィジェット
第 2 引数　コールバック関数の ID

コールバック関数を解除するプログラム

ソース 3–5 のプログラムを例に，具体的に説明します．このプログラムを実行すると，図 3.8 のように 2 つのボタンが並んだウィンドウが表示されます．

左のボタンをクリックすると，ソースコードの 52–53 行目で設定したコールバック関数 cb_button が呼び出されて，“Callback function is called.” というメッセージがターミナルに表示されます．

一方，右のボタンをクリックすると，59–60 行目で設定したコールバック関数 disconnect_handler が呼び出されます．この関数内で，左のボタンに対するコールバック関数を解除します．

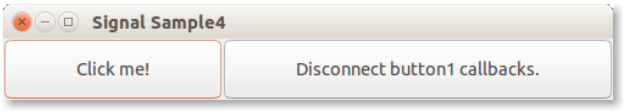


図 3.8　コールバック関数の解除

```
$ ./signal-sample4 ↵
Callback function is called.
Callback function is diconnected.
Callback function is already diconnected.
```

左のボタンのコールバック関数を解除するためには，左のボタンのウィジェットとそれに対するコールバック関数のハンドラ ID を知る必要があります．最も簡単な解決方法は，これらの変数をグローバル変数として扱うことです．

もう 1 つのより良い方法は，GObject 型の変数にその他の変数を関連付ける GTK+ の機能を利用することです．これが 56–57 行目の関数 g_object_set_data です．この関数は，第 2 引数に指定した文字列をキーとして，第 3 引数に指定した変数を，第 1 引数のオブジェクトに関連付けるものです．

```
void
g_object_set_data (GObject *object, const gchar *key, gpointer data);
```

第 1 引数　オブジェクト
第 2 引数　関連付ける変数を識別する文字列
第 3 引数　関連付ける変数

反対に，関連付けた変数を取得するには，関数 g_object_get_data を使用します．

```
gpointer g_object_get_data (GObject *object, const gchar *key);
```

第 1 引数　オブジェクト
第 2 引数　関連付けた変数を識別する文字列

また，関数 disconnect_handler 内では，関数 g_signal_handler_disconnect が何度も実行されるのを回避するため，関数 g_signal_handler_is_connected を使って，コールバック関数が設定されているかどうか調べています．

```
gboolean g_signal_handler_is_connected (gpointer instance,
                                        gulong   handler_id);
```

第 1 引数　コールバック関数を登録したオブジェクト
第 2 引数　コールバック関数の ID

ソース 3–5 関数 g_signal_connect_swapped の例：signal-sample4.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_button (GtkWidget *widget, gpointer user_data)
5  {
6    g_print ("Callback␣function␣is␣called.\n");
7  }
8
9  static void
10 disconnect_handler (GtkWidget *widget, gpointer user_data)
11 {
12   GtkWidget *button;
13   gulong    handle;
14
15   button = GTK_WIDGET (g_object_get_data (G_OBJECT (widget), "button1"));
16   handle = (gulong) g_object_get_data (G_OBJECT (widget), "handle");
17
18   if (g_signal_handler_is_connected (button, handle))
19     {
20       g_signal_handler_disconnect (button, handle);
21       g_print ("Callback␣function␣is␣diconnected.\n");
22       return;
23     }
24   else
25     {
```

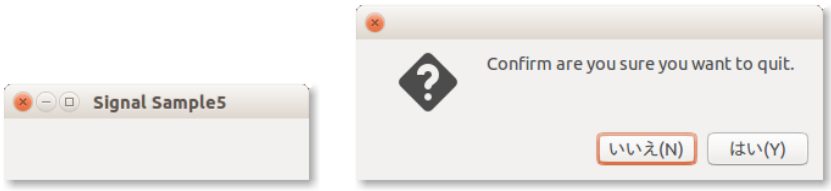


図 3.9　シグナルの伝搬

```
26        g_print ("Callback␣function␣is␣already␣diconnected.\n");
27      }
28  }
29
30  int
31  main (int argc, char *argv[])
32  {
33    GtkWidget *window;
34    GtkWidget *hbox;
35    GtkWidget *button1;
36    GtkWidget *button2;
37    gulong     handle;
38
39    gtk_init (&argc, &argv);
40
41    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
42    gtk_window_set_title (GTK_WINDOW (window), "Signal␣Sample4");
43    gtk_widget_set_size_request (window, 500, 50);
44    g_signal_connect (G_OBJECT (window), "destroy",
45                      G_CALLBACK (gtk_main_quit), NULL);
46
47    hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
48    gtk_container_add (GTK_CONTAINER (window), hbox);
49
50    button1 = gtk_button_new_with_label ("Click␣me!");
51    gtk_box_pack_start (GTK_BOX (hbox), button1, TRUE, TRUE, 0);
52    handle = g_signal_connect (G_OBJECT (button1), "clicked",
53                               G_CALLBACK (cb_button), NULL);
54
55    button2 = gtk_button_new_with_label ("Disconnect␣button1␣callbacks.");
56    g_object_set_data (G_OBJECT (button2), "button1", (gpointer) button1);
57    g_object_set_data (G_OBJECT (button2), "handle",  (gpointer) handle);
58    gtk_box_pack_start (GTK_BOX (hbox), button2, TRUE, TRUE, 0);
59    g_signal_connect (G_OBJECT (button2), "clicked",
60                      G_CALLBACK(disconnect_handler), NULL);
61
62    gtk_widget_show_all (window);
63    gtk_main ();
64
65    return 0;
66  }
```

3.3.4　シグナルの伝搬

シグナルは，そのシグナルが発生したウィジェットからその親ウィジェットへと，次々に伝搬していきます．その仕組みや動作を解説するのは大変なので，ここではプログラムを作成する際に役に立つテクニックを紹介します．これはウィンドウの「閉じる」ボタンをクリックしたときに，本当に終了してもいいか確認してからプログラムを終了させるというものです．

ウィンドウの「閉じる」ボタンをクリックすると，**delete-event シグナル**が発生します．ソース 3–6 では，46–47 行目でこのシグナルに対するコールバック関数を設定しています．コールバック関数の内容は 3–28 行目です．ここでは図 3.9 右のようなダイアログを表示して，本当に終了してもいいかどうか確認しています．ダイアログの使い方については，7.5 節（p. 180）を参照してください．ここでのポイントは，表示されたダイアログでイエスと答えるかノーと答えるかで，関数の戻り値が異なることです．

delete-event シグナルに対するコールバック関数の戻り値が FALSE の場合，続いて **destroy シグナル**が発生します．対応するコールバック関数 cb_destroy を 48–49 行目で設定しているので，この関数が呼び出されてプログラムが終了します．

しかし，delete-event シグナルに対するコールバック関数の戻り値が TRUE の場合，destroy シグナルは発生せずプログラムは続行します．

ソース 3–6 シグナルの伝搬：signal-sample5.c

```
1  #include <gtk/gtk.h>
2
3  static gboolean
4  cb_delete (GtkWidget *widget, gpointer user_data)
5  {
6    GtkWidget *dialog;
7    gint      result;
8
9    dialog
10     = gtk_message_dialog_new (GTK_WINDOW (widget),
11                               GTK_DIALOG_MODAL |
12                               GTK_DIALOG_DESTROY_WITH_PARENT,
13                               GTK_MESSAGE_QUESTION,
14                               GTK_BUTTONS_YES_NO,
15                               "Confirm are you sure you want to quit.");
16
17   result = gtk_dialog_run (GTK_DIALOG (dialog));
18   gtk_widget_destroy (dialog);
19
20   if (result == GTK_RESPONSE_YES)
21     {
22       return FALSE;
23     }
24   else
25     {
26       return TRUE;
27     }
28 }
29
30 static void
31 cb_destroy (GtkWidget *widget, gpointer user_data)
32 {
33   gtk_main_quit ();
34 }
35
36 int
37 main (int argc, char *argv[])
38 {
39   GtkWidget *window;
40
41   gtk_init (&argc, &argv);
42
43   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
44   gtk_window_set_title (GTK_WINDOW (window), "Signal Sample5");
45   gtk_widget_set_size_request (window, 250, 50);
46   g_signal_connect (G_OBJECT (window), "delete-event",
47                     G_CALLBACK (cb_delete), NULL);
48   g_signal_connect (G_OBJECT (window), "destroy",
49                     G_CALLBACK (cb_destroy), NULL);
50
51   gtk_widget_show_all (window);
52   gtk_main ();
53
54   return 0;
55 }
```

4 GLib

GLib は，C 言語で定義された関数を拡張した関数や，プラットフォームに依存しない基本データ型等を提供する，GTK+ の基盤となるライブラリです．GLib では基本データ型に加えて，リストやツリー，ハッシュといったデータ型も提供し，これらのデータを扱う便利な関数も実装されています．この章では，GLib で定義されたデータ型や便利な関数を紹介します．

4.1 GLib で定義された基本データ型

GLib ではプログラミングの簡便さやプラットフォームに依存しないソースコード作成を支援するために，C 言語の基本データ型に対応する型を表 4.1 のように定義しています．

表 4.1　GLib で定義された基本データ型

GLib で定義されたデータ型	対応する基本データ型
gboolean	int
gpointer	void*
gconstpointer	const void *
gchar	char
guchar	unsigned char
gint	int
guint	unsigned int
gshort	short
gushort	unsigned short
glong	long
gulong	unsigned long
gfloat	float
gdouble	double
gint8	符号付き 8 ビット整数
guint8	符号なし 8 ビット整数
gint16	符号付き 16 ビット整数
guint16	符号なし 16 ビット整数
gint32	符号付き 32 ビット整数
guint32	符号なし 32 ビット整数
gint64	符号付き 64 ビット整数
guint64	符号なし 64 ビット整数

4.2 便利な関数

ここでは GLib で提供されている関数をカテゴリ別にいくつか紹介します.

4.2.1 文字列操作関数

- g_strdup
  文字列をコピーして，新しく確保した領域へのポインタを返します.

```
gchar* g_strdup (const gchar *str);
```

第 1 引数	コピーする文字列
戻り値	コピーした文字列

```
gchar *copy;
copy = g_strdup ("Hello GTK+ World!");
g_print ("%s\n", copy);
```

- g_strdup_printf
  指定したフォーマットと引数に与えたパラメータにより文字列を作成して，新しく確保した領域へのポインタを返します．関数 sprintf と同じ働きをしますが，あらかじめ作成される文字列用の領域を確保しておく必要がありません.

```
gchar* g_strdup_printf (const gchar *format, ...);
```

第 1 引数	生成したい文字列のフォーマット
第 2 引数以降	フォーマットに応じた値
戻り値	生成した文字列

```
int n;
for (n = 0; n < 10; n++)
  {
    gchar *text;
    text = g_strdup_printf ("chapter-%d", n);
    g_print ("%s\n", text);
    g_free (text);
  }
```

- g_strsplit
  文字列 string を区切り文字 delimiter で最大 max_tokens 個に分割する関数です.

```
gchar** g_strsplit (const gchar *string,
                    const gchar *delimiter,
                    gint         max_tokens);
```

第 1 引数	入力文字列
第 2 引数	区切り文字
第 3 引数	最大分割数．1 より小さい値を指定する入力文字列を完全に分割する
戻り値	生成した文字列の配列．配列の最後は NULL で終わる

```
const gchar *string = "This is a sample of g_strsplit.";
gchar       **text;
int         n;
```

```
text = g_strsplit (string, " ", 0);
for (n = 0; text[n] != NULL; n++)
  {
    g_print ("%s\n", text[n]);
  }
g_strfreev (text);
```

- g_strfreev
  関数 g_strsplit 等で確保された文字列の配列領域を解放する関数です．

```
void g_strfreev (gchar **str_array);
```

第 1 引数	解放するメモリ領域の先頭アドレス

4.2.2 メモリアロケーション関数

- g_new0
  指定したデータ型 struct_type のメモリ領域を n_structs 分確保し，0 で初期化して，その先頭アドレスを返します．この関数は g_malloc0 のマクロとして定義されています．

```
#define g_new0(struct_type, n_structs) \
    ((struct_type *) g_malloc0 (((gsize) sizeof (struct_type)) * \
                                ((gsize) (n_structs))))
gpointer g_malloc0 (gulong n_bytes);
```

第 1 引数	生成するデータ型
第 2 引数	データ数
戻り値	確保したメモリ領域の先頭アドレス

```
gint *array;
int  n;
array = g_new0 (gint, 10);
for (n = 0; n < 10; n++)
  {
    g_print ("%d\n", array[n]);
  }
g_free (array);
```

- g_free
  マクロ g_new0 等で確保されたメモリ領域を解放する関数です．

```
void g_free (gpointer mem);
```

第 1 引数	解放するメモリ領域の先頭アドレス

4.2.3 ファイルアクセス関数

- g_file_test
  GFileTest で定義されたファイルの状態を調べる関数です．第 2 引数に与える値によって，ファイルが存在するか，ファイルがディレクトリであるかなどを調べることができます．

```
gboolean g_file_test (const gchar *filename, GFileTest test);
```

第 1 引数	ファイル名
第 2 引数	テストしたい内容に応じた値
戻り値	調べる内容に合致した場合は TRUE，合致しなければ FALSE を返す

表 4.2　GFileTest の値

値	説明
G_FILE_TEST_IS_REGULAR	ファイルかどうか調べる．
G_FILE_TEST_IS_SYMLINK	シンボリックリンクかどうか調べる．
G_FILE_TEST_IS_DIR	ディレクトリかどうか調べる．
G_FILE_TEST_IS_EXECUTABLE	実行形式のファイルかどうか調べる．
G_FILE_TEST_EXISTS	ファイルが存在するかどうか調べる．

- g_dir_open
  ディレクトリをオープンする関数です．オープンしたディレクトリ内のファイル名を調べるには関数 g_dir_read_name を使用します．

```
GDir* g_dir_open (const gchar *path,
                  guint       flags,
                  GError      **error);
```

第 1 引数	オープンしたいディレクトリ名
第 2 引数	フラグ
第 3 引数	エラー情報
戻り値	ディレクトリ構造体

- g_dir_read_name
  関数 g_dir_open でオープンしたディレクトリ内のファイル名を調べる関数です．呼び出されるごとに順にファイル名を返していきます．最後まで読み込んだ場合は NULL が返ります．

```
G_CONST_RETURN gchar* g_dir_read_name (GDir *dir);
```

第 1 引数	ディレクトリ構造体
戻り値	ファイル名

- g_dir_close
  関数 g_dir_open でオープンしたディレクトリをクローズする関数です．

```
void g_dir_close (GDir *dir);
```

第 1 引数	ディレクトリ構造体

サンプルプログラム

上記の関数を使用したプログラムの例をソース 4–1 に示します．このプログラムは引数に指定したディレクトリ内のファイルの一覧を表示するプログラムです．

ディレクトリのオープン（16 行目）

関数 g_dir_open の第 1 引数にオープンするディレクトリ名を指定してディレクトリをオープンします．現在の仕様では第 2 引数には 0 のみを指定することになっています．第 3 引数には GError 構造体へのポインタを指定しますが，エラーの詳細が必要ない場合には NULL を与えておけばよいでしょう．ディレクトリのオープンに失敗すると NULL が返ってきます．

ディレクトリ内のファイルの表示（21–30 行目）

21 行目からの while ループでは，関数 g_dir_read_name によってオープンしたディレクトリ内のファイル名を 1 つずつ取得していきます．そして関数 g_file_test でファイルがディレクトリかどうか調べるために，関数 g_build_filename を使用してファ

イル名にパスを追加しています．関数 g_build_filename ではファイル名用のメモリ領域が新しく領域確保されるので，29 行目で関数 g_free で使用しなくなったメモリ領域を解放しています．

ディレクトリのクローズ（31 行目）
最後に関数 g_dir_close でディレクトリをクローズしています．

プログラムの実行結果を以下に示します．

```
$ gcc file-sample.c -o file-sample `pkg-config --cflags --libs glib-2.0` ↵
$ ./file-sample . ↵
file-sample.c   (file)
file-sample     (file)
file-sample.o   (file)
Makefile        (file)
test-dir        (directory)
$
```

ソース 4–1 ファイル操作の例：file-sample.c

```
 1 #include <glib.h>
 2 #include <stdlib.h>
 3
 4 int
 5 main (int argc, char *argv[])
 6 {
 7   GDir  *dir;
 8   gchar        *currentdir;
 9
10   if (argc != 2)
11     {
12       g_print ("Usage: ./file-sample directory\n");
13       exit (1);
14     }
15   currentdir = argv[1];
16   dir = g_dir_open (currentdir, 0, NULL);
17   if (dir)
18     {
19       const gchar *name;
20
21       while (name = g_dir_read_name (dir))
22         {
23           gchar    *path;
24           gboolean is_dir;
25
26           path = g_build_filename (currentdir, name, NULL);
27           is_dir = g_file_test (path, G_FILE_TEST_IS_DIR);
28           g_print ("%s\t(%s)\n", name, (is_dir) ? "directory" : "file");
29           g_free (path);
30         }
31       g_dir_close (dir);
32     }
33   return 0;
34 }
```

4.2.4 Unicode に関する関数

日本語の文字コードには，シフト JIS や日本語 EUC が使われてきました．最近使用されることの多くなった文字コードに，国際規格の「Unicode」があります．GLib でも，ファイル名の文字列に Unicode（UTF-8）を採用しています．

このため GLib では，ユーザーの使用している各国環境（**ロケール**）で規定される文字コードと，UTF-8 の相互変換を行う関数を提供しています．以下にその例を示します．

- g_locale_to_utf8
  現在のロケールで与えられた文字列を，UTF-8 でエンコードされた文字列に変換する関数です．この関数では新しい文

字列用のメモリ領域が確保されるので，作成した文字列が使用されなくなった場合には，関数 g_free でメモリ領域を解放してください．

```
gchar* g_locale_to_utf8 (const gchar *opsysstring,
                         gssize       len,
                         gsize        *bytes_read,
                         gsize        *bytes_written,
                         GError       **error);
```

第 1 引数	文字列
第 2 引数	文字列の長さ．−1 を指定すると文字列全体を変換する
第 3 引数	読み込んだデータのバイト数
第 4 引数	書き込んだデータのバイト数
第 5 引数	エラー構造体
戻り値	変換した文字列

- g_locale_from_utf8
  UTF8 でエンコードされた文字列を，現在のロケールの文字コードに変換する関数です．この関数では新しい文字列用のメモリ領域が確保されるので，作成した文字列が使用されなくなった場合には，関数 g_free でメモリ領域を解放してください．

```
gchar* g_locale_from_utf8 (const gchar *utf8string,
                           gssize       len,
                           gsize        *bytes_read,
                           gsize        *bytes_written,
                           GError       **error);
```

第 1 引数	文字列
第 2 引数	文字列の長さ．−1 を指定すると文字列全体を変換する
第 3 引数	読み込んだデータのバイト数
第 4 引数	書き込んだデータのバイト数
第 5 引数	エラー構造体
戻り値	変換した文字列

- g_filename_to_utf8
  ファイル名の文字列を UTF8 でエンコードされた文字列に変換する関数です．この関数では新しい文字列用のメモリ領域が確保されるので，作成した文字列が使用されなくなった場合には，関数 g_free でメモリ領域を解放してください．

```
gchar* g_filename_to_utf8 (const gchar *opsysstring,
                           gssize       len,
                           gsize        *bytes_read,
                           gsize        *bytes_written,
                           GError       **error);
```

第 1 引数	文字列
第 2 引数	文字列の長さ．−1 を指定すると文字列全体を変換する
第 3 引数	読み込んだデータのバイト数
第 4 引数	書き込んだデータのバイト数
第 5 引数	エラー構造体
戻り値	変換した文字列

- g_filename_from_utf8
  UTF8 でエンコードされた文字列をファイル名用のエンコーディングに変換する関数です．この関数では新しい文字列用のメモリ領域が確保されるので，作成した文字列が使用されなくなった場合には，関数 g_free でメモリ領域を解放してください．

```
gchar* g_filename_from_utf8 (const gchar *utf8string,
                             gssize      len,
                             gsize       *bytes_read,
                             gsize       *bytes_written,
                             GError      **error);
```

第 1 引数	文字列
第 2 引数	文字列の長さ．−1 を指定すると文字列全体を変換する
第 3 引数	読み込んだデータのバイト数
第 4 引数	書き込んだデータのバイト数
第 5 引数	エラー構造体
戻り値	変換した文字列

4.2.5 乱数に関する関数

glib で実装されている乱数発生関数です．乱数発生アルゴリズムにはメルセンヌ・ツイスターと呼ばれる非常に高品質な擬似乱数列を生成できるものを使用しています．

- g_rand_new_with_seed
  乱数のシードを与えて乱数の初期化を行います．

```
GRand * g_rand_new_with_seed (guint32 seed);
```

第 1 引数	乱数のシード
戻り値	Grand 構造体へのポインタ

- g_rand_new
  乱数の初期化を行います．乱数のシードは現在の時刻かもしあれば/dev/urandom から取得します．

```
GRand * g_rand_new (void);
```

戻り値	Grand 構造体へのポインタ

- g_rand_free
  Grand 構造体型変数の領域を解放します．

```
void g_rand_free (GRand *rand_);
```

第 1 引数	Grand 構造体へのポインタ

- g_rand_int
  0 から $2^{32}-1$ までの整数の中から一様乱数に従って生成した乱数を返します．

```
guint32 g_rand_int (GRand *rand_);
```

第 1 引数	Grand 構造体へのポインタ
戻り地	0 から $2^{32}-1$ までの整数

- g_rand_int_range
  指定した範囲の整数の中から一様乱数に従って生成した乱数を返します．

```
gint32 g_rand_int_range (GRand *rand_, gint32 begin, gint32 end);
```

第 1 引数	Grand 構造体へのポインタ
第 2 引数	生成する乱数の最小値
第 3 引数	生成する乱数の最大値
戻り地	begin から end-1 までの整数

```
GRand *rand = g_rand_new();
g_print ("%d\n", g_rand_int (rand));
g_rand_free (rand);
```

- g_rand_double
  [0 : 1) の実数の中から一様乱数に従って生成した乱数を返します.

```
gdouble g_rand_double (GRand *rand_);
```

第 1 引数	Grand 構造体へのポインタ
戻り地	[0 : 1) の範囲の実数

- g_rand_double_range
  指定した範囲の実数の中から一様乱数に従って生成した乱数を返します.

```
gdouble g_rand_double_range (GRand *rand_, gdouble begin, gdouble end);
```

第 1 引数	Grand 構造体へのポインタ
第 2 引数	生成する乱数の最小値
第 3 引数	生成する乱数の最大値
戻り地	[$begin$: end) の範囲の実数

4.2.6　その他の便利な関数

- g_get_home_dir
  ユーザのホームディレクトリを取得する関数です.

```
G_CONST_RETURN gchar* g_get_home_dir (void);
```

戻り値	ホームディレクトリ名

```
g_print ("My home directory is %s.\n", g_get_home_dir ());
```

- g_get_current_dir
  現在のディレクトリを取得する関数です. この関数で取得したメモリ領域は関数 g_free で解放する必要があります.

```
gchar* g_get_current_dir (void);
```

戻り値	現在のディレクトリ名

```
gchar *currentdir;
currentdir = g_get_current_dir ();
g_print ("The current directory is %s.\n", currentdir);
g_free (currentdir);
```

- g_path_get_basename
  パスから最後のファイル名を返します. パスがディレクトリを指す場合には最後のディレクトリ名を返します.

```
gchar* g_path_get_basename (const gchar *file_name);
```

第 1 引数	パス
戻り値	パスの最後のファイル名もしくはディレクトリ名

- g_path_get_dirname
  パスから最後のファイル名を取り除いたディレクトリ部分を返します．

```
gchar* g_path_get_dirname (const gchar *file_name);
```

第 1 引数	パス
戻り値	パスから最後のファイル名を取り除いたディレクトリ名

```
gchar *basename;
gchar *dirname;
basename = g_path_get_basename (filename);
dirname = g_path_get_dirname (filename);
g_print ("basename = %s\n", basename);
g_print ("dirname = %s\n", dirname);
g_free (basename);
g_free (dirname);
```

- g_build_filename
  引数に与えた文字列をファイル名用の区切り文字（例えば/）で結合して，1 つの文字列を作成します．引数の最後は NULL で終わる必要があります．

```
gchar* g_build_filename (const gchar *first_element, ...);
```

第 1 引数以降	文字列
戻り値	生成されたパス

```
const gchar *dirname = "/home/gtk";
const gchar *basename = "sample.c";
gchar *filename;
filename = g_build_filename (dirname, basename, NULL);
g_print ("filename = %s\n", filename);
g_free (filename);
```

4.3 連結リスト

GLib では基本的なデータ型のほかに，よく用いられる便利なデータ型が定義されています．その中で本節では連結リスト について，次の節でハッシュについて説明します．

リストには単方向リストと双方向リストがありますが，ここでは双方向リストについてだけ説明します．GLib では双方向リストを次のように定義しています．ご覧のとおり，データを指す変数のデータ型として gpointer 型を用いているので，さまざまな型のデータをリストに与えることができます．

```
struct GList {
  gpointer data;
  GList    *next;
  GList    *prev;
};
```

4.3.1 GList に対する関数

GList 型の変数に対するさまざまな関数が定義されています．以下にいくつかの関数を紹介します．

- g_list_append
  リストにデータを追加する関数です．引数 list に NULL を与えると，新しいリストを作成してデータを追加します．

```
GList* g_list_append (GList *list, gpointer data);
```

第 1 引数	リスト
第 2 引数	追加するデータ
戻り値	リストの先頭アドレス

- g_list_insert
  指定した位置にデータを挿入する関数です．指定した位置がリストの範囲内にない場合には，リストの最後尾にデータを追加します．

```
GList* g_list_insert (GList    *list,
                      gpointer data,
                      gint     position);
```

第 1 引数	リスト
第 2 引数	追加するデータ
第 3 引数	データを追加する位置
戻り値	リストの先頭アドレス

- g_list_delete_link
  リストから指定した位置のノードを削除する関数です．リストのデータが動的に領域を確保したものである場合には，この関数を呼び出す前に，削除するノードのデータ領域を解放しておく必要があります．

```
GList* g_list_delete_link (GList *list, GList *link_);
```

第 1 引数	リスト
第 2 引数	削除するノードの先頭アドレス
戻り値	リストの先頭アドレス

- g_list_free
  リストを解放する関数です．リストのデータが動的に領域を確保したものである場合には，この関数を呼び出す前に，それらの領域を解放しておく必要があります．

```
void g_list_free (GList *list);
```

第 1 引数	リスト

- g_list_foreach
  リストの各ノードに対して関数 func を実行する関数です．

```
void g_list_foreach (GList    *list,
                     GFunc    func,
                     gpointer user_data);
```

第 1 引数	リスト
第 2 引数	ノードに対して適用する関数
第 3 引数	第 2 引数の関数の引数に与えるデータ

GFunc は次のように定義されています．この関数の第 1 引数にはリストのデータが渡されます．第 2 引数には，関数 g_list_foreach の第 3 引数に指定した変数が渡されます．

```
void (*GFunc) (gpointer data, gpointer user_data);
```

- g_list_length
  リストの長さを返す関数です．

```
guint g_list_length (GList *list);
```

第 1 引数　リスト
戻り値　リストの長さ

- g_list_first
  リストの先頭のノードを返す関数です.

```
GList* g_list_first (GList *list);
```

第 1 引数　リスト
戻り値　先頭ノードのアドレス

- g_list_last
  リストの末尾のノードを返す関数です.

```
GList* g_list_last (GList *list);
```

第 1 引数　リスト
戻り値　末尾ノードのアドレス

- g_list_previous
  現在のノードの前のノードを返します.

```
#define g_list_previous(list) \
        ((list) ? (((GList *)(list))->prev) : NULL)
```

第 1 引数　リスト
戻り値　前のノードのアドレス

- g_list_next
  現在のノードの次のノードを返します.

```
#define g_list_next(list) ((list) ? (((GList *)(list))->next) : NULL)
```

第 1 引数　リスト
戻り値　次のノードのアドレス

- g_list_nth
  n 番目のノードを返します.

```
GList* g_list_nth (GList *list, guint n);
```

第 1 引数　リスト
第 2 引数　ノード番号
戻り値　指定したノードのアドレス

- g_list_nth_data
  n 番目のリストのデータを返します.

```
gpointer g_list_nth_data (GList *list, guint n);
```

第 1 引数　リスト
第 2 引数　ノード番号
戻り値　指定したノードのデータ

- g_list_sort
  ノードを compare_func に従ってソートする関数です.

```
GList* g_list_sort (GList *list, GCompareFunc compare_func);
```

第 1 引数	リスト
第 2 引数	ソート関数
戻り値	ソートされたリストの先頭アドレス

ノードをソートする関数 GCompareFunc は次のように定義された関数です．

```
gint (*GCompareFunc) (gconstpointer a, gconstpointer b);
```

4.3.2　リスト操作の例: ソート

ここで GList を用いたリスト操作の例を見てみましょう．ソース 4–2 は “January” から “December” までの文字列をデータに持つリストを作成し，文字列を昇順にソートして表示するものです．実行結果を以下に示します．ソート後の表示を見ると，文字列が辞書順にソートされているのがわかります．

```
$ gcc list-sort.c -o list-sort `pkg-config --cflags --libs glib-2.0` ↵
$ ./list-sort ↵
January February March April May June July August September October November December
April August December February January July June March May November October September
```

変数の初期化（18–21 行目）
GList 型の変数は最初に関数 g_list_append を呼び出すときのために NULL に初期化しておきます．変数 data はリスト用の文字列データです．

リストの作成（24–27 行目）
関数 g_list_append を用いて，リストを作成します．基本的に関数の第 1 引数に与えた変数に関数の戻り値を代入してリストを作成していきます．

初期リストの表示（28–31 行目）
関数 g_list_nth_data を用いて，リストの n 番目のデータを順に取得し，表示します．

リストのソート（34 行目）
関数 g_list_sort を用いて，リストのデータを昇順にソートします．データのソートには単純に関数 strcmp を使用します．

リストの表示（35 行目）
関数 g_list_foreach を用いて，昇順にソートされたリストのデータを表示します．先ほどは for 文を使ってリストのすべてのデータにアクセスしましたが，ここではそれを関数 g_list_foreach で代用しました．

リストの解放（38 行目）
最後に関数 g_list_free により，リストの領域を解放します．

ソース 4–2 リストのソート：list-sort.c

```
1  #include <glib.h>
2
3  static gint
4  compare_function (gconstpointer a, gconstpointer b)
5  {
6    return strcmp ((gchar *) a, (gchar *) b);
7  }
8
9  static void
10 print_data (gpointer data, gpointer user_data)
11 {
12   g_print ("%s␣", (gchar *) data);
```

```
13 }
14
15 int
16 main (int argc, char *argv[])
17 {
18   GList *list = NULL;
19   gchar *data[] = {"January", "February", "March", "April", "May",
20                    "June", "July", "August", "September", "October",
21                    "November", "December"};
22   int    n;
23
24   for (n = 0; n < 12; n++)
25     {
26       list = g_list_append (list, (gpointer) data[n]);
27     }
28   for (n = 0; n < 12; n++)
29     {
30       g_print ("%s␣", (gchar *) g_list_nth_data (list, n));
31     }
32   g_print ("\n");
33
34   list = g_list_sort (list, compare_function);
35   g_list_foreach (list, (GFunc) print_data, NULL);
36   g_print ("\n");
37
38   g_list_free (list);
39
40   return 0;
41 }
```

4.3.3 リスト操作の例: データの追加・削除

先の例では，リストの生成とソートを扱いました．次の例では，データの追加と削除の例を見てみましょう（ソース 4–3）．この例では，データの追加（挿入）に関数 g_list_insert を，データの削除に関数 g_list_delete_link を使用しています．

初期リストの作成（10–13 行目）
はじめに，January，February，April の文字列をデータに持つリストを作成します．先ほどの例とは違い，文字列を関数 g_strdup を使ってコピーして，それをデータとして使用します．

リストの挿入（20 行目）
2 番目と 3 番目のノードの間に新しいデータ March を持つノードを挿入します．

リストの削除（27–28 行目）
このリストは関数 g_strdup で動的に確保したメモリ領域をデータとして使用しているので，ノードを削除する前に，関数 g_free により領域を解放します．4 番目（添字は 3）のノードのデータを取得するには，関数 g_list_nth_data を使います．削除するノードは関数 g_list_nth を使って取得します．

リストの解放（35–36 行目）
リストデータは動的に確保したメモリ領域を使用していたので，関数 g_list_free を使用する前に，関数 g_list_foreach を使って各データ領域を関数 g_free で解放しています．

```
$ gcc list-operation.c -o list-operation ‘pkg-config --cflags --libs glib-2.0‘ ↵
$ ./list-operation ↵
January February April
January February March April
January February March
```

ソース 4–3 リストの追加・削除：list-operation.c

```
1  #include <glib.h>
2
3  int
4  main (int argc, char *argv[])
5  {
6    GList *list = NULL, *work;
7    gchar *data[] = {"January", "February", "April"};
8    int    n;
9
10   for (n = 0; n < 3; n++)
11     {
12       list = g_list_append (list, (gpointer) g_strdup (data[n]));
13     }
14   for (work = list; work; work = g_list_next (work))
15     {
16       g_print ("%s␣", (gchar *) work->data);
17     }
18   g_print ("\n");
19
20   list = g_list_insert (list, (gpointer) g_strdup ("March"), 2);
21   for (work = list; work; work = g_list_next (work))
22     {
23       g_print ("%s␣", (gchar *) work->data);
24     }
25   g_print ("\n");
26
27   g_free (g_list_nth_data (list, 3));
28   list = g_list_delete_link (list, g_list_nth (list, 3));
29   for (work = list; work; work = g_list_next (work))
30     {
31       g_print ("%s␣", (gchar *) work->data);
32     }
33   g_print ("\n");
34
35   g_list_foreach (list, (GFunc) g_free, NULL);
36   g_list_free (list);
37
38   return 0;
39 }
```

4.4 ハッシュ

リストに並んでハッシュもよく使われるデータ構造です．GLib では **GHashTable** という構造体で定義されており，g_hash_table_から始まる関数によってハッシュテーブルを操作します．

4.4.1 GHashTable に対する関数

GHashTable 型の変数に対するさまざまな関数が定義されています．以下にいくつかの関数を紹介します．

- g_hash_table_new
  ハッシュテーブルを作成する関数です．hash_func が NULL の場合は関数 g_direct_hash を使用します．関数 key_equal_func は，2 つのキーが等しいかどうかをチェックする比較関数で，これは GHashTable の中にあるキーを検索するときに使用します．

```
GHashTable* g_hash_table_new (GHashFunc  hash_func,
                              GEqualFunc key_equal_func);
```

第 1 引数	ハッシュ関数
第 2 引数	キーチェック関数
戻り値	生成されたハッシュテーブル

- g_hash_table_new_full
  関数 g_hash_table_new と同様に新規に GHashTable を生成する関数です．この関数では，GHashTable から要素を削除

する際に，キーと値が確保していたメモリを解放するために呼び出す関数を指定することが可能です．

```
GHashTable*
g_hash_table_new_full (GHashFunc       hash_func,
                       GEqualFunc      key_equal_func,
                       GDestroyNotify  key_destroy_func,
                       GDestroyNotify  value_destroy_func);
```

第 1 引数	ハッシュ関数
第 2 引数	キーチェック関数
第 3 引数	キー用のメモリ領域を解放する関数
第 4 引数	値用のメモリ領域を解放する関数
戻り値	生成されたハッシュテーブル

- g_hash_table_insert
  新しいキーと値を挿入する関数です．

```
void g_hash_table_insert (GHashTable *hash_table,
                          gpointer    key,
                          gpointer    value);
```

第 1 引数	ハッシュテーブル
第 2 引数	キー
第 3 引数	値

- g_hash_table_size
  GHashTable の中にある要素の数を返す関数です．

```
guint g_hash_table_size (GHashTable *hash_table);
```

第 1 引数	ハッシュテーブル
戻り値	ハッシュテーブルに格納された要素数

- g_hash_table_lookup
  GHashTable の中にあるキーを検索する関数です．

```
gpointer g_hash_table_lookup (GHashTable     *hash_table,
                              gconstpointer key);
```

第 1 引数	ハッシュテーブル
第 2 引数	キー
戻り値	指定したキーに対する値

- g_hash_table_destroy
  GHashTable を破棄する関数です．キーとその値を直接メモリ上に確保した場合は，まずそれらを解放する必要があります．

```
void g_hash_table_destroy (GHashTable *hash_table);
```

第 1 引数	ハッシュテーブル

4.4.2 ハッシュテーブルの例

ここでは，GHashTable を使ったハッシュ操作の簡単な例を示します（ソース 4–4）．この例ではキーも値も文字列として，ハッシュテーブルを構成し，キーから値を検索して表示します．

```
$ gcc hash-sample.c -o hash-sample `pkg-config --cflags --libs glib-2.0` ↵
$ ./hash-sample ↵
Key: large => Value: +1
Key: normal => Value: +0
Key: small => Value: -1
```

ハッシュテーブルの作成（10–12 行目）
関数 g_hash_table_new_full を使ってハッシュテーブルを作成します．この例では，キーも値も文字列としているので，GDestroyNotify 関数として g_free を指定しています．

値の登録（13–15 行目）
関数 g_hash_table_insert により，指定したキーで値を登録します．

値の参照（17–23 行目）
for 文により登録されている値を表示します．値の検索には関数 g_hash_table_lookup を使用します．

テーブルの解放（24 行目）
使わなくなったハッシュテーブルを破棄するには，関数 g_hash_table_destroy を使います．ハッシュテーブルを作成する際に，関数 g_hash_table_new_full でキーと値の領域を解放する関数を指定してあるので，関数 g_hash_table_destroy が呼び出されたときに自動的にキーと値の領域が解放されます．関数 g_hash_table_new でハッシュテーブルを作成した場合は，関数 g_hash_table_destroy を呼び出す前に，キーと値の領域を解放しておく必要があります．

ソース 4–4 ハッシュテーブルの操作：hash-sample.c

```
 1 #include <glib.h>
 2
 3 int
 4 main (int argc, char *argv[])
 5 {
 6   GHashTable *hash;
 7   gchar      *data[] = {"large", "normal", "small"};
 8   int         n;
 9
10   hash = g_hash_table_new_full (g_str_hash, g_str_equal,
11                                 (GDestroyNotify) g_free,
12                                 (GDestroyNotify) g_free);
13   g_hash_table_insert (hash, g_strdup (data[0]), g_strdup ("+1"));
14   g_hash_table_insert (hash, g_strdup (data[1]), g_strdup ("+0"));
15   g_hash_table_insert (hash, g_strdup (data[2]), g_strdup ("-1"));
16
17   for (n = 0; n < 3; n++)
18     {
19       g_print ("Key: %s => Value: %s\n",
20                data[n],
21                (gchar *) g_hash_table_lookup (hash,
22                                               (gconstpointer) data[n]));
23     }
24   g_hash_table_destroy (hash);
25
26   return 0;
27 }
```

4.5　イベントループ

GLib では一定のインターバルで定期的に関数を呼び出す機能が提供されています．関数 g_timeout_add を使うと関数の引数に指定した関数を指定したインターバルで呼び出すことができます．

```
guint g_timeout_add (guint       interval,
                     GSourceFunc function,
                     gpointer    data);
```

第 1 引数	インターバル
第 2 引数	呼び出す関数
第 3 引数	呼び出す関数に与えるデータ
戻り値	イベント ID

インターバルの値は 1/1000 秒単位で指定します．したがって interval = 1000 とすると，1 秒間隔で関数を呼び出すことになります．また，呼び出す関数 GSourceFunc のプロトタイプ宣言は次のようになっています．

```
gboolean (*GSourceFunc) (gpointer data);
```

関数 g_timeout_add の第 3 引数には，呼び出す関数の引数に渡すデータを与えます．この関数の戻り値には，作成されたイベントの ID が返ります．

この関数のループを停止するには，関数 g_source_remove を使用します．関数の引数には関数 g_timeout_add の戻り値から取得したイベント ID を指定します．

```
gboolean g_source_remove (guint tag);
```

第 1 引数	イベント ID
戻り値	成功したら TRUE，失敗したら FALSE を返します

ソース 4–5 にイベントループを利用したタイマープログラムを紹介します．関数 g_timeout_add によって 1 秒ごとに関数 count_down を呼び出し，変数 count の値が 0 になったときにプログラムを終了します．

ソース 4–5　タイマープログラム：timer-sample.c

```
1  #include <gtk/gtk.h>
2
3  static gint count = 10;
4  static guint timer_id;
5
6  static
7  gboolean count_down (gpointer user_data)
8  {
9    GtkLabel *label = GTK_LABEL (user_data);
10   gchar    format[] = "Count␣%2d";
11   gchar            str[9];
12
13   g_sprintf (str, format, --count);
14   gtk_label_set_text (label, str);
15
16   if (count == 0)
17     {
18       g_source_remove (timer_id);
19       gtk_main_quit ();
20     }
21 }
22
23 int
24 main (int argc, char *argv[])
25 {
26   GtkWidget *window;
27   GtkWidget *label;
28
29   gtk_init (&argc, &argv);
30
31   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
32   gtk_window_set_title  (GTK_WINDOW (window), "Timer␣Sample");
33   gtk_widget_set_size_request (window, 200, -1);
34   gtk_container_set_border_width (GTK_CONTAINER (window), 10);
35
```

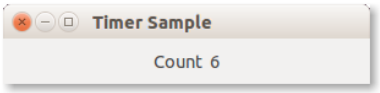


図 4.1　タイマーサンプル

```
36   label = gtk_label_new ("Count␣10");
37   gtk_container_add (GTK_CONTAINER (window), label);
38
39   timer_id = g_timeout_add (1000, (GSourceFunc) count_down, label);
40
41   gtk_widget_show_all (window);
42   gtk_main ();
43
44   return 0;
45 }
```

5 cairo による図形の描画

5.1 cairo とは

cairo は 2 次元グラフィックスライブラリの 1 つです．2 次元グラフィックスの形式は，画像を各画素の色情報で表現するラスタ形式と呼ばれるものと，画像を線の端点の座標などの幾何的な情報で表現するベクタ形式と呼ばれるものに分類できます．

SVG 画像フォーマットをはじめとして，ベクタ形式の画像は，描画する内容を幾何情報として保存しているため，画像を拡大縮小しても元の情報を失うことなく高品質に表示できます．cairo は，このベクタ形式の画像を表示するためのグラフィックスライブラリです．もちろん GTK+ でも cairo が使用されており，現在最も注目されているライブラリと言っても過言ではないでしょう．そのため，cairo は C 言語だけではなく，C++ や Java，Mono をはじめとして，Perl や Python，Ruby などのスクリプト言語に対応したバインディングが公開されています．

cairo の学習には以下の Web ページが参考になります．本章でも参考にさせていただきました．

- Cairo Tutorial http://cairographics.org/tutorial/
  cairo 本家のチュートリアルです．
- The Cairo graphics tutorial http://zetcode.com/tutorials/cairographicstutorial/
  ZetCode で公開しているチュートリアルシリーズの cairo に関するチュートリアルです．本書ではここで紹介されているサンプルプログラムを，そのままの形もしくは多少アレンジした形で使用しています．
- Rubyist Magazine cairo: 2 次元画像描画ライブラリ
  http://jp.rubyist.net/magazine/?0019-cairo
  cairo の Ruby バインディング rcairo による cairo の紹介ページです．

5.2 図形描画の概念と手順

cairo で図形を描画するためには，以下に挙げる項目について理解している必要があります．

- サーフェス
- コンテキスト
- ソース
- パス

サーフェスは，GDK ライブラリでいうドローアブルに対応するもので，絵を描画する「キャンバス」のようなものです．それに対してコンテキスト**コンテキスト**は，描画の仕方を制御するもので，キャンバスに対応させれば「画家」ということができるかもしれません．**ソース**は「筆」に当たるものです．

これが本当の絵であれば，「画家」が手に持った「筆」で直接「キャンバス」に描画するでしょう．しかし，cairo では描画する内容を，いったん**パス**で表現します．パスとは，描画する図形の軌跡のことです．このパスに沿って，ソースに指定した色や模様を使って，サーフェス上に絵を描画します．

別の見方をすると，キャンバスの上にマスクとなる用紙を重ね，パスの部分だけマスクを切り抜き，その上からソースを描画することで，サーフェス上にパスの内容を描画すると言うこともできます．

図 5.1 は，上記の説明を模式的に示したものです．まず，図 5.1(a) のようにソースに色を設定し，次に図 5.1(b) のように四角形のパスを設定します．この状態でソースの内容をサーフェスに描画すると，パスの部分だけマスクが切り抜かれて，サーフェス上に四角形が描画されるというわけです．パスを作成してからソースを設定しても，同様の結果が得られます．

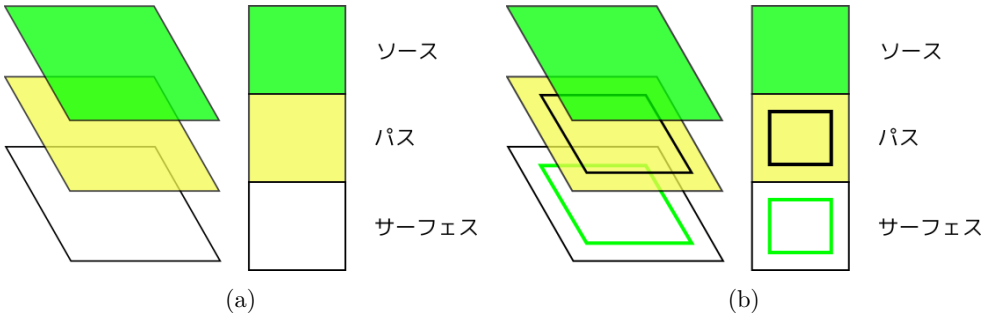


図 5.1　cairo での図形描画の概念: (a) ソース（色）の設定，(b) パスの設定と描画

実際の図形の描画は，次の手順で行います．

1. 出力先に合わせてサーフェスを作成する．
2. コンテキストを作成する．
3. 図形を描画する．
4. 描画処理を終了する．

それぞれの手順の具体的な内容を，次節以降で説明していきます．

5.3　サーフェスの作成

サーフェスは，出力先の種類に応じて作成しなければなりません．ここでは以下に示す出力先に応じた，サーフェスの作成方法を紹介します．

- GTK+ アプリケーションのドローアブル
- 画像
- PS（EPS）ファイル
- SVG ファイル

5.3.1　GTK+ アプリケーションのドローアブル

GTK+ アプリケーションのドローアブルをサーフェスとする場合，ユーザが明示的にサーフェスを作成する必要はありません．出力先のドローアブルを引数として，関数 gdk_cairo_create を呼び出すことで，指定したドローアブルをサーフェスとするコンテキストが生成されます．

```
cairo_t* gdk_cairo_create (GdkWindow *window);
```

第 1 引数	GdkWindow 型の変数
戻り値	コンテキスト

また，描画用のウィジェットとして **GtkDrawingArea** を利用することができます．このウィジェットには **draw シグナル**が設定されていますので，図形の描画に関する記述はこのシグナルに対するコールバック関数内に書くとよいでしょう．draw シグナルに対するコールバック関数は以下のような形式をしており，第 2 引数にコンテキストが自動的に与えられます．使用例はソース 5–1 を参照してください．

```
gboolean user_function (GtkWidget *widget
                        cairo_t   *cr,
                        gpointer  user_data);
```

5.3.2 画像用のサーフェス

画像用のサーフェスを作成するには，画像サイズと画像形式を指定して白紙状態のサーフェスを作成する方法と，既にある画像データを元にサーフェスを作成する方法があります．

画像サイズと画像形式を指定してサーフェスを作成するには，関数 cairo_image_surface_create を使用します．

```
cairo_surface_t* cairo_image_surface_create (cairo_format_t format,
                                             int            width,
                                             int            height);
```

第 1 引数　画像形式（表 5.1 を参照）
第 2 引数　画像の幅
第 3 引数　画像の高さ
戻り値　サーフェス

表 5.1　画像サーフェスの種類

値	説明
CAIRO_FORMAT_ARGB32	アルファチャネルを持つ RGB 画像
CAIRO_FORMAT_RGB24	RGB 画像
CAIRO_FORMAT_A8	アルファチャネルを持つ 8 ビットのグレースケール画像
CAIRO_FORMAT_A1	アルファチャネルを持つ 1 ビット画像

画像データからサーフェスを作成する

画像データからサーフェスを作成するには，メモリ上のデータを用いる場合と，画像ファイル（PNG 形式）から作成する方法があります．メモリ上のデータからサーフェスを作成する方法は，そのためのデータを用意するよりも，5.11.3 節（p. 88）で紹介するように GdkPixbuf データをソースとして設定するほうが簡単です．

メモリ上のデータを用いるには，次の関数 cairo_image_surface_create_for_data を使用します．

```
cairo_surface_t*
cairo_image_surface_create_for_data (unsigned char  *data,
                                     cairo_format_t format,
                                     int            width,
                                     int            height,
                                     int            stride);
```

第 1 引数　画像データ
第 2 引数　画像形式（表 5.1 を参照）
第 3 引数　画像の幅
第 4 引数　画像の高さ
第 5 引数　画像の 1 行分のデータ数
戻り値　サーフェス

PNG ファイルから作成する場合には，次の関数 cairo_image_surface_create_from_png を使います．

```
cairo_surface_t*
cairo_image_surface_create_from_png (const char *filename);
```

第 1 引数　PNG 形式の画像ファイル名
戻り値　サーフェス

サーフェスを PNG ファイルに出力する

また逆に，サーフェスの内容を PNG 形式の画像ファイルとして出力する関数 cairo_surface_write_to_png も用意されています．

```
cairo_status_t cairo_surface_write_to_png (cairo_surface_t *surface,
                                           const char      *filename);
```

第 1 引数	サーフェス
戻り値	cairo_status_t で定義された処理結果に応じた値

5.3.3　PS（EPS）ファイル

出力先を PS ファイルにする場合には，関数 cairo_ps_surface_create を使用します．画像の幅と高さの指定はポイント（1/72 インチ）で指定します．

```
cairo_surface_t* cairo_ps_surface_create (const char *filename,
                                          double     width_in_points,
                                          double     height_in_points);
```

第 1 引数	ファイル名
第 2 引数	画像の幅
第 3 引数	画像の高さ
戻り値	サーフェス

さらに，出力先を EPS ファイルに設定したい場合には，上記の関数で作成したサーフェスに対して，関数 cairo_ps_surface_set_eps を呼び出します．

```
void cairo_ps_surface_set_eps (cairo_surface_t *surface,
                               cairo_bool_t    eps);
```

第 1 引数	サーフェス
第 2 引数	EPS 形式にするかどうかを TRUE もしくは FALSE で指定する

PS ファイルや次節の SVG ファイルのような複数ページをサポートする画像形式では，サーフェスに描画した内容を出力ファイルに反映させるために，関数 cairo_show_page を呼び出す必要があります．

```
void cairo_show_page (cairo_t *cr);
```

これらの関数を呼び出すと，描画内容をページに反映させると同時に，サーフェスの内容をクリアします．描画内容を次のページでも使用したい場合には，代わりに関数 cairo_copy_page を呼び出します．

```
void cairo_copy_page (cairo_t *cr);
```

また，ファイルへの出力を終了する際は，関数 cairo_surface_destroy を呼び出さなければいけません．

```
void cairo_surface_destroy (cairo_surface_t *surface);
```

5.3.4　SVG ファイル

出力先を SVG ファイルにする場合には，関数 cairo_svg_surface_create を使用します．

```
cairo_surface_t* cairo_svg_surface_create (const char *filename,
                                           double     width_in_points,
                                           double     height_in_points);
```

第 1 引数	ファイル名
第 2 引数	画像の幅
第 3 引数	画像の高さ
戻り値	サーフェス

5.4　コンテキストの作成

サーフェスを作成した後，それを引数として関数 cairo_create によってコンテキストを作成します．

```
cairo_t* cairo_create (cairo_surface_t *target);
```

第 1 引数	サーフェス
戻り値	コンテキスト

サーフェスが GTK+ アプリケーションのドローアブルの場合は，そのドローアブルを引数にして関数 gdk_cairo_create を呼び出すことで，コンテキストを作成します．

5.5　線分の描画

本節では線分の描画と，線分の属性を設定する方法について説明します．5.2 節（p. 69）で説明したように，図形の描画はパスを作成することによって行われます．このパスを構成する最も単純な要素は線分です．

線分のパスを作成して，描画する手順を以下に示します．

1. パスの始点を設定（関数 cairo_move_to）
2. パスの終点を設定（関数 cairo_line_to）
3. 作成したパスの内容を描画（関数 cairo_stroke）

```
void cairo_move_to (cairo_t *cr, double x, double y);

void cairo_line_to (cairo_t *cr, double x, double y);

void cairo_stroke (cairo_t *cr);
```

もしパスの始点が指定されていない場合には，関数 cairo_line_to が関数 cairo_move_to の役割を果たすので，関数 cairo_line_to を 2 回呼び出しても，線分のパスを作成できます．また，1 つの線分パスを作成した後，続けて関数 cairo_line_to を呼び出すと，直前に作成した線分の終点と新たに指定した点を結んだ線分のパスが作成され，折れ線のパスが作成されます．

cairo では，パスを作成しただけでは図形は描画されません．パスの内容を描画するために，関数 cairo_stroke を呼び出します．関数 cairo_stroke は，パスの内容を描画すると同時に，そのパスをクリアします．パスを再利用したい場合は，描画後もパスを保持していると便利です．このような場合には，関数 cairo_stroke の代わりに関数 cairo_stroke_preserve を使用します．

```
void cairo_stroke_preserve (cairo_t *cr);
```

例えば，始点 $(50, 50)$ と終点 $(100, 100)$ を結んだ線分を描画したい場合には，次のようにします．

```
cairo_t *cr;
cairo_move_to (cr, 50, 50);
cairo_line_to (cr, 100, 100);
cario_stroke (cr);
```

5.6　線分の属性と描画サンプル

cairo では，線分のパスに対して次の属性を設定できます．

- 線分の太さ
- 線端の種類
- 接続の種類
- 線分の種類

以下では，線分の属性を変えて描画したプログラム例と合わせて説明します．

5.6.1　線分の太さ

線分の太さは，関数 cairo_set_line_width で変更できます．

```
void cairo_set_line_width (cairo_t *cr, double width);
```

線分の太さは，標準では 2.0 画素に設定されています．

5.6.2　線端の種類

線端の種類には，表 5.2 に示した cairo_line_cap_t 列挙体で定義された 3 種類があります．描画される線端は図 5.2 のようになります．

表 5.2　線端の種類

値	説明
CAIRO_LINE_CAP_BUTT	始点と終点をそのまま描画する．
CAIRO_LINE_CAP_ROUND	線端を丸く描画する．
CAIRO_LINE_CAP_SQUARE	線端を線幅の半分だけはみだして描画する．

図 5.2 に示したように，CAIRO_LINE_CAP_ROUND と CAIRO_LINE_CAP_SQUARE を設定した場合は，線分の両端は指定した座標よりも線の太さの半分のサイズだけはみ出します．これらの値を設定するためには，関数 cairo_set_line_cap を使用します．

```
void cairo_set_line_cap (cairo_t *cr, cairo_line_cap_t line_cap);
```

第 1 引数　コンテキスト
第 2 引数　線端の種類（表 5.2 を参照）

図 5.2　線端の種類

ソース 5–1 に，図 5.2 のソースコードを示します．このソース 5–1 では描画する線分の色を，関数 cairo_set_source_rgb を使って設定しています．線分の色も線分の属性だと思われるかもしれませんが，色情報はソースの属性です．ソースは標準では RGB の単色（初期状態では黒）に設定されています．cairo ではソースとして，単色だけでなく，グラデーション等のパターンや画像等も設定することができます．

ソース 5–1　線端の種類：cairo-line-cap.c

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

gboolean
cb_draw (GtkWidget      *widget,
         cairo_t        *cr,
         gpointer       user_data)
{
  double    line_width = 30.0;

  cairo_set_line_width (cr, line_width);

  cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
  cairo_set_line_cap (cr, CAIRO_LINE_CAP_BUTT);
  cairo_move_to (cr,  50.0, 50.0);
```

```
15   cairo_line_to (cr, 350.0, 50.0);
16   cairo_stroke (cr);
17
18   cairo_set_source_rgb (cr, 0.0, 1.0, 0.0);
19   cairo_set_line_cap (cr, CAIRO_LINE_CAP_ROUND);
20   cairo_move_to (cr,  50.0, 100.0);
21   cairo_line_to (cr, 350.0, 100.0);
22   cairo_stroke (cr);
23
24   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
25   cairo_set_line_cap (cr, CAIRO_LINE_CAP_SQUARE);
26   cairo_move_to (cr,  50.0, 150.0);
27   cairo_line_to (cr, 350.0, 150.0);
28   cairo_stroke (cr);
29
30   cairo_set_line_width (cr, 1);
31   cairo_set_source_rgb (cr, 0.0, 0.0, 0.0);
32
33   cairo_move_to (cr, 50.0,  15.0);
34   cairo_line_to (cr, 50.0, 185.0);
35   cairo_stroke (cr);
36
37   cairo_move_to (cr, 50.0 - line_width / 2.0,  15.0);
38   cairo_line_to (cr, 50.0 - line_width / 2.0, 185.0);
39   cairo_stroke (cr);
40
41   return FALSE;
42 }
43
44 int
45 main (int argc, char *argv[])
46 {
47   GtkWidget *window;
48   GtkWidget *canvas;
49
50   gtk_init (&argc, &argv);
51
52   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
53   gtk_window_set_title (GTK_WINDOW (window), "cairo_line_cap␣Sample");
54   gtk_widget_set_size_request (window, 400, 200);
55   g_signal_connect (G_OBJECT (window), "destroy",
56                     G_CALLBACK (gtk_main_quit), NULL);
57
58   canvas = gtk_drawing_area_new ();
59   gtk_container_add (GTK_CONTAINER (window), canvas);
60   g_signal_connect (G_OBJECT (canvas), "draw",
61                     G_CALLBACK (cb_draw), NULL);
62
63   gtk_widget_show_all (window);
64   gtk_main ();
65
66   return 0;
67 }
```

5.6.3　接続の種類

線分と線分をつなぐ接続の種類には，表 5.3 に示した cairo_line_join_t 列挙体で定義された 3 種類があります．実際に描画した例は図 5.3 のようになります．

表 5.3　接続の種類

値	説明
CAIRO_LINE_JOIN_MITER	接続部分をとがらせる．
CAIRO_LINE_JOIN_ROUND	接続部分を丸くする．
CAIRO_LINE_JOIN_BEVEL	接続部分のとがった部分をカットしたような形にする．

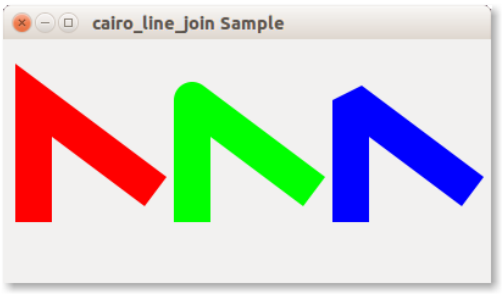


図 5.3　接続の種類

これらの値を設定するためには関数 cairo_set_line_join を使用します．

```
void cairo_set_line_join (cairo_t *cr, cairo_line_join_t line_join);
```

第 1 引数　コンテキスト
第 2 引数　接続の種類（表 5.3 を参照）

ソース 5–2 に，図 5.3 のソースコードを示します．関数 cairo_set_line_join の使い方に関しては説明する必要はないでしょう．

このソースコード中では，5.12 節（p. 92）で説明する関数 cairo_translate を使用しています．この関数は，パスを作成する際の座標系原点を，この関数で指定した値だけ平行移動させます．22 行目から始まる for ループ中で同じ座標を指定して 3 つの折れ線を描画していますが，ループの最後の行で関数 cairo_translate を呼び出しているので，横方向に平行移動した位置にそれぞれ折れ線が描画されています．

ソース 5–2 接続の種類：cairo_line_join.c（一部を抜粋）

```
 1 #include <gtk/gtk.h>
 2
 3 gboolean
 4 cb_draw (GtkWidget      *widget,
 5           cairo_t        *cr,
 6           gpointer       user_data)
 7 {
 8   cairo_line_join_t join_type[] = {CAIRO_LINE_JOIN_MITER,
 9                                    CAIRO_LINE_JOIN_ROUND,
10                                    CAIRO_LINE_JOIN_BEVEL};
11   double color[3][3] = {{1.0, 0.0, 0.0},
12                         {0.0, 1.0, 0.0},
13                         {0.0, 0.0, 1.0}};
14   int n;
15   cairo_set_line_width (cr, 30.0);
16
17   for (n = 0; n < 3; n++)
18     {
19       cairo_set_source_rgb (cr, color[n][0], color[n][1], color[n][2]);
20       cairo_set_line_join (cr, join_type[n]);
21       cairo_move_to (cr,  25.0, 150.0);
22       cairo_line_to (cr,  25.0,  50.0);
23       cairo_line_to (cr, 125.0, 125.0);
24       cairo_stroke (cr);
25       cairo_translate (cr, 130.0, 0.0);
26     }
27   return FALSE;
28 }
```

5.6.4　線分の種類

線分は，関数 cairo_set_dash を使用してそのパターンを設定することで，破線や点線等として描画することができます．

```
void cairo_set_dash (cairo_t       *cr,
                     const double  *dashes,
                     int           num_dashes,
                     double        offset);
```

第 1 引数	コンテキスト
第 2 引数	破線のパターン
第 3 引数	第 2 引数の配列の要素数
第 4 引数	オフセット

この関数を使用して破線と鎖線を描画した例を図 5.4 に示します．ソースコードはソース 5–3 です．

図 5.4　線分の種類

関数 cairo_set_dash の使い方

図 5.4 の例を用いて，関数 cairo_set_dash の引数の意味と，設定方法を説明します．

まず，破線のパターンを設定するのが第 2 引数です．第 2 引数は double 型の配列で，この配列には，前景の長さと背景の長さを交互に格納します．

図 5.4 の一番上の破線では，ソース 5–3 の 10 行目でパターンを設定しており，前景を 40，背景を 5 の間隔で繰り返す破線となっています．また真ん中の鎖線では，前景 40，背景 5，前景 5，背景 5 というパターンの鎖線を設定しています．

第 3 引数は，第 2 引数で与える配列の要素数です．

第 4 引数では，パターンの開始位置を設定します．つまり，正の値を与えると，線分のパターンを左にシフトし，負の値を与えると，反対に線分のパターンを右にシフトしたような結果が得られます．図 5.4 の一番下の鎖線では真ん中の鎖線と同じパターンを描画していますが，第 4 引数を −5 としているので，真ん中の鎖線を右にシフトしたように描画されていることがわかると思います．

ソース 5–3 線分の種類：cairo_line_dash.c（一部を抜粋）

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

gboolean
cb_draw (GtkWidget      *widget,
         cairo_t        *cr,
         gpointer       user_data)
{
  double    dash_pattern1[] = {40.0, 5.0};
  double    dash_pattern2[] = {40.0, 5.0, 5.0, 5.0};

  cairo_set_line_width (cr, 5.0);

  cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
  cairo_set_dash (cr, dash_pattern1,
                  sizeof (dash_pattern1) / sizeof (dash_pattern1[0]),
                  0);
  cairo_move_to (cr,  50.0, 50.0);
  cairo_line_to (cr, 350.0, 50.0);
  cairo_stroke (cr);

  cairo_set_source_rgb (cr, 0.0, 1.0, 0.0);
  cairo_set_dash (cr, dash_pattern2,
                  sizeof (dash_pattern2) / sizeof (dash_pattern2[0]),
                  0);
  cairo_move_to (cr,  50.0, 100.0);
  cairo_line_to (cr, 350.0, 100.0);
  cairo_stroke (cr);

  cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
```

```
30   cairo_set_dash (cr, dash_pattern2,
31                   sizeof (dash_pattern2) / sizeof (dash_pattern2[0]),
32                   -5);
33   cairo_move_to (cr,  50.0, 150.0);
34   cairo_line_to (cr, 350.0, 150.0);
35   cairo_stroke (cr);
36
37   return FALSE;
38 }
```

5.7　矩形の描画

矩形を描画するには，線分と同様に矩形のパスを作成して，関数 cairo_stroke を呼び出します．矩形のパスを作成するには，関数 cairo_rectangle を使用します．x と y に矩形の左上の座標を，width と height にそれぞれ矩形の幅と高さを指定します．

```
void cairo_rectangle (cairo_t *cr,
                      double   x,
                      double   y,
                      double   width,
                      double   height);
```

第 1 引数	コンテキスト
第 2 引数	左上の x 座標
第 3 引数	左上の y 座標
第 4 引数	幅
第 5 引数	高さ

矩形描画の例を図 5.5 に，対応するソースコードをソース 5–4 に示します．

真ん中の矩形と右の矩形は塗りつぶされています．矩形のような閉じたパスの内部を塗りつぶすには，関数 cairo_fill を使用します．関数 cairo_stroke と同様に，描画後もパスを保持するには関数 cairo_fill_preserve を使用します．

```
void cairo_fill (cairo_t *cr);

void cairo_fill_preserve (cairo_t *cr);
```

また，真ん中と右の矩形からは，線分の太さが 1 よりも大きい場合，パスの描画と内部の塗りつぶしの順番によって描画結果が異なることがわかります．

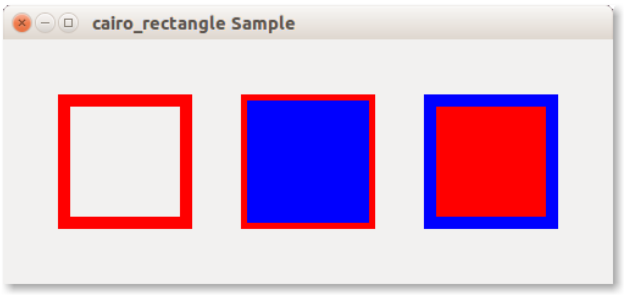


図 5.5　矩形の描画

ソース 5–4　矩形の描画：cairo_rectangle.c（一部を抜粋）

```
1 #include <gtk/gtk.h>
2
3 gboolean
4 cb_draw (GtkWidget      *widget,
5          cairo_t        *cr,
6          gpointer       user_data)
7 {
```

```
8    cairo_set_line_width (cr, 10.0);
9
10   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
11   cairo_rectangle (cr,  50.0, 50.0, 100.0, 100.0);
12   cairo_stroke (cr);
13
14   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
15   cairo_rectangle (cr, 200.0, 50.0, 100.0, 100.0);
16   cairo_stroke_preserve (cr);
17   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
18   cairo_fill (cr);
19
20   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
21   cairo_rectangle (cr, 350.0, 50.0, 100.0, 100.0);
22   cairo_fill_preserve (cr);
23   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
24   cairo_stroke (cr);
25
26   return FALSE;
27 }
```

5.8　多角形の描画

本節では，閉じたパスを描画する方法を説明します．節のタイトルは「多角形の描画」となっていますが，パスの一部が曲線でも問題ありません．

図 5.6 に，3 通りの方法で描画した多角形の例を示します．左の多角形は，5.5 節（p. 73）で説明した方法で，始点から終点（この場合始点と終点は同一の点）までのパスを作成して描画したものです．始点と終点がきれいにつながっていないことがわかります．

真ん中の多角形は，始点まで戻らないで 1 つ前の点を終点としてパスを作成して，描画したものです．この場合，最後の線分に当たるパスを作成していないので，線分が描画されていませんが，内部の領域は正しく描画されています．

右の多角形は，パスは真ん中の多角形と同様ですが，描画前に関数 cairo_close_path を呼び出しています．この関数はパスを閉じるための関数で，始点と終点を結んだ閉じたパスを作成します．

```
void cairo_close_path (cairo_t *cr);
```

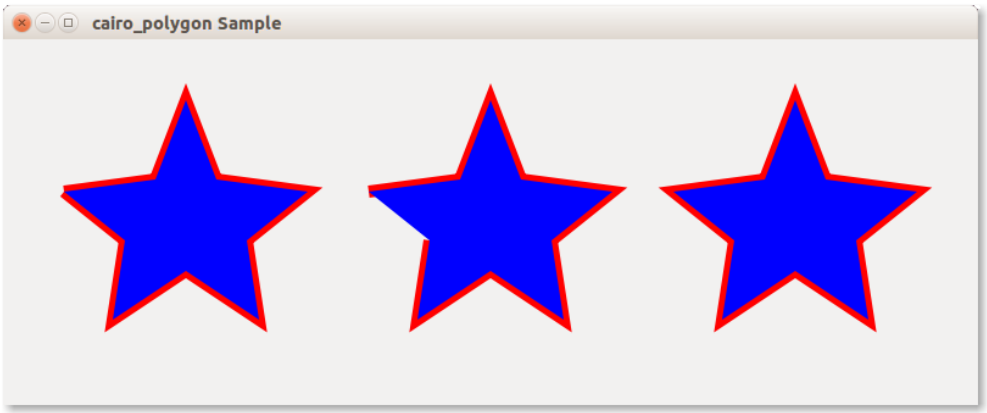


図 5.6　多角形の描画

ソース 5–5 に図 5.6 のソースコードを示します．

ソース 5–5　多角形の描画：cairo_polygon.c（一部を抜粋）

```
1  #include <gtk/gtk.h>
2
3  gboolean
4  cb_draw (GtkWidget      *widget,
5           cairo_t        *cr,
6           gpointer       user_data)
7  {
8    double    points[11][2] = {{50.0, 125.0}, {125.0, 115.0},
```

```
 9                                {150.0, 50.0}, {175.0, 115.0},
10                                {250.0, 125.0}, {200.0, 165.0},
11                                {210.0, 230.0}, {150.0, 190.0},
12                                {90.0, 230.0}, {100.0, 165.0},
13                                {50.0, 125.0}};
14   int        n;
15
16   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
17   cairo_set_line_width (cr, 10.0);
18   for (n = 0; n < 11; n++)
19     {
20       cairo_line_to (cr, points[n][0], points[n][1]);
21     }
22   cairo_stroke_preserve (cr);
23   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
24   cairo_set_line_width (cr, 1.0);
25   cairo_fill (cr);
26   cairo_translate (cr, 250.0, 0.0);
27
28   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
29   cairo_set_line_width (cr, 10.0);
30   for (n = 0; n < 10; n++)
31     {
32       cairo_line_to (cr, points[n][0], points[n][1]);
33     }
34   cairo_stroke_preserve (cr);
35   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
36   cairo_set_line_width (cr, 1.0);
37   cairo_fill (cr);
38   cairo_translate (cr, 250.0, 0.0);
39
40   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
41   cairo_set_line_width (cr, 10.0);
42   for (n = 0; n < 10; n++)
43     {
44       cairo_line_to (cr, points[n][0], points[n][1]);
45     }
46   cairo_close_path (cr);
47   cairo_stroke_preserve (cr);
48   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
49   cairo_set_line_width (cr, 1.0);
50   cairo_fill (cr);
51
52   return FALSE;
53 }
```

5.9 円弧の描画

円弧を描画するには，関数 cairo_arc を使用します．

```
void cairo_arc (cairo_t *cr, double  xc, double  yc, double radius,
                double  angle1, double angle2);
```

第 1 引数	コンテキスト
第 2 引数	円中心の x 座標
第 3 引数	円中心の y 座標
第 4 引数	円の半径
第 5 引数	開始角度
第 6 引数	終了角度

図 5.7 に，関数 cairo_arc で与える引数と，描画される円弧の関係を示します．xc と yc は円弧の中心座標を，radius は円弧の半径を表します．angle1 と angle2 は円弧の開始角度と終了角度で，それぞれ円弧の中心から水平方向右方向を 0° として，単位ラジアンの絶対角度で与えます．

円弧の描画例を図 5.8 に示します．円弧パス内部を塗りつぶした例も示しました．図 5.8 のソースコードはソース 5–6 です．

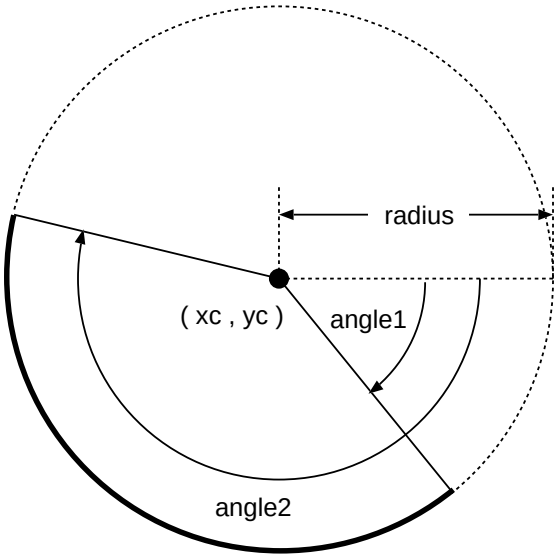


図 5.7 円弧描画のためのパラメータ

ソース 5–6 円弧の描画：cairo_arc.c（一部を抜粋）

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

gboolean
cb_draw (GtkWidget      *widget,
         cairo_t        *cr,
         gpointer       user_data)
{
  cairo_set_line_width (cr, 10.0);
  cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);

  cairo_arc (cr, 100.0, 100.0, 80.0, 0.0, 2.0 * M_PI);
  cairo_stroke (cr);

  cairo_arc (cr, 300.0, 100.0, 80.0, 0.0, 3.0 * M_PI / 4.0);
  cairo_stroke (cr);

  cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);

  cairo_arc (cr, 300.0, 100.0, 80.0,
             5.0 * M_PI / 4.0, 7.0 * M_PI / 4.0);
  cairo_fill (cr);

  return FALSE;
}
```

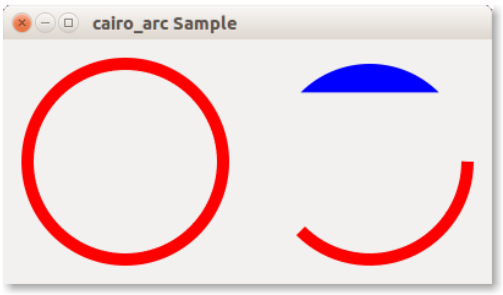


図 5.8 円弧の描画

5.10　曲線の描画

関数 cairo_curve_to を使用すると，ベジエ曲線（Bézier curve）を描画できます．

```
void cairo_curve_to (cairo_t *cr,
                     double  x1, double y1,
                     double  x2, double y2,
                     double  x3, double y3);
```

第 1 引数	コンテキスト
第 2 引数	ベジェ曲線の制御点 1
第 3 引数	ベジェ曲線の制御点 2
第 4 引数	終点

関数 cairo_move_to などでパスの始点を決めた後，関数 cairo_curve_to を呼び出します．座標 (x1，y1) と (x2，y2) はベジエ曲線を描画するための制御点で，(x3，y3) は終点の座標を表します．ここではベジエ曲線や制御点についての詳細は省略します．

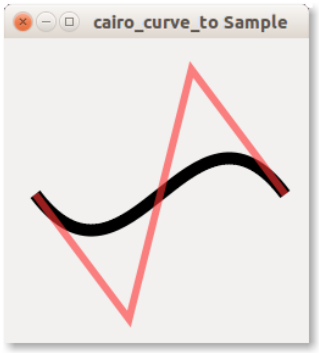


図 5.9　ベジエ曲線の描画

曲線の描画とは関係ありませんが，このソースではソースの色設定に関数 cairo_set_source_rgba を使用して，透明度を持った色を設定して線分を描画しています．

ソース 5–7　ベジエ曲線の描画：cairo_curve.c（一部を抜粋）

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

gboolean
cb_draw (GtkWidget      *widget,
         cairo_t        *cr,
         gpointer       user_data)
{
  double    x0 = 25.6,   y0 = 128.0;
  double    x1 = 102.4,  y1 = 230.4;
  double    x2 = 153.6,  y2 = 25.6;
  double    x3 = 230.4,  y3 = 128.0;

  cairo_set_line_width (cr, 10.0);
  cairo_set_source_rgb (cr, 0.0, 0.0, 0.0);

  cairo_move_to (cr, x0, y0);
  cairo_curve_to (cr, x1, y1, x2, y2, x3, y3);
  cairo_stroke (cr);

  cairo_set_line_width (cr, 6.0);
  cairo_set_source_rgba (cr, 1.0, 0.2, 0.2, 0.6);

  cairo_move_to (cr, x0, y0);
  cairo_line_to (cr, x1, y1);
  cairo_line_to (cr, x2, y2);
  cairo_line_to (cr, x3, y3);
```

```
27    cairo_stroke (cr);
28
29    return FALSE;
30  }
```

5.11　ソースの設定

ソースは，パスを描画する際の色やパターンに相当するものです．ソースとして使用できるものを次に挙げます．

- 単色
- グラデーション
- 画像
- サーフェス

5.11.1　単色のソース

ユーザが設定しなければ，初期状態でソースは単色のサーフェスです．サーフェスの色を設定するためには，関数 cairo_set_source_rgb もしくは関数 cairo_set_source_rgba を使用します．

```
void cairo_set_source_rgb (cairo_t *cr,
                           double red, double green, double blue);
```

第 1 引数	コンテキスト
第 2 引数	赤成分の値
第 3 引数	緑成分の値
第 4 引数	青成分の値

```
void cairo_set_source_rgba (cairo_t *cr,
                            double red, double green, double blue,
                            double alpha);
```

第 1 引数	コンテキスト
第 2 引数	赤成分の値
第 3 引数	緑成分の値
第 4 引数	青成分の値
第 5 引数	アルファ成分の値

RGB 値とアルファ値は，すべて 0 から 1 の範囲の値で与えます．透明度を変えながら矩形を描画した例を図 5.10 に示します．

図 5.10　透明度の設定

ソース 5–8 RGBA サーフェスの設定：cairo_transparency.c（一部を抜粋）

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

gboolean
cb_draw (GtkWidget      *widget,
         cairo_t        *cr,
         gpointer       user_data)
{
  int        n;

  cr = gdk_cairo_create (drawable);

  for (n = 1; n <= 10; n++)
    {
      cairo_set_source_rgba (cr, 0.0, 0.0, 1.0, n * 0.1);
      cairo_rectangle (cr, 50.0 * n, 20.0, 40.0, 40.0);
      cairo_fill (cr);
    }
  return FALSE;
}
```

5.11.2　グラデーションパターンのソース

cairo で作成できるグラデーションパターンには次の 2 通りがあります．

- 線形のグラデーションパターン
- 放射状のグラデーションパターン

グラデーションパターンの作成とグラデーションパターンの描画は次の手順で行います．

1. パターンの大きさの設定
2. グラデーションパターンの設定
3. パターンをソースに設定
4. パスを描画

手順 1，2 によって作成したパターンをソースとして設定するには，関数 cairo_set_source を使用します．

```
void cairo_set_source (cairo_t *cr, cairo_pattern_t *source);
```

第 1 引数	コンテキスト
第 2 引数	パターン

手順 1，2 については，このあとグラデーションパターンごとに説明していきます．

線形のグラデーションパターン

線形のグラデーションパターンを作成するには，まず関数 cairo_pattern_create_linear によって，グラデーションパターンを作成する矩形の左上と右下の座標を指定します．

```
cairo_pattern_t* cairo_pattern_create_linear (double x0, double y0,
                                              double x1, double y1);
```

第 1 引数	グラデーション開始の x 座標
第 2 引数	グラデーション開始の y 座標
第 3 引数	グラデーション終了の x 座標
第 4 引数	グラデーション終了の y 座標
戻り値	パターン

パターン領域を作成する際の座標と，パスを作成する際の座標は，同じ座標系となっています．したがって，パターンを作成した位置と描画する位置がずれている場合には，意図したように正しく描画されません．そのため，パターンを作成する際には，描画する領域や描画の仕方を考慮して，パターンを作成する位置と大きさを指定しなければいけません．ただし，5.12 節

(p. 92) で紹介する cairo_matrix_t 型の変換行列を用いて，作成したパターンを変換（実際には描画領域を変換）することによって，作成したパターンの位置や大きさにとらわれず，パターンを描画することが可能です．

パターン領域を作成したら，次は具体的にどのようなグラデーションにするのかを，次の関数を使用して設定します．

```
void cairo_pattern_add_color_stop_rgb (cairo_pattern_t *pattern,
                                       double          offset,
                                       double          red,
                                       double          green,
                                       double          blue);
```

第 1 引数	パターン
第 2 引数	オフセット
第 3 引数	赤成分の値
第 4 引数	緑成分の値
第 5 引数	青成分の値

```
void cairo_pattern_add_color_stop_rgba (cairo_pattern_t *pattern,
                                        double          offset,
                                        double          red,
                                        double          green,
                                        double          blue,
                                        double          alpha);
```

第 1 引数	パターン
第 2 引数	オフセット
第 3 引数	赤成分の値
第 4 引数	緑成分の値
第 5 引数	青成分の値
第 6 引数	アルファ成分の値

それぞれの関数の第 3 から第 5 引数には，グラデーションパターンに使用する RGB 情報を 0 から 1 の範囲で指定します．第 2 引数は，その色を作成したパターン領域のどの位置に配置するかを指定する値で，0 から 1 までの範囲で指定します．

図 5.11 を例に，線形グラデーションパターンの作成方法について具体的に説明していきます．ソースコードは，ソース 5–9 になります．この例では，次に示す 3 種類のグラデーションパターンを作成しています．

- 水平方向のグラデーション（2 つの色の混合）
- 垂直方向のグラデーション（アルファ値を線形に変化）
- 斜め方向のグラデーション

水平方向のグラデーション グラデーションの方向は，関数 cairo_pattern_create_linear に与える座標パラメータで決定します．水平方向のグラデーションを作成したい場合には，水平方向の座標をパターンの大きさだけ変化させます．一方，垂直方向の座標は変化量を 0 に，すなわち同じ値を与えます．

ソース 5–9 の pattern1 では 16 行目でパターン領域を決定していますが，水平方向は 50.0 から 350.0 まで増加し，垂直方向の座標は同じ値（50.0）になっています．

次に 17–18 行目で、RGB 値を変化させたグラデーションパターンを設定しています．そして作成したパターンを，19 行目でソースに設定して，20 行目で同じ幅の矩形パスを作成して，21 行目で描画しています．

垂直方向のグラデーション 次に、垂直方向のグラデーション（pattern2）を作成する場合には，水平方向のグラデーションの場合とは反対に，垂直方向の座標を変化させ（120.0 から 170.0），水平方向の座標に同じ値（50.0）を与えます（23 行目）．

さらに，この例では RGB 値は変化させず，アルファ値を 1 から 0 にグラデーションさせたパターンを，関数 cairo_pattern_add_color_stop_rgba で作成して（24–25 行目），描画しています（26–28 行目）．

斜め方向のグラデーション 最後に斜め方向のグラデーションパターン（pattern3）について説明します．関数 cairo_pattern_create_linear に与えるパラメータを垂直方向も水平方向も変化させることで（30 行目），2 つの座標 (x0, y0)–(x1, y1) を結んだ方向に，グラデーションが生成されます．さらに pattern3 では，パターンの位置と大きさが上記の 2 つのグラデーションの場合と異なっています。pattern1 と pattern2 では，作成されたパターンと描画されるパスが一致していま

図 5.11　線形グラデーションパターン

した。pattern3 では，グラデーションパターンを (0，0)–(50，50) の領域で作成して，実際に描画する領域は位置も大きさも変化させています．

パターンを異なる位置に描画したい　関数 cairo_translate を使用して，パスを作成する座標がパターンを作成した位置と一致するように，座標系を平行移動します（36 行目）．37 行目でパスを作成する際には，パターンを作成した位置とパスの始点が一致していることがわかると思います．

パターンより大きな描画領域　描画領域がパターンよりも大きい場合に，パターン外の領域をどう描画するかは，関数 cairo_pattern_set_extend で設定します．

```
void cairo_pattern_set_extend (cairo_pattern_t *pattern,
                               cairo_extend_t  extend);
```

描画の方法には，表 5.4 に示す cairo_extend_t 列挙体で定義された 4 種類の方法があり，初期状態では CAIRO_EXTEND_NONE が設定されています．この例では，35 行目で CAIRO_EXTEND_REPEAT が設定されているため，パターンが繰り返し描画されます．

表 5.4　パターン外の領域の描画設定

値	説明
CAIRO_EXTEND_NONE	何も描画しない．
CAIRO_EXTEND_REPEAT	パターンをタイル状に繰り返し描画する．
CAIRO_EXTEND_REFLECT	パターンを折り返してタイル状に描画する．
CAIRO_EXTEND_PAD	パターンの端の値で描画する．

ソース 5–9　線形グラデーションパターン：cairo_pattern_linear.c（一部を抜粋）

```
1  #include <gtk/gtk.h>
2
3  gboolean
4  cb_draw (GtkWidget      *widget,
5           cairo_t        *cr,
6           gpointer       user_data)
7  {
8    cairo_pattern_t *pattern1;
9    cairo_pattern_t *pattern2;
10   cairo_pattern_t *pattern3;
11
12   pattern1 = cairo_pattern_create_linear (50.0, 50.0, 350.0, 50.0);
13   cairo_pattern_add_color_stop_rgb (pattern1, 0.0, 1.0, 0.0, 0.0);
14   cairo_pattern_add_color_stop_rgb (pattern1, 1.0, 1.0, 1.0, 1.0);
15   cairo_set_source (cr, pattern1);
16   cairo_rectangle (cr, 50.0, 50.0, 300.0, 50.0);
17   cairo_fill (cr);
18
19   pattern2 = cairo_pattern_create_linear (50.0, 120.0, 50.0, 170.0);
20   cairo_pattern_add_color_stop_rgba (pattern2, 0.0, 0.0, 1.0, 0.0, 1.0);
```

```
21   cairo_pattern_add_color_stop_rgba (pattern2, 1.0, 0.0, 1.0, 0.0, 0.0);
22   cairo_set_source (cr, pattern2);
23   cairo_rectangle (cr, 50.0, 120.0, 300.0, 50.0);
24   cairo_fill (cr);
25
26   pattern3 = cairo_pattern_create_linear (0, 0.0, 50.0, 50.0);
27   cairo_pattern_add_color_stop_rgb (pattern3, 0.0, 0.0, 0.0, 0.0);
28   cairo_pattern_add_color_stop_rgb (pattern3, 0.5, 1.0, 1.0, 0.0);
29   cairo_pattern_add_color_stop_rgb (pattern3, 1.0, 0.0, 0.0, 0.0);
30   cairo_pattern_set_extend (pattern3, CAIRO_EXTEND_REPEAT);
31   cairo_set_source (cr, pattern3);
32   cairo_translate (cr, 50.0, 190.0);
33   cairo_rectangle (cr, 0.0, 0.0, 300.0, 70.0);
34   cairo_fill (cr);
35
36   cairo_pattern_destroy (pattern1);
37   cairo_pattern_destroy (pattern2);
38   cairo_pattern_destroy (pattern3);
39
40   return FALSE;
41 }
```

放射状のグラデーションパターン

放射状のグラデーションパターンを作成するには，まず関数 cairo_pattern_create_radial によって，グラデーションパターンを作成する 2 つの円領域を指定します．

```
cairo_pattern_t* cairo_pattern_create_radial (double cx0, double cy0,
                                              double radius0,
                                              double cx1, double cy1,
                                              double radius1);
```

第 1 引数	一つ目の円中心の x 座標
第 2 引数	一つ目の円中心の y 座標
第 3 引数	一つ目の円中心の半径
第 4 引数	二つ目の円中心の x 座標
第 5 引数	二つ目の円中心の y 座標
第 6 引数	二つ目の円中心の半径
戻り値	パターン

第 1 引数から第 3 引数で指定する円はパターンの内側の円を表し，第 4 引数から第 6 引数で指定する円は外側の円を表します．そして，内側の円の円弧から外側の円の円弧に向かって，放射状にグラデーションが生成されます．内側の円の内部は，関数 cairo_pattern_add_color_stop_rgb で設定された内側の円に一番近い色で描画されます．

図 5.12 に放射状のグラデーションパターンで円を描画した例を，そのソースコードをソース 5–10 に示します．

ソース 5–10 放射状グラデーションパターン：cairo_pattern_radial.c（一部を抜粋）

```
1  #include <gtk/gtk.h>
2  #include <math.h>
3
4  gboolean
5  cb_draw (GtkWidget      *widget,
6           cairo_t        *cr,
7           gpointer       user_data)
8  {
9    cairo_pattern_t *pattern;
10
11   pattern = cairo_pattern_create_radial (160.0, 160.0, 30.0,
12                                          200.0, 200.0, 200.0);
13   cairo_pattern_add_color_stop_rgb (pattern, 0.5, 1.0, 0.0, 0.0);
14   cairo_pattern_add_color_stop_rgb (pattern, 1.0, 1.0, 1.0, 1.0);
15   cairo_set_source (cr, pattern);
```

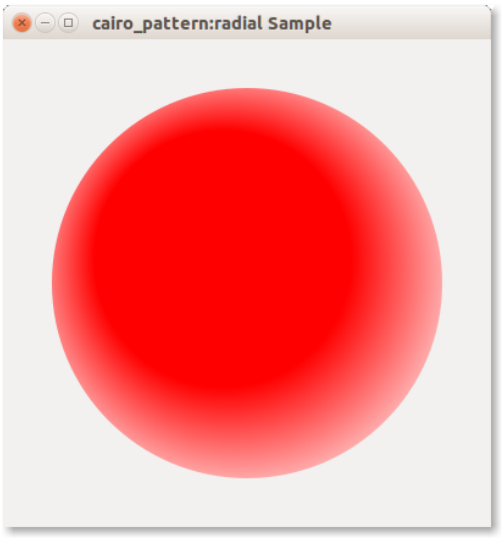


図 5.12　放射状グラデーションパターン

```
16    cairo_arc (cr, 200.0, 200.0, 160.0, 0.0, 2.0 * M_PI);
17    cairo_fill (cr);
18
19    cairo_pattern_destroy (pattern);
20
21    return FALSE;
22  }
```

5.11.3　画像データをソースにする

画像データをソースにする方法は，次の 2 通りあります．

- 画像データを直接ソースに設定する方法
- 画像データを一度サーフェスに描画した後で，そのサーフェスをソースに設定する方法

本項では，1 つ目の方法についてのみ説明します．2 つ目の方法は，次項で説明することにします．

画像データをソースに設定するには，関数 gdk_cairo_set_source_pixbuf を使用します．これは GdkPixbuf 形式の画像データをソースに設定する関数です（GdkPixbuf については 6 章 GdkPixbuf で説明します）．

また，第 3，4 引数は，画像データの原点をソースのどの位置に置くかを指定します．

```
void gdk_cairo_set_source_pixbuf (cairo_t          *cr,
                                  const GdkPixbuf  *pixbuf,
                                  double           pixbuf_x,
                                  double           pixbuf_y);
```

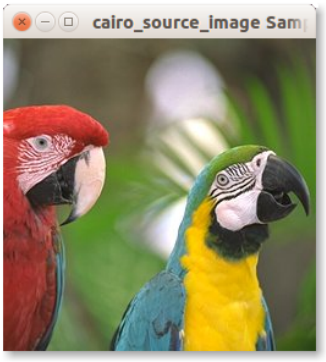


図 5.13　画像ソースの描画

画像データをソースにしてそのまま描画した結果を図 5.13 に，またそのソースコードをソース 5–11 に示します．まず main

関数内の 36 行目で画像ファイルから GdkPixbuf 形式で画像を読み込んで，49 行目でコールバック関数 cb_draw の第 3 引数として設定しています．これによって 4 行目から定義されているコールバック関数の第 3 引数 user_data に 36 行目で生成した GdkPixbuf データが入ります．

次にコールバック関数内の 14 行目で関数 gdk_cairo_set_source_pixbuf を呼び出して，画像データをソースとして設定し，15 行目で関数 cairo_paint によって，ソースの内容を描画しています．この関数はソースの内容をそのままサーフェスに描画する関数で，今回の場合や，サーフェスを現在のソースの色で塗りつぶしたりする場合に，パスを作成することなく描画できるため便利です．

```
void cairo_paint (cairo_t *cr);
```

また，関数 cairo_paint_with_alpha を使うと，透明度を設定してソースの内容を描画できます．

```
void cairo_paint_with_alpha (cairo_t *cr, double alpha);
```

第 1 引数　コンテキスト
第 2 引数　アルファ値

ソース 5–11 画像ソースの描画：cairo_source_image.c

```
1  #include <gtk/gtk.h>
2  #include <stdlib.h>
3
4  gboolean
5  cb_draw (GtkWidget      *widget,
6           cairo_t        *cr,
7           gpointer       user_data)
8  {
9    gdk_cairo_set_source_pixbuf (cr, (GdkPixbuf *) user_data, 0.0, 0.0);
10   cairo_paint (cr);
11
12   return FALSE;
13 }
14
15 int
16 main (int argc, char *argv[])
17 {
18   GtkWidget *window;
19   GtkWidget *canvas;
20   GdkPixbuf *pixbuf;
21
22   if (argc != 2)
23     {
24       g_print ("%s␣imagefile\n", argv[0]);
25       exit (1);
26     }
27   gtk_init (&argc, &argv);
28
29   pixbuf = gdk_pixbuf_new_from_file (argv[1], NULL);
30
31   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
32   gtk_window_set_title (GTK_WINDOW (window),
33                         "cairo_source_image␣Sample");
34   gtk_widget_set_size_request (window,
35                                gdk_pixbuf_get_width (pixbuf),
36                                gdk_pixbuf_get_height (pixbuf));
37   g_signal_connect (G_OBJECT (window), "destroy",
38                     G_CALLBACK (gtk_main_quit), NULL);
39
40   canvas = gtk_drawing_area_new ();
41   gtk_container_add (GTK_CONTAINER (window), canvas);
42   g_signal_connect (G_OBJECT (canvas), "draw",
43                     G_CALLBACK (cb_draw), pixbuf);
44
45   gtk_widget_show_all (window);
46   gtk_main ();
47   g_object_unref (pixbuf);
48
49   return 0;
50 }
```

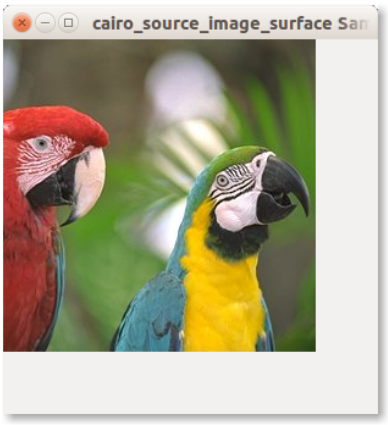


図 5.14　画像サーフェスソースの描画

5.11.4　サーフェスをソースにする

サーフェスは図形が描画される対象ですが，同時にソースとして使用することができます．ここでは次に挙げる 2 通りの方法を説明します．

- サーフェスをそのままソースとして使用する方法
- サーフェスをパターンとして登録して，そのパターンをソースとして使用する方法

サーフェスをそのままソースとして使用する方法

サーフェスをソースとして設定するには，関数 cairo_set_source_surface を使用します．第 3，4 引数には，ソースに設定するサーフェスの原点を，ソースのどの位置に置くかを指定します．

```
void cairo_set_source_surface (cairo_t         *cr,
                               cairo_surface_t *surface,
                               double          x,
                               double          y);
```

図 5.14 に，画像サーフェスをソースにしてそのまま描画した結果を示します．また，図 5.14 のソースコードをソース 5–12 に示します．

この例では，5.3.2 節（p. 71）で紹介した関数 cairo_image_surface_create_from_png を使用して，PNG 形式の画像データからソースに設定するためのサーフェスを作成しています（32 行目）．そしてコールバック関数 cb_draw 中で，関数 cairo_set_source_surface によってこのサーフェスをソースとして設定しています（15 行目）．

この例では描画用のサーフェスを，ソースとして使用するサーフェスよりも大きくしてみました．図 5.14 を見るとわかるように，ソースの外側の領域は何も描画されていません．もし，ソースとして設定したサーフェスの内容をタイル状に繰り返して描画したい場合には，次に説明する方法を使います．

ソース 5–12 画像サーフェスソースの描画：cairo_source_image_surface.c

```
 1 #include <gtk/gtk.h>
 2
 3 gboolean
 4 cb_draw (GtkWidget      *widget,
 5           cairo_t        *cr,
 6           gpointer       user_data)
 7 {
 8   cairo_surface_t *surface;
 9
10   surface = (cairo_surface_t *) user_data;
11   cairo_set_source_surface (cr, surface, 0.0, 0.0);
12   cairo_paint (cr);
13
14   return FALSE;
15 }
16
17 int
```

```
18 main (int argc, char *argv[])
19 {
20   GtkWidget       *window;
21   GtkWidget       *canvas;
22   cairo_surface_t *surface;
23
24   gtk_init (&argc, &argv);
25
26   surface = cairo_image_surface_create_from_png ("./Parrots.png");
27
28   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
29   gtk_window_set_title (GTK_WINDOW (window),
30                         "cairo_source_image_surface␣Sample");
31   gtk_widget_set_size_request
32     (window,
33      cairo_image_surface_get_width (surface)  * 1.2,
34      cairo_image_surface_get_height (surface) * 1.2);
35   g_signal_connect (G_OBJECT (window), "destroy",
36                     G_CALLBACK (gtk_main_quit), NULL);
37
38   canvas = gtk_drawing_area_new ();
39   gtk_container_add (GTK_CONTAINER (window), canvas);
40   g_signal_connect (G_OBJECT (canvas), "draw",
41                     G_CALLBACK (cb_draw), surface);
42
43   gtk_widget_show_all (window);
44   gtk_main ();
45
46   cairo_surface_destroy (surface);
47
48   return 0;
49 }
```

サーフェスをパターンとして使用する方法

サーフェスをパターンとして設定するには，関数 cairo_pattern_create_for_surface を使用します．

```
cairo_pattern_t*
cairo_pattern_create_for_surface (cairo_surface_t *surface);
```

この関数でパターンを作成してしまえば，前節の 5.11.2 節（p. 84）で紹介した方法が利用できます．

ソース 5–13 パターン化した画像サーフェスソースの描画：cairo_pattern_image.c（一部を抜粋）

```
1  #include <gtk/gtk.h>
2
3  gboolean
4  cb_draw (GtkWidget     *widget,
5           cairo_t       *cr,
6           gpointer      user_data)
7  {
8    cairo_surface_t *surface;
9    cairo_pattern_t *pattern;
10   double          points[10][2] = {{ 50.0, 125.0}, {125.0, 115.0},
11                                    {150.0,  50.0}, {175.0, 115.0},
12                                    {250.0, 125.0}, {200.0, 165.0},
13                                    {210.0, 230.0}, {150.0, 190.0},
14                                    { 90.0, 230.0}, {100.0, 165.0}};
15   int n;
16
17   surface = (cairo_surface_t *) user_data;
18   pattern = cairo_pattern_create_for_surface (surface);
19   cairo_set_source (cr, pattern);
20   cairo_pattern_set_extend (cairo_get_source (cr), CAIRO_EXTEND_REPEAT);
21
22   for (n = 0; n < 10; n++)
23     {
24       cairo_line_to (cr, points[n][0], points[n][1]);
25     }
26   cairo_close_path (cr);
27   cairo_fill (cr);
28
29   cairo_pattern_destroy (pattern);
30
```

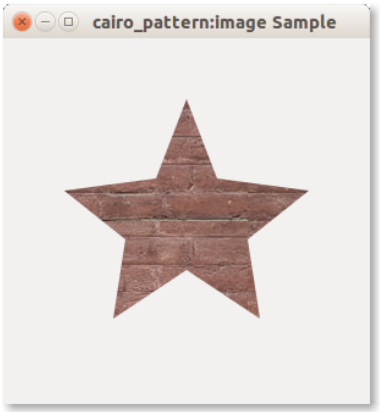


図 5.15　パターン化した画像サーフェスソースの描画

```
31    return FALSE;
32 }
```

5.12　座標系の変換

本節では，パスを作成する座標系の変換について説明します．座標系を変換することで，描画する図形を平行移動したり，拡大縮小，回転することができます．座標系を変換する基本的な関数は次の 3 つです．

```
void cairo_translate (cairo_t *cr, double tx, double ty);
```

第 1 引数　コンテキスト
第 2 引数　座標系の原点の移動先の x 座標
第 3 引数　座標系の原点の移動先の y 座標

```
void cairo_scale (cairo_t *cr, double sx, double sy);
```

第 1 引数　コンテキスト
第 2 引数　座標系の横方向のスケーリングパラメータ
第 3 引数　座標系の縦方向のスケールパラメータ

```
void cairo_rotate (cairo_t *cr, double angle);
```

第 1 引数　コンテキスト
第 2 引数　座標系の回転パラメータ

上記の関数は，平行移動，拡大縮小，回転をそれぞれ単独で行う関数ですが，これらの変換を行列で表現して行列を引数にして座標系の変換を行う関数が，関数 cairo_transform です．

```
void cairo_transform (cairo_t *cr, const cairo_matrix_t *matrix);
```

第 1 引数　コンテキスト
第 2 引数　座標系の変換を表した行列パラメータ

既に関数 cairo_translate についてはこれまで紹介した例で何度か出てきていますが，残りの 3 つの関数も含めて，図 5.16（ソース 5–14）を例に，それぞれの関数の使い方を説明します．図 5.16 では，それぞれ次のような変換を行いながら円を描画しています．

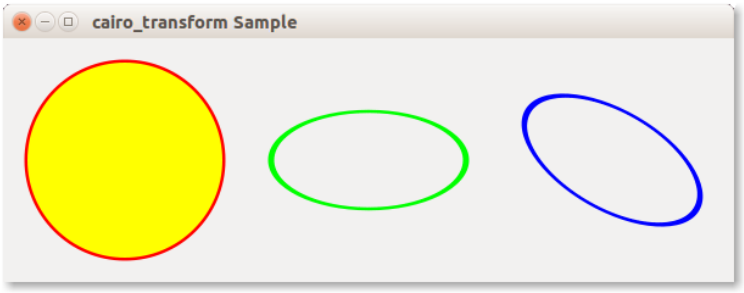


図 5.16　座標系の変換

1. 平行移動（図 5.16 左の円）
   まず，18–20 行目で座標系の原点を (100，100) に平行移動した後，中心 (0，0)，半径 80 の円を描画しています．関数 cairo_translate によって座標系を平行移動しているため，原点を中心に円のパスを作成しても，実際は (100，100) を中心に円が描画されています．
2. 縮小（図 5.16 中の円）
   次に 23–26 行目で，もう一度座標系を平行移動した後，関数 cairo_scale によって垂直方向の座標系を 1/2 に縮小して，中心 (0，0)，半径 80 の円を描画しています．垂直方向のスケールを 1/2 に変換したため，楕円として描画されています．
3. 回転（図 5.16 右の円）
   29–34 行目で座標系を回転して，円を描画しています．ここでは，前のステップで座標系のスケールを変換しているため，一度もとのスケールに戻した後（27 行目），座標系を $\pi/6$ だけ回転しています．回転角度はラジアン[*1] で指定し，水平右方向を 0 として，時計回りに正の角度を取ります（図 5.8 を参照）．また，ここでは前回のステップで描画した楕円を回転したような画像を描画するために，もう一度垂直方向の座標系を 1/2 に縮小しています．
4. 行列による座標系の変換（図 5.16 左の円）
   座標系の変換を行うとそれまでの変換内容が行列として保持されます．そこで，これまでの変換行列を関数 cairo_get_matrix によって取得して，関数 cairo_matrix_invert によってその逆行列を計算しています．そして，計算した逆行列を引数に与え，関数 cairo_transform を呼び出しています（36–38 行目）．すなわち，これまでの逆変換を行ったことになるので，座標変換を行わない状態に戻ります．この状態で (100，100) を中心とした半径 80 の円を描画すると，最初に描画した円と重なることが確認できます．

```
void cairo_get_matrix (cairo_t *cr, cairo_matrix_t *matrix);

cairo_status_t cairo_matrix_invert (cairo_matrix_t *matrix);
```

ソース 5–14　座標系の変換：cairo_transform.c（一部を抜粋）

```
1  #include <gtk/gtk.h>
2  #include <math.h>
3
4  gboolean
5  cb_draw (GtkWidget      *widget,
6           cairo_t        *cr,
7           gpointer       user_data)
8  {
9    cairo_matrix_t mat;
10
11   cairo_set_line_width (cr, 5.0);
12
13   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
14   cairo_translate (cr, 100.0, 100.0);
15   cairo_arc (cr, 0.0, 0.0, 80.0, 0.0, 2.0 * M_PI);
16   cairo_stroke (cr);
17
18   cairo_set_source_rgb (cr, 0.0, 1.0, 0.0);
19   cairo_translate (cr, 200.0, 0.0);
20   cairo_scale (cr, 1.0, 0.5);
21   cairo_arc (cr, 0.0, 0.0, 80.0, 0.0, 2.0 * M_PI);
22   cairo_stroke (cr);
```

[*1] Radian: 角度の単位．360 度が 2π ラジアンとなる．

```
23
24    cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
25    cairo_translate (cr, 200.0, 0.0);
26    cairo_scale (cr, 1.0, 2.0);
27    cairo_rotate (cr, M_PI / 6.0);
28    cairo_scale (cr, 1.0, 0.5);
29    cairo_arc (cr, 0.0, 0.0, 80.0, 0.0, 2.0 * M_PI);
30    cairo_stroke (cr);
31
32    cairo_get_matrix (cr, &mat);
33    cairo_matrix_invert (&mat);
34    cairo_transform (cr, &mat);
35
36    cairo_set_source_rgb (cr, 1.0, 1.0, 0.0);
37    cairo_arc (cr, 100.0, 100.0, 80.0, 0.0, 2.0 * M_PI);
38    cairo_fill (cr);
39
40    return FALSE;
41 }
```

5.13　テキストの描画

cairo を用いると，テキストも簡単に描画できます．ここでは最も簡単なテキストの描画方法を紹介します．図 5.17 （ソース 5–15）の例では，フォントの設定とテキストの描画を行う関数を扱っています．テキストを描画するには，関数 cairo_show_text を使用します．

```
void cairo_show_text (cairo_t *cr, const char *utf8);
```

第 1 引数	コンテキスト
第 1 引数	表示する文字列

使い方はいたって簡単で，表示したい文字列（UTF-8 形式）を関数の第 2 引数に与えるだけです．この場合，標準で設定されているフォントで，指定した文字列が描画されます．標準で設定されているフォントは環境によって異なるかもしれませんが，基本的には sans-serif に設定されています．また，フォントを変更するには，関数 cairo_select_font_face を使用します．

```
void cairo_select_font_face (cairo_t              *cr,
                             const char           *family,
                             cairo_font_slant_t   slant,
                             cairo_font_weight_t  weight);
```

第 1 引数	コンテキスト
第 2 引数	フォントの種類
第 3 引数	フォントのスラント（表 5.5 を参照）
第 4 引数	フォントのウェイト（表 5.6 を参照）

第 2 引数には使用したいフォント名を，第 3，4 引数にはフォントのスラントとウェイトを指定します．これらは表??と表 5.6 に示した cairo_font_slant_t 列挙体と cairo_font_weight_t 列挙体で定義された値を指定します．なお，フォントによってスラント設定が反映されない場合があるようです．

表 5.5　フォントのスラント設定

値	説明
CAIRO_FONT_SLANT_NORMAL	標準
CAIRO_FONT_SLANT_ITALIC	イタリック体
CAIRO_FONT_SLANT_OBLIQUE	斜体

また，フォントの大きさを変更するには，関数 cairo_set_font_size を使用します．

```
void cairo_set_font_size (cairo_t *cr, double size);
```

表 5.6　フォントのウェイト設定

値	説明
CAIRO_FONT_WEIGHT_NORMAL	標準
CAIRO_FONT_WEIGHT_BOLD	ボールド体

ソース 5–15 テキストの描画：cairo_text.c（一部を抜粋）

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

gboolean
cb_draw (GtkWidget      *widget,
         cairo_t        *cr,
         gpointer       user_data)
{
  cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
  cairo_select_font_face (cr, "FreeSans",
                          CAIRO_FONT_SLANT_ITALIC,
                          CAIRO_FONT_WEIGHT_NORMAL);
  cairo_set_font_size (cr, 24.0);
  cairo_move_to (cr, 50.0, 50.0);
  cairo_show_text (cr, "This is a text drawing sample.");

  cairo_set_source_rgb (cr, 0.0, 1.0, 0.0);

  cairo_select_font_face (cr, "VL ゴシック",
                          CAIRO_FONT_SLANT_NORMAL,
                          CAIRO_FONT_WEIGHT_NORMAL);
  cairo_move_to (cr, 50.0, 100.0);
  cairo_show_text (cr, "これはテキスト描画のサンプルです。");

  cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
  cairo_select_font_face (cr, "さざなみ明朝",
                          CAIRO_FONT_SLANT_NORMAL,
                          CAIRO_FONT_WEIGHT_BOLD);
  cairo_move_to (cr, 50.0, 150.0);
  cairo_show_text (cr, "これはテキスト描画のサンプルです。");

  return FALSE;
}
```

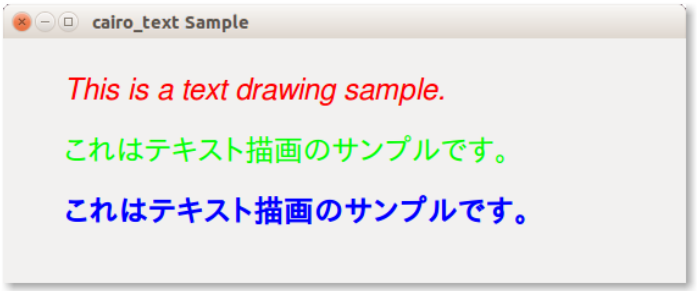


図 5.17　テキストの描画

5.14　クリッピング

クリッピングの概念図を，図 5.18 に示します．クリッピングは，ソースの範囲を限定することに相当します．図 5.18 のようにソースの内側の矩形領域のみをクリッピングすると，そのクリッピング領域からはみ出たパスは描画されません．ソース領域をクリッピングするには，クリッピングしたい領域のパスを作成して，次に関数 cairo_clip を呼び出します．

```
void cairo_clip (cairo_t *cr);
```

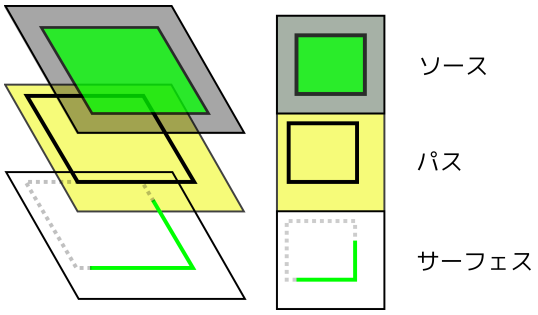


図 5.18　クリッピングの概念図

図 5.19 （ソース 5–16）にクリッピングの例を示します．この例では画像をソースに設定して，円のパスを作成してクリッピングを行い，関数 cairo_paint を呼び出します．クリッピングを行わないと画像全体が描画されるのに対し，ここではクリッピングされた円領域の内部だけが描画されています．

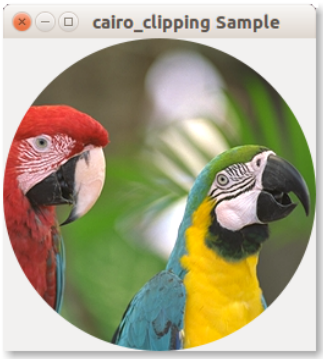


図 5.19　クリッピング

ソース 5–16 ソース領域のクリッピング：cairo_clipping.c（一部を抜粋）

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

gboolean
cb_draw (GtkWidget      *widget,
         cairo_t        *cr,
         gpointer       user_data)
{
  cairo_surface_t *surface;
  double          x, y, r;

  surface = (cairo_surface_t *) user_data;
  cairo_set_source_surface (cr, surface, 0.0, 0.0);

  x = cairo_image_surface_get_width (surface) / 2.0;
  y = cairo_image_surface_get_height (surface) / 2.0;
  r = x < y ? x : y;
  cairo_arc (cr, x, y, r, 0.0, 2.0 * M_PI);
  cairo_clip (cr);
  cairo_paint (cr);

  return FALSE;
}
```

5.15 マスク

前節で説明したクリッピングは，ソース領域の有効な領域をアルファ値 1，無効な領域をアルファ値 0 と設定していたと考えると，マスクは 0 から 1 までの中間値も扱えるものと考えることができます．クリッピングする領域はパスで指定できましたが，マスクの場合はその領域の各点についてどの程度の割合を使うかを指定する必要があります．そのため，マスクの設定はパターンもしくはサーフェスで設定されたアルファ値を利用して行います．パターンとサーフェスによってマスクを設定する関数として，次の関数が用意されています．

```
void cairo_mask (cairo_t *cr, cairo_pattern_t *pattern);

void cairo_mask_surface (cairo_t         *cr,
                         cairo_surface_t *surface,
                         double          surface_x,
                         double          surface_y);
```

第 1 引数	コンテキスト
第 2 引数	サーフェス
第 3 引数	サーフェスの左上の x 座標
第 4 引数	サーフェスの左上の y 座標

5.11.2 節（p. 87）で扱った，放射状のグラデーションパターンをマスクとして利用した例を，図 5.20 に示します．マスクの設定にはアルファ値が使用されるため，パターン作成時の色の値はマスクに影響を与えません．

ソースコードはソース 5–17 です．27 行目でマスクの設定を行っていますが，クリッピングの例とは異なり，関数 cairo_paint 等の描画関数を呼び出していません．ここからわかるように，関数 cairo_mask はマスクを設定すると同時に，描画も行います．

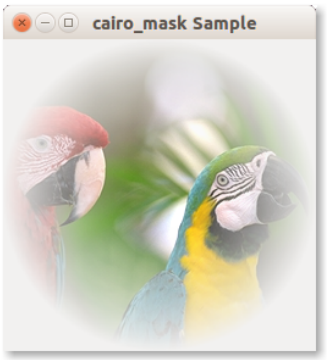


図 5.20 マスク

ソース 5–17 ソース領域のマスク：cairo_mask.c（一部を抜粋）

```
1  #include <gtk/gtk.h>
2
3  gboolean
4  cb_draw (GtkWidget     *widget,
5           cairo_t       *cr,
6           gpointer      user_data)
7  {
8    cairo_pattern_t *pattern;
9    cairo_surface_t *surface;
10   double          x, y, r;
11
12   surface = (cairo_surface_t *) user_data;
13   cairo_set_source_surface (cr, surface, 0.0, 0.0);
14
15   x = cairo_image_surface_get_width (surface) / 2.0;
16   y = cairo_image_surface_get_height (surface) / 2.0;
17   r = x < y ? x : y;
18
19   pattern = cairo_pattern_create_radial (x, y, 0.2 * r, x, y, r);
```

```
20    cairo_pattern_add_color_stop_rgba (pattern, 0.0, 1.0, 1.0, 1.0, 1.0);
21    cairo_pattern_add_color_stop_rgba (pattern, 1.0, 1.0, 1.0, 1.0, 0.0);
22
23    cairo_mask (cr, pattern);
24
25    cairo_pattern_destroy (pattern);
26
27    return FALSE;
28  }
```

5.16　合成

本節では，図形の合成について説明します．ここでいう画像の合成とは，2 つの図形を重ねて描画する際に，重なった領域だけ描画したり，お互いの非共通領域（排他的論理和；XOR）を描画したりすることです．

どのような合成が行われるかを，表 5.8 に示しました．また，それぞれの合成方法を用いた合成結果を図 5.21 に示します．ソースコードはソース 5–18 です．

図形の合成の手順は次のようになります．

1. 2 つの図形を別々に描画するために，2 つのサーフェスを作成します（描画処理を行うために同時にコンテキストも作成します）．実際には，合成した図形を描画するための本来のサーフェスが既に用意されていることが多いため，関数 cairo_surface_create_similar を用いて，新しいサーフェスを作成すればよいでしょう．このとき，関数の第 2 引数に CAIRO_CONTEXT_COLOR_ALPHA を指定しないと，正しく合成結果が描画されないので注意が必要です．

```
cairo_surface_t*
cairo_surface_create_similar (cairo_surface_t *other,
                              cairo_content_t content,
                              int              width,
                              int              height);
```

第 1 引数	設定のもとになるサーフェス
第 2 引数	コンテンツ（表 5.7 を参照）
第 3 引数	新しいサーフェスの幅
第 4 引数	新しいサーフェスの高さ
戻り値	サーフェス

表 5.7　cairo_content_t の種類

値	説明
CAIRO_CONTENT_COLOR	カラーのコンテンツ
CAIRO_CONTENT_ALPHA	アルファチャンネルのみのコンテンツ
CAIRO_CONTENT_COLOR_ALPHA	カラーとアルファチャネルを持つコンテンツ

以下の説明では，一方をソースサーフェス，もう一方を出力サーフェスと呼ぶことにます．

2. それぞれのサーフェスに図形を描画します．
3. 出力サーフェスに合成方法を設定します．合成方法の設定には，関数 cairo_set_operator を使用します．

```
void cairo_set_operator (cairo_t *cr, cairo_operator_t op);
```

4. 出力サーフェスのためのソースとして，ソースサーフェスを設定します．これには，関数 cairo_source_set_surface を使用します．
5. 関数 cairo_paint を呼び出して描画します．

表 5.8 合成オペレータ

値	説明
CAIRO_OPERATOR_CLEAR	出力サーフェスをクリアする.
CAIRO_OPERATOR_SOURCE	出力サーフェスをソースサーフェスの内容で置き換える.
CAIRO_OPERATOR_OVER	出力サーフェスの図形上にソースサーフェスの図形を重ね描きする.
CAIRO_OPERATOR_IN	出力サーフェスの図形領域内にあるソースサーフェスの図形領域のみを描画する.
CAIRO_OPERATOR_OUT	出力サーフェスの図形領域外にあるソースサーフェスの図形領域のみを描画する.
CAIRO_OPERATOR_ATOP	出力サーフェスの図形上に，出力サーフェスの図形領域内にあるソースサーフェスの図形領域を描画する.
CAIRO_OPERATOR_DEST	出力サーフェスの図形のみ描画する.
CAIRO_OPERATOR_DEST_OVER	ソースサーフェスの図形上に出力サーフェスの図形を重ね描きする.
CAIRO_OPERATOR_DEST_IN	ソースサーフェスの図形領域内にある出力サーフェスの図形領域のみを描画する.
CAIRO_OPERATOR_DEST_OUT	ソースサーフェスの図形領域外にある出力サーフェスの図形領域のみを描画する.
CAIRO_OPERATOR_DEST_ATOP	ソースサーフェスの図形の上に，ソースサーフェスの図形領域内にある出力サーフェスの図形領域を描画する.
CAIRO_OPERATOR_XOR	ソースサーフェスの図形と出力サーフェスの図形の XOR を描画する.
CAIRO_OPERATOR_ADD	重なった部分の値はソースと出力の値を加算した値で描画する.
CAIRO_OPERATOR_SATURATE	CAIRO_OPERATOR_DEST_OVER と同じ描画結果が得られる.

ソース 5–18 図形の合成：cairo_composite.c（一部を抜粋）

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

static void
draw (cairo_t          *cr,
      double           x,
      double           y,
      double           w,
      double           h,
      cairo_operator_t op,
      gchar            *operator_str) {
  cairo_t         *source_cr, *dest_cr;
  cairo_surface_t *source_surface, *dest_surface;

  source_surface =
    cairo_surface_create_similar (cairo_get_target (cr),
                                  CAIRO_CONTENT_COLOR_ALPHA, w, h);
  dest_surface =
    cairo_surface_create_similar (cairo_get_target (cr),
                                  CAIRO_CONTENT_COLOR_ALPHA, w, h);
  source_cr = cairo_create (source_surface);
  cairo_set_source_rgb (source_cr, 0.0, 0.0, 1.0);
  cairo_rectangle (source_cr, x + 10.0, y + 20.0, 50.0, 50.0);
  cairo_fill (source_cr);

  dest_cr  = cairo_create (dest_surface);
  cairo_set_source_rgb (dest_cr, 1.0, 0.0, 0.0);
  cairo_rectangle (dest_cr, x, y, 50.0, 50.0);
  cairo_fill (dest_cr);

  cairo_set_operator (dest_cr, op);
  cairo_set_source_surface (dest_cr, source_surface, 0.0, 0.0);
  cairo_paint (dest_cr);
```

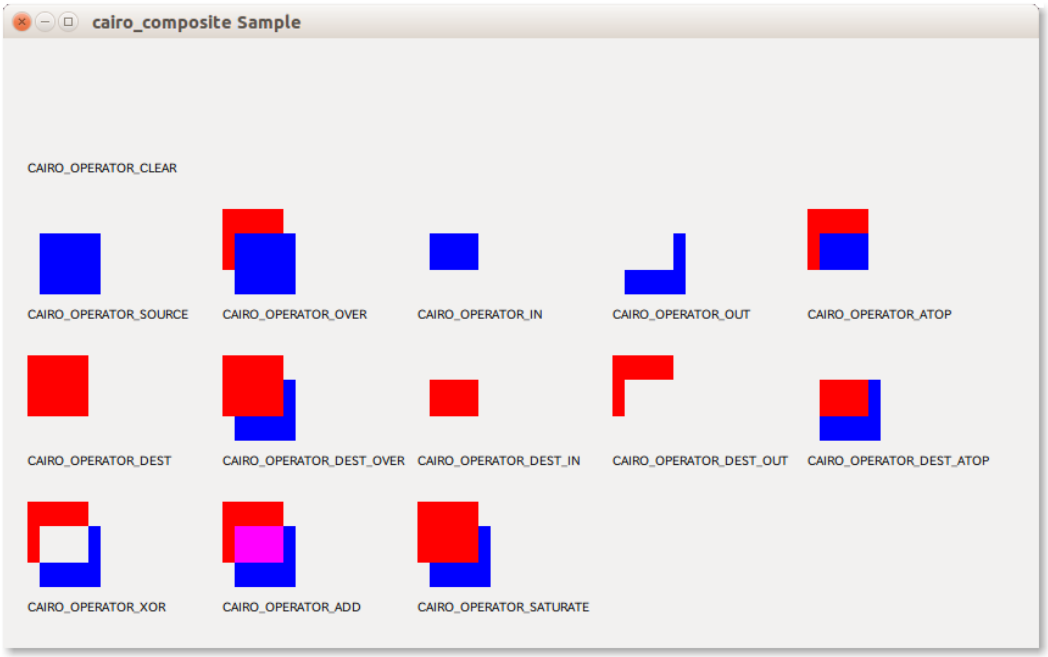


図 5.21　図形の合成

```
33
34    cairo_set_source_surface (cr, dest_surface, 0.0, 0.0);
35    cairo_paint (cr);
36
37    cairo_set_source_rgb (cr, 0.0, 0.0, 0.0);
38    cairo_move_to (cr, x, y + 90.0);
39    cairo_show_text (cr, operator_str);
40
41    cairo_surface_destroy (source_surface);
42    cairo_surface_destroy (dest_surface);
43  }
44
45  static gboolean
46  cb_draw (GtkWidget      *widget,
47           cairo_t        *cr,
48           gpointer       user_data)
49  {
50    cairo_operator_t operator[] = {CAIRO_OPERATOR_CLEAR,
51                                   CAIRO_OPERATOR_SOURCE,
52                                   CAIRO_OPERATOR_OVER,
53                                   CAIRO_OPERATOR_IN,
54                                   CAIRO_OPERATOR_OUT,
55                                   CAIRO_OPERATOR_ATOP,
56                                   CAIRO_OPERATOR_DEST,
57                                   CAIRO_OPERATOR_DEST_OVER,
58                                   CAIRO_OPERATOR_DEST_IN,
59                                   CAIRO_OPERATOR_DEST_OUT,
60                                   CAIRO_OPERATOR_DEST_ATOP,
61                                   CAIRO_OPERATOR_XOR,
62                                   CAIRO_OPERATOR_ADD,
63                                   CAIRO_OPERATOR_SATURATE};
64    gchar *operator_strs[] = {"CAIRO_OPERATOR_CLEAR",
65                              "CAIRO_OPERATOR_SOURCE",
66                              "CAIRO_OPERATOR_OVER",
67                              "CAIRO_OPERATOR_IN",
68                              "CAIRO_OPERATOR_OUT",
69                              "CAIRO_OPERATOR_ATOP",
70                              "CAIRO_OPERATOR_DEST",
71                              "CAIRO_OPERATOR_DEST_OVER",
72                              "CAIRO_OPERATOR_DEST_IN",
73                              "CAIRO_OPERATOR_DEST_OUT",
74                              "CAIRO_OPERATOR_DEST_ATOP",
75                              "CAIRO_OPERATOR_XOR",
76                              "CAIRO_OPERATOR_ADD",
```

```
77                                "CAIRO_OPERATOR_SATURATE"};
78   gint x,y, w, h, n;
79
80   w = gtk_widget_set_allocated_width (widget);
81   h = gtk_widget_set_allocated_height (widget);
82   x = 20;
83   y = 20;
84   draw (cr, x, y, w, h, operator[0], operator_strs[0]);
85
86   for (x = 20, y = 140, n = 1; n <= 5; x += 160, n++)
87     {
88       draw (cr, x, y, w, h, operator[n], operator_strs[n]);
89     }
90   for (x = 20, y = 260, n = 6; n <= 10; x += 160, n++)
91     {
92       draw (cr, x, y, w, h, operator[n], operator_strs[n]);
93     }
94   for (x = 20, y = 380, n = 11; n < 14; x += 160, n++)
95     {
96       draw (cr, x, y, w, h, operator[n], operator_strs[n]);
97     }
98   return FALSE;
99 }
```

6 GdkPixbuf

GdkPixbuf は，画像ファイルの読み書きなど，画像データの扱いを担当するライブラリです．以前は独立したパッケージとしてメンテナンスされていましたが，現在では GTK+ のパッケージに取り込まれ，GTK+ との親和性が高くなっています．この章では GdkPixbuf が提供する機能のうち，以下の項目について解説します．

- ファイルの読み書き
- 画像情報の取得
- 画像の描画

そして，最後に OpenCV というコンピュータビジョンライブラリで使用されている xml 形式の画像フォーマットを読み込むためのオリジナル画像ローダを作成する方法について解説します．

6.1 画像ファイルの読み書き

6.1.1 画像の読み込み

GdkPixbuf では，関数 gdk_pixbuf_new_from_file を使うことで，画像ファイルから画像データを読み込むことができます．GdkPixbuf で扱える画像フォーマットを表 6.1 に示します．

```
GdkPixbuf* gdk_pixbuf_new_from_file (const char *filename,
                                     GError     **error);
```

GError は次のように定義されています．関数の引数に与える場合には，値を NULL に初期化してから与える必要があります．

```
typedef struct {
  GQuark       domain;
  gint         code;
  gchar        *message;
} GError;
```

ファイルの読み込み等でエラーが発生した場合は，関数の戻り値は NULL となり，GError 構造体の変数 error にはエラー情報が格納されます．以下は読み込み許可のないファイルと存在しないファイルを引数に与えた場合のエラーコードとエラーメッセージの例です．

```
$ ./read_image cannotread.png
error->code : 2
error->message :
 ファイル 'cannotread.png' のオープンに失敗しました: 許可がありません
```

表 6.1　GdkPixbuf で扱うことができる画像フォーマット

画像フォーマット指定	読み込み	書き込み
bmp	○	○
ico	○	○
jpeg	○	○
png	○	○
ani	○	×
gif	○	×
icns	○	×
pnm	○	×
qtif	○	×
ras	○	×
svg	○	×
tga	○	×
tiff	○	×
wmf	○	×
xbm	○	×
xpm	○	×

```
$ ./read_image nothing.png
error->code : 4
error->message :
 ファイル 'nothing.png' のオープンに失敗しました: そのようなファイルやディレクトリはありません
```

GdkPixbuf 構造体を生成するには，そのほかに次のような関数があります．

- gdk_pixbuf_new
  画像サイズ等を指定して GdkPixbuf 構造体を生成します．

```
GdkPixbuf* gdk_pixbuf_new (GdkColorspace colorspace,
                           gboolean      has_alpha,
                           int           bits_per_sample,
                           int           width,
                           int           height);
```

第 1 引数	GdkColorspace
第 2 引数	アルファチャンネルの有無
第 3 引数	1 点のビット数
第 4 引数	画像の幅
第 5 引数	画像の高さ
戻り値	GdkPixbuf データ

GdkColorspace には GDK_COLORSPACE_RGB を与えます．また，has_alpha は透過領域を持つ画像を作成する場合は TRUE を，そうでない場合は FALSE を与えます．画像は一般に赤，緑，青の 3 つの色情報で表現されます．has_alpha を TRUE とした場合には，3 つの色情報に透過度を表す情報も加わります．メモリ上ではこれらの情報は，画像の点ごとに並んで格納されますが，イメージとしては，それぞれの情報を持つ平面が重なって画像を表現すると考えたほうがわかりやすいかもしれません．この平面を**チャネル**もしくは**プレーン**と呼びます．また，画像の各点のチャネルごとの情報量を表す値 bits_per_sample は現在では 8（単位はビット）のみサポートしています．

- gdk_pixbuf_new_from_data
  画像データから GdkPixbuf 構造体を生成します．画像データは guchar 型の 1 次元配列として与えます．

```
GdkPixbuf*
gdk_pixbuf_new_from_data (const guchar           *data,
                          GdkColorspace          colorspace,
                          gboolean               has_alpha,
                          int                    bits_per_sample,
                          int                    width,
                          int                    height,
                          int                    rowstride,
                          GdkPixbufDestroyNotify destroy_fn,
                          gpointer               destroy_fn_data);
```

GdkPixbufDestroyNotify は次のように定義されており，画像データを解放する関数を与えます．destroy_fn_data には GdkPixbufDestroyNotify 関数の第 2 引数に与えるユーザデータを指定します．

```
void (*GdkPixbufDestroyNotify) (guchar *pixels, gpointer data);
```

また，第 7 引数の rowstride は，画像の 1 行分のバイト数を表します．つまり，幅 W のカラー画像の場合，$3W$ となります（アルファチャネルがない場合）．しかし，幅が 4 の倍数ではない画像を読み込んだ場合は異なる値となります．これは処理の効率化のために，1 行分のデータ数が 4 バイトの倍数になるように，実際には使用しない余計な領域を追加していることが原因です．具体的には次のように計算できます．

$$rowstride = 3W + (4 - 3W\%4)$$

ここで $a\%b$ は a を b で割った余りを表します．

- gdk_pixbuf_new_from_xpm_data
  XPM データから GdkPixbuf 構造体を生成します．

```
GdkPixbuf* gdk_pixbuf_new_from_xpm_data (const char **data);
```

- gdk_pixbuf_new_from_inline
  インラインデータから GdkPixbuf 構造体を生成します．

```
GdkPixbuf* gdk_pixbuf_new_from_inline (gint         data_length,
                                       const guint8 *data,
                                       gboolean     copy_pixels,
                                       GError       **error);
```

data_length にはデータの長さを与えますが，−1 を指定するとすべてのデータを使用します．また，copy_pixels に TRUE を指定するとデータをローカルにコピーします．
インラインデータは gdk-pixbuf-csource というコマンドラインツールを使って作成できます．

```
$ gdk-pixbuf-csource --name="icon_pixbuf" /usr/share/pixmap/gnome-terminal.png
> icon.h ↵
```

インラインデータは次のような形式をしています．変数名は gdk-pixbuf-csource コマンドの –name オプションで指定できます．

```
/* GdkPixbuf RGBA C-Source image dump 1-byte-run-length-encoded */

#ifdef __SUNPRO_C
#pragma align 4 (icon_pixbuf)
#endif
#ifdef __GNUC__
static const guint8 icon_pixbuf[] __attribute__ ((__aligned__ (4))) =
#else
static const guint8 icon_pixbuf[] =
#endif
{ ""
  /* Pixbuf magic (0x47646b50) */
  "GdkP"
  /* length: header (24) + pixel_data (7104) */
  "\0\0\33\330"
  /* pixdata_type (0x2010002) */
  "\2\1\0\2"
  /* rowstride (192) */
  "\0\0\0\300"
```

```
20  /* width (48) */
21  "\0\0\0""0"
22  /* height (48) */
23  "\0\0\0""0"
24  /* pixel_data: */
25  "\377\0\0\0\0\222\0\0\0\0\6\0\0\0\1\0\0\0\2\0\0\0\4\0\0\0\7\0..."
26  "\0\0\12\204\0\0\0\13\233\0\0\0\14\203\0\0\0\13\6\0\0\0\12\0\..."
27  "\0\0\7\0\0\0\4\0\0\0\2\0\0\0\1\202\0\0\0\0\4\0\0\0\2\0\0\0""..."
28  "\1\1\1\377\246\0\0\0\377\20\0\0\0K\0\0\0\14\0\0\0\7\0\0\0\2\..."
29  ...
```

6.1.2　画像の書き込み

GdkPixbuf 形式の画像データをファイルに保存するには関数 gdk_pixbuf_save を使用します．

```
gboolean gdk_pixbuf_save (GdkPixbuf *pixbuf,
                          const char *filename,
                          const char *type,
                          GError **error,
                          ...);
```

第 1 引数	GdkPixbuf データ
第 2 引数	ファイル名
第 3 引数	ファイルタイプ
第 4 引数	エラー構造体
戻り値	成功したら TRUE. それ以外は FLASE

現在 GdkPixbuf で書き込みに対応している画像フォーマットは，JPEG，PNG，ICO, BMP, TIFF のみです（表 6.1 を参照）．引数 type にはそれぞれの画像フォーマットに応じて，”jpeg”，”png”，”ico”, ”bmp”を指定します．GdkPixbuf 形式のデータを JPEG フォーマットで保存する一番簡単な例は次のようになります．

```
gdk_pixbuf_save (pixbuf, "sample.jpg", "jpeg", NULL, NULL);
```

第 5 引数からは，保存する画像フォーマットに応じたパラメータを，キーとその値を組にして与えます．関数の最後は NULL で終わる必要があります．パラメータには表 6.2 に挙げるようなものがあります．

表 6.2　GdkPixbuf で画像を保存する際のパラメーター

画像フォーマット指定	キー	キーの値
jpeg	quality	0–100
png	compression	0–9
ico	depth	16, 24, 32

6.2　画像情報の取得

GdkPixbuf 構造体からは，画像の大きさや，プレーン数，画素値などのさまざまな情報を取得できます．これらの情報は，以下の関数を使って取得可能です．

- gdk_pixbuf_get_width
  画像の幅を返します．

```
int gdk_pixbuf_get_width (const GdkPixbuf *pixbuf);
```

- gdk_pixbuf_get_height
  画像の高さを返します．

```
int gdk_pixbuf_get_height (const GdkPixbuf *pixbuf);
```

- gdk_pixbuf_get_n_channels
  画像のチャネル（プレーン）数を返します．RGB 画像であれば，チャンネル数は 3，アルファチャンネルを持つ場合には 4 となります．

```
int gdk_pixbuf_get_n_channels (const GdkPixbuf *pixbuf);
```

- gdk_pixbuf_get_has_alpha
  画像がアルファチャネルを持つかどうかを調べます．アルファチャネルを持つ場合は TRUE を，そうでない場合は FALSE を返します．

```
gboolean gdk_pixbuf_get_has_alpha (const GdkPixbuf *pixbuf);
```

- gdk_pixbuf_get_rowstride
  画像の 1 行分のバイト数を返します．

```
int gdk_pixbuf_get_rowstride (const GdkPixbuf *pixbuf);
```

- gdk_pixbuf_get_pixels
  画素データの先頭ポインタを返します．

```
guchar* gdk_pixbuf_get_pixels (const GdkPixbuf *pixbuf);
```

6.3 画像の表示

これまで紹介した関数を使用すれば，画像データを簡単にメモリ上に読み込んで扱うことができます．では，画像データをメモリ上で扱うだけではなく，ウィンドウ上に表示するにはどうすればいいでしょうか．この節では，画像をウィンドウ上に表示する方法をいくつか紹介します．

6.3.1 GtkImage ウィジェットによる画像の表示

チュートリアルの 2.4 節（p. 15）でも紹介したように，画像を表示する一番簡単な方法は**イメージウィジェット**（GtkImage）を使用する方法です．GtkImage ウィジェットは名前の通り画像を扱うウィジェットです．GtkImage ウィジェットを使用した画像の表示の例を，ソース 6–1 に示します．

ソース 6–1 GtkImage ウィジェットによる画像の表示：display_gtkimage.c

```
1  #include <gtk/gtk.h>
2  #include <stdlib.h>
3
4  int
5  main (int argc, char *argv[])
6  {
7    GtkWidget *window;
8    GtkWidget *image;
9
10   if (argc < 2)
11     {
```

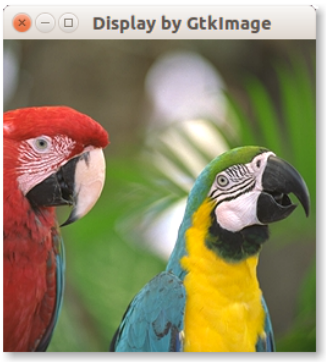


図 6.1 GtkImage ウィジェットによる画像の表示

```
12        g_print ("Usage:␣./display1␣imagefile\n");
13        exit (0);
14      }
15    gtk_init (&argc, &argv);
16
17    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
18    gtk_window_set_title (GTK_WINDOW (window), "Display␣Image␣1");
19    g_signal_connect (G_OBJECT (window), "destroy",
20                      G_CALLBACK (gtk_main_quit), NULL);
21    image = gtk_image_new_from_file (argv[1]);
22    gtk_container_add (GTK_CONTAINER (window), image);
23
24    gtk_widget_show_all (window);
25    gtk_main ();
26
27    return 0;
28  }
```

ソース 6–1 を見てもわかるように，GtkImag ウィジェットを GtkWindow ウィジェットにパックするだけで，画像を表示できます．この例では GdkPixbuf を使用せずに，関数 gtk_image_new_from_file を使用することで，画像ファイルから直接 GtkImage ウィジェットを作成できます．

次の関数を使用することで，GdkPixbuf から GtkImage ウィジェットを作成することも可能です．

```
GtkWidget* gtk_image_new_from_pixbuf (GdkPixbuf *pixbuf);
```

逆に次の関数を使用することで，GtkImage ウィジェットから GdkPixbuf 形式のデータを取得することもできます．

```
GdkPixbuf* gtk_image_get_pixbuf (GtkImage *image);
```

6.3.2　GtkDrawingArea ウィジェットによる画像の表示

5 章でも紹介したように, 画像の表示には GtkDrawingArea ウィジェットを利用することもできます．

GtkDrawingArea ウィジェットは GtkImage ウィジェットに比べて扱いが多少面倒ですが，表示した画像の上にさらに図形を描画するといったことが可能になります．ソース 5–11 では単純に読み込んだ画像を表示していただけなので, 今回は複数の画像を並べて一つの画面に表示するようにしてみます．

これを実現するにはソース 5–11 と同様に, 関数 gdk_cairo_set_source_pixbuf を使用するのですが, この関数の第 3 引数と第 4 引数で表示する画像の左上の点をウィジェット上のどの位置に配置するかを指定します．GtkDrawingArea ウィジェットを使った画像描画のソースコードをソース 6–2 に示します．

ソース 6–2　GtkDrawingArea ウィジェットによる画像の表示：display_by_gtkdrawingarea.c

```
1  #include <gtk/gtk.h>
2  #include <stdlib.h>
3
4  static gboolean
5  cb_draw (GtkWidget      *widget,
6           cairo_t        *cr,
7           gpointer       user_data)
8  {
9    GdkPixbuf *image;
10   int width, height;
11
12   image = (GdkPixbuf *) user_data;
13   width = gdk_pixbuf_get_width (image);
14   height = gdk_pixbuf_get_height (image);
15
16   gdk_cairo_set_source_pixbuf (cr, image, 0, 0);
17   cairo_paint (cr);
18   gdk_cairo_set_source_pixbuf (cr, image, width, height);
19   cairo_paint (cr);
20
21   return FALSE;
22 }
23
24 int
25 main (int argc, char *argv[])
26 {
27   GtkWidget *window;
```

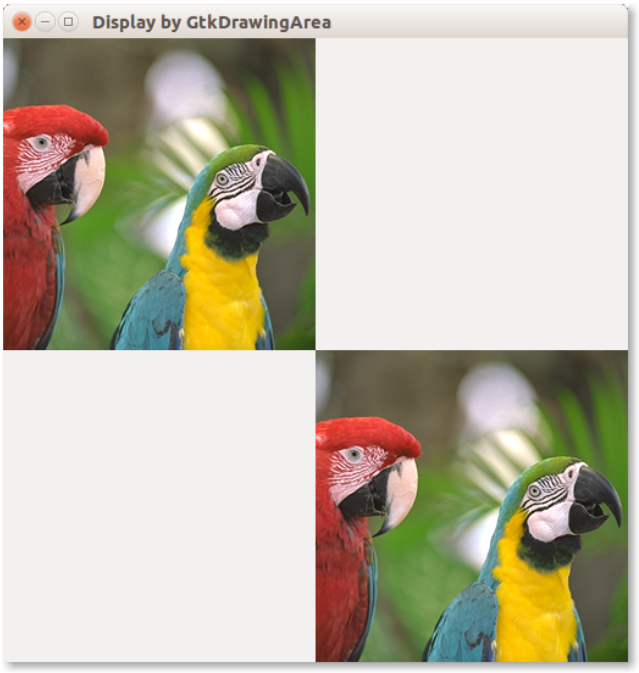


図 6.2　GtkDrawingArea ウィジェットによる画像の表示

```
28   GtkWidget *canvas;
29   GdkPixbuf *pixbuf;
30
31   if (argc < 2)
32     {
33       g_print ("Usage:␣display3␣imagefile\n");
34       exit (0);
35     }
36   gtk_init (&argc, &argv);
37
38   pixbuf = gdk_pixbuf_new_from_file (argv[1], NULL);
39
40   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
41   gtk_window_set_title (GTK_WINDOW (window), "Display␣by␣GtkDrawingArea");
42   g_signal_connect (G_OBJECT (window), "destroy",
43                     G_CALLBACK(gtk_main_quit), NULL);
44
45   canvas = gtk_drawing_area_new ();
46   gtk_widget_set_size_request (canvas,
47                                gdk_pixbuf_get_width (pixbuf) * 2,
48                                gdk_pixbuf_get_height (pixbuf) * 2);
49   g_signal_connect (G_OBJECT (canvas), "draw",
50                     G_CALLBACK (cb_draw), (gpointer) pixbuf);
51   gtk_container_add (GTK_CONTAINER (window), canvas);
52
53   gtk_widget_show_all (window);
54   gtk_main ();
55
56   g_object_unref (pixbuf);
57
58   return 0;
59 }
```

6.4　オリジナル画像ローダの作成

表 6.1 でも紹介したように GTK+ では標準で様々な形式の画像ファイルを読み書きすることが可能です．これらの画像ローダーはモジュール形式で提供されており，標準の画像ローターに加えてユーザーが独自の画像ローダーを作成し，使用できる仕組みになっています．本節では，コンピュータビジョンライブラリ OpenCV で扱う xml 形式の画像フォーマットを例に独自の画像ローダを作成する方法を説明します．

6.4.1　画像ローダの骨格

まず画像ローダの骨格となるソースコードをソース 6–3–1 に示します．最低限実装しなければならないのは，以下に挙げる画像を読み込みための関数です．

- gdk_pixbuf__cv_image_load
- gdk_pixbuf__cv_image_begin_load
- gdk_pixbuf__cv_image_stop_load
- gdk_pixbuf__cv_image_load_increment

一つ目の関数は関数 gdk_pixbuf_new_from_file から呼び出される関数です．残りの 3 つは GdkPixbufLoader を使用して画像データを読み込みながら表示するときに使用します．また，画像ローダの情報として，以下に挙げる項目が必要です．

- fill_vtable
- fill_info

一つ目は上で紹介した関数へのポインタ情報で，二つ目が画像ローダの名前や拡張子などの画像フォーマットに関する情報です．

ソース 6–3–1 画像ローダの骨格

```
#include <stdio.h>
#include <string.h>
#include <glib/gi18n.h>
#include <gmodule.h>
#include <gtk/gtk.h>
#include <gdk-pixbuf/gdk-pixbuf.h>
#include <gdk-pixbuf/gdk-pixbuf-io.h>


static GdkPixbuf* gdk_pixbuf__cv_image_load (FILE *f, GError **error);

static gpointer   gdk_pixbuf__cv_image_begin_load (GdkPixbufModuleSizeFunc size_func,
                                                   GdkPixbufModulePreparedFunc func,
                                                   GdkPixbufModuleUpdatedFunc func2,
                                                   gpointer user_data,
                                                   GError **error);
static gboolean  gdk_pixbuf__cv_image_stop_load (gpointer context, GError **error);
static gboolean  gdk_pixbuf__cv_image_load_increment (gpointer context,
                                                      const guchar *buf,
                                                      guint size,
                                                      GError **error);

static gpointer
gdk_pixbuf__cv_image_begin_load (GdkPixbufModuleSizeFunc        size_func,
                                 GdkPixbufModulePreparedFunc    prepared_func,
                                 GdkPixbufModuleUpdatedFunc     updated_func,
                                 gpointer                       user_data,
                                 GError                         **error)
{

}

static gboolean
gdk_pixbuf__cv_image_stop_load  (gpointer       data,
                                 GError         **error)
{
}

static gboolean
gdk_pixbuf__cv_image_load_increment (gpointer           data,
                                     const guchar       *buf,
                                     guint              size,
                                     GError             **error)
{
}

G_MODULE_EXPORT void
fill_vtable (GdkPixbufModule *module)
{
  module->load           = gdk_pixbuf__cv_image_load;
```

```
51   module->begin_load     = gdk_pixbuf__cv_image_begin_load;
52   module->stop_load      = gdk_pixbuf__cv_image_stop_load;
53   module->load_increment = gdk_pixbuf__cv_image_load_increment;
54 }
55
56 G_MODULE_EXPORT void
57 fill_info (GdkPixbufFormat *info)
58 {
59   static GdkPixbufModulePattern signature[] = { };
60   static gchar * mime_types[] = { };
61   static gchar * extensions[] = { };
62   info->name        = "";
63   info->signature   = signature;
64   info->description = "";
65   info->mime_types  = mime_types;
66   info->extensions  = extensions;
67   info->flags       = GDK_PIXBUF_FORMAT_THREADSAFE;
68   info->license     = "";
69 }
```

6.4.2 画像ファーマット情報

fill_vtable には実装した関数の名前をそのまま指定すればよいのでソース 6–3–1 に示した通りで問題ありません．次に fill_info に今回作成する OpenCV で扱う xml 形式の画像フォーマットの情報を入力していきます．必要な情報は以下に挙げる 7 つの項目です．

- 画像ローダー名
- 画像フォーマットを特定するヘッダーパターン (GdkPixbufModulePattern)
- 画像ローダーの説明
- mime タイプ
- 拡張子
- 画像フォーマットの読み込み，書き込み対応を表すフラグ
- ライセンス

画像ローダー名と画像ローダーの説明はローダーの製作者が自由に設定して構いません．また，mime タイプももしその画像フォーマットに対して mime タイプが未定義であれば他の画像フォーマットを例にして設定します．ここでは”x-image/opencv”としました．また，今回扱う画像フォーマットは xml 形式のファイルですので拡張子は”xml”のままにしました．そして，この画像ローダーのライセンスは他のローダーと合わせて LGPL としました．またフラグについては，今回は画像ファイルの読み込みのみ実装するため，GDK_PIXBUF_FORMAT_THREADSAFE のみ指定しました．書き込みにも対応したローダーを実装する場合には GDK_PIXBUF_FORMAT_WRITABLE も合わせて論理和の形で指定します．

最後に画像フォーマットを特定するヘッダーパターンを構造体 (GdkPixbufModulePattern) に与えます．GdkPixbufModulePattern は次のように定義されています．

```
typedef struct {
  char *prefix;
  char *mask;
  int  relevance;
} GdkPixbufModulePattern;
```

ヘッダーパターンは文字列パターン prefix とマスクパターン mask のペアによって指定します．マスクに用いる文字は’ ’, ’!’, ’x’, ’z’, ’n’ のいづれかであり，それぞれ一致，不一致，無視，0，非 0 を意味します．例えば prefix が ”abcdx” で mask が ” !x z” である場合，入力文字列”auud\0” はヘッダーパターンと一致することになります．もし prefix の文字列と完全に一致する文字パターンをヘッダーパターンとする場合には mask には NULL を指定して下さい．今回の例では，ソース 6–3–2 に示すようにヘッダーパターンを指定しました．

ソース 6–3–2 画像フォーマット情報

```
1 G_MODULE_EXPORT void
2 fill_info (GdkPixbufFormat *info)
3 {
4   static GdkPixbufModulePattern signature[] = {
5      {"<?xml␣version=\"1.0\"?>\n<opencv_storage>", NULL, 100},
6      {NULL, NULL, 0}
```

```
  };
  static gchar * mime_types[] = {
    "image/x-opencv",
    NULL
  };
  static gchar * extensions[] = {
    "xml",
    NULL
  };
  info->name        = "opencv";
  info->signature   = signature;
  info->description = "The␣OpenCV␣image␣format";
  info->mime_types  = mime_types;
  info->extensions  = extensions;
  info->flags       = GDK_PIXBUF_FORMAT_THREADSAFE;
  info->license     = "LGPL";
}
```

6.4.3　画像読み込み関数 gdk_pixbuf__cv_image_load の実装

本節では二つあった画像読み込み関数のうちの gdk_pixbuf__cv_image_load の実装方法について説明していきます．ソース 6–3–1 の関数のプロトタイプ宣言の部分を見るとわかるように，この関数は引数として FILE 構造体へのポインタを取り，戻り値として GdkPixbuf 構造体へのポインタを返す仕様になっています．従って，この関数内ですることは，FILE 構造体から画像フォーマットに従って画像のヘッダ情報とデータを読み込んで，GdkPixbuf 形式の画像データを作成することです．ここでは処理を以下に示す三つパートに分けて，それぞれの機能を実装することにします．

- FILE 構造体からのデータ読み込み
- 画像のヘッダ情報解析
- GdkPixbuf 形式の画像データ作成

FILE 構造体からのデータ読み込み

まず FILE 構造体から画像データをすべて読み込みメモリ領域に格納します．このとき，データのサイズは不明ですので，1024 バイトずつデータを読み出しながらメモリ領域にコピーしています．ソースコードはソース 6–3–3 のようになります．

ソース 6–3–3　FILE 構造体からのデータ読み込み

```
guchar buf[1024];
guchar *buffer = NULL;
int    nreads = -1;
size_t total = 0;

while (nreads != 0) {
  nreads = fread (buf, 1, 1024, f);
  buffer = g_renew (guchar, buffer, total + nreads);
  memcpy (buffer + total, buf, nreads);
  total += nreads;
}
```

画像のヘッダ情報解析

次に GdkPixbuf 形式のデータを作成するために，読み込んだ画像データの中から画像サイズ等のヘッダ情報を取得します．ヘッダ情報の解析方法はそれぞれの画像フォーマットによって異なりますので，ここではヘッダ情報の具体的な解析部分について割愛します．

GdkPixbuf 形式のデータを作成するには最低限画像の幅と高さの情報が必要になります．また通常画像の輝度情報は 8 ビットの符号なし整数で表されますが，OpenCV では浮動小数点で表される輝度情報を扱えたり，任意のチャネル数を扱えたりするため，ここではソース 6–3–4 に示すような引数を持つヘッダ解析関数を実装して画像の幅，高さ，チャネル数，画素値の型を取得しています．

ソース 6–3–4 画像ヘッダの解析関数

```
1 static gboolean
2 read_header (guchar* buffer,
3               gint    buffer_size,
4               gint*   nreads,
5               gint*   width,
6               gint*   height,
7               gint*   nplanes,
8               guchar* ptype)
```

GdkPixbuf 形式の画像データ作成

画像のヘッダ情報を取得したら, それをもとに画像の輝度データをメモリから読み出し, 関数 gdk_pixbuf_new_from_data により GdkPixbuf 形式の画像データを作成します.

6.4.4　画像読み込み関数 gdk_pixbuf__cv_image_load_increment の実装

最後にもう一つの画像読み込み関数 gdk_pixbuf__cv_image_load_increment を実装します. この関数は画像データを読み込みながら同時に画面に表示していくインクリメンタル・ロードの際に使われる関数です. インクリメンタル・ロードは次の手順で行われます.

1. 関数 gdk_pixbuf__cv_image_begin_load で画像データ読み込みの際に使う構造体を初期化する.
2. 関数 gdk_pixbuf__cv_image_load_incremental で画像データ読み込む.
3. 関数 gdk_pixbuf__cv_image_stop_load で GdkPixbuf 形式の画像データを作成する.

関数 gdk_pixbuf__cv_image_begin_load の実装

まず, インクリメンタル・ロードの初めに関数 gdk_pixbuf__cv_image_begin_load で画像データ読み込みの際に使う構造体を初期化します. この関数は関数 gdk_pixbuf_loader_load_module により画像ローダーが読み込まれる際に呼び出され, 関数 gdk_pixbuf__cv_image_load_incremental の第 1 引数として渡され参照されます.

この関数内で初期化される構造体には次に示すように三つの関数へのポインタに加えて, 画像の大きさやプレーン数など必要な情報をメンバに設定しています.

```
typedef struct {
  GdkPixbufModuleUpdatedFunc   updated_func;
  GdkPixbufModulePreparedFunc  prepared_func;
  GdkPixbufModuleSizeFunc      size_func;
  gpointer                     user_data;
  GError                       **error;
  guchar                       *buffer;
  size_t                       read_bytes;
  size_t                       header_offset;
  gboolean                     got_header;
  gint                         width;
  gint                         height;
  gint                         nplanes;
  guchar                       ptype;
} OpenCVLoaderContext;
```

ソース 6–3–5 に具体的に実装した関数 gdk_pixbuf__cv_image_begin_load を示します. 上で説明した構造体の領域確保とメンバの初期化を行い, 領域確保した構造体の先頭アドレスを関数の戻り値として返します.

ソース 6–3–5 関数 gdk_pixbuf__cv_image_begin_load の実装

```
1 static gpointer
2 gdk_pixbuf__cv_image_begin_load (GdkPixbufModuleSizeFunc        size_func,
3                                  GdkPixbufModulePreparedFunc    prepared_func,
4                                  GdkPixbufModuleUpdatedFunc     updated_func,
```

```
                                    gpointer                        user_data,
                                    GError                          **error)
{
  OpenCVLoaderContext *context;

  context = g_new (OpenCVLoaderContext, 1);
  if (!context) {
    g_set_error (error,
                 GDK_PIXBUF_ERROR,
                 GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY,
                 "Insufficient memory to load OpenCV context struct");
    return NULL;
  }
  context->buffer        = NULL;
  context->read_bytes    = 0;
  context->got_header    = FALSE;
  context->updated_func  = updated_func;
  context->prepared_func = prepared_func;
  context->size_func     = size_func;
  context->user_data     = user_data;
  context->error         = error;

  return (gpointer) context;
}
```

関数 gdk_pixbuf__cv_image_load_increment の実装

次に関数 gdk_pixbuf__cv_image_load_incremental を実装します．ここでは，逐次的に読み込まれる画像データをバッファに保存する処理（ソース 6–3–6 15 行目から 18 行目）と，画像のヘッダ情報を読み込む処理（ソース 6–3–6 20 行目から 35 行目）を行うことにします．また，ヘッダの解析が終了した時点で，GdkPixbufModuleSizeFunc 型の関数を呼び出しています．ヘッダ解析処理は画像フォーマットごとに異なるため，ここでは詳細は解説しません（詳細はソース 6–3–8 を参照してください）．

ソース 6–3–6 関数 gdk_pixbuf__cv_image_load_increment の実装

```
static gboolean
gdk_pixbuf__cv_image_load_increment (gpointer           data,
                                     const guchar       *buf,
                                     guint              size,
                                     GError             **error)
{
  OpenCVLoaderContext *context = (OpenCVLoaderContext *) data;
  gboolean ret;

  g_return_val_if_fail (context != NULL, FALSE);
  g_return_val_if_fail (buf != NULL, FALSE);

  if (size == 0) return TRUE;

  context->buffer = g_renew (guchar, context->buffer,
                             context->read_bytes + size);
  memcpy (context->buffer + context->read_bytes, buf, size);
  context->read_bytes += size;

  if (!context->got_header) {
    ret = read_header (context->buffer,
                       context->read_bytes,
                       &context->header_offset,
                       &context->width,
                       &context->height,
                       &context->nplanes,
                       &context->ptype);
    if (ret) {
      gint width  = context->width;
      gint height = context->height;
      (*context->size_func) (&width, &height, context->user_data);
      if (width == 0 || height == 0) return FALSE;
      context->got_header = TRUE;
    }
  }
  return TRUE;
}
```

関数 gdk_pixbuf__cv_image_stop_load の実装

最後に, 関数 gdk_pixbuf__cv_image_stop_load を実装します. この関数が呼び出されるまでにすべての画像データがバッファに読み込みと画像のヘッダ情報解析が終わっていますので, この関数内では, 読み込んだ画像データから GdkPixbuf 形式のデータを作成しています. そして, 画像データの生成が終了した時点で, GdkPixbufModulePreparedFunc 型の関数を呼び出しています.

ソース 6–3–7 関数 gdk_pixbuf__cv_image_stop_load の実装

```
1  static gboolean
2  gdk_pixbuf__cv_image_stop_load  (gpointer       data,
3                                   GError         **error)
4  {
5    OpenCVLoaderContext *context = (OpenCVLoaderContext *) data;
6    OpenCVImageInfo     *info;
7    GdkPixbuf           *pixbuf;
8    gboolean            ret;
9
10   g_return_val_if_fail (context != NULL, TRUE);
11
12   info = g_new (OpenCVImageInfo, 1);
13   if (!info) {
14     g_set_error (error,
15                  GDK_PIXBUF_ERROR,
16                  GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY,
17                  "Insufficient␣memory␣to␣load␣OpenCV␣image␣info␣struct");
18     return FALSE;
19   }
20   info->width    = context->width;
21   info->height   = context->height;
22   info->nplanes  = context->nplanes;
23   info->ptype    = context->ptype;
24
25   ret = read_image (context->buffer + context->header_offset,
26                     context->read_bytes - context->header_offset + 1, info);
27   if (!ret) {
28     g_set_error (error,
29                  GDK_PIXBUF_ERROR,
30                  GDK_PIXBUF_ERROR_FAILED,
31                  "Could␣not␣comletely␣read␣image␣header");
32     return FALSE;
33   }
34   pixbuf = gdk_pixbuf_new_from_data (info->data, GDK_COLORSPACE_RGB,
35                                      FALSE,
36                                      8,
37                                      info->width,
38                                      info->height,
39                                      info->width * 3,
40                                      (GdkPixbufDestroyNotify)
41                                      gdk_pixbuf_cv_data_free,
42                                      info);
43   if (!pixbuf) {
44     g_set_error (error,
45                  GDK_PIXBUF_ERROR,
46                  GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY,
47                  "Insufficient␣memory␣to␣allocate␣pixbuf␣struct");
48     return FALSE;
49   }
50   g_object_set_data (G_OBJECT(pixbuf), "image-param", (gpointer) info);
51
52   if (context->prepared_func) {
53     (*context->prepared_func) (pixbuf, NULL, context->user_data);
54   }
55   if (context) {
56     if (context->buffer) {
57       g_free (context->buffer);
58       context->buffer = NULL;
59     }
60     g_free (context);
61     context = NULL;
62   }
63   return TRUE;
64 }
```

6.4.5　オリジナル画像ローダのコンパイル

完成したソースコードをソース 6–3–8 に示します．作成したオリジナル画像ローダをコンパイルするには次のようにします．GdkPixbuf の画像ローダをコンパイルする際には-DGDK_PIXBUF_ENABLE_BACKEND オプションをつける必要がありますので注意してください．

```
$ gcc io-opencv.c -o libpixbufloader-opencv.so `pkg-config --cflags --libs gtk+-3.0` -fPIC -shared
 -DGDK_PIXBUF_ENABLE_BACKEND ↵
```

6.4.6　オリジナル画像ローダの登録

画像ローダのコンパイルに成功したら，最後にシステム上で作成した画像ローダを使用できるように以下の手順で設定を行います．

1. コンパイルした画像ローダのコピー
2. 画像ローダの登録

画像ローダのコピー

Ubuntu 16.04 では，画像ローダが/usr/lib/x86_64-linux-gnu/gdk-pixbuf-2.0/2.10.0/loaders/以下にインストールされています．以下のコマンドを使って作成した画像ローダをコピーしましょう．

```
$ sudo install -s libpixbufloader-opencv.so /usr/lib/x86_64-linux-gnu/gdk-pixbuf-2.0/2.10.0/
loaders/ ↵
```


お使いの OS で画像ローダがインストールされているディレクトリが不明な場合は，find コマンドを使用して，以下のように調べるとよいでしょう．

```
$ sudo find / -name "libpixbufloader*" -print ↵
```

画像ローダの登録

画像ローダのコピーが終わったら，gdk-pixbuf-query-loaders コマンドを実行して，作成した画像ローダの情報が表示されるかどうか確認しましょう．gdk-pixbuf-query-loaders コマンドを実行した結果の一度を以下に示します．

```
$ gdk-pixbuf-query-loaders | less↵
...
"/usr/lib/x86_64-linux-gnu/gdk-pixbuf-2.0/2.10.0/loaders/libpixbufloader-opencv.so"
"opencv" 4 "gdk-pixbuf" "The OpenCV image format" "LGPL"
"image/x-opencv" ""
"xml" ""
"<opencv_storage>" "" 100

"/usr/lib/x86_64-linux-gnu/gdk-pixbuf-2.0/2.10.0/loaders/libpixbufloader-ico.so"
"ico" 5 "gdk-pixbuf" "Windows icon" "LGPL"
...
```

上記のように作成した画像ローダの情報が含まれていれば成功です．最後に以下のコマンドを実行してこの情報を正式に登録して終了です．

```
$ sudo gdk-pixbuf-query-loaders --update-cache↵
```

最後に作成した OpenCV 画像ローダのソースコード完全版をソース 6–3–8 に示します.

ソース 6–3–8 OpenCV 画像ローダのソースコード: io-opencv.c

```
#include <stdio.h>
#include <string.h>
#include <glib/gi18n.h>
#include <gmodule.h>
#include <gtk/gtk.h>
#include <gdk-pixbuf/gdk-pixbuf.h>
#include <gdk-pixbuf/gdk-pixbuf-io.h>

typedef struct _OpenCVImageInfo
{
  gint    width;
  gint    height;
  gint    nplanes;
  guchar  ptype;
  guchar* data;
} OpenCVImageInfo;

typedef struct _OpenCVLoaderContext
{
  GdkPixbufModuleUpdatedFunc    updated_func;
  GdkPixbufModulePreparedFunc   prepared_func;
  GdkPixbufModuleSizeFunc       size_func;
  gpointer                      user_data;
  GError                        **error;
  guchar                        *buffer;
  size_t                        read_bytes;
  size_t                        header_offset;
  gboolean                      got_header;
  gint                          width;
  gint                          height;
  gint                          nplanes;
  guchar                        ptype;
} OpenCVLoaderContext;

static GdkPixbuf* gdk_pixbuf__cv_image_load (FILE *f, GError **error);

static gpointer   gdk_pixbuf__cv_image_begin_load (GdkPixbufModuleSizeFunc size_func,
                                                   GdkPixbufModulePreparedFunc func,
                                                   GdkPixbufModuleUpdatedFunc func2,
                                                   gpointer user_data,
                                                   GError **error);
static gboolean  gdk_pixbuf__cv_image_stop_load (gpointer context, GError **error);
static gboolean  gdk_pixbuf__cv_image_load_increment (gpointer context,
                                                      const guchar *buf,
                                                      guint size,
                                                      GError **error);

/**
 * <>で囲まれた文字列を読み込む
 *
 * @param [in] buf 文字列へのポインタ
 * @param [in] buf_size バッファサイズ
 * @param [out] nreads 読み込んだデータ数
 *
 * @return 読み込んだ文字列
 */
static gchar*
read_line (guchar* buf,
           int     buf_size,
           int*    nreads)
{
  int    n;
  int    offset = 0;
  gchar* line;
```

```
65    gchar* ptr;
66
67    *nreads = 0;
68
69    while (buf[*nreads] != '<' && *nreads != buf_size - 1) {
70      offset++;
71      (*nreads)++;
72    }
73    if (*nreads == buf_size - 1) {
74      *nreads = 0;
75      return NULL;
76    }
77    while (buf[*nreads] != '>' && *nreads != buf_size - 1) {
78      (*nreads)++;
79    }
80    if (*nreads == buf_size - 1) {
81      *nreads = 0;
82      return NULL;
83    }
84    line = g_new (gchar, (*nreads + 2 - offset));
85    ptr  = line;
86
87    for (n = 0; n < (*nreads + 1 - offset); n++) {
88      *ptr++ = buf[offset + n];
89    }
90    *ptr = '\0';
91
92    (*nreads)++;
93
94    return line;
95  }
96
97  /**
98   * 文字列から数値を読み込む
99   *
100  * @param [in] buf 文字列へのポインタ
101  * @param [in] buf_size バッファサイズ
102  * @param [out] nreads 読み込んだデータ数
103  *
104  * @return 読み込んだ整数値
105  */
106 static gint
107 read_value (guchar* buf,
108             int     buf_size,
109             int*    nreads)
110 {
111   int    n;
112   gchar* line;
113   gchar* ptr;
114   gint   value;
115
116   *nreads = 0;
117
118   while (buf[*nreads] != '<' && *nreads != buf_size - 1) (*nreads)++;
119   if (*nreads == buf_size - 1) {
120     *nreads = 0;
121     return -1;
122   }
123   line = g_new (gchar, *nreads);
124   ptr  = line;
125
126   for (n = 0; n < *nreads; n++) *ptr++ = buf[n];
127   *ptr = '\0';
128
129   value = atoi (line);
130   g_free (line);
131
132   return value;
133 }
134
135 /**
136  * <>で囲まれた文字列を読み込む
137  *
138  * @param [in] buf 文字列へのポインタ
139  * @param [in] buf_size バッファサイズ
140  * @param [out] nreads 読み込んだデータ数
```

```
141  *
142  * @return 読み込んだ文字列
143  */
144 static gchar*
145 read_string(guchar* buf,
146             int     buf_size,
147             int*    nreads)
148 {
149   int    n;
150   gchar* line;
151   gchar* ptr;
152
153   *nreads = 0;
154
155   while (buf[*nreads] != '<' && *nreads != buf_size - 1) (*nreads)++;
156   if (*nreads == buf_size - 1) {
157     *nreads = 0;
158     return NULL;
159   }
160   line = g_new (gchar, *nreads);
161   ptr  = line;
162
163   for (n = 0; n < *nreads; n++) *ptr++ = buf[n];
164   *ptr = '\0';
165
166   return line;
167 }
168
169 /**
170  * 画素値の型とプレーン数を解析する
171  *
172  * @param [in] buf 文字列へのポインタ
173  * @param [out] ptype 画素値の型を表す文字列
174  * @param [out] nplanes プレーン数
175  */
176 static void
177 parse_image_type (guchar*    buf,
178                   guchar*    ptype,
179                   gint*      nplanes)
180 {
181   gint    value;
182   guchar* work;
183   int     n;
184
185   *ptype = buf[strlen(buf) - 2];
186
187   work = g_new (guchar, strlen (buf) - 2);
188   buf++;
189   for (n = 0; n < strlen (buf) - 2; n++) *(work + n) = *(buf + n);
190   *(work + n) = '\0';
191
192   *nplanes = atoi (work);
193
194   g_free (work);
195 }
196
197 /**
198  * 画像データを読み込んで表示用データに変換する
199  *
200  * @param [in] buf 文字列へのポインタ
201  * @param [in] buf_size バッファサイズ
202  * @param [out] info 画像情報を格納する構造体へのポインタ
203  */
204 static void
205 set_data (guchar*          buf,
206           gint             buf_size,
207           OpenCVImageInfo* info)
208 {
209   int     x, y, p, n;
210   guchar* buf_ptr;
211   guchar* dptr;
212   guchar* sptr;
213   guchar  string[128];
214   gint    dsize;
215   gint    empty_planes;
216
```

```
217   dptr  = info->data;
218   dsize = info->width * info->height * 3;
219   empty_planes = 3 - info->nplanes;
220
221   buf_ptr = buf;
222
223   switch (info->ptype) {
224   case 'u':
225     {
226       guchar copy;
227
228       for (y = 0; y < info->height; y++) {
229         for (x = 0; x < info->width; x++) {
230           for (p = 0; p < info->nplanes; p++) {
231             while (*buf_ptr == '␣' || *buf_ptr == '\t' ||
232                    *buf_ptr == '\n') buf_ptr++;
233             sptr = string;
234             while (*buf_ptr != '␣' && *buf_ptr != '\t' && *buf_ptr != '<') {
235               *sptr++ = *buf_ptr++;
236             }
237             *sptr = '\0';
238             if (p < 3) *dptr++ = (guchar) atoi (string);
239           }
240           if (info->nplanes < 3) {
241             copy = *(dptr - 1);
242             for (p = 0; p < empty_planes; p++) *dptr++ = copy;
243           }
244         }
245       }
246     }
247     break;
248   case 'c':
249     {
250       guchar copy;
251
252       for (y = 0; y < info->height; y++) {
253         for (x = 0; x < info->width; x++) {
254           for (p = 0; p < info->nplanes; p++) {
255             while (*buf_ptr == '␣' || *buf_ptr == '\t' ||
256                    *buf_ptr == '\n') buf_ptr++;
257             sptr = string;
258             while (*buf_ptr != '␣' && *buf_ptr != '\t' && *buf_ptr != '<') {
259               *sptr++ = *buf_ptr++;
260             }
261             *sptr = '\0';
262             if (p < 3) *dptr++ = (guchar) (128 + atoi (string));
263           }
264           if (info->nplanes < 3) {
265             copy = *(dptr - 1);
266             for (p = 0; p < empty_planes; p++) *dptr++ = copy;
267           }
268         }
269       }
270     }
271     break;
272   case 'w':
273     {
274       gushort* array;
275       gushort  copy;
276       gint     min = G_MAXUSHORT, max = 0;
277       gint     val;
278       gdouble  coef;
279
280       array = g_new0 (gushort, dsize);
281       n = 0;
282
283       for (y = 0; y < info->height; y++) {
284         for (x = 0; x < info->width; x++) {
285           for (p = 0; p < info->nplanes; p++) {
286             while (*buf_ptr == '␣' || *buf_ptr == '\t' ||
287                    *buf_ptr == '\n') buf_ptr++;
288             sptr = string;
289             while (*buf_ptr != '␣' && *buf_ptr != '\t' && *buf_ptr != '<') {
290               *sptr++ = *buf_ptr++;
291             }
292             *sptr = '\0';
```

```
293             if (p < 3) {
294               val = atoi (string);
295               if (val < min) min = val;
296               if (val > max) max = val;
297               array[n++] = (gushort) val;
298             }
299           }
300           if (info->nplanes < 3) {
301             copy = array[n-1];
302             for (p = 0; p < empty_planes; p++) array[n++] = copy;
303           }
304         }
305       }
306       if (min != max) {
307         coef = 255.0 / (max - min);
308       } else {
309         coef = 255.0 / max;
310       }
311       for (n = 0; n < dsize; n++) {
312         *dptr++ = (guchar) ((array[n] - min) * coef);
313       }
314       g_free (array);
315     }
316     break;
317   case 's':
318     {
319       gshort* array;
320       gshort  copy;
321       gint    min = G_MAXSHORT, max = G_MINSHORT;
322       gint    val;
323       gdouble coef;
324
325       array = g_new0 (gshort, dsize);
326       n = 0;
327
328       for (y = 0; y < info->height; y++) {
329         for (x = 0; x < info->width; x++) {
330           for (p = 0; p < info->nplanes; p++) {
331             while (*buf_ptr == '␣' || *buf_ptr == '\t' ||
332                    *buf_ptr == '\n') buf_ptr++;
333             sptr = string;
334             while (*buf_ptr != '␣' && *buf_ptr != '\t' && *buf_ptr != '<') {
335               *sptr++ = *buf_ptr++;
336             }
337             *sptr = '\0';
338             if (p < 3) {
339               val = atoi (string);
340               if (val < min) min = val;
341               if (val > max) max = val;
342               array[n++] = (gshort) val;
343             }
344           }
345           if (info->nplanes < 3) {
346             copy = array[n-1];
347             for (p = 0; p < empty_planes; p++) array[n++] = copy;
348           }
349         }
350       }
351       if (min != max) {
352         coef = 255.0 / (max - min);
353       } else {
354         coef = 255.0 / max;
355       }
356       for (n = 0; n < dsize; n++) {
357         *dptr++ = (guchar) ((array[n] - min) * coef);
358       }
359       g_free (array);
360     }
361     break;
362   case 'i':
363     {
364       gint*   array;
365       gint    copy;
366       gint    min = G_MAXINT32, max = G_MININT32;
367       gint    val;
368       gdouble coef;
```

```
369
370       array = g_new0 (gint, dsize);
371       n = 0;
372
373       for (y = 0; y < info->height; y++) {
374         for (x = 0; x < info->width; x++) {
375           for (p = 0; p < info->nplanes; p++) {
376             while (*buf_ptr == ' ' || *buf_ptr == '\t' ||
377                    *buf_ptr == '\n') buf_ptr++;
378             sptr = string;
379             while (*buf_ptr != ' ' && *buf_ptr != '\t' && *buf_ptr != '<') {
380               *sptr++ = *buf_ptr++;
381             }
382             *sptr = '\0';
383             if (p < 3) {
384               val = atoi (string);
385               if (val < min) min = val;
386               if (val > max) max = val;
387               array[n++] = val;
388             }
389           }
390           if (info->nplanes < 3) {
391             copy = array[n-1];
392             for (p = 0; p < empty_planes; p++) array[n++] = copy;
393           }
394         }
395       }
396       if (min != max) {
397         coef = 255.0 / (max - min);
398       } else {
399         coef = 255.0 / max;
400       }
401       for (n = 0; n < dsize; n++) {
402         *dptr++ = (guchar) ((array[n] - min) * coef);
403       }
404       g_free (array);
405     }
406     break;
407   case 'f':
408     {
409       gfloat* array;
410       gfloat  copy;
411       gfloat  min = G_MAXFLOAT, max = G_MINFLOAT;
412       gfloat  val;
413       gdouble coef;
414
415       array = g_new0 (gfloat, dsize);
416       n = 0;
417
418       for (y = 0; y < info->height; y++) {
419         for (x = 0; x < info->width; x++) {
420           for (p = 0; p < info->nplanes; p++) {
421             while (*buf_ptr == ' ' || *buf_ptr == '\t' ||
422                    *buf_ptr == '\n') buf_ptr++;
423             sptr = string;
424             while (*buf_ptr != ' ' && *buf_ptr != '\t' && *buf_ptr != '<') {
425               *sptr++ = *buf_ptr++;
426             }
427             *sptr = '\0';
428             if (p < 3) {
429               val = atoi (string);
430               if (val < min) min = val;
431               if (val > max) max = val;
432               array[n++] = val;
433             }
434           }
435           if (info->nplanes < 3) {
436             copy = array[n-1];
437             for (p = 0; p < empty_planes; p++) array[n++] = copy;
438           }
439         }
440       }
441       if (min != max) {
442         coef = 255.0 / (max - min);
443       } else {
444         coef = 255.0 / max;
```

```
445       }
446       for (n = 0; n < dsize; n++) {
447         *dptr++ = (guchar) ((array[n] - min) * coef);
448       }
449       g_free (array);
450     }
451     break;
452   case 'd':
453     {
454       gdouble* array;
455       gdouble  copy;
456       gdouble  min = G_MAXDOUBLE, max = G_MINDOUBLE;
457       gdouble  val;
458       gdouble  coef;
459
460       array = g_new0 (gdouble, dsize);
461       n = 0;
462
463       for (y = 0; y < info->height; y++) {
464         for (x = 0; x < info->width; x++) {
465           for (p = 0; p < info->nplanes; p++) {
466             while (*buf_ptr == '␣' || *buf_ptr == '\t' ||
467                    *buf_ptr == '\n') buf_ptr++;
468             sptr = string;
469             while (*buf_ptr != '␣' && *buf_ptr != '\t' && *buf_ptr != '<') {
470               *sptr++ = *buf_ptr++;
471             }
472             *sptr = '\0';
473             if (p < 3) {
474               val = atoi (string);
475               if (val < min) min = val;
476               if (val > max) max = val;
477               array[n++] = val;
478             }
479           }
480           if (info->nplanes < 3) {
481             copy = array[n-1];
482             for (p = 0; p < empty_planes; p++) array[n++] = copy;
483           }
484         }
485       }
486       if (min != max) {
487         coef = 255.0 / (max - min);
488       } else {
489         coef = 255.0 / max;
490       }
491       for (n = 0; n < dsize; n++) {
492         *dptr++ = (guchar) ((array[n] - min) * coef);
493       }
494       g_free (array);
495     }
496     break;
497   default:
498     break;
499   }
500 }
501
502 /**
503  * 画像情報(大きさやデータ型など)を読み込む
504  *
505  * @param [in] 文字列へのポインタ
506  * @param [in] buf_size バッファサイズ
507  * @param [out] nreads 読み込んだデータ数
508  * @param [out] width 画像の幅
509  * @param [out] height 画像の高さ
510  * @param [out] nplanes プレーン数
511  * @param [out] ptype 画素値の型
512  *
513  * @return 正常に処理が終了すればTRUE、そうでなければFALSE
514  */
515 static gboolean
516 read_header (guchar* buffer,
517              gint    buffer_size,
518              gint*   nreads,
519              gint*   width,
```

```
520             gint*   height,
521             gint*   nplanes,
522             guchar* ptype)
523 {
524   guchar* line;
525   gboolean ret = TRUE;
526   gint    offset = 0;
527
528   /* xmlヘッダのチェック */
529   line = read_line (buffer, buffer_size, nreads);
530   if (!line || (line && strcmp (line, "<?xml␣version=\"1.0\"?>") != 0)) {
531     ret = FALSE;
532   }
533   if (line) g_free (line);
534   if (!ret) return FALSE;
535
536   offset += *nreads;
537   buffer_size -= *nreads;
538
539   line = read_line (buffer + offset, buffer_size, nreads);
540   if (!line || (line && strcmp (line, "<opencv_storage>") != 0)) {
541     ret = FALSE;
542   }
543   if (line) g_free (line);
544   if (!ret) return FALSE;
545
546   offset += *nreads;
547   buffer_size -= *nreads;
548
549   /* 画像サイズの取得 */
550   *width  = -1;
551   *height = -1;
552   while (1) {
553     line = read_line (buffer + offset, buffer_size, nreads);
554     if (line) {
555       offset += *nreads;
556       buffer_size -= *nreads;
557
558       if (strcmp (line, "<width>") == 0 ||
559           strcmp (line, "<cols>") == 0) {
560         *width = read_value (buffer + offset, buffer_size, nreads);
561         g_free (line);
562         if (*width != -1 && *height != -1) break;
563       } else if (strcmp (line, "<height>") == 0 ||
564                  strcmp (line, "<rows>") == 0) {
565         *height = read_value (buffer + offset, buffer_size, nreads);
566         g_free (line);
567         if (*width != -1 && *height != -1) break;
568       }
569     } else {
570       g_free (line);
571       return FALSE;
572     }
573   }
574   /* 画素値の型の取得 */
575   while (1) {
576     offset += *nreads;
577     buffer_size -= *nreads;
578     line = read_line (buffer + offset, buffer_size, nreads);
579     if (line) {
580       if (strcmp (line, "<dt>") != 0) {
581         g_free (line);
582       } else {
583         g_free (line);
584         break;
585       }
586     } else {
587       return FALSE;
588     }
589   }
590   offset += *nreads;
591   buffer_size -= *nreads;
592   line = read_string (buffer + offset, buffer_size, nreads);
593   if (line) {
594     parse_image_type (line, ptype, nplanes);
595     *nreads = offset + *nreads;
```

```
596     g_free (line);
597     return TRUE;
598   } else {
599     return FALSE;
600   }
601 }
602
603 /**
604  * 文字列から画像情報を取得する
605  *
606  * @param [in] buffer 文字列へのポインタ
607  * @param [in] buffer_size バッファサイズ
608  * @param [out] info 画像情報を格納した構造体へのポインタ
609  *
610  * @return 正常に処理が終了すればTRUE、そうでなければFALSE
611  */
612 static gboolean
613 read_image (guchar*          buffer,
614             gint             buffer_size,
615             OpenCVImageInfo* info)
616 {
617   guchar*  line;
618   gint     nreads;
619   gint     offset = 0;
620   gboolean ret = TRUE;
621
622   while (1) {
623     line = read_line (buffer + offset, buffer_size, &nreads);
624     if (!line) {
625       ret = FALSE;
626       break;
627     } else {
628       offset += nreads;
629       buffer_size -= nreads;
630       if (strcmp (line, "<data>") == 0) {
631         break;
632       } else {
633         g_free (line);
634       }
635     }
636   }
637   if (line) g_free (line);
638   if (!ret) return FALSE;
639
640   info->data = g_new (guchar, info->width * info->height * 3);
641   set_data (buffer + offset, buffer_size, info);
642
643   return TRUE;
644 }
645
646 /**
647  * 画像情報を格納した構造体のメモリ領域を解放する
648  *
649  * @param [in] 画像データへのポインタ
650  * @param [in] data 画像情報を格納した構造体へのポインタ
651  */
652 void gdk_pixbuf_cv_data_free (guchar    *pixels,
653                               gpointer  data)
654 {
655   OpenCVImageInfo *info;
656
657   g_free (pixels);
658   if (data) {
659     info = (OpenCVImageInfo *) data;
660     if (info && info->data) {
661       g_free (info->data);
662       info->data = NULL;
663     }
664     g_free (info);
665   }
666 }
667
668 static GdkPixbuf*
669 gdk_pixbuf__cv_image_load (FILE   *f,
670                            GError **error)
671 {
```

```
672   OpenCVImageInfo     *info;
673   GdkPixbuf           *pixbuf;
674   guchar              *data;
675   guchar              buf[1024];
676   guchar              *buffer = NULL;
677   guchar              *ptr;
678   int                 width, height;
679   int                 nreads = -1;
680   size_t              total = 0;
681   size_t              header_offset;
682   gboolean            ret;
683
684   while (nreads != 0) {
685     nreads = fread (buf, 1, 1024, f);
686     buffer = g_renew (guchar, buffer, total + nreads);
687     memcpy (buffer + total, buf, nreads);
688     total += nreads;
689   }
690   info = g_new (OpenCVImageInfo, 1);
691   if (!info) {
692     g_set_error (error,
693                  GDK_PIXBUF_ERROR,
694                  GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY,
695                  "Insufficient␣memory␣to␣load␣OpenCV␣image␣info␣struct");
696     return NULL;
697   }
698   ret = read_header (buffer,
699                      total,
700                      &header_offset,
701                      &info->width,
702                      &info->height,
703                      &info->nplanes,
704                      &info->ptype);
705   if (!ret) {
706     g_set_error (error,
707                  GDK_PIXBUF_ERROR,
708                  GDK_PIXBUF_ERROR_FAILED,
709                  "Could␣not␣comletely␣read␣image␣header");
710     return NULL;
711   }
712   ret = read_image (buffer + header_offset, total - header_offset + 1, info);
713   if (!ret) {
714     g_set_error (error,
715                  GDK_PIXBUF_ERROR,
716                  GDK_PIXBUF_ERROR_FAILED,
717                  "Could␣not␣comletely␣read␣image␣header");
718     return NULL;
719   }
720   pixbuf = gdk_pixbuf_new_from_data (info->data, GDK_COLORSPACE_RGB,
721                                      FALSE,
722                                      8,
723                                      info->width, info->height,
724                                      info->width * 3,
725                                      (GdkPixbufDestroyNotify)
726                                      gdk_pixbuf_cv_data_free,
727                                      info);
728   if (!pixbuf) {
729     g_set_error (error,
730                  GDK_PIXBUF_ERROR,
731                  GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY,
732                  "Insufficient␣memory␣to␣allocate␣pixbuf␣struct");
733     return NULL;
734   }
735   g_object_set_data (G_OBJECT(pixbuf), "image-param", (gpointer) info);
736
737   g_free (buffer);
738
739   return pixbuf;
740 }
741
742 static gpointer
743 gdk_pixbuf__cv_image_begin_load (GdkPixbufModuleSizeFunc        size_func,
744                                  GdkPixbufModulePreparedFunc    prepared_func,
745                                  GdkPixbufModuleUpdatedFunc     updated_func,
746                                  gpointer                       user_data,
747                                  GError                         **error)
748 {
```

```
749   OpenCVLoaderContext *context;
750
751   context = g_new (OpenCVLoaderContext, 1);
752   if (!context) {
753     g_set_error (error,
754                  GDK_PIXBUF_ERROR,
755                  GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY,
756                  "Insufficient␣memory␣to␣load␣OpenCV␣context␣struct");
757     return NULL;
758   }
759   context->buffer        = NULL;
760   context->read_bytes    = 0;
761   context->got_header    = FALSE;
762   context->updated_func  = updated_func;
763   context->prepared_func = prepared_func;
764   context->size_func     = size_func;
765   context->user_data     = user_data;
766   context->error         = error;
767
768   return (gpointer) context;
769 }
770
771 static gboolean
772 gdk_pixbuf__cv_image_stop_load  (gpointer       data,
773                                  GError         **error)
774 {
775   OpenCVLoaderContext *context = (OpenCVLoaderContext *) data;
776   OpenCVImageInfo     *info;
777   GdkPixbuf           *pixbuf;
778   gboolean            ret;
779
780   g_return_val_if_fail (context != NULL, TRUE);
781
782   info = g_new (OpenCVImageInfo, 1);
783   if (!info) {
784     g_set_error (error,
785                  GDK_PIXBUF_ERROR,
786                  GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY,
787                  "Insufficient␣memory␣to␣load␣OpenCV␣image␣info␣struct");
788     return FALSE;
789   }
790   info->width   = context->width;
791   info->height  = context->height;
792   info->nplanes = context->nplanes;
793   info->ptype   = context->ptype;
794
795   ret = read_image (context->buffer + context->header_offset,
796                     context->read_bytes - context->header_offset + 1, info);
797   if (!ret) {
798     g_set_error (error,
799                  GDK_PIXBUF_ERROR,
800                  GDK_PIXBUF_ERROR_FAILED,
801                  "Could␣not␣comletely␣read␣image␣header");
802     return FALSE;
803   }
804   pixbuf = gdk_pixbuf_new_from_data (info->data, GDK_COLORSPACE_RGB,
805                                      FALSE,
806                                      8,
807                                      info->width,
808                                      info->height,
809                                      info->width * 3,
810                                      (GdkPixbufDestroyNotify)
811                                      gdk_pixbuf_cv_data_free,
812                                      info);
813   if (!pixbuf) {
814     g_set_error (error,
815                  GDK_PIXBUF_ERROR,
816                  GDK_PIXBUF_ERROR_INSUFFICIENT_MEMORY,
817                  "Insufficient␣memory␣to␣allocate␣pixbuf␣struct");
818     return FALSE;
819   }
820   g_object_set_data (G_OBJECT(pixbuf), "image-param", (gpointer) info);
821
822   if (context->prepared_func) {
823     (*context->prepared_func) (pixbuf, NULL, context->user_data);
824   }
825   if (context) {
```

```
826     if (context->buffer) {
827       g_free (context->buffer);
828       context->buffer = NULL;
829     }
830     g_free (context);
831     context = NULL;
832   }
833   return TRUE;
834 }
835
836 static gboolean
837 gdk_pixbuf__cv_image_load_increment (gpointer           data,
838                                      const guchar       *buf,
839                                      guint              size,
840                                      GError             **error)
841 {
842   OpenCVLoaderContext *context = (OpenCVLoaderContext *) data;
843   gboolean ret;
844
845   g_return_val_if_fail (context != NULL, FALSE);
846   g_return_val_if_fail (buf != NULL, FALSE);
847
848   if (size == 0) return TRUE;
849
850   context->buffer = g_renew (guchar, context->buffer,
851                              context->read_bytes + size);
852   memcpy (context->buffer + context->read_bytes, buf, size);
853   context->read_bytes += size;
854
855   if (!context->got_header) {
856     ret = read_header (context->buffer,
857                        context->read_bytes,
858                        &context->header_offset,
859                        &context->width,
860                        &context->height,
861                        &context->nplanes,
862                        &context->ptype);
863     if (ret) {
864       gint width  = context->width;
865       gint height = context->height;
866       (*context->size_func) (&width, &height, context->user_data);
867       if (width == 0 || height == 0) return FALSE;
868       context->got_header = TRUE;
869     }
870   }
871   return TRUE;
872 }
873
874 G_MODULE_EXPORT void
875 fill_vtable (GdkPixbufModule *module)
876 {
877   module->load           = gdk_pixbuf__cv_image_load;
878   module->begin_load     = gdk_pixbuf__cv_image_begin_load;
879   module->stop_load      = gdk_pixbuf__cv_image_stop_load;
880   module->load_increment = gdk_pixbuf__cv_image_load_increment;
881 }
882
883 G_MODULE_EXPORT void
884 fill_info (GdkPixbufFormat *info)
885 {
886   static GdkPixbufModulePattern signature[] = {
887     { "<opencv_storage>", NULL, 100 },
888     { NULL, NULL, 0 }
889   };
890   static gchar * mime_types[] = {
891     "image/x-opencv",
892     NULL
893   };
894   static gchar * extensions[] = {
895     "xml",
896     NULL
897   };
898   info->name        = "opencv";
899   info->signature   = signature;
900   info->description = _("The␣OpenCV␣image␣format");
901   info->mime_types  = mime_types;
902   info->extensions  = extensions;
```

```
903   info->flags       = GDK_PIXBUF_FORMAT_THREADSAFE;
904   info->license     = "LGPL";
905 }
```

7 ウィジェットリファレンス

この章ではさまざまなウィジェットについて，具体的なサンプルプログラムを通してその使い方を説明します．

7.1 ボタンウィジェット

ボタンウィジェットは，GUI アプリケーションを作成するうえで欠かすことのできない重要なウィジェットです．ボタンウィジェットには，ここでは次の代表的な 3 つのボタンについて紹介します．

- 普通のボタン（GtkButton）... ある動作のトリガー役をする
- チェックボタン（GtkCheckButton）... ある項目のオン・オフを設定する
- ラジオボタン（GtkRadioButton）... 複数の項目のなかから選択を行う

7.1.1 普通のボタン

ボタンウィジェット（GtkButton）は，その名前の通りボタンの役割をするウィジェットです．

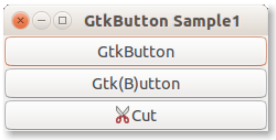


図 7.1　普通のボタン

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkButton
```

シグナルとコールバック関数

表 7.1 にボタンウィジェットのシグナルを示します．通常は **clicked シグナル**のみで十分ですが，ボタンを押したときと離したときで別の動作をさせたい場合には **pressed シグナル**や **released シグナル**を利用します．上記のシグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkButton *button, gpointer user_data);
```

表 7.1　ボタンウィジェットのシグナル

シグナル	説明
activate	ボタンを押して離すという一連の動作，すなわちボタンがクリックされたときに発生するシグナル
clicked	ボタンを押して離すという一連の動作，すなわちボタンがクリックされたときに発生するシグナル
enter	マウスポインタがボタン領域に入ったときに発生するシグナル
leave	マウスポインタがボタン領域から出たときに発生するシグナル
pressed	ボタンが押されたときに発生するシグナル
released	ボタンが離されたときに発生するシグナル

ウィジェットの作成

ボタンウィジェット (GtkButton) を作成する関数は次の 4 つです．

- gtk_button_new
  ラベルのない普通のボタンを作成する関数です．

```
GtkWidget* gtk_button_new (void);
```

- gtk_button_new_with_label
  ラベル付きのボタン（図 7.1 上段）を作成する関数です．

```
GtkWidget* gtk_button_new_with_label (const gchar *label);
```

- gtk_button_new_with_mnemonic
  アクセラレータ機能付きボタン（図 7.1 中段）を作成する関数です．アクセラレータキーに設定したい文字の前にアンダースコアを挿入します．

```
GtkWidget* gtk_button_new_with_mnemonic (const gchar *label);
```

- gtk_button_new_from_icon_name
  アイコンボタン（図 7.1 下段）を作成する関数です．アイコンに対応する文字列から取得します．第 2 引数には表??に示したアイコンの大きさを指定します．

```
GtkWidget* gtk_button_new_from_icon_name (const gchar *icon_name,
                                          GtkIconSize size);
```

  この関数で作成したボタンにはアイコンしか表示されません．図 7.1 下段のようにテキストも同時に表示するためには，関数 gtk_button_set_always_show_image を呼び出して，常に画像を表示する設定にした上で，関数 gtk_button_set_label を呼び出して表示するテキストをセットします．

```
void
gtk_button_set_always_show_image (GtkButton *button,
                                  gboolean  always_show);
```

```
button = gtk_button_new_from_icon_name ("edit-cut", GTK_ICON_SIZE_BUTTON);
gtk_button_set_always_show_image (GTK_BUTTON(button), TRUE);
gtk_button_set_label (GTK_BUTTON(button), "Cut");
```

ウィジェットのプロパティ設定

ボタンのプロパティの中で使用度の高そうなものを表 7.3 に示します．ウィジェットのプロパティを取得する関数は基本的には，gtk_ウィジェット名_get_プロパティという名前となります（異なる場合もありますので注意してください．）．また，プロパティを設定する関数は，gtk_ウィジェット名_set_プロパティという名前です．

例えば，ボタンウィジェットの label プロパティを取得したり，ラベルを更新したりするには次の関数を使用します．

- gtk_button_get_label
  ボタンウィジェットのラベルに設定されている文字列を返します．

```
G_CONST_RETURN gchar* gtk_button_get_label (GtkButton *button);
```

表 7.2　GtkIconSize の値

値	説明
GTK_ICON_SIZE_INVALID	Invalid size.
GTK_ICON_SIZE_MENU	メニュー用の大きさ（16 ピクセル）
GTK_ICON_SIZE_SMALL_TOOLBAR	小さなツールバー用の大きさ（16 ピクセル）
GTK_ICON_SIZE_LARGE_TOOLBAR	大きなツールバー用の大きさ（24 ピクセル）
GTK_ICON_SIZE_BUTTON	ボタン用の大きさ（16 ピクセル）
GTK_ICON_SIZE_DIALOG	ダイアログ用の大きさ（48 ピクセル）

表 7.3　GtkButoon のプロパティ

データ型	プロパティ
gboolean	always-show-image
GtkWidget*	image
GtkPositoinType	image-position
gchar*	label
gfloat	xalign
gfloat	yalign

- gtk_button_set_label
  ボタンウィジェットのラベルを更新します．

```
void gtk_button_set_label (GtkButton *button, const gchar *label);
```

サンプルプログラム

ボタンウィジェットのサンプルプログラムをソース 7–1–1 に示します．このプログラムは，ボタンをクリックした回数を記憶して，クリックのたびにボタンのラベルにクリック回数を表示するものです．プログラム起動時は図 7.2 左のように表示されますが，ボタンをクリックするとコールバック関数 cb_button_clicked が呼び出され，ラベルが図 7.2 右のように更新されます．

ソース 7-1-1 ボタンウィジェットのサンプルプログラム：gtkbutton-sample2.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_button_clicked (GtkButton *widget,
5                     gpointer  user_data)
6  {
7    static int count = 0;
8    char       buf[1024];
9
10   sprintf (buf, "%d␣time(s)␣clicked.", ++count);
11   gtk_button_set_label (widget, buf);
12 }
13
14 int
15 main (int argc, char *argv[])
16 {
17   GtkWidget *window;
18   GtkWidget *box;
19   GtkWidget *button;
20
21   gtk_init (&argc, &argv);
22
```

図 7.2　ボタンウィジェットのサンプルプログラム

```
23   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
24   gtk_window_set_title (GTK_WINDOW (window), "GtkButton␣Sample2");
25   gtk_widget_set_size_request (window, 240, -1);
26   g_signal_connect (G_OBJECT (window), "destroy",
27                     G_CALLBACK (gtk_main_quit), NULL);
28
29   box = gtk_box_new (GTK_ORIENTATION_VERTICAL, 0);
30   gtk_container_add (GTK_CONTAINER (window), box);
31
32   button = gtk_button_new_with_label ("Please␣click␣me.");
33   gtk_box_pack_start (GTK_BOX (box), button, TRUE, TRUE, 0);
34   g_signal_connect (G_OBJECT (button), "clicked",
35                     G_CALLBACK (cb_button_clicked), NULL);
36
37   gtk_widget_show_all (window);
38   gtk_main ();
39
40   return 0;
41 }
```

7.1.2　チェックボタン

チェックボタンウィジェット（**GtkCheckButton**）はボタンウィジェットから派生したウィジェットで，ボタンを押すことでボタン上にチェックマークを表示できます．アプリケーションの設定でオプションの使用の有無をユーザーが設定したいときに使用します．

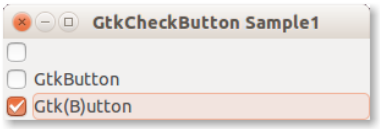


図 7.3　チェックボタン

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkButton
                               +----GtkToggleButton
                                     +----GtkCheckButton
```

シグナルとコールバック関数

表 7.4 にチェックボタンウィジェットのシグナルを示します．オブジェクトの階層構造を見てもわかるように，チェックボタンはトグルボタンの機能を継承したウィジェットです．**toggled シグナル**はトグルボタンから継承したシグナルです．

表 7.4　チェックボタン（トグルボタン）ウィジェットのシグナル

シグナル	説明
toggled	チェックボタンの状態が変化したときに発生するシグナル

toggled シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkToggleButton *togglebutton, gpointer user_data);
```

ウィジェットの作成

チェックボタンウィジェット（GtkCheckButton）を作成する関数は次の 3 つです．

- gtk_check_button_new
  ラベルなしのチェックボタン（図 7.3 上段）を作成する関数です．

```
GtkWidget* gtk_check_button_new (void);
```

- gtk_check_button_new_with_label
  ラベル付きのチェックボタン（図 7.3 中段）を作成する関数です．

```
GtkWidget* gtk_check_button_new_with_label (const gchar *label);
```

- gtk_check_button_new_with_mnemonic
  アクセラレータ機能付きチェックボタン (図 7.3 下段) を作成する関数です．アクセラレータキーに設定したい文字の前にアンダースコアを挿入します．

```
GtkWidget* gtk_check_button_new_with_mnemonic (const gchar *label);
```

ウィジェットのプロパティ設定

チェックボタンがアクティブ（チェックされている状態）かどうかを調べるには，トグルボタンに対する関数を使います．関数 gtk_toggle_button_get_active の戻り値が TRUE なら状態がアクティブ，FALSE なら状態がアクティブではないということになります．

```
gboolean gtk_toggle_button_get_active (GtkToggleButton *toggle_button);
```

トグルボタンの状態を設定するには関数 gtk_toggle_button_set_active を使います．

```
void gtk_toggle_button_set_active (GtkToggleButton *toggle_button,
                                   gboolean         is_active);
```

また，関数 gtk_toggle_button_toggled を使うと，トグルボタンの今の状態を反転（トグル）できます．

```
void gtk_toggle_button_toggled (GtkToggleButton *toggle_button);
```

サンプルプログラム

チェックボタンウィジェットのサンプルプログラムをソース 7–1–2 に示します．このプログラムは，チェックボタンの toggled シグナルに対するコールバック関数 cb_button_toggled 内でチェックボタンの状態を調べて，ターミナル上に表示します．

ソース 7-1-2 チェックボタンウィジェットのサンプルプログラム：gtkcheckbutton-sample2.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_button_toggled (GtkToggleButton *widget,
5                     gpointer         user_data)
6  {
7    gboolean active;
8    gchar    *str[] = {"FALSE", "TRUE"};
9
10   active = gtk_toggle_button_get_active (widget);
11   g_print ("Check␣button␣state␣:␣%s\n", str[active]);
12 }
13
14 int
15 main (int argc, char *argv[])
16 {
17   GtkWidget *window;
18   GtkWidget *box;
19   GtkWidget *button;
20
21   gtk_init (&argc, &argv);
22
23   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
24   gtk_window_set_title (GTK_WINDOW (window), "GtkCheckButton␣Sample2");
25   gtk_widget_set_size_request (window, 300, 40);
26   g_signal_connect (G_OBJECT (window), "destroy",
27                     G_CALLBACK (gtk_main_quit), NULL);
28
29   box = gtk_box_new (GTK_ORIENTATION_VERTICAL, 0);
```

```
30   gtk_container_add (GTK_CONTAINER (window), box);
31
32   button = gtk_check_button_new_with_label ("Please␣click␣me.");
33   gtk_box_pack_start (GTK_BOX (box), button, TRUE, TRUE, 0);
34   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button), TRUE);
35   g_signal_connect (G_OBJECT (button), "toggled",
36                     G_CALLBACK (cb_button_toggled), NULL);
37
38   gtk_widget_show_all (window);
39   gtk_main ();
40
41   return 0;
42 }
```

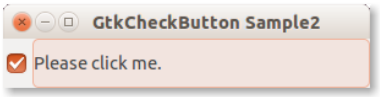


図 7.4　チェックボタンウィジェットのサンプルプログラム

7.1.3　ラジオボタン

ラジオボタンウィジェット（**GtkRadioButton**）はボタンウィジェットから派生したウィジェットで，複数の項目から 1 つの項目を選択するときに使用します．

図 7.5　ラジオボタン

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
        +----GtkWidget
               +----GtkContainer
                      +----GtkBin
                             +----GtkButton
                                    +----GtkToggleButton
                                           +----GtkCheckButton
                                                  +----GtkRadioButton
```

シグナルとコールバック関数

表 7.5 にチェックボタンウィジェットのシグナルを示します．toggled シグナルは先ほどのチェックボタンと同様にトグルボタンに対するシグナルです．**group-changed シグナル**はラジオボタンのシグナルで，そのボタンのグループが変更されたときに発生するシグナルです．

表 7.5　ラジオボタンウィジェットのシグナル

シグナル	説明
group-changed	ラジオボタンのグループが変化したときに発生するシグナル
toggled	ボタンの状態が変化したときに発生するシグナル

group-changed シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkRadioButton *radiobutton, gpointer user_data);
```

ウィジェットの作成

ラジオボタンウィジェット（GtkRadioButton）を作成する関数は次の 6 つです．

- gtk_radio_button_new
  ラベルのないラジオボタン（図 7.5 上段）を作成する関数です．

```
GtkWidget* gtk_radio_button_new (GSList *group);
```

- gtk_radio_button_new_with_label
  ラベル付きのラジオボタン（図 7.5 中段）を作成する関数です．

```
GtkWidget* gtk_radio_button_new_with_label (GSList      *group,
                                            const gchar *label);
```

- gtk_radio_button_new_with_mnemonic
  アクセラレータ機能付きラジオボタン（図 7.5 下段）を作成する関数です．アクセラレータキーに設定したい文字の前にアンダースコアを挿入します．

```
GtkWidget* gtk_radio_button_new_with_mnemonic (GSList      *group,
                                               const gchar *label);
```

以下の関数はラジオボタンウィジェットを引数に与えて，そのラジオボタンと同じグループを持つ新しいラジオボタンを作成する関数です．

- gtk_radio_button_new_from_widget

```
GtkWidget* gtk_radio_button_new_from_widget (GtkRadioButton *group);
```

- gtk_radio_button_new_with_label_from_widget

```
GtkWidget*
gtk_radio_button_new_with_label_from_widget (GtkRadioButton *group,
                                             const gchar    *label);
```

- gtk_radio_button_new_with_mnemonic_from_widget

```
GtkWidget*
gtk_radio_button_new_with_mnemonic_from_widget (GtkRadioButton *group,
                                                const gchar    *label);
```

ラジオボタンを作成する例をソース 7–1–3 に示します．最初のラジオボタンを作成するときは，引数に与えるグループを必ず NULL にします．そして，同じグループのラジオボタンを作成する場合には，先に作成したラジオボタンを引数に与えた関数 gtk_radio_button_new_from_widget によってラジオボタンを作成するようにします．

ソース 7-1-3 ラジオボタンウィジェットの生成

```
1  GtkWidget *radio_button[2];
2
3  radio_button[0] = gtk_radio_button_new (NULL);
4  radio_button[1] =
5    gtk_radio_button_new_from_widget (GTK_RADIO_BUTTON (radio_button[0]));
```

ウィジェットのプロパティ設定

- ボタンの状態
  ラジオボタンもチェックボタンと同様に，そのボタンが選択されているかどうか示すプロパティを持っています．このプロパティを調べるためには，関数 gtk_toggle_button_get_active を用います．
- グループ
  またラジオボタン特有のプロパティとしてグループがあります．ラジオボタンが属しているグループを取得するには関数 gtk_radio_button_get_group を用います．

```
GSList* gtk_radio_button_get_group (GtkRadioButton *radio_button);
```

反対にラジオボタンのグループを別のグループに変更するには，関数 gtk_radio_button_set_group を用います．

```
void gtk_radio_button_set_group (GtkRadioButton *radio_button,
                                 GSList         *group);
```

サンプルプログラム

ラジオボタンウィジェットのサンプルプログラムをソース 7–1–4 に示します．このプログラムは，ラジオボタンの toggled シグナルに対するコールバック関数 cb_button_toggled 内でボタンの状態を調べて，どのボタンが選択されているかをターミナル上に表示します．

また，ここではコールバック関数に渡すデータとして，整数を設定しています．このような場合は GINT_TO_POINTER という特別なマクロを使用してデータを変換してください（11 行目）．反対に gpointer 型に変換されたデータをもとの整数に戻す場合には GPOINTER_TO_INT を使用します．

図 7.6　ラジオボタンウィジェットのサンプルプログラム

ソース 7-1-4 ラジオボタンウィジェットのサンプルプログラム：gtkradiobutton-sample2.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_button_toggled (GtkRadioButton *widget, gpointer user_data)
5  {
6    g_print ("Catch␣the␣toggled␣signal␣from␣the␣radio␣button%d.\n",
7             GPOINTER_TO_INT (user_data));
8    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (widget)))
9      {
10       g_print ("The␣radio␣button%d␣is␣selected.\n",
11                GPOINTER_TO_INT (user_data));
12     }
13 }
14
15 int
16 main (int argc, char *argv[])
17 {
18   GtkWidget *window;
19   GtkWidget *box;
20   GtkWidget *button[3];
21   GSList    *group = NULL;
22
23   gtk_init (&argc, &argv);
24
25   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
26   gtk_window_set_title (GTK_WINDOW (window), "GtkRadioButton␣Sample2");
27   gtk_widget_set_size_request (window, 300, -1);
28   g_signal_connect (G_OBJECT (window), "destroy",
29                     G_CALLBACK (gtk_main_quit), NULL);
30
31   box = gtk_box_new (GTK_ORIENTATION_VERTICAL, 0);
32   gtk_container_add (GTK_CONTAINER (window), box);
33
34   button[0] = gtk_radio_button_new_with_label (group, "Red");
35   gtk_box_pack_start (GTK_BOX (box), button[0], TRUE, TRUE, 0);
36   button[1] =
37     gtk_radio_button_new_with_label_from_widget (GTK_RADIO_BUTTON (button[0]),
38                                                  "Green");
39   gtk_box_pack_start (GTK_BOX (box), button[1], TRUE, TRUE, 0);
40   button[2] =
41     gtk_radio_button_new_with_label_from_widget (GTK_RADIO_BUTTON (button[0]),
42                                                  "Blue");
43   gtk_box_pack_start (GTK_BOX (box), button[2], TRUE, TRUE, 0);
44
45   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button[1]), TRUE);
46
```

```
47   g_signal_connect (G_OBJECT (button[0]), "toggled",
48                     G_CALLBACK (cb_button_toggled), GINT_TO_POINTER (0));
49   g_signal_connect (G_OBJECT (button[1]), "toggled",
50                     G_CALLBACK (cb_button_toggled), GINT_TO_POINTER (1));
51   g_signal_connect (G_OBJECT (button[2]), "toggled",
52                     G_CALLBACK (cb_button_toggled), GINT_TO_POINTER (2));
53
54   gtk_widget_show_all (window);
55   gtk_main ();
56
57   return 0;
58 }
```

7.2 コンテナウィジェット

コンテナウィジェット（GtkContainer）とは，GtkWindow のような他のウィジェットを内部に配置するウィジェットを指します．ここで紹介するコンテナウィジェットは，単に他のウィジェットを内部に配置するだけでなく，そのほかにも特殊な機能を持ちます．

7.2.1 ウィンドウ

ウィンドウウィジェット（GtkWindow）は，アプリケーションの基本となるウィジェットです．ウィンドウウィジェットにはその他のウィジェットを配置するコンテナとしての役割と，ウィンドウを最大化したりアイコン化したりといったアプリケーションを操作する役割があります．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkWindow
```

ウィジェットの作成

ウィンドウの作成には関数 gtk_window_new を使います．引数には，GtkWindowType で定義された値（表 7.6 を参照）で，ウィンドウタイプを指定します．標準のウィンドウの場合は GTK_WINDOW_TOPLEVEL を指定します．ポップアップウィンドウの場合には GTK_WINDOW_POPUP を指定します．

```
GtkWidget* gtk_window_new (GtkWindowType type);
```

表 7.6 GtkWindowType の値

値	説明
GTK_WINDOW_TOPLEVEL	通常のウィンドウを作成する
GTK_WINDOW_POPUP	ポップアップウィンドウを作成する

子ウィジェットの配置

フレームウィジェットにウィジェットを配置するには GtkContainer ウィジェットの関数 gtk_container_add を使用します．

ウィジェットのプロパティ設定

ここではたくさんあるウィンドウウィジェットのプロパティをいくつか紹介します．

- ウィンドウタイトル
  ウィンドウのタイトルバーに表示するタイトルです．タイトルの取得や設定は，関数 gtk_window_get_title と関数 gtk_window_set_title で行います．

  ```
  const gchar* gtk_window_get_title (GtkWindow *window);
  ```

```
void gtk_window_set_title (GtkWindow *window, const gchar *title);
```

- ウィンドウサイズの変更設定
  関数 gtk_window_set_resizable を使うと，ユーザがウィンドウサイズを変更できるかどうかを設定できます．現在の設定は，関数 gtk_window_get_resizable で取得できます．

```
gboolean gtk_window_get_resizable (GtkWindow *window);

void gtk_window_set_resizable (GtkWindow *window,
                               gboolean  resizable);
```

- アイコン
  ウィンドウのタイトルバーに表示されたり，ウィンドウをアイコン化した際に表示されるアイコンです．

```
GdkPixbuf* gtk_window_get_icon (GtkWindow *window);

void gtk_window_set_icon (GtkWindow *window, GdkPixbuf *icon);
```

- ウィンドウの装飾
  ウィンドウは通常タイトルバーなどの装飾が表示されますが，関数 gtk_window_set_decorated の第 2 引数に FALSE を指定すると，ウィンドウの装飾が表示されなくなります．

```
void gtk_window_set_decorated (GtkWindow *window,
                               gboolean setting);
```

  関数 gtk_window_get_decorated を使用すると現在の設定を取得できます．

```
gboolean gtk_window_get_decorated (GtkWindow *window);
```

ウィンドウの操作

以下にウィンドウの操作に関するいくつかの項目を紹介します．

- ウィンドウのサイズを指定する
  関数 gtk_widget_set_size_request の第 2 引数と第 3 引数でウィンドウの横と縦のサイズを指定します．

```
void gtk_widget_set_size_request (GtkWidget *widget,
                                  gint      width,
                                  gint      height);
```

- ウィンドウのサイズを固定する
  ウィンドウサイズを固定するには関数 gtk_window_set_resizable を使用します．関数の第 2 引数に FALSE を指定すると，ユーザがマウスのドラッグ等でウィンドウのサイズを変更できなくなります．
- ウィンドウのサイズを変更する
  ウィンドウサイズを変更するには関数 gtk_window_resize を使用します．関数の第 2 引数と第 3 引数にウィンドウの横と縦のサイズを指定します．

```
void gtk_window_resize (GtkWindow *window,
                        gint      width,
                        gint      height);
```

  関数 gtk_widget_set_size_request を使用することもできます．
- ウィンドウのサイズを画面の大きさに合わせる
  ウィンドウのサイズを画面の大きさと一致させるには，画面の大きさを取得して，関数 gtk_window_resize もしくは gtk_widget_set_size_request でウィンドウの大きさを変更します．画面の大きさを取得する 1 つの方法は，関数 gdk_screen_width と関数 gdk_screen_height を使用することです．

```
gint gdk_screen_width (void);

gint gdk_screen_height (void);
```

- ウィンドウをフルスクリーン表示する
  ウィンドウをフルスクリーン表示するには関数 gtk_window_fullscreen を使用します．反対にフルスクリーン状態を解除するには関数 gtk_window_unfullscreen を使用します．

```
void gtk_window_fullscreen (GtkWindow *window);
```

```
void gtk_window_unfullscreen (GtkWindow *window);
```

- ウィンドウをアイコン化する
  ウィンドウをアイコン化するには関数 gtk_window_iconify を使用します．反対にアイコン化状態を解除するには関数 gtk_window_deiconify を使用します．

```
void gtk_window_iconify (GtkWindow *window);

void gtk_window_deiconify (GtkWindow *window);
```

- ウィンドウを最大化する
  ウィンドウを最大化するには関数 gtk_window_maximize を使用します．反対に最大化状態を解除するには関数 gtk_window_unmaximize を使用します．

```
void gtk_window_maximize (GtkWindow *window);

void gtk_window_unmaximize (GtkWindow *window);
```

- ウィンドウの表示位置を指定する
  ウィンドウの表示位置を指定するには関数 gtk_window_move を使用します．関数の第 2 引数と第 3 引数にウィンドウの左上の座標を指定します．

```
void gtk_window_move (GtkWindow *window, gint x, gint y);
```

7.2.2 フレーム

フレームウィジェット（GtkFrame）は，枠を持ったコンテナウィジェットで，配置したウィジェットの外側に枠を表示してアプリケーションの見た目を良くしてくれます．また，枠に対して見出しを付けることもできます．このウィジェットを使うと，目的や機能ごとにウィジェットをグループ化できて，見た目が良くなるだけではなく，操作性も向上させられます．

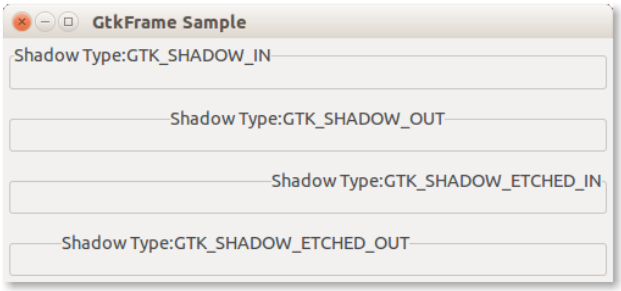


図 7.7 フレーム表示の例

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkFrame
```

ウィジェットの作成

フレームの作成には関数 gtk_frame_new を使います．引数にはフレームの見出しのテキストを与えます．

```
GtkWidget* gtk_frame_new (const gchar *label);
```

子ウィジェットの配置

フレームウィジェットにウィジェットを配置するには，GtkContainer ウィジェットの関数 gtk_container_add を使用します．

ウィジェットのプロパティ設定

フレームウィジェットのプロパティには次の 3 つの項目が存在します.

- 見出しテキスト
  見出しテキストは次の関数を使って取得したり，設定したりできます.

```
G_CONST_RETURN gchar* gtk_frame_get_label (GtkFrame *frame);

void gtk_frame_set_label (GtkFrame *frame, const gchar *label);
```

- 見出し位置
  見出しをフレームのどの位置に表示するかを設定します．第 2 引数の xalign が 0 なら左，1 なら右に表示します．また，第 3 引数の yalign が 0 なら上，1 なら下に表示します。

```
void gtk_frame_get_label_align (GtkFrame *frame,
                                gfloat *xalign, gfloat *yalign);

void gtk_frame_set_label_align (GtkFrame *frame,
                                gfloat xalign, gfloat yalign);
```

- フレームの装飾
  フレームの装飾の種類は表 7.7 に示す **GtkShadowType** で指定します.

```
GtkShadowType gtk_frame_get_shadow_type (GtkFrame *frame);

void gtk_frame_set_shadow_type (GtkFrame *frame, GtkShadowType type);
```

表 7.7　フレームの種類

種類	説明
GTK_SHADOW_NONE	装飾なし
GTK_SHADOW_IN	内側にへこむように影を付ける
GTK_SHADOW_OUT	外側に飛び出すように影を付ける
GTK_SHADOW_ETCHED_IN	枠だけ内側にへこむように影を付ける
GTK_SHADOW_ETCHED_OUT	枠だけ外側に飛び出すように影を付ける

7.2.3　ペイン

ペインウィジェット（**GtkPaned**）は，仕切りで区切られた 2 つのコンテナスペースにウィジェットを配置することのできるウィジェットです．ペインの特徴は，この仕切りをマウスでドラッグすることで，2 つの領域の大きさを変えられることです（図 7.8）.

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkPaned
```

ウィジェットの作成

ペインには，領域を横に 2 つに分割する横型のペインと，領域を縦に 2 つに分割する縦型のペインの 2 種類があります．ペインウィジェットを作成する関数は，関数 gtk_paned_new です.

```
GtkWidget* gtk_paned_new (GtkOrientation orientation);
```

図 7.8 ペインによる領域の分割

子ウィジェットの配置

ペインにウィジェットを配置するには，関数 gtk_paned_pack1 と gtk_paned_pack2 を使用します．左右（または上下）どちらの領域にウィジェットを配置するかによって，使用する関数を使い分ける必要があります．

第 3 引数は，ペインの領域が子ウィジェットよりも大きくなったときに，子ウィジェットの領域をペインの領域に合わせて拡張するかどうかを指定します．第 4 引数は，ペインの領域は子ウィジェットよりも小さくなったときに，子ウィジェットの領域をペインの領域に合わせて縮小するかどうかを指定します．

```
void
gtk_paned_pack1 (GtkPaned *paned,
                 GtkWidget *child, gboolean resize, gboolean shrink);

void
gtk_paned_pack2 (GtkPaned *paned,
                 GtkWidget *child, gboolean resize, gboolean shrink);
```

第 1 引数：	ペインウィジェット
第 2 引数：	配置する子ウィジェット
第 3 引数：	子ウィジェット領域の拡張の可否
第 4 引数：	子ウィジェット領域の縮小の可否

次の関数を使うと簡単にペインに子ウィジェットを配置できます．この関数は，上記の関数の第 3 引数に FALSE，第 4 引数に TRUE を指定したものと同様の働きをします．

```
void gtk_paned_add1 (GtkPaned *paned, GtkWidget *child);

void gtk_paned_add2 (GtkPaned *paned, GtkWidget *child);
```

7.2.4 ツールバー

ツールバーウィジェット（GtkToolbar）は，アイコン付きのボタンなどを並べてアプリケーションの操作性を高めるために使われるウィジェットです．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkToolbar
```

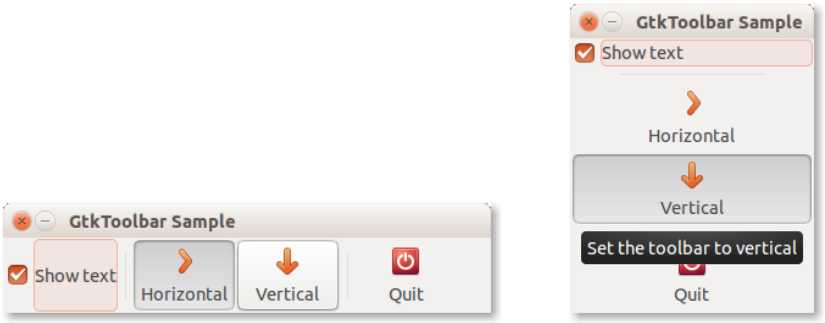


図 7.9　ツールバー

ウィジェットの作成

ツールバーウィジェット（GtkToolbar）の作成には関数 gtk_toolbar_new を使います．

```
GtkWidget* gtk_toolbar_new (void);
```

子ウィジェットの配置

ツールバーにウィジェットを配置するには，関数 gtk_toolbar_insert を使用します．

```
void gtk_toolbar_insert (GtkToolbar  *toolbar,
                         GtkToolItem *item,
                         gint        pos);
```

第 1 引数：　ツールバーウィジェット
第 2 引数：　配置するツールバーアイテム
第 3 引数：　アイテムの追加位置．負の値を指定すると末尾に追加される

ツールアイテムの作成

ツールアイテム（GtkToolItem）には，次の 3 種類があります．

- 普通のアイテム
  さまざまなウィジェットを配置することのできる汎用性の高いツールアイテムです．
- ボタンアイテム
  ボタンウィジェットを配置したツールアイテムです．ツールボタンウィジェットには，普通のボタン，トグルボタン，ラジオボタンの 3 種類があります．
- セパレータアイテム
  アイテムをグループごとに仕切る役割をするツールアイテムです．

それぞれのツールアイテムの作成方法を説明します．

普通のアイテムの作成　普通のツールアイテムを作成するには，関数 gtk_tool_item_new を使用します．

```
GtkToolItem* gtk_tool_item_new (void);
```

そして，アイテムに使用したいウィジェットを作成し，関数 gtk_container_add を使用して，ウィジェットをアイテムに配置します．

普通のボタンアイテムの作成　普通のボタンアイテムを作成するには，関数 gtk_tool_button_new を使用します．

```
GtkToolItem* gtk_tool_button_new (GtkWidget   *icon_widget,
                                  const gchar *label);
```

第 1 引数：　アイコン用ウィジェット
第 2 引数：　ラベル
戻り値：　ツールアイテム

トグルボタンアイテムの作成 トグルボタンアイテムを作成するには，関数 gtk_toggle_tool_button_new を使用します．

```
GtkToolItem* gtk_toggle_tool_button_new (void);
```

ラジオボタンアイテムの作成 ラジオボタンアイテムを作成するには，関数 gtk_radio_tool_button_new を使用します．

```
GtkToolItem* gtk_radio_tool_button_new (GSList *group);
```

ラジオボタンアイテムの作成手順はラジオボタンと同様です．最初のラジオボタンアイテムを作成するときには，それぞれの関数の第 1 引数には NULL を与えます．続くラジオボタンアイテムを作成するには，関数の第 1 引数に関数 gtk_radio_tool_button_get_group で取得した最初のアイテムのグループを与えます．

```
GSList* gtk_radio_tool_button_get_group (GtkRadioToolButton *button);
```

トグルボタンアイテムとラジオボタンアイテムを作成する関数の引数を見ると気づくと思いますが，これらのアイテムを作成する際には，ラベルもアイコンも指定することができません．ラベルやアイコンを配置するには，それぞれ以下の関数を使用します．

```
void gtk_tool_button_set_label (GtkToolButton *button, const gchar *label);

void gtk_tool_button_set_icon_name (GtkToolButton *button, const gchar *icon_name);

void gtk_tool_button_set_icon_name (GtkToolButton *button, GtkWidget *icon_widget);
```

アイコンは，アイコンテーマで定義された文字列で指定する方法と別の関数を使って作成したアイコンウィジェットを指定する方法があります．

セパレータアイテムの作成 セパレータアイテムを作成するには，関数 gtk_separator_tool_item_new を使用します．

```
GtkToolItem* gtk_separator_tool_item_new (void);
```

シグナルとコールバック関数

表 7.9 にツールバーウィジェットのシグナルを示します．**orientation-changed シグナル**に対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkToolbar     *toolbar,
                    GtkOrientation orientation,
                    gpointer       user_data);
```

GtkOrientation は次のように定義されており，現在のツールバーの方向が変数 orientation に入ります．

表 7.8 GtkOrientation の値

値	説明
GTK_ORIENTATION_HORIZONTAL	横方向の配置
GTK_ORIENTATION_VERTICAL	縦方向の配置

popup-context-menu シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．変数 x，y にはマウスカーソルの座標が，変数 button にはボタン番号が入ります．キーが押された場合には値は −1 となります．

```
gboolean user_function (GtkToolbar *toolbar,
                        gint       x,
                        gint       y,
                        gint       button,
                        gpointer   user_data);
```

表 7.9 ツールバーウィジェットのシグナル

シグナル	説明
orientation-changed	ツールバーの方向が変化したときに発生するシグナル
popup-context-menu	ポップアップメニューを表示するためにマウスの右ボタンをクリックしたり，キーが押されたりしたときに発生するシグナル
style-changed	ツールバーのスタイルが変更されたときに発生するシグナル

style-changed シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkToolbar      *toolbar,
                    GtkToolbarStyle style,
                    gpointer        user_data);
```

GtkToolbarStyle は次のように定義されており，現在のツールバーのスタイルが変数 style に入ります．

表 7.10　GtkToolbarStype の値

値	説明
GTK_TOOLBAR_ICONS	アイコンのみのスタイル
GTK_TOOLBAR_TEXT	テキストのみのスタイル
GTK_TOOLBAR_BOTH	アイコンとテキストのスタイル
GTK_TOOLBAR_BOTH_HORIZ	アイコンとテキストが横に並んだスタイル

ウィジェットのプロパティ設定

ツールバーウィジェットのプロパティには次の 3 つの項目が存在します．

- ツールバーの方向（ツールバーアイテムが配置される方向）
  ツールバーの方向は次の関数を使って取得したり，設定したりできます．

```
void gtk_orientable_set_orientation (GtkOrientable  *orientable,
                                     GtkOrientation orientation);

GtkOrientation gtk_orientable_get_orientation (GtkOrientable *orientable);
```

- アイコンサイズ
  ツールバーアイテムのアイコンの大きさです．アイコンサイズは GtkIconSize で定義された値を指定します（表??を参照）．

```
void gtk_toolbar_set_icon_size (GtkToolbar  *toolbar,
                                GtkIconSize icon_size);

GtkIconSize gtk_toolbar_get_icon_size (GtkToolbar *toolbar);
```

サンプルプログラム

ツールバーウィジェットのサンプルプログラムをソース 7–2–1 に示します．プログラムの実行結果は図 7.9（p. 144）になります．

ソース 7-2-1 ツールバーウィジェットのサンプルプログラム：gtktoolbar-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_show_text (GtkWidget *widget, gpointer user_data)
5  {
6    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (widget)))
7      {
8        gtk_toolbar_set_style (GTK_TOOLBAR (user_data), GTK_TOOLBAR_BOTH);
9      }
10   else
11     {
12       gtk_toolbar_set_style (GTK_TOOLBAR (user_data), GTK_TOOLBAR_ICONS);
13     }
14 }
15
16 static void
17 cb_set_horizontal (GtkToggleToolButton *widget, gpointer user_data)
18 {
19   if (gtk_toggle_tool_button_get_active (widget))
20     {
21       GtkOrientable *orientable;
```

```
22
23        orientable
24          = GTK_ORIENTABLE (g_object_get_data (G_OBJECT (user_data),
25                                               "toolbar"));
26        gtk_orientable_set_orientation (orientable,
27                                        GTK_ORIENTATION_HORIZONTAL);
28        gtk_widget_set_size_request (GTK_WIDGET (user_data), 400, -1);
29      }
30  }
31
32  static void
33  cb_set_vertical (GtkToggleToolButton *widget, gpointer user_data)
34  {
35    if (gtk_toggle_tool_button_get_active (widget))
36      {
37        GtkOrientable *orientable;
38
39        orientable
40          = GTK_ORIENTABLE (g_object_get_data (G_OBJECT (user_data),
41                                               "toolbar"));
42        gtk_orientable_set_orientation (orientable,
43                                        GTK_ORIENTATION_VERTICAL);
44        gtk_widget_set_size_request (GTK_WIDGET (user_data), 200, -1);
45      }
46  }
47
48  int
49  main (int argc, char *argv[])
50  {
51    GtkWidget    *window;
52    GtkWidget    *toolbar;
53    GtkWidget    *widget;
54    GtkToolItem *item;
55
56    gtk_init (&argc, &argv);
57
58    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
59    gtk_window_set_title (GTK_WINDOW (window), "GtkToolbar␣Sample");
60    gtk_window_set_resizable (GTK_WINDOW (window), FALSE);
61    g_signal_connect (G_OBJECT (window), "destroy",
62                      G_CALLBACK (gtk_main_quit), NULL);
63    gtk_widget_set_size_request (window, 400, -1);
64
65    toolbar = gtk_toolbar_new ();
66    gtk_container_add (GTK_CONTAINER (window), toolbar);
67    gtk_toolbar_set_style (GTK_TOOLBAR (toolbar), GTK_TOOLBAR_BOTH);
68
69    g_object_set_data (G_OBJECT (window), "toolbar", (gpointer) toolbar);
70
71    item = gtk_tool_item_new ();
72    widget = gtk_check_button_new_with_label ("Show␣text");
73    gtk_container_add (GTK_CONTAINER (item), widget);
74    gtk_tool_item_set_tooltip_text (item, "Toggle␣whether␣show␣text");
75    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (widget), TRUE);
76    g_signal_connect (G_OBJECT (widget), "toggled",
77                      G_CALLBACK (cb_show_text), (gpointer) toolbar);
78    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
79
80    item = gtk_separator_tool_item_new ();
81    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
82
83    item = gtk_radio_tool_button_new (NULL);
84    gtk_tool_button_set_icon_name (GTK_TOOL_BUTTON (item), "go-next");
85    gtk_tool_button_set_label (GTK_TOOL_BUTTON (item), "Horizontal");
86    gtk_toggle_tool_button_set_active (GTK_TOGGLE_TOOL_BUTTON (item), TRUE);
87    gtk_tool_item_set_tooltip_text (item, "Set␣the␣toolbar␣to␣horizontal");
88    g_signal_connect (G_OBJECT (item), "toggled",
89                      G_CALLBACK (cb_set_horizontal), (gpointer) window);
90    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
91
92    item = gtk_radio_tool_button_new_from_widget (GTK_RADIO_TOOL_BUTTON (item));
93    gtk_tool_button_set_icon_name (GTK_TOOL_BUTTON (item), "go-down");
94    gtk_tool_button_set_label (GTK_TOOL_BUTTON (item), "Vertical");
95    gtk_tool_item_set_tooltip_text (item, "Set␣the␣toolbar␣to␣vertical");
96    g_signal_connect (G_OBJECT (item), "toggled",
97                      G_CALLBACK (cb_set_vertical), (gpointer) window);
```

```
98    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
99
100   item = gtk_separator_tool_item_new ();
101   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
102
103   widget = gtk_image_new_from_icon_name ("application-exit",
104                                          GTK_ICON_SIZE_LARGE_TOOLBAR);
105   item = gtk_tool_button_new (widget, "Quit");
106   gtk_tool_item_set_tooltip_text (item, "Exit␣this␣program");
107   g_signal_connect_swapped (G_OBJECT (item), "clicked",
108                             G_CALLBACK (gtk_main_quit), (gpointer) window);
109   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
110
111   gtk_widget_show_all (window);
112   gtk_main ();
113
114   return 0;
115 }
```

7.2.5　ノートブック

ノートブックウィジェット（**GtkNotebook**）は，タブ見出しの付いたコンテナウィジェット（図 7.10）です．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkNotebook
```

シグナルとコールバック関数

表 7.11 にノートブックウィジェットの代表的なシグナルを示します．**switch-page シグナル**に対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkNotebook *notebook,
                    GtkWidget   *page,
                    guint        page_num,
                    gpointer     user_data);
```

表 7.11　ノートブックウィジェットのシグナル

シグナル	説明
switch-page	ページが切り替わったときときに発生するシグナル

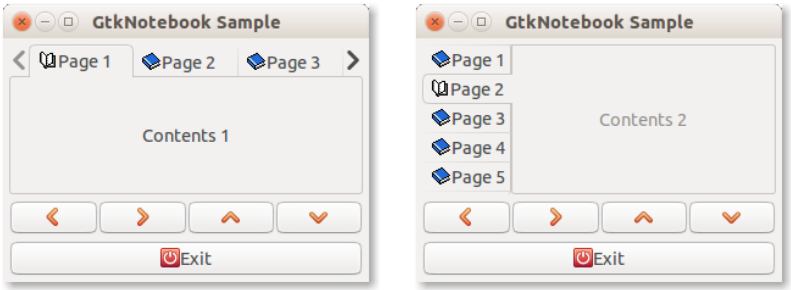


図 7.10　ノートブックウィジェット

ウィジェットの作成

ノートブックウィジェットを作成する関数は gtk_notebook_new です.

```
GtkWidget* gtk_notebook_new (void);
```

子ウィジェットの配置

ノートブックウィジェットにウィジェットを配置するには次の関数を使用します.

- gtk_notebook_append_page
  ノートブックウィジェットにウィジェットを追加します.
  child には配置する子ウィジェットを指定します. tab_label にはタブ領域に表示するウィジェット（例えばラベルウィジェット）を指定します. 関数の戻り値として, 追加したページ番号（ページ番号は 0 から開始）が返ります.

```
gint gtk_notebook_append_page (GtkNotebook *notebook,
                               GtkWidget   *child,
                               GtkWidget   *tab_label);
```

第 1 引数：	ノートブックウィジェット
第 2 引数：	配置する子ウィジェット
第 3 引数：	タブ領域に配置するウィジェット
戻り値：	追加したページの位置番号

- gtk_notebook_append_page_menu
  ノートブックウィジェットにウィジェットを追加します. 関数 gtk_notebook_append_page との違いはポップアップ用のラベルウィジェットを指定する点です.

```
gint gtk_notebook_append_page_menu (GtkNotebook *notebook,
                                    GtkWidget   *child,
                                    GtkWidget   *tab_label,
                                    GtkWidget   *menu_label);
```

第 1 引数：	ノートブックウィジェット
第 2 引数：	配置する子ウィジェット
第 3 引数：	タブ領域に配置するウィジェット
第 4 引数：	ポップアップメニュー用のウィジェット
戻り値：	追加したページの位置番号

このほかに, 現在のページの前に新しいページを追加する関数 gtk_notebook_prepend_page と関数 gtk_notebook_prepend_page_menu や, 指定した位置に新しいページを挿入する関数 gtk_notebook_insert_page と関数 gtk_notebook_insert_page_menu が存在します.

```
gint gtk_notebook_prepend_page (GtkNotebook *notebook,
                                GtkWidget   *child,
                                GtkWidget   *tab_label);

gint gtk_notebook_prepend_page_menu (GtkNotebook *notebook,
                                     GtkWidget   *child,
                                     GtkWidget   *tab_label,
                                     GtkWidget   *menu_label);

gint gtk_notebook_insert_page (GtkNotebook *notebook,
                               GtkWidget   *child,
                               GtkWidget   *tab_label,
                               gint         position);

gint gtk_notebook_insert_page_menu (GtkNotebook *notebook,
                                    GtkWidget   *child,
                                    GtkWidget   *tab_label,
```

```
                                    GtkWidget    *menu_label,
                                    gint         position);
```

ウィジェットのプロパティ設定

ノートブックウィジェットのプロパティには次の 3 つの項目が存在します.

- ポップアップメニュー表示の可/不可

```
void gtk_notebook_popup_enable (GtkNotebook *notebook);

void gtk_notebook_popup_disable (GtkNotebook *notebook);
```

- 現在のページ番号
  現在表示されているページ番号を取得したり，指定したページを表示したりします.

```
void gtk_notebook_set_current_page (GtkNotebook *notebook,
                                    gint         page_num);

gint gtk_notebook_get_current_page (GtkNotebook *notebook);
```

- ページ数
  次の関数によってページ数を取得します.

```
gint gtk_notebook_get_n_pages (GtkNotebook *notebook);
```

- スクロールの可/不可
  タブが領域内に収まらない場合に，タブをスクロールするための矢印を表示するかどうかを設定します.

```
void gtk_notebook_set_scrollable (GtkNotebook *notebook,
                                  gboolean    scrollable);

gboolean gtk_notebook_get_scrollable (GtkNotebook *notebook);
```

- タブの表示位置
  タブの表示位置を設定したり，タブの表示位置を切り替えたりします.

```
void gtk_notebook_set_tab_pos (GtkNotebook     *notebook,
                               GtkPositionType pos);

GtkPositionType gtk_notebook_get_tab_pos (GtkNotebook *notebook);
```

サンプルプログラム

ソース 7–2–3 にノートブックウィジェットのサンプルプログラムを示します.

ソース 7-2-3 ノートブックウィジェットのサンプルプログラム：gtknotebook-sample.c

```
 1 #include <gtk/gtk.h>
 2
 3 GdkPixbuf *book_open;
 4 GdkPixbuf *book_close;
 5
 6 static void
 7 set_page_image (GtkNotebook *notebook,
 8                 gint         page_num,
 9                 GdkPixbuf    *pixbuf)
10 {
11   GtkWidget *page_widget;
12   GtkWidget *icon;
13
14   page_widget = gtk_notebook_get_nth_page (notebook, page_num);
15   icon = (GtkWidget *)
16     g_object_get_data (G_OBJECT (page_widget), "tab_icon");
17   gtk_image_set_from_pixbuf (GTK_IMAGE (icon), pixbuf);
18   icon = (GtkWidget *)
19     g_object_get_data (G_OBJECT (page_widget), "menu_icon");
20   gtk_image_set_from_pixbuf (GTK_IMAGE (icon), pixbuf);
```

```
21 }
22
23 static void
24 page_switch (GtkNotebook     *notebook,
25              GtkWidget       *page,
26              gint            page_num,
27              gpointer        user_data)
28 {
29   gint        old_page_num;
30
31   old_page_num = gtk_notebook_get_current_page (notebook);
32   if (page_num == old_page_num) return;
33   set_page_image (notebook, page_num, book_open);
34   if (old_page_num != -1)
35     {
36       set_page_image (notebook, old_page_num, book_close);
37     }
38 }
39
40 static void
41 cb_tab_position_left (GtkWidget *widget, gpointer user_data)
42 {
43   gtk_notebook_set_tab_pos (GTK_NOTEBOOK (user_data), GTK_POS_LEFT);
44 }
45
46 static void
47 cb_tab_position_right (GtkWidget *widget, gpointer user_data)
48 {
49   gtk_notebook_set_tab_pos (GTK_NOTEBOOK (user_data), GTK_POS_RIGHT);
50 }
51
52 static void
53 cb_tab_position_top (GtkWidget *widget, gpointer user_data)
54 {
55   gtk_notebook_set_tab_pos (GTK_NOTEBOOK (user_data), GTK_POS_TOP);
56 }
57
58 static void
59 cb_tab_position_bottom (GtkWidget *widget, gpointer user_data)
60 {
61   gtk_notebook_set_tab_pos (GTK_NOTEBOOK (user_data), GTK_POS_BOTTOM);
62 }
63
64 int
65 main (int argc, char *argv[])
66 {
67   GtkWidget *window;
68   GtkWidget *vbox;
69   GtkWidget *hbox;
70   GtkWidget *notebook;
71   GtkWidget *box;
72   GtkWidget *label;
73   GtkWidget *label_box;
74   GtkWidget *menu_box;
75   GtkWidget *icon;
76   GtkWidget *button;
77   gchar     buf[1024];
78   int       n;
79   GtkWidget *widget;
80
81   gtk_init (&argc, &argv);
82
83   book_open  = gdk_pixbuf_new_from_file ("stock_book_open.png",  NULL);
84   book_close = gdk_pixbuf_new_from_file ("stock_book_close.png", NULL);
85
86   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
87   gtk_window_set_title  (GTK_WINDOW (window), "GtkNotebook␣Sample");
88   gtk_window_set_resizable (GTK_WINDOW(window), TRUE);
89   gtk_widget_set_size_request (window, 300, 200);
90   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
91   g_signal_connect (G_OBJECT (window), "destroy",
92                     G_CALLBACK (gtk_main_quit), NULL);
93
94   vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 5);
95   gtk_container_add (GTK_CONTAINER (window), vbox);
96
```

```
97  notebook = gtk_notebook_new ();
98  gtk_notebook_set_scrollable (GTK_NOTEBOOK (notebook), TRUE);
99  gtk_notebook_popup_enable (GTK_NOTEBOOK (notebook));
100 g_signal_connect (G_OBJECT (notebook), "switch-page",
101                    G_CALLBACK (page_switch), NULL);
102 gtk_box_pack_start (GTK_BOX (vbox), notebook, TRUE, TRUE, 0);
103
104 for (n = 1; n <= 5; n++)
105   {
106     box   = gtk_box_new (GTK_ORIENTATION_VERTICAL, 0);
107
108     sprintf (buf, "Contents␣%d", n);
109     label = gtk_label_new (buf);
110     gtk_box_pack_start (GTK_BOX (box), label, TRUE, TRUE, 0);
111
112     label_box = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
113     icon = gtk_image_new_from_pixbuf (book_close);
114     g_object_set_data (G_OBJECT (box), "tab_icon", icon);
115     gtk_box_pack_start (GTK_BOX (label_box), icon, FALSE, TRUE, 0);
116
117     sprintf (buf, "Page␣%d", n);
118     label = gtk_label_new (buf);
119     gtk_box_pack_start (GTK_BOX (label_box), label, FALSE, TRUE, 0);
120
121     gtk_widget_show_all (label_box);
122
123     menu_box = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
124     icon = gtk_image_new_from_pixbuf (book_close);
125     g_object_set_data (G_OBJECT (box), "menu_icon", icon);
126     gtk_box_pack_start (GTK_BOX (menu_box), icon, FALSE, TRUE, 0);
127
128     label = gtk_label_new (buf);
129     gtk_box_pack_start (GTK_BOX (menu_box), label, FALSE, TRUE, 0);
130
131     gtk_widget_show_all (menu_box);
132
133     gtk_notebook_append_page_menu (GTK_NOTEBOOK (notebook),
134                                    box, label_box, menu_box);
135   }
136 widget = gtk_notebook_get_nth_page (GTK_NOTEBOOK(notebook), 1);
137 gtk_widget_set_sensitive (widget, FALSE);
138
139 hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
140 gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
141 {
142   button =
143     gtk_button_new_from_icon_name ("go-previous", GTK_ICON_SIZE_BUTTON);
144   g_signal_connect (G_OBJECT (button), "clicked",
145                     G_CALLBACK (cb_tab_position_left), notebook);
146   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
147
148   button =
149     gtk_button_new_from_icon_name ("go-next", GTK_ICON_SIZE_BUTTON);
150   g_signal_connect (G_OBJECT (button), "clicked",
151                     G_CALLBACK (cb_tab_position_right), notebook);
152   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
153
154   button = gtk_button_new_from_icon_name ("go-up", GTK_ICON_SIZE_BUTTON);
155   g_signal_connect (G_OBJECT (button), "clicked",
156                     G_CALLBACK (cb_tab_position_top), notebook);
157   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
158
159   button = gtk_button_new_from_icon_name ("go-down", GTK_ICON_SIZE_BUTTON);
160   g_signal_connect (G_OBJECT (button), "clicked",
161                     G_CALLBACK (cb_tab_position_bottom), notebook);
162   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
163 }
164 button =
165   gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
166 gtk_button_set_label (GTK_BUTTON (button), "Exit");
167 gtk_button_set_always_show_image (GTK_BUTTON (button), TRUE);
168 g_signal_connect (G_OBJECT (button), "clicked",
169                   G_CALLBACK (gtk_main_quit), NULL);
170 gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
171
172 gtk_widget_show_all (window);
```

```
173   gtk_main ();
174
175   return 0;
176 }
```

7.2.6 エクスパンダ

エクスパンダウィジェット（**GtkExpander**）は，配置したウィジェットを表示したり隠したり切り替えることができるウィジェットです．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkExpander
```

シグナルとコールバック関数

表 7.12 にエクスパンダウィジェットのシグナルを示します．**activate シグナル**に対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkExpander *toolbar,
                    gpointer     user_data);
```

表 7.12 エクスパンダウィジェットのシグナル

シグナル	説明
activate	子ウィジェットを表示したり，隠したりしたときに発生するシグナル

ウィジェットの作成

エクスパンダウィジェットを作成するには次の関数を使用します．

- gtk_expander_new
  ラベル文字を指定してウィジェットを作成します．

```
GtkWidget* gtk_expander_new (const gchar *label);
```

- gtk_expander_new_with_mnemonic
  アクセラレータ機能付きラベル文字を指定してウィジェットを作成します．

```
GtkWidget* gtk_expander_new_with_mnemonic (const gchar *label);
```

子ウィジェットの配置

エクスパンダウィジェットにウィジェットを配置するには関数 gtk_container_add を使用します．

ウィジェットのプロパティ設定

エクスパンダウィジェットのプロパティを以下に示します．

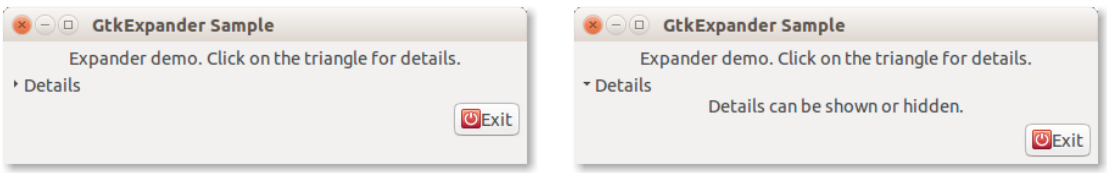


図 7.11 エクスパンダウィジェット

- ウィジェットの展開状態
  エクスパンダウィジェットが展開している（子ウィジェットが表示されている）状態か，そうでないかを設定します．

```
void gtk_expander_set_expanded (GtkExpander *expander,
                                gboolean    expanded);

gboolean gtk_expander_get_expanded (GtkExpander *expander);
```

- ラベル
  ラベルウィジェットのラベルを次の関数で設定します．

```
void gtk_expander_set_label (GtkExpander *expander,
                             const gchar *label);

G_CONST_RETURN gchar*
gtk_expander_get_label (GtkExpander *expander);
```

サンプルプログラム

ソース 7–2–4 にエクスパンダウィジェットのサンプルプログラムを示します．エクスパンダウィジェットはコンテナとして特殊な機能を持っていますが，使うのはとても簡単です．

ソース 7-2-4 エクスパンダウィジェットのサンプルプログラム：gtkexpander-sample.c

```
1  #include <gtk/gtk.h>
2
3  int
4  main (int argc, char *argv[])
5  {
6    GtkWidget *window;
7    GtkWidget *vbox;
8    GtkWidget *hbox;
9    GtkWidget *label;
10   GtkWidget *button;
11   GtkWidget *expander;
12
13   gtk_init (&argc, &argv);
14
15   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
16   gtk_window_set_title (GTK_WINDOW (window), "GtkExpander␣Sample");
17   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
18   g_signal_connect (G_OBJECT (window), "destroy",
19                     G_CALLBACK (gtk_main_quit), NULL);
20
21   vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 5);
22   gtk_container_add (GTK_CONTAINER (window), vbox);
23
24   label =
25     gtk_label_new ("Expander␣demo.␣Click␣on␣the␣triangle␣for␣details.");
26   gtk_box_pack_start (GTK_BOX (vbox), label, FALSE, FALSE, 0);
27
28   expander = gtk_expander_new ("Details");
29   gtk_box_pack_start (GTK_BOX (vbox), expander, FALSE, FALSE, 0);
30   gtk_expander_set_expanded (GTK_EXPANDER (expander), TRUE);
31
32   label = gtk_label_new ("Details␣can␣be␣shown␣or␣hidden.");
33   gtk_container_add (GTK_CONTAINER (expander), label);
34
35   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
36   gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
37
38   button =
39     gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
40   gtk_button_set_label (GTK_BUTTON (button), "Exit");
41   gtk_button_set_always_show_image (GTK_BUTTON (button), TRUE);
42   g_signal_connect (G_OBJECT (button), "clicked",
43                     G_CALLBACK (gtk_main_quit), NULL);
44   gtk_box_pack_end (GTK_BOX (hbox), button, FALSE, FALSE, 0);
45
46   gtk_widget_show_all (window);
47   gtk_main ();
```

```
48
49   return 0;
50 }
```

7.3 入力ウィジェット

この節では，キーボードから文字列を入力する際に使用するウィジェットについて説明します．

7.3.1 エントリ

エントリウィジェット（GtkEntry）は，比較的短い文字列（例えばファイル名）を入力するためのウィジェットです．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkEntry
```

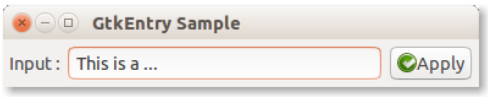


図 7.12 エントリウィジェット

シグナルとコールバック関数

表 7.13 にエントリウィジェットのシグナルを示します．エントリウィジェットには **activate シグナル**のほかにも **copy-clipboard シグナル**や **paste-clipboard シグナル** などのいくつかのシグナルが存在しますが，それほど使用する頻度は高くないので，ここでは省略しています．

activate シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkEntry *entry, gpointer user_data);
```

表 7.13 エントリウィジェットのシグナル

シグナル	説明
activate	エンターキーが押されたときに発生するシグナル

ウィジェットの作成

エントリウィジェットを作成するには関数 gtk_entry_new を使用します．

```
GtkWidget* gtk_entry_new (void);
```

ウィジェットのプロパティ設定

エントリウィジェットのプロパティには次の項目が存在します．そのほかにもいくつかのプロパティが存在します．

- 入力可能かどうか
  エントリに対してユーザが入力可能かどうかを表します．これらの関数の第 1 引数には **GtkEditable** 型の変数を与える必要がありますので，GtkEntry 型の変数を GtkEditable 型の変数に型変換する必要があります．

```
gboolean gtk_editable_get_editable (GtkEditable *editable);

void gtk_editable_set_editable (GtkEditable *editable,
                                gboolean     is_editable);
```

- 外枠の有無
  次の関数で外枠の表示の有無を設定します．

```
gboolean gtk_entry_get_has_frame (GtkEntry *entry);

void gtk_entry_set_has_frame (GtkEntry *entry, gboolean setting);
```

- シークレット文字
  パスワードのように，入力した文字をそのまま表示するのではなく，別の文字（例えば '*'）に置き換えて表示したい場合に，代わりに表示する文字です．

```
void gtk_entry_set_invisible_char (GtkEntry *entry, gunichar ch);

gunichar gtk_entry_get_invisible_char (GtkEntry *entry);
```

- シークレット文字を表示するかどうか
  入力した文字を上記の隠し文字で表示するかどうかを設定します．

```
void gtk_entry_set_visibility (GtkEntry *entry, gboolean visible);

gboolean gtk_entry_get_visibility (GtkEntry *entry);
```

- 入力されているテキスト
  現在エントリウィジェットに入力されている文字列です．

```
void gtk_entry_set_text (GtkEntry *entry, const gchar *text);

G_CONST_RETURN gchar* gtk_entry_get_text (GtkEntry *entry);
```

サンプルプログラム

ソース 7–3–1 にエントリウィジェットのサンプルプログラムを示します．このプログラムは，エントリウィジェットにキーボードから文字列を入力して，エンターキーを押すかボタンを押すと，エントリに入力された文字列を端末に表示します．

ソース 7–3–1 エントリウィジェットのサンプルプログラム：gtkentry-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_entry (GtkEntry *entry, gpointer user_data)
5  {
6    g_print ("%s\n", gtk_entry_get_text (entry));
7  }
8
9  static void
10 cb_button (GtkButton *button, gpointer user_data)
11 {
12   g_print ("%s\n", gtk_entry_get_text (GTK_ENTRY (user_data)));
13 }
14
15 int
16 main (int argc, char *argv[])
17 {
18   GtkWidget *window;
19   GtkWidget *hbox;
20   GtkWidget *label;
21   GtkWidget *entry;
22   GtkWidget *button;
23
24   gtk_init (&argc, &argv);
25
26   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
27   gtk_window_set_title  (GTK_WINDOW (window), "GtkEntry␣Sample");
28   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
29   g_signal_connect (G_OBJECT (window), "destroy",
30                     G_CALLBACK (gtk_main_quit), NULL);
31
32   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
33   gtk_container_add (GTK_CONTAINER (window), hbox);
34
```

```
35   label = gtk_label_new ("Input␣:");
36   gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);
37
38   entry = gtk_entry_new ();
39   g_signal_connect (G_OBJECT (entry), "activate",
40                     G_CALLBACK (cb_entry), NULL);
41   gtk_box_pack_start (GTK_BOX (hbox), entry, TRUE, TRUE, 0);
42
43   button
44     = gtk_button_new_from_icon_name ("emblem-default", GTK_ICON_SIZE_BUTTON);
45   gtk_button_set_label (GTK_BUTTON (button), "Apply");
46   gtk_button_set_always_show_image (GTK_BUTTON (button), TRUE);
47   g_signal_connect (G_OBJECT (button), "clicked",
48                     G_CALLBACK (cb_button), (gpointer) entry);
49   gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
50
51   gtk_widget_show_all (window);
52   gtk_main ();
53
54   return 0;
55 }
```

7.3.2 コンボボックスエントリ

コンボボックステキスト（**GtkComboBoxText**）は，テキスト（エントリ機能付きも可能）のコンボボックスウィジェットです．コンボボックスはあらかじめ設定された項目からある項目を選択するウィジェットですが，コンボボックステキストはエントリ機能を有効にすることで文字列の入力も追加可能なウィジェットです．

コンボボックス（テキスト）を扱うためには **GtkTreeView** ウィジェットの知識が必要となります．より詳しく理解するためには，GtkTreeView ウィジェットの解説（7.6 節，p. 194）を先に読まれることをおすすめします．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkComboBox
                               +----GtkComboBoxText
```

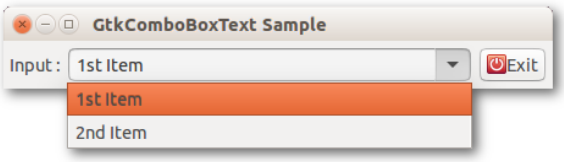


図 7.13 コンボボックステキストウィジェット

シグナルとコールバック関数

表 7.14 にコンボボックスエントリウィジェットのシグナルを示します．このシグナルはコンボボックスに対するシグナルです．**changed シグナル**に対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkComboBox *widget,
                    gpointer     user_data);
```

ウィジェットの作成

コンボボックステキストウィジェットを作成する関数を以下に示します．

- gtk_combo_box_text_new
  シンプルなテキスト用のコンボボックスを作成する関数です．

表 7.14　コンボボックステキストウィジェットのシグナル

シグナル	説明
changed	コンボアイテムが変更したときに発生するシグナル.

```
GtkWidget* gtk_combo_box_text_new (void);
```

- gtk_combo_box_text_new_with_entry
  エントリウィジェットによる入力機能のあるコンボボックステキストを作成する関数です.

```
GtkWidget* gtk_combo_box_text_new_with_entry (void);
```

ウィジェットのプロパティ設定

- 選択されているアイテムのインデックス
  現在選択されているコンボアイテムが何番目のアイテムかを知りたいときには関数 gtk_combo_box_get_active を使用します.

```
gint gtk_combo_box_get_active (GtkComboBox *combo_box);
```

  また，標準値としてあらかじめアイテムを選択（または自動選択）したいときには，関数 gtk_combo_box_set_active を使用します．アイテムのインデックスは 0 から始まります.

```
void
gtk_combo_box_set_active (GtkComboBox *combo_box, gint index_);
```

- 選択されているアイテムの文字列
  現在選択されているコンボアイテムの文字列を知りたいときには，関数 gtk_combo_box_text_get_active_text を使用します.

```
gchar *gtk_combo_box_text_get_active_text (GtkComboBoxText *combo_box);
```

サンプルプログラム

ソース 7–3–3 にコンボボックステキストウィジェットのサンプルプログラムを示します．コンボボックスにはあらかじめ 2 つの文字列が登録されていて，アイテムを選択すると選択したアイテムのインデックスと文字列を端末に表示します.

また，エントリ内に文字列を入力（エンターキーの入力で確定）すると，その文字列が新しいアイテムとして登録されます．そして，アイテム数が 5 を超えた場合には，先頭のアイテムを削除して，新しいアイテムを追加し，アイテム数が常に 5 つになるようにしてあります．この機能は, 22–67 行目で定義したコールバック関数内で実装しています．このコールバック関数はコンボボックステキストに対するものではなく, 子ウィジェットのエントリウィジェットに対するものなので, 99 行目で関数 gtk_bin_get_child を使用してエントリウィジェットを取得して, activate シグナルに対するコールバック関数を設定しています.

ソース 7–3–3 コンボボックステキストウィジェットのサンプルプログラム：gtkcomboboxtext-sample.c

```
1  #include <gtk/gtk.h>
2
3  #define COMBO_LIST_LIMIT 5
4
5  static void
6  cb_combo_changed (GtkComboBox *combo_box,
7                    gpointer     user_data)
8  {
9    gint pos;
10   gchar *text;
11
12   pos = gtk_combo_box_get_active (combo_box);
13   if (pos != -1)
14     {
15       text =
16         gtk_combo_box_text_get_active_text (GTK_COMBO_BOX_TEXT (combo_box));
17       g_print ("Item number is %d (%s)\n", pos, text);
18       g_free (text);
19     }
20 }
21
```

```
22 static void
23 cb_combo_entry_activate (GtkEntry *entry,
24                          gpointer user_data)
25 {
26   GtkComboBoxText *combobox;
27   GtkTreeModel    *model;
28   GtkTreeIter     iter;
29   gchar           *text;
30   gboolean        exist_flag = FALSE;
31   const gchar     *new_text;
32
33   new_text = gtk_entry_get_text (entry);
34   if (!new_text || g_strcmp0 (new_text, "") == 0) return;
35
36   combobox = GTK_COMBO_BOX_TEXT (user_data);
37   model = gtk_combo_box_get_model (GTK_COMBO_BOX (combobox));
38   if (gtk_tree_model_get_iter_first (model, &iter))
39     {
40       do {
41         gtk_tree_model_get (model, &iter, 0, &text, -1);
42         if (g_strcmp0 (new_text, text) == 0)
43           {
44             exist_flag = TRUE;
45           }
46       } while (!exist_flag && gtk_tree_model_iter_next (model, &iter));
47     }
48   if (!exist_flag)
49     {
50       gtk_combo_box_text_append_text (combobox, new_text);
51       int nlists = 0;
52       if (gtk_tree_model_get_iter_first (model, &iter))
53         {
54           do {
55             nlists++;
56           } while (gtk_tree_model_iter_next (model, &iter));
57         }
58       if (nlists > COMBO_LIST_LIMIT)
59         {
60           gtk_tree_model_get_iter_first (model, &iter);
61           gtk_tree_model_get (model, &iter, 0, &text, -1);
62           gtk_combo_box_text_remove (combobox, 0);
63           g_print ("Remove␣the␣first␣item␣label␣'%s'.\n", text);
64         }
65       g_print ("Append␣a␣new␣item␣label␣'%s'.\n", new_text);
66     }
67 }
68
69 int
70 main (int argc, char *argv[])
71 {
72   GtkWidget *window;
73   GtkWidget *hbox;
74   GtkWidget *label;
75   GtkWidget *combo;
76   GtkWidget *button;
77
78   gtk_init (&argc, &argv);
79
80   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
81   gtk_window_set_title  (GTK_WINDOW (window), "GtkComboBoxText␣Sample");
82   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
83   g_signal_connect (G_OBJECT (window), "destroy",
84                     G_CALLBACK (gtk_main_quit), NULL);
85
86   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
87   gtk_container_add (GTK_CONTAINER (window), hbox);
88
89   label = gtk_label_new ("Input␣:");
90   gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);
91
92   combo = gtk_combo_box_text_new_with_entry ();
93   gtk_combo_box_text_append_text (GTK_COMBO_BOX_TEXT (combo), "1st␣Item");
94   gtk_combo_box_text_append_text (GTK_COMBO_BOX_TEXT (combo), "2nd␣Item");
95   gtk_combo_box_set_active (GTK_COMBO_BOX (combo), 0);
96
97   g_signal_connect (G_OBJECT (combo), "changed",
```

```
98                      G_CALLBACK (cb_combo_changed), NULL);
99    g_signal_connect (G_OBJECT (gtk_bin_get_child (GTK_BIN (combo))), "activate",
100                     G_CALLBACK (cb_combo_entry_activate), combo);
101   gtk_box_pack_start (GTK_BOX (hbox), combo, TRUE, TRUE, 0);
102
103   button
104     = gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
105   gtk_button_set_label (GTK_BUTTON (button), "Exit");
106   gtk_button_set_always_show_image (GTK_BUTTON (button), TRUE);
107   g_signal_connect (G_OBJECT (button), "clicked",
108                     G_CALLBACK (gtk_main_quit), NULL);
109   gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
110
111   gtk_widget_show_all (window);
112   gtk_main ();
113
114   return 0;
115 }
```

7.3.3　スピンボタン

スピンボタンウィジェット（**GtkSpinButton**）は，数値を入力するためのウィジェットで，キーボードから入力するエントリウィジェットとボタンウィジェットが並んだ複合ウィジェットです．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkEntry
                   +----GtkSpinButton
```

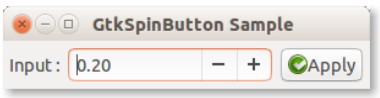


図 7.14　スピンボタンウィジェット

シグナルとコールバック関数

表 7.15 にスピンボタンウィジェットのシグナルを示します．**value-changed シグナル**が発生するのは，エントリウィジェットに数値を直接入力してエンターキーを押したときか，ボタンを押して数値が変化したときです．

value-changed シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．

```
gboolean user_function (GtkSpinButton *spinbutton,
                        gpointer       user_data);
```

表 7.15　スピンボタンウィジェットのシグナル

シグナル	説明
value-changed	数値が変化したときに発生するシグナル．

ウィジェットの作成

スピンボタンを作成するには次の 2 通りの方法があります．

- gtk_spin_button_new
  GtkAdjustment で数値の範囲等の設定をして，関数の引数に与えます．climb_rate はボタンを押したときの数値の増減値です．digits には小数点以下何桁まで表示するかを指定します．

```
GtkWidget* gtk_spin_button_new (GtkAdjustment *adjustment,
```

```
                                    gdouble       climb_rate,
                                    guint         digits);
```

第 1 引数：	アジャストメントウィジェット
第 2 引数：	ボタンによる増減値
第 3 引数：	表示桁数
戻り値：	スピンウィジェット

- gtk_spin_button_new_with_range
  GtkAdjustment の代わりに, 数値の範囲と増減幅を指定してウィジェットを作成する関数です.

```
GtkWidget* gtk_spin_button_new (gdouble min, gdouble max, gdouble step);
```

第 1 引数：	最小値
第 2 引数：	最大値
第 3 引数：	増減幅
戻り値：	スピンウィジェット

ソース 7–3–4 に，関数 gtk_spin_button_new によるスピンボタン作成の例を示します.

ソース 7–3–4 スピンボタンウィジェットの作成

```
1  GtkWidget *spinbutton;
2  GtkAdjustment *adjustment;
3  gdouble   value = 1.0, min = 0.0, max = 100.0;
4  gdouble   step = 0.1, page = 1.0, page_size = 10.0, digits = 1;
5
6  adjustment = gtk_adjustment_new (value, min, max, step, page, page_size);
7  spinbutton = gtk_spin_button_new (adjustment, step, digits);
```

ウィジェットのプロパティ設定

スピンボタンウィジェットのプロパティには次の項目が存在します.

- 数値
  次の関数で指定したり, 取得したりできます.

```
void gtk_spin_button_set_value (GtkSpinButton *spin_button, gdouble value);

gdouble gtk_spin_button_get_value (GtkSpinButton *spin_button);
```

- ボタンを押したときの数値の増減値
  この値は次の関数で指定したり，取得したりできます.

```
void gtk_spin_button_set_increments (GtkSpinButton *spin_button,
                                     gdouble       step,
                                     gdouble       page);
```

第 1 引数：	スピンボタン
第 2 引数：	増減幅
第 3 引数：	ページ増減幅

```
void gtk_spin_button_get_increments (GtkSpinButton *spin_button,
                                     gdouble       *step,
                                     gdouble       *page);
```

サンプルプログラム

ソース 7–3–5 にエントリウィジェットのサンプルプログラムを示します．このプログラムは，エントリウィジェットにキーボードから文字列を入力して，エンターキーを押すかボタンを押すと，エントリに入力された文字列を端末に表示します．

ソース 7–3–5 スピンボタンウィジェットのサンプルプログラム：gtkspinbutton-sample.c

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

static void
cb_value_changed (GtkSpinButton *spinbutton, gpointer user_data)
{
  g_print ("value␣=␣%f\n", gtk_spin_button_get_value (spinbutton));
}

static void
cb_button (GtkButton *button, gpointer user_data)
{
  g_print ("value␣=␣%f\n",
           gtk_spin_button_get_value (GTK_SPIN_BUTTON (user_data)));
}

int
main (int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *hbox;
  GtkWidget *label;
  GtkWidget *spinbutton;
  GtkWidget *button;
  gdouble   min = 0.0, max = 100.0, step = 0.1;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "GtkSpinButton␣Sample");
  gtk_container_set_border_width (GTK_CONTAINER (window), 5);
  gtk_widget_set_size_request (window, 300, -1);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
  gtk_container_add (GTK_CONTAINER (window), hbox);

  label = gtk_label_new ("Input␣:");
  gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);

  spinbutton = gtk_spin_button_new_with_range (min, max, step);
  gtk_spin_button_set_digits (GTK_SPIN_BUTTON (spinbutton), 2);
  gtk_spin_button_set_wrap (GTK_SPIN_BUTTON (spinbutton), TRUE);

  g_signal_connect (G_OBJECT (spinbutton), "value-changed",
                    G_CALLBACK (cb_value_changed), NULL);
  gtk_box_pack_start (GTK_BOX (hbox), spinbutton, TRUE, TRUE, 0);

  button
    = gtk_button_new_from_icon_name ("emblem-default", GTK_ICON_SIZE_BUTTON);
  gtk_button_set_label (GTK_BUTTON (button), "Apply");
  gtk_button_set_always_show_image (GTK_BUTTON (button), TRUE);
  g_signal_connect (G_OBJECT (button), "clicked",
                    G_CALLBACK (cb_button), (gpointer) spinbutton);
  gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);

  gtk_widget_show_all (window);
  gtk_main ();

  return 0;
}
```

7.3.4 テキストビュー

テキストビューウィジェット（GtkTextView）は，複数行にわたるテキストを表示するためのウィジェットです．

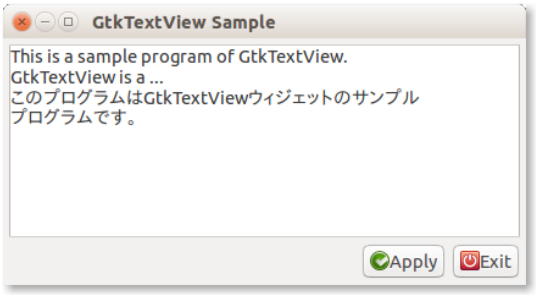


図 7.15 テキストビューウィジェット

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkTextView
```

ウィジェットの作成

テキストビューウィジェットを作成するには，関数 gtk_text_view_new か関数 gtk_text_view_new_with_buffer を使用します．

```
GtkWidget* gtk_text_view_new (void);

GtkWidget* gtk_text_view_new_with_buffer (GtkTextBuffer *buffer);
```

簡単なテキストを表示するだけなら，関数 gtk_text_view_new を使いましょう．自動的に標準のテキストバッファが作成されます．

ウィジェットのプロパティ設定

テキストビューウィジェットで最もよく使用されるのは，ウィジェットへのテキストの設定と取得でしょう．これらは次の関数を用いて行います．

これらの関数を使用するには GtkTextBuffer を取得する必要があります．GtkTextView ウィジェットから GtkTextBuffer を取得するには，関数 gtk_text_view_get_buffer を使用します．

```
GtkTextBuffer* gtk_text_view_get_buffer (GtkTextView *text_view);
```

- gtk_text_buffer_set_text
  テキストは UTF8 エンコーディングでなければいけません．ロケール指定のエンコーディングを UTF8 エンコーディングに変換するには，関数 g_locale_to_utf8 を使用します．変数 len にはテキストの長さをバイト数で指定します．−1 を指定すると終端記号までの文字を指定したことになります．

```
void gtk_text_buffer_set_text (GtkTextBuffer *buffer,
                               const gchar   *text,
                               gint          len);
```

- gtk_text_buffer_get_text
  start から end までの行のテキストを取得します．取得したテキストは UTF8 エンコーディングで新しく領域確保されたテキストです．このテキストを使用しなくなった場合には，関数 g_free によって領域を解放する必要があります．

```
gchar*
gtk_text_buffer_get_text (GtkTextBuffer     *buffer,
                          const GtkTextIter *start,
                          const GtkTextIter *end,
                          gboolean          include_hidden_chars);
```

テキスト位置 GtkTextIter を取得する関数には次のような種類があります．

- gtk_text_buffer_get_iter_at_line
  指定した行番号の GtkTextIter を取得します．行番号は 0 から開始します．

```
void gtk_text_buffer_get_iter_at_line (GtkTextBuffer *buffer,
                                       GtkTextIter   *iter,
                                       gint          line_number);
```

- gtk_text_buffer_get_start_iter
  開始行の GtkTextIter を取得します．

```
void gtk_text_buffer_get_start_iter (GtkTextBuffer *buffer,
                                     GtkTextIter   *iter);
```

- gtk_text_buffer_get_end_iter
  最終行の GtkTextIter を取得します．

```
void gtk_text_buffer_get_end_iter (GtkTextBuffer *buffer,
                                   GtkTextIter   *iter);
```

サンプルプログラム

ソース 7–3–6 にテキストビューウィジェットのサンプルプログラムを示します．テキストビューウィジェットには自由に入力ができるようになっていて，適用ボタンを押すと入力されているすべてのテキストがターミナルに表示されるようになっています．

ソース 7-3-6 テキストビューウィジェットのサンプルプログラム：gtktextview-sample.c

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

static void
set_text (GtkTextView *textview, const gchar *text)
{
  GtkTextBuffer *buffer;

  buffer = gtk_text_view_get_buffer (textview);
  gtk_text_buffer_set_text (buffer, text, -1);
}

static void
print_text (GtkWidget *widget, gpointer user_data)
{
  GtkTextBuffer *buffer;
  GtkTextIter   start, end;
  gchar         *utf8_text;

  buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (user_data));
  gtk_text_buffer_get_start_iter (buffer, &start);
  gtk_text_buffer_get_end_iter (buffer, &end);
  utf8_text = gtk_text_buffer_get_text (buffer, &start, &end, TRUE);
  g_print ("%s\n", utf8_text);
  g_free (utf8_text);
}

int
main (int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *vbox;
  GtkWidget *hbox;
  GtkWidget *scrolledwindow;
  GtkWidget *textview;
  GtkWidget *button;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "GtkTextView␣Sample");
```

```
41   gtk_widget_set_size_request (window, 300, 100);
42   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
43   g_signal_connect (G_OBJECT (window), "destroy",
44                     G_CALLBACK (gtk_main_quit), NULL);
45
46   vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 5);
47   gtk_container_add (GTK_CONTAINER (window), vbox);
48
49   scrolledwindow = gtk_scrolled_window_new (NULL, NULL);
50   gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW
51                                        (scrolledwindow),
52                                        GTK_SHADOW_ETCHED_OUT);
53   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolledwindow),
54                                   GTK_POLICY_AUTOMATIC,
55                                   GTK_POLICY_AUTOMATIC);
56   gtk_box_pack_start (GTK_BOX (vbox), scrolledwindow, TRUE, TRUE, 0);
57
58   textview = gtk_text_view_new ();
59   gtk_container_add (GTK_CONTAINER (scrolledwindow), textview);
60   set_text (GTK_TEXT_VIEW (textview),
61             "This␣is␣a␣sample␣program␣of␣GtkTextView.\n"
62             "GtkTextView␣is␣a␣...\n"
63             "このプログラムはGtkTextViewウィジェットのサンプル\n"
64             "プログラムです。");
65
66   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
67   gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
68
69   button
70     = gtk_button_new_from_icon_name ("application-exit", GTK_ICON_SIZE_BUTTON);
71   gtk_button_set_label (GTK_BUTTON (button), "Exit");
72   gtk_button_set_always_show_image (GTK_BUTTON (button), TRUE);
73   g_signal_connect (G_OBJECT (button), "clicked",
74                     G_CALLBACK (gtk_main_quit), NULL);
75   gtk_box_pack_end (GTK_BOX (hbox), button, FALSE, FALSE, 0);
76
77   button
78     = gtk_button_new_from_icon_name ("emblem-default", GTK_ICON_SIZE_BUTTON);
79   gtk_button_set_label (GTK_BUTTON (button), "Apply");
80   gtk_button_set_always_show_image (GTK_BUTTON (button), TRUE);
81   g_signal_connect (G_OBJECT (button), "clicked",
82                     G_CALLBACK (print_text), (gpointer) textview);
83   gtk_box_pack_end (GTK_BOX (hbox), button, FALSE, FALSE, 0);
84
85   gtk_widget_show_all (window);
86   gtk_main ();
87
88   return 0;
89 }
```

7.4 メニューウィジェット

7.4.1 メニューバー

メニューバーウィジェット（**GtkMenuBar**）は，ウィンドウの上部などに配置して，さまざまな操作を支援するウィジェットです（図 7.16）.

図 7.16　メニューバー

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkMenuShell
                         +----GtkMenuBar
```

ウィジェットの作成

メニューバーの作成には関数 gtk_menu_bar_new を使用します.

```
GtkWidget* gtk_menu_bar_new (void);
```

メニューバーウィジェットは単にメニューを配置するためのウィジェットです．メニューを作成するには，関数 gtk_menu_new でメニューウィジェットを作成し，関数 gtk_menu_item_new などを使って作成したメニューアイテムを，メニューウィジェットに配置します.

```
GtkWidget* gtk_menu_new (void);
```

メニューアイテムの作成

メニューアイテムには大きく分類して 5 種類のメニューアイテムがあります.

1. 普通のメニューアイテム
   ラベルで構成されるメニューアイテムです．メニューアイテムを作成する関数には，次の 3 つの関数があります.
   - gtk_menu_item_new
     ラベルのないメニューアイテムを作成します．あまり使用することはないでしょう.

     ```
     GtkWidget* gtk_menu_item_new (void);
     ```

   - gtk_menu_item_new_with_label
     ラベルのみのメニューアイテムを作成します.

     ```
     GtkWidget* gtk_menu_item_new_with_label (const gchar *label);
     ```

   - gtk_menu_item_new_with_mnemonic
     アクセラレータ機能付きのラベルを持ったメニューアイテムを作成します.

     ```
     GtkWidget* gtk_menu_item_new_with_mnemonic (const gchar *label);
     ```

2. チェックボタンのメニューアイテム
   - gtk_check_menu_item_new
     ラベルなしのチェックボタンメニューアイテムを作成します.

     ```
     GtkWidget* gtk_check_menu_item_new (void);
     ```

- gtk_check_menu_item_new_with_label
  ラベル付きのチェックボタンメニューアイテムを作成します.

```
GtkWidget* gtk_check_menu_item_new_with_label (const gchar *label);
```

- gtk_check_menu_item_new_with_mnemonic
  アクセラレータ機能のあるラベル付きのチェックボタンメニューアイテムを作成します.

```
GtkWidget* gtk_check_menu_item_new_with_mnemonic (const gchar *label);
```

チェックメニューアイテムのチェック状態を設定するには，関数 gtk_check_menu_item_set_active を使用します.

```
void gtk_check_menu_item_set_active (GtkCheckMenuItem *check_menu_item,
                                     gboolean          is_active);
```

3. ラジオボタンのメニューアイテム
   - gtk_radio_menu_item_new
     ラベルなしのラジオボタンメニューアイテムを作成します．新しいグループのメニューアイテムを作成する場合には，引数に NULL を与えます.

```
GtkWidget* gtk_radio_menu_item_new (GSList *group);
```

   - gtk_radio_menu_item_new_from_widget
     ラベルなしのラジオボタンメニューアイテムを作成します．既にあるラジオボタンメニューアイテムを引数に与えることによって，そのアイテムと同じグループのラジオボタンメニューアイテムを作成します.

```
GtkWidget* gtk_radio_menu_item_new_from_widget (GtkRadioMenuItem *group);
```

   - gtk_radio_menu_item_new_with_label
     ラベル付きのラジオボタンメニューアイテムを作成します．新しいグループのメニューアイテムを作成する場合には，引数に NULL を与えます.

```
GtkWidget* gtk_radio_menu_item_new_with_label (GSList      *group,
                                               const gchar *label);
```

   - gtk_radio_menu_item_new_with_label_from_widget
     ラベル付きのラジオボタンメニューアイテムを作成します．既にあるラジオボタンメニューアイテムを引数に与えることによって，そのアイテムと同じグループのラジオボタンメニューアイテムを作成します.

```
GtkWidget*
gtk_radio_menu_item_new_with_label_from_widget (GtkRadioMenuItem *group,
                                                const gchar      *label);
```

   - gtk_radio_menu_item_new_with_mnemonic
     アクセラレータ機能のあるラベル付きのラジオボタンメニューアイテムを作成します.

```
GtkWidget* gtk_radio_menu_item_new_with_mnemonic (GSList      *group,
                                                  const gchar *label);
```

   - gtk_radio_menu_item_new_with_mnemonic_from_widget
     アクセラレータ機能のあるラベル付きのラジオボタンメニューアイテムを作成します．既にあるラジオボタンメニューアイテムを引数に与えることによって，そのアイテムと同じグループのラジオボタンメニューアイテムを作成します.

```
GtkWidget*
gtk_radio_menu_item_new_with_mnemonic_from_widget (GtkRadioMenuItem *group,
                                                   const gchar      *label);
```

4. その他のメニューアイテム
   その他のメニューアイテムには, セパレータがあります.
   - gtk_separator_menu_item_new

```
GtkWidget* gtk_separator_menu_item_new (void);
```

アクセラレータキーの設定

メニューアイテムには，そのメニューアイテムを選択しなくてもその操作を実行できるように，アクセラレータキーが設定されていることがよくあります．アクセラレータキーを設定するには，関数 gtk_widget_add_accelerator を使用します．

```
void gtk_widget_add_accelerator (GtkWidget       *widget,
                                 const gchar     *accel_signal,
                                 GtkAccelGroup   *accel_group,
                                 guint           accel_key,
                                 GdkModifierType accel_mods,
                                 GtkAccelFlags   accel_flags);
```

第 1 引数	アクセラレータキーを設定するウィジェット
第 2 引数	アクセラレータキーが押されたときに発生させるシグナル
第 3 引数	アクセラレータグループ
第 4 引数	アクセラレータキー
第 5 引数	装飾子
第 6 引数	アクセラレータフラグ

アクセラレータキーの設定例をソース 7–4–1 に示します．アクセラレータキー GDK_KEY_O, 装飾子 GDK_CONTROL_MASK を指定することで，CTRL+O（コントロールキーを押しながら o）を押すことでメニューアイテムを選択のと同じ動作が起こります．GtkAccelGroup は，ショートカットキーを一括管理するためのものと考えてください．

ソース 7—4–1 アクセラレータキー設定のサンプル

```
1  GtkWindow     *window;
2  GtkWidget     *item;
3  GtkAccelGroup *accel_group;
4
5  accel_group = gtk_accel_group_new ();
6  gtk_window_add_accel_group (window, accel_group);
7  item = gtk_menu_item_new_with_mnemonic ("_Open");
8  gtk_widget_add_accelerator (item, "activate", accel_group,
9                              GDK_KEY_O, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
```

GdkModifierType は次のように定義されています．

```
typedef enum
{
  GDK_SHIFT_MASK    = 1 << 0,
  GDK_LOCK_MASK     = 1 << 1,
  GDK_CONTROL_MASK  = 1 << 2,
  GDK_MOD1_MASK     = 1 << 3,
  GDK_MOD2_MASK     = 1 << 4,
  GDK_MOD3_MASK     = 1 << 5,
  GDK_MOD4_MASK     = 1 << 6,
  GDK_MOD5_MASK     = 1 << 7,
  GDK_BUTTON1_MASK  = 1 << 8,
  GDK_BUTTON2_MASK  = 1 << 9,
  GDK_BUTTON3_MASK  = 1 << 10,
  GDK_BUTTON4_MASK  = 1 << 11,
  GDK_BUTTON5_MASK  = 1 << 12,
  GDK_MODIFIER_RESERVED_13_MASK = 1 << 13,
   ...
  GDK_MODIFIER_RESERVED_25_MASK = 1 << 25,
  GDK_SUPER_MASK    = 1 << 26,
  GDK_HYPER_MASK    = 1 << 27,
  GDK_META_MASK     = 1 << 28,
  GDK_MODIFIER_RESERVED_29_MASK = 1 << 29,
```

```
  GDK_RELEASE_MASK   = 1 << 30,
  GDK_MODIFIER_MASK = 0x5c001fff
} GdkModifierType;
```

GtkAccelFlags は次のように定義されています．

```
typedef enum
{
  GTK_ACCEL_VISIBLE = 1 << 0,
  GTK_ACCEL_LOCKED  = 1 << 1,
  GTK_ACCEL_MASK    = 0x07
} GtkAccelFlags;
```

ショートカットキーをメニューに表示する場合には，GTK_ACCEL_VISIBLE を指定します．また，設定したキーを固定するためには GTK_ACCEL_LOCKED を指定します．両方設定したい場合には，GTK_ACCEL_VISIBLE | GTK_ACCEL_LOCKED のように論理和で指定するか，GTK_ACCEL_MASK を指定します．

階層的なメニューの作成

1 つのメニューの中にさらに階層的にサブメニューを作成することはよくあります． サブメニューを作成するには，関数 gtk_menu_item_set_submenu を使います．

```
void gtk_menu_item_set_submenu (GtkMenuItem *menu_item,
                                GtkWidget   *submenu);
```

サブメニュー作成の手順は以下のようになります．

1. 親メニューの作成

   ```
   menu = gtk_menu_new ();
   ```

2. メニューアイテムの作成&セット

   ```
   item = gtk_menu_item_new ();
   gtk_menu_shell_append (GTK_MENU_SHELL (menu), item);
   ```

3. サブメニューの作成

   ```
   submenu = gtk_menu_new ();
   ```

4. サブメニューのセット

   ```
   gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), submenu);
   ```

5. サブメニューアイテムの作成&セット

   ```
   subitem = gtk_menu_item_new ();
   gtk_menu_shell_append (GTK_MENU_SHELL (submenu), subitem);
   ```

サンプルプログラム

ソース 7–4–2 に，図 7.16 のメニューを作成するソースを示します．File メニューは，普通のメニューアイテムで構成されています．View メニューは，チェックメニューアイテムとラジオメニューアイテム，サブメニューの例です．

このプログラムではメニューに設定したショートカットキーを親ウィンドウに登録することで，親ウィンドウ上で設定したショートカットキーを有効にしています．ウィンドウ上でショートカットキーを有効にするには，関数 gtk_window_add_accel_group を使用します．

```
void gtk_window_add_accel_group (GtkWindow     *window,
                                 GtkAccelGroup *accel_group);
```

ソース 7-4-2 メニューバーのサンプルプログラム：gtkmenubar-sample.c

```
1 #include <gtk/gtk.h>
2 #include <gdk/gdkkeysyms.h>
3
4 static void
```

```
5  cb_quit (GtkWidget *widget, gpointer data)
6  {
7    gtk_main_quit ();
8  }
9
10 static GtkWidget*
11 create_menu (GtkWindow *window)
12 {
13   GtkWidget     *menubar;
14   GtkWidget     *menu;
15   GtkWidget     *menuitem_top;
16   GtkWidget     *menuitem;
17   GtkAccelGroup *accel_group;
18   GSList        *group = NULL;
19
20   accel_group = gtk_accel_group_new ();
21   gtk_window_add_accel_group (window, accel_group);
22
23   menubar = gtk_menu_bar_new ();
24
25   menuitem_top = gtk_menu_item_new_with_mnemonic ("_File");
26   gtk_menu_shell_append (GTK_MENU_SHELL (menubar), menuitem_top);
27
28   menu = gtk_menu_new ();
29   gtk_menu_item_set_submenu (GTK_MENU_ITEM (menuitem_top), menu);
30   {
31     menuitem = gtk_menu_item_new_with_mnemonic ("_New");
32     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
33     gtk_widget_add_accelerator (menuitem, "activate", accel_group,
34                                 GDK_KEY_N, GDK_CONTROL_MASK,
35                                 GTK_ACCEL_VISIBLE | GTK_ACCEL_LOCKED);
36
37     menuitem = gtk_menu_item_new_with_mnemonic ("_Open");
38     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
39     gtk_widget_add_accelerator (menuitem, "activate", accel_group,
40                                 GDK_KEY_O, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
41
42     menuitem = gtk_menu_item_new_with_mnemonic ("_Save");
43     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
44     gtk_widget_add_accelerator (menuitem, "activate", accel_group,
45                                 GDK_KEY_S, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
46
47     menuitem = gtk_menu_item_new_with_mnemonic ("Save␣_as");
48     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
49     gtk_widget_add_accelerator (menuitem, "activate", accel_group,
50                                 GDK_KEY_S, GDK_CONTROL_MASK | GDK_SHIFT_MASK,
51                                 GTK_ACCEL_VISIBLE);
52
53     menuitem = gtk_separator_menu_item_new ();
54     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
55
56     menuitem = gtk_menu_item_new_with_mnemonic ("_Quit");
57     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
58     gtk_widget_add_accelerator (menuitem, "activate", accel_group,
59                                 GDK_KEY_Q, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
60     g_signal_connect (G_OBJECT(menuitem),
61                       "activate", G_CALLBACK (cb_quit), NULL);
62   }
63   menuitem_top = gtk_menu_item_new_with_mnemonic ("_View");
64   gtk_menu_shell_append (GTK_MENU_SHELL (menubar), menuitem_top);
65   menu = gtk_menu_new ();
66   gtk_menu_item_set_submenu (GTK_MENU_ITEM (menuitem_top), menu);
67   {
68     menuitem = gtk_check_menu_item_new_with_label ("Show␣Hidden␣Folders");
69
70     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
71     gtk_widget_add_accelerator (menuitem, "activate", accel_group,
72                                 GDK_KEY_H, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
73
74     menuitem = gtk_menu_item_new_with_label ("Sort␣by␣...");
75     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
76
77     menu = gtk_menu_new ();
78     gtk_menu_item_set_submenu (GTK_MENU_ITEM (menuitem), menu);
79     {
80       menuitem = gtk_radio_menu_item_new_with_label (group, "File␣type");
```

```
81        gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
82        gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM (menuitem), TRUE);
83
84        group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM (menuitem));
85        menuitem = gtk_radio_menu_item_new_with_label (group, "File␣size");
86        gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
87
88        group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM (menuitem));
89        menuitem = gtk_radio_menu_item_new_with_label (group, "Update␣time");
90        gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
91      }
92    }
93    return menubar;
94  }
95
96  int
97  main (int argc, char *argv[])
98  {
99    GtkWidget *window;
100   GtkWidget *menubar;
101
102   gtk_init (&argc, &argv);
103
104   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
105   gtk_window_set_title (GTK_WINDOW (window), "GtkMenubar␣Sample");
106   gtk_widget_set_size_request (window, 300, -1);
107   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
108   g_signal_connect (G_OBJECT (window), "destroy",
109                     G_CALLBACK (gtk_main_quit), NULL);
110
111   menubar = create_menu (GTK_WINDOW (window));
112   gtk_container_add (GTK_CONTAINER (window), menubar);
113
114   gtk_widget_show_all (window);
115   gtk_main ();
116
117   return 0;
118 }
```

7.4.2 ポップアップメニュー

ポップアップメニュー（**GtkPopupMenu**）は，マウスの右ボタンをクリックしたときに表示されるメニューです（図 7.17）．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkObject
             +----GtkWidget
                   +----GtkContainer
                         +----GtkMenuShell
                               +----GtkMenu
```

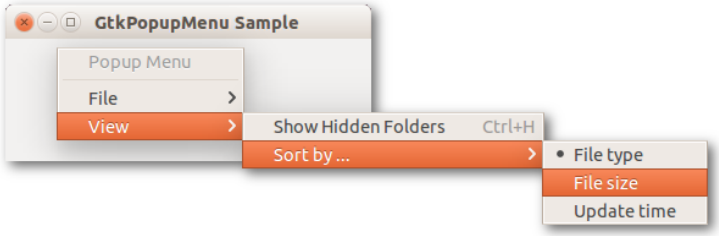


図 7.17 ポップアップメニュー

ウィジェットの作成

ポップアップメニューの作成は，先に説明したメニューの作成と同様です．違いは，メニューバーウィジェットにメニューを配置するのではなく，マウスボタンのクリック等の動作によってメニューを表示する点です．

ポップアップメニューの表示

ポップアップメニューを表示するには，関数 gtk_menu_popup を使用します．

```
void gtk_menu_popup (GtkMenu             *menu,
                     GtkWidget           *parent_menu_shell,
                     GtkWidget           *parent_menu_item,
                     GtkMenuPositionFunc func,
                     gpointer            data,
                     guint               button,
                     guint32             activate_time);
```

第 1 引数には表示するポップアップメニューを指定します．第 2 引数から第 5 引数までは通常 NULL を指定しておけば大丈夫です．第 6 引数は任意の値を与えても動作に違いがありません．通常は 0 でいいでしょう．第 7 引数は，ポップアップメニューを表示する時間を指定します．マウスボタンのクリックイベントに連動してポップアップメニューを表示する場合には，**GdkEventButton** 型の変数のメンバ time を指定します．この値が使用できない場合には，代わりに関数 gtk_get_current_event_time を使用するといいでしょう．

```
guint32 gtk_get_current_event_time (void);
```

サンプルプログラム

ソース 7–4–3 にポップアップメニューの例を示します．ウィンドウ上でマウスの右ボタンをクリックするとポップアップメニューが表示されます．どのボタンがクリックされたかを調べるには，GdkEventButton 型の変数のメンバ button を使用します．この値が 1，2，3 のとき，それぞれマウスの左ボタン，中ボタン，右ボタンに対応します．

ソース 7–4–3 ポップアップメニューのサンプルプログラム：gtkpopupmenu-sample.c

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

static void
cb_quit (GtkWidget *widget, gpointer user_data)
{
  gtk_main_quit ();
}

static gboolean
cb_popup_menu (GtkWidget      *widget,
               GdkEventButton *event,
               gpointer       user_data)
{
  GtkMenu *popupmenu = GTK_MENU (user_data);

  if (event->button == 3) /* 右ボタンクリック */
    {
      gtk_menu_popup (popupmenu, NULL, NULL, NULL, NULL, 0, event->time);
    }
  return FALSE;
}

static GtkWidget*
create_popupmenu (GtkWindow *parent)
{
  GtkWidget     *popupmenu;
  GtkWidget     *menu;
  GtkWidget     *menuitem_top;
  GtkWidget     *menuitem;
  GtkAccelGroup *accel_group;

  accel_group = gtk_accel_group_new ();
  gtk_window_add_accel_group (parent, accel_group);

```

```
36  popupmenu = gtk_menu_new ();
37
38  menuitem_top = gtk_menu_item_new_with_label ("Popup␣Menu");
39  gtk_widget_set_sensitive (menuitem_top, FALSE);
40  gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), menuitem_top);
41
42  menuitem_top = gtk_separator_menu_item_new ();
43  gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), menuitem_top);
44
45  menuitem_top = gtk_menu_item_new_with_mnemonic ("_File");
46  gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), menuitem_top);
47
48  menu = gtk_menu_new ();
49  gtk_menu_item_set_submenu (GTK_MENU_ITEM (menuitem_top), menu);
50  {
51    menuitem = gtk_menu_item_new_with_mnemonic ("_New");
52    gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
53    gtk_widget_add_accelerator (menuitem, "activate", accel_group,
54                                GDK_KEY_N, GDK_CONTROL_MASK,
55                                GTK_ACCEL_VISIBLE | GTK_ACCEL_LOCKED);
56
57    menuitem = gtk_menu_item_new_with_mnemonic ("_Open");
58    gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
59    gtk_widget_add_accelerator (menuitem, "activate", accel_group,
60                                GDK_KEY_O, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
61
62    menuitem = gtk_menu_item_new_with_mnemonic ("_Save");
63    gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
64    gtk_widget_add_accelerator (menuitem, "activate", accel_group,
65                                GDK_KEY_S, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
66
67    menuitem = gtk_menu_item_new_with_mnemonic ("Save␣_as");
68    gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
69    gtk_widget_add_accelerator (menuitem, "activate", accel_group,
70                                GDK_KEY_S, GDK_CONTROL_MASK | GDK_SHIFT_MASK,
71                                GTK_ACCEL_VISIBLE);
72
73    menuitem = gtk_separator_menu_item_new ();
74    gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
75
76    menuitem = gtk_menu_item_new_with_mnemonic ("_Quit");
77    gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
78    gtk_widget_add_accelerator (menuitem, "activate", accel_group,
79                                GDK_KEY_Q, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
80    g_signal_connect (G_OBJECT(menuitem),
81                      "activate", G_CALLBACK (cb_quit), NULL);
82  }
83  menuitem_top = gtk_menu_item_new_with_mnemonic ("_View");
84  gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), menuitem_top);
85  menu = gtk_menu_new ();
86  gtk_menu_item_set_submenu (GTK_MENU_ITEM (menuitem_top), menu);
87  {
88    menuitem = gtk_check_menu_item_new_with_label ("Show␣Hidden␣Folders");
89
90    gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
91    gtk_widget_add_accelerator (menuitem, "activate", accel_group,
92                                GDK_KEY_H, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
93
94    menuitem = gtk_menu_item_new_with_label ("Sort␣by␣...");
95    gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
96
97    menu = gtk_menu_new ();
98    gtk_menu_item_set_submenu (GTK_MENU_ITEM (menuitem), menu);
99    {
100     menuitem = gtk_radio_menu_item_new_with_label (NULL, "File␣type");
101     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
102     gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM (menuitem), TRUE);
103
104     menuitem =
105       gtk_radio_menu_item_new_with_label_from_widget (GTK_RADIO_MENU_ITEM (menuitem),
106                                                       "File␣size");
107     gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
108
109     menuitem =
110       gtk_radio_menu_item_new_with_label_from_widget (GTK_RADIO_MENU_ITEM (menuitem),
111                                                       "Update␣time");
```

```
112       gtk_menu_shell_append (GTK_MENU_SHELL (menu), menuitem);
113     }
114   }
115   return popupmenu;
116 }
117
118 int
119 main (int argc, char *argv[])
120 {
121   GtkWidget *window;
122   GtkWidget *eventbox;
123   GtkWidget *popupmenu;
124
125   gtk_init (&argc, &argv);
126
127   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
128   gtk_window_set_title (GTK_WINDOW (window), "GtkPopupMenu␣Sample");
129   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
130   gtk_widget_set_size_request (window, 300, 100);
131   g_signal_connect (G_OBJECT (window), "destroy",
132                     G_CALLBACK (gtk_main_quit), NULL);
133
134   popupmenu = create_popupmenu (GTK_WINDOW (window));
135   gtk_widget_show_all (popupmenu);
136
137   eventbox = gtk_event_box_new ();
138   gtk_container_add (GTK_CONTAINER (window), eventbox);
139   g_signal_connect (G_OBJECT (eventbox), "button-press-event",
140                     G_CALLBACK (cb_popup_menu), popupmenu);
141
142   gtk_widget_show_all (window);
143   gtk_main ();
144
145   return 0;
146 }
```

7.4.3　GtkBuilder を使ったメニュー作成

ここでは, 2.7 節でも紹介した, GtkBuilder を使ったメニューの作成方法について解説します.

GtkBuilder を使ったメニュー作成の流れ

2.7.1 節でも説明しましたが, もう一度ここで GtkBuilder を使ったメニュー作成の流れをまとめます. 手順 3 と 4 については, 2.7 節で説明したことと変わりませんので, これ以降の節で手順 1 と 2 の部分について更に詳しい説明をしていきます.

1. XML 形式でメニュー構成を記述
2. **GActionEntry** 構造体でメニューアイテムに対するコールバック関数などを記述
3. 関数 g_action_map_add_action_entries を用いてコールバックをアプリケーションに登録
4. GtkBuilder を用いて XML 形式で記述したメニュー構成からメニューを作成し, そのメニューをアプリケーション上に配置

メニュー構成アイテムの定義

メニューアイテムの構成は XML 形式で記述し, 以下のように <interface> タグから始まり, <menu></menu> タグ内に記述します.

```
<interface>
  <menu id='appmenu'>
  </menu>
</interface>
```

そして, メニューアイテムは <item></item> で記述します. メニューアイテムの属性は <attribute> タグで指定し, 属性の種類を name=' ' という形で指定します. 属性の種類は以下のようになります.

表 7.16 メニューアイテムの属性

属性	説明
label	メニューに表示するラベル
action	GActionEntry 構造体の要素と関連付けるための文字列
accel	アクセラレータキー
icon	メニューに表示するアイコン
target	メニューアイテムのコールバック関数に与えるパラメータ

メニューアイテムには通常のメニューアイテム, トグルメニューアイテム, ラジオメニューアイテムのようにそれぞれ異なる機能を持ったメニューアイテムが存在しますが, XML 形式のメニュー構成定義では, メニューアイテムを定義する際に, それがどの機能を持ったメニューアイテムであるかを記述しません. メニューアイテムの種類は, GActionEntry 構造体による記述によって指定することになります.

メニューアイテムの記述の例を以下に示します.

- 通常のアイテム

```
<item>
  <attribute name='label'>New</attribute>
  <attribute name='action'>app.new</attribute>
  <attribute name='accel'>&lt;Control&gt;n</attribute>
  <attribute name='icon'>/usr/share/icons/.../document-new.svg</attribute>
</item>
```

- トグルアイテム
  トグルアイテムの定義は通常のアイテムの定義と全く同じです.

```
<item>
  <attribute name='label'>Show Hidden Folder</attribute>
  <attribute name='action'>app.toggle_item</attribute>
</item>
```

- ラジオアイテム
  ラジオアイテムは複数のアイテムの中からどれかを選択する形式ですので, 複数のアイテムに対して同一の action 属性を定義します. そして, どのアイテムが選択されたかどうかを判別できるように, target 属性にそのアイテム固有の数値や文字列を指定しておきます.

```
<item>
  <attribute name='label'>File Type</attribute>
  <attribute name='action'>app.radio_item</attribute>
  <attribute name='target'>type</attribute>
</item>
<item>
  <attribute name='label'>File Size</attribute>
  <attribute name='action'>app.radio_item</attribute>
  <attribute name='target'>size</attribute>
</item>
```

ラジオアイテムの target 属性には上記の例で示した文字列の他に数値（整数, 実数）を指定することもできます. ただし, 数値を指定する場合にはその型を明示的に示す必要があります.

整数を target 属性に指定する場合は以下の例のように, type='i' を追加します（i は integer の i）. また, 実数を指定する場合には, type='d' を追加します.

```
<item>
  <attribute name='label'>File Type</attribute>
  <attribute name='action'>app.radio_item</attribute>
  <attribute name='target' type='i'>1</attribute>
</item>
<item>
  <attribute name='label'>File Size</attribute>
  <attribute name='action'>app.radio_item</attribute>
  <attribute name='target' type='i'>2</attribute>
</item>
```

GActionEntry 構造体の設定

GActionEntry 構造体は次のように定義されています.

```
struct GActionEntry {
  const gchar *name;
  void (* activate) (GSimpleAction *action,
                     Gvariant      *parameter,
                     gpointer      user_data);
  const gchar *parameter_type;
  const gchar *state;
  void (* change_state) (GSimpleAction *action,
                         GVariant      *value,
                         gpointer      user_data);
};
```

それぞれのメンバの説明は次のとおりです.

1. name ... メニューアイテム定義（attribute の action 属性）と関連付けするための文字列
2. activate ... メニューアイテムが選択されたときに呼び出されるコールバック関数
3. parameter_type ... パラメータタイプ
4. state ... 状態
5. change_state ... メニューアイテムの状態が変わったときに呼び出されるコールバック関数

ラジオアイテムに対しては, parameter_type と state を適切に設定する必要があります. メニュー定義中の attribute の target 属性のタイプと parameter_type の関係を以下にまとめます.

表 7.17　parameter_type の種類

値	target 属性のタイプ
"s"	文字列（type 指定はなし）
"i"	整数（type='i'）
"d"	実数（type='d'）

state の値は, トグルアイテムであれば, "true" もしくは "false" を指定します. "true" を指定した場合にはトグルアイテムがオンになった状態になります. また, ラジオアイテムの場合は, 初期状態で選択するアイテムの target 属性で指定した値を与えます. 以下に target 属性が文字列の場合と整数の場合の指定の例を示します.

target 属性が文字列の場合にはその文字列をシングルクォートで括る必要がありますので注意が必要です.

文字列の場合:

```
static GActionEntry entries[] = {
  {"radio_item", cb_radio, "s", "'type'", NULL},
};
```

整数の場合:

```
static GActionEntry entries[] = {
  {"radio_item", cb_radio, "i", "1", NULL},
};
```

コールバック関数

GtkBuilder を使ってメニューを作成した場合, トグルアイテムのオン, オフや, ラジオアイテムの選択されているアイテムの表示の切り替えがじ自動的に行われません. そのため, コールバック関数内で明示的に状態の切り替えを行う必要があります.

状態の切り替えには関数 g_simple_action_set_state や関数 g_action_change_state を使用します. 具体的な実装方法は, ソース 7–4–4 の 25 行目と, 37–41 行目を参考にしてください.

サンプルプログラム

ソース 7–4–2 と同じメニューを GtkBuilder で実現するプログラムのソースコードをソース 7–4–4 に示します. また, ソース 7–4–5 にメニュー定義ファイルの内容を示します. ICONPATH の部分は実際には具体的な絶対パスを記述しています. 図 7.18 に実行結果を示します. 図 7.16 と異なるのは, メニューアイテムにアイコンが表示されている点です.

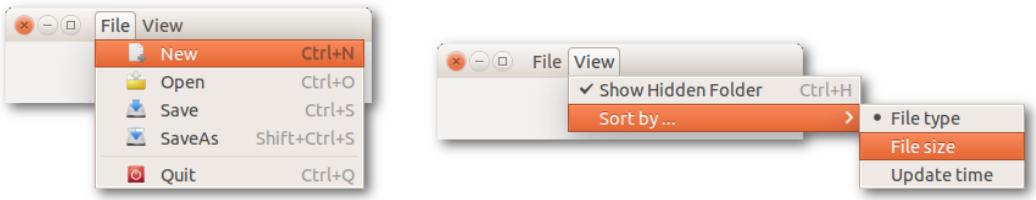


図 7.18 GtkBuilder を使用したメニュー

ソース 7–4–4 GtkBuilder を使用したメニューのサンプルプログラム : menu_by_gtkbuilder.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_new (GSimpleAction   *action,
5          GVariant        *paramter,
6          gpointer        user_data)
7  {
8    g_print ("New␣menu␣item␣is␣selected.\n");
9  }
10
11 static void
12 cb_quit (GSimpleAction  *action,
13           GVariant       *parameter,
14           gpointer       user_data)
15 {
16   GApplication *app = G_APPLICATION (user_data);
17   g_application_quit (app);
18 }
19
20 static void
21 cb_radio (GSimpleAction *action,
22           GVariant      *parameter,
```

```
23            gpointer      user_data)
24 {
25   g_simple_action_set_state (action, parameter);
26   g_print ("state␣=␣%s\n", g_variant_get_string (parameter, NULL));
27 }
28
29 static void
30 cb_toggle (GSimpleAction        *action,
31            GVariant             *parameter,
32            gpointer             user_data)
33 {
34   GVariant *state;
35   gboolean new_state;
36
37   state = g_action_get_state (G_ACTION(action));
38   new_state = !g_variant_get_boolean (state);
39   g_action_change_state (G_ACTION(action),
40                          g_variant_new_boolean (new_state));
41   g_variant_unref (state);
42 }
43
44 static GActionEntry entries[] = {
45   {"new", cb_new, NULL, NULL, NULL},
46   {"open", NULL, NULL, NULL, NULL},
47   {"save", NULL, NULL, NULL, NULL},
48   {"save-as", NULL, NULL, NULL, NULL},
49   {"quit", cb_quit, NULL, NULL, NULL},
50   {"toggle_item", cb_toggle, NULL, "true", NULL},
51   {"radio_item", cb_radio, "s", "'File␣type'", NULL},
52 };
53
54 static void cb_activate (GApplication   *app,
55                          gpointer       user_data)
56 {
57   GtkWidget *window;
58
59   window = gtk_application_window_new (GTK_APPLICATION (app));
60   gtk_widget_set_size_request (window, 300, 50);
61
62   GtkBuilder *builder;
63   builder = gtk_builder_new ();
64   gtk_builder_add_from_file (builder, "./menu.ui", NULL);
65   GMenuModel *menubar;
66   menubar = (GMenuModel *) gtk_builder_get_object (builder, "appmenu");
67   gtk_application_set_menubar (GTK_APPLICATION (app), menubar);
68
69   gtk_widget_show_all (window);
70 }
71
72 int
73 main (int argc, char *argv[])
74 {
75   GtkApplication *app;
76
77   app = gtk_application_new ("test.gtk.menu", 0);
78   g_action_map_add_action_entries (G_ACTION_MAP (app),
79                                    entries, G_N_ELEMENTS (entries),
80                                    app);
81   g_signal_connect (app, "activate", G_CALLBACK(cb_activate), NULL);
82   g_application_run (G_APPLICATION(app), 0, NULL);
83
84   return 0;
85 }
```

ソース 7–4–5 メニュー定義ファイル: menu.ui

```
1 <interface>
2   <menu id='appmenu'>
3     <submenu>
4       <attribute name='label'>File</attribute>
5       <section>
6         <item>
7           <attribute name='label'>New</attribute>
8           <attribute name='action'>app.new</attribute>
```

```
9           <attribute name='accel'>&lt;Control&gt;n</attribute>
10          <attribute name='icon'>ICONPATH/document-new.svg</attribute>
11        </item>
12        <item>
13          <attribute name='label'>Open</attribute>
14          <attribute name='action'>app.open</attribute>
15          <attribute name='accel'>&lt;Control&gt;o</attribute>
16          <attribute name='icon'>ICONPATH/document-open.svg</attribute>
17        </item>
18        <item>
19          <attribute name='label'>Save</attribute>
20          <attribute name='action'>app.save</attribute>
21          <attribute name='accel'>&lt;Control&gt;s</attribute>
22          <attribute name='icon'>ICONPATH/document-save.svg</attribute>
23        </item>
24        <item>
25          <attribute name='label'>Save as</attribute>
26          <attribute name='action'>app.save-as</attribute>
27          <attribute name='accel'>&lt;Control&gt;&lt;Shift&gt;s</attribute>
28          <attribute name='icon'>ICONPATH/document-save-as.svg</attribute>
29        </item>
30      </section>
31      <section>
32        <item>
33          <attribute name='label'>Quit</attribute>
34          <attribute name='action'>app.quit</attribute>
35          <attribute name='accel'>&lt;Control&gt;q</attribute>
36          <attribute name='icon'>ICONPATH/system-shutdown.svg</attribute>
37        </item>
38      </section>
39    </submenu>
40    <submenu>
41      <attribute name='label'>View</attribute>
42      <section>
43        <item>
44          <attribute name='label'>Show Hidden Folder</attribute>
45          <attribute name='action'>app.toggle_item</attribute>
46          <attribute name='accel'>&lt;Control&gt;h</attribute>
47        </item>
48        <submenu>
49          <attribute name='label'>Sort by ...</attribute>
50          <section>
51            <item>
52              <attribute name='label'>File type</attribute>
53              <attribute name='action'>app.radio_item</attribute>
54              <attribute name='target'>File type</attribute>
55            </item>
56            <item>
57              <attribute name='label'>File size</attribute>
58              <attribute name='action'>app.radio_item</attribute>
59              <attribute name='target'>File size</attribute>
60            </item>
61            <item>
62              <attribute name='label'>Update time</attribute>
63              <attribute name='action'>app.radio_item</attribute>
64              <attribute name='target'>Update time</attribute>
65            </item>
66          </section>
67        </submenu>
68      </section>
69    </submenu>
70  </menu>
71 </interface>
```

7.5 ダイアログ

ダイアログには，ユーザがレイアウトを比較的自由に設定できる汎用の GtkDialog から，特殊な用途に使用する GtkMessageDialog などまで用意されています．ここでは，さまざまなダイアログを紹介します．

7.5.1 ダイアログボックス

GtkDialog は，ダイアログのなかでも最も汎用性が高いウィジェットで，ユーザが比較的自由にレイアウトすることが可能です．この後紹介するダイアログは，このウィジェットがベースになっています．

図 7.19　ダイアログ

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkWindow
                               +----GtkDialog
```

ウィジェットの作成

GtkDialog ウィジェットを作成するには次の 2 つの関数を使います．

- gtk_dialog_new
  関数 gtk_dialog_new は空のダイアログを作成します．ダイアログにメッセージやボタンを配置するにはユーザがレイアウトして配置する必要があります．

```
GtkWidget* gtk_dialog_new (void);
```

- gtk_dialog_new_with_buttons
  この関数はダイアログの作成を簡便化する引数をとります．GtkDialogFlags ではダイアログのいくつかのパラメータを指定します．

```
GtkWidget*
gtk_dialog_new_with_buttons (const gchar     *title,
                             GtkWindow       *parent,
                             GtkDialogFlags  flags,
                             const gchar     *first_button_text,
                             GtkResponseType response_type,
                             ...);
```

第 1 引数	タイトル
第 2 引数	親ウィンドウ
第 3 引数	ダイアログフラグ
第 4 引数	ボタンラベル
第 5 引数	レスポンスタイプ
戻り値	ダイアログ

GtkDialogFlags は次のように定義されています．
また，ボタンの指定は第 4 引数からはボタンラベルとレスポンス ID を組にして指定します．この関数の引数は NULL で終わらなければいけません．この関数を使用したダイアログの作成例を示します．

表 7.18 GtkDialogFlags の値

値	説明
GTK_DIALOG_MODAL	モーダルダイアログを作成します
GTK_DIALOG_DESTROY_WITH_PARENT	親ウィジェットが閉じたときにダイアログも閉じる
GTK_DIALOG_USE_HEADER_BAR	ヘッダバーを使用する

```
GtkWidget *dialog
  = gtk_dialog_new_with_buttons ("My dialog",
                                 main_app_window,
                                 GTK_DIALOG_MODAL | GTK_DIALOG_DESTROY_WITH_PARENT,
                                 "_OK", GTK_RESPONSE_ACCEPT,
                                 "_Cancel", GTK_RESPONSE_REJECT,
                                 NULL);
```

レスポンス ID は次のように定義されています．

```
typedef enum
{
  GTK_RESPONSE_NONE         = -1,
  GTK_RESPONSE_REJECT       = -2,
  GTK_RESPONSE_ACCEPT       = -3,
  GTK_RESPONSE_DELETE_EVENT = -4,
  GTK_RESPONSE_OK           = -5,
  GTK_RESPONSE_CANCEL       = -6,
  GTK_RESPONSE_CLOSE        = -7,
  GTK_RESPONSE_YES          = -8,
  GTK_RESPONSE_NO           = -9,
  GTK_RESPONSE_APPLY        = -10,
  GTK_RESPONSE_HELP         = -11
} GtkResponseType;
```

指定したレスポンス ID は，対応するボタンがクリックされたときに関数 gtk_dialog_run の戻り値として返ります．

```
gint gtk_dialog_run (GtkDialog *dialog);
```

サンプルプログラム

ソース 7–5–1 にダイアログウィジェットのサンプルプログラムを示します．このプログラムを実行すると，図 7.20 左のウィンドウが表示されます．Save ボタンを押すと，図 7.20 右のダイアログが表示されます．OK か Cancel のどちらのボタンを押したかに応じて，ターミナルにどのレスポンスが返ってきたかを表示します．

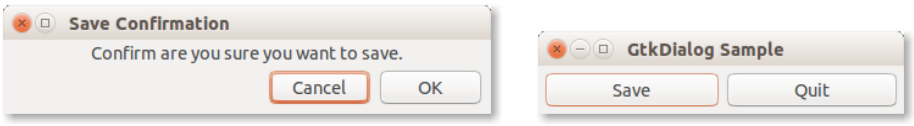


図 7.20 ダイアログウィジェットのサンプルプログラム

ソース 7-5-1 ダイアログウィジェットのサンプルプログラム：gtkdialog-sample.c

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

static void
cb_button (GtkButton *button, gpointer user_data)
{
  GtkWidget *dialog;
  GtkWidget *parent;
  GtkWidget *content_area;
  GtkWidget *label;
  gint       response;

  parent = GTK_WIDGET (user_data);
  dialog = gtk_dialog_new_with_buttons ("Save Confirmation",
                                        GTK_WINDOW(parent),
                                        GTK_DIALOG_MODAL |
                                        GTK_DIALOG_DESTROY_WITH_PARENT,
                                        "_Cancel", GTK_RESPONSE_CANCEL,
                                        "_OK", GTK_RESPONSE_OK,
                                        NULL);
  content_area = gtk_dialog_get_content_area (GTK_DIALOG (dialog));
  label = gtk_label_new ("Confirm are you sure you want to save.");
  gtk_container_add (GTK_CONTAINER (content_area), label);
  gtk_widget_set_size_request (dialog, 400, -1);
  gtk_widget_show_all (dialog);

  response = gtk_dialog_run (GTK_DIALOG (dialog));
  if (response == GTK_RESPONSE_OK)
    {
      g_print ("OK button was pressed.\n");
    }
  else if (response == GTK_RESPONSE_CANCEL)
    {
      g_print ("Cancel button was pressed.\n");
    }
  else
    {
      g_print ("Another response was recieved.\n");
    }
  gtk_widget_destroy (dialog);
}

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

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title  (GTK_WINDOW (window), "GtkDialog Sample");
  gtk_widget_set_size_request (window, 300, -1);
  gtk_container_set_border_width (GTK_CONTAINER (window), 5);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
  gtk_container_add (GTK_CONTAINER (window), hbox);

  button = gtk_button_new_with_label ("Save");
  g_signal_connect (G_OBJECT (button), "clicked",
                    G_CALLBACK (cb_button), (gpointer) window);
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);

  button = gtk_button_new_with_label ("Quit");
  g_signal_connect (G_OBJECT (button), "clicked",
                    G_CALLBACK (gtk_main_quit), NULL);
  gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
```

```
72
73    gtk_widget_show_all (window);
74    gtk_main ();
75
76    return 0;
77 }
```

7.5.2 メッセージダイアログボックス

メッセージダイアログ（**GtkMessageDialog**）では，情報ダイアログ，警告ダイアログ，質問ダイアログ，エラーダイアログの 4 つのダイアログを作成できます．さらに GtkButtonsType によってボタンのレイアウトを簡単に設定できます．

(a) 情報ダイアログ　(b) 警告ダイアログ

(c) 質問ダイアログ　(d) エラーダイアログ

図 7.21　メッセージダイアログ

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkWindow
                               +----GtkDialog
                                     +----GtkMessageDialog
```

ウィジェットの作成

GtkMessageDialog ウィジェットを作成するには次の 2 つの関数を使用します．

- gtk_mesage_dialog_new
  ダイアログの設定（GtkDialogFlags），メッセージタイプ（GtkMessageType），ボタンタイプ（GtkButtonsType），メッセージを指定することで，簡単なダイアログを作成します（図 7.21 を参照）．

```
GtkWidget* gtk_message_dialog_new (GtkWindow      *parent,
                                   GtkDialogFlags flags,
                                   GtkMessageType type,
                                   GtkButtonsType buttons,
                                   const gchar    *message_format,
                                   ...);
```

第 1 引数	親ウィンドウ
第 2 引数	ダイアログフラグ
第 3 引数	メッセージタイプ
第 4 引数	ボタンタイプ
第 5 引数	メッセージフォーマット
戻り値	メッセージダイアログ

GtkMessageType は表 7.19 のように定義されています．

表 7.19　メッセージダイアログの種類

メッセージタイプ	説明
GTK_MESSAGE_INFO	情報ダイアログを作成する（図 7.21(a)）
GTK_MESSAGE_WARNING	警告ダイアログを作成する（図 7.21(b)）
GTK_MESSAGE_QUESTION	質問ダイアログを作成する（図 7.21(c)）
GTK_MESSAGE_ERROR	エラーダイアログを作成する（図 7.21(d)）

GtkButtonsType は表??のように定義されています．

表 7.20　メッセージダイアログのボタンの種類

ボタンタイプ	説明
GTK_BUTTON_NONE	ボタンなし
GTK_BUTTON_OK	OK ボタン
GTK_BUTTON_CLOSE	CLOSE ボタン
GTK_BUTTON_CANCEL	CANSEL ボタン
GTK_BUTTON_YES_NO	YES ボタンと NO ボタン
GTK_BUTTON_OK_CANCEL	OK ボタンと CANCEL ボタン

- gtk_mesage_dialog_new_with_markup
  メッセージを，マークアップしたテキストで指定できるメッセージダイアログです．

```
GtkWidget*
gtk_message_dialog_new_with_markup (GtkWindow      *parent,
                                    GtkDialogFlags flags,
                                    GtkMessageType type,
                                    GtkButtonsType buttons,
                                    const gchar    *message_format,
                                    ...);
```

第 1 引数	親ウィンドウ
第 2 引数	ダイアログフラグ
第 3 引数	メッセージタイプ
第 4 引数	ボタンタイプ
第 5 引数	メッセージフォーマット
戻り値	メッセージダイアログ

ウィジェットのプロパティ設定

関数 gtk_message_dialog_set_markup を使ってマークアップされたテキストを設定できます．

```
void gtk_message_dialog_set_markup (GtkMessageDialog *message_dialog,
                                    const gchar      *str);
```

第 1 引数	メッセージダイアログ
第 2 引数	マークアップテキスト

サンプルプログラム

ソース 7–5–2 にメッセージダイアログウィジェットのサンプルプログラムを示します．押したボタンに応じてメッセージダイアログが表示されます（図 7.22）．

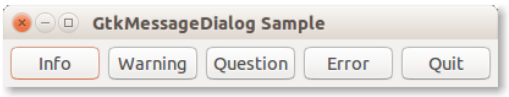


図 7.22 メッセージダイアログのサンプルプログラム

ソース 7–5–2 メッセージダイアログウィジェットのサンプルプログラム：gtkmessagedialog-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  show_dialog (GtkButton *button, gpointer user_data)
5  {
6    GtkWidget      *dialog;
7    GtkWidget      *window;
8    GtkButtonsType btype[] = {GTK_BUTTONS_CLOSE,
9                              GTK_BUTTONS_OK_CANCEL,
10                             GTK_BUTTONS_YES_NO,
11                             GTK_BUTTONS_OK};
12   GtkMessageType mtype = (GtkMessageType) user_data;
13   gchar          *string[] = {"GTK_MESSAGE_INFO",
14                               "GTK_MESSAGE_WARNING",
15                               "GTK_MESSAGE_QUESTION",
16                               "GTK_MESSAGE_ERROE"};
17   gint           result;
18
19   window = GTK_WIDGET(g_object_get_data (G_OBJECT (button), "window"));
20   dialog =
21     gtk_message_dialog_new (GTK_WINDOW (window),
22                             GTK_DIALOG_MODAL |
23                             GTK_DIALOG_DESTROY_WITH_PARENT,
24                             mtype,
25                             btype[mtype],
26                             "This␣dialog␣is␣a␣GtkMessageDialog␣sample\n"
27                             "(GtkMessageType␣=␣%s)",
28                             string[mtype]);
29   result = gtk_dialog_run (GTK_DIALOG (dialog));
30
31   switch (result)
32     {
33     case GTK_RESPONSE_OK:
34       g_print ("GTK_RESPONSE_OK␣is␣recieved.\n");
35       break;
36     case GTK_RESPONSE_CANCEL:
37       g_print ("GTK_RESPONSE_CANCEL␣is␣recieved.\n");
38       break;
39     case GTK_RESPONSE_CLOSE:
40       g_print ("GTK_RESPONSE_CLOSE␣is␣recieved.\n");
41       break;
42     case GTK_RESPONSE_YES:
43       g_print ("GTK_RESPONSE_YES␣is␣recieved.\n");
44       break;
45     case GTK_RESPONSE_NO:
46       g_print ("GTK_RESPONSE_NO␣is␣recieved.\n");
47       break;
48     default:
49       g_print ("Another␣response␣is␣recieved.\n");
50       break;
51     }
52   gtk_widget_destroy (dialog);
53 }
54
55 int
56 main (int argc, char *argv[])
57 {
58   GtkWidget *window;
59   GtkWidget *hbox;
```

```
60   GtkWidget *button;
61   gchar     *label[] = {"Info", "Warning", "Question", "Error"};
62   int        n;
63
64   gtk_init (&argc, &argv);
65
66   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
67   gtk_window_set_title (GTK_WINDOW (window), "GtkMessageDialog␣Sample");
68   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
69   g_signal_connect (G_OBJECT (window), "destroy",
70                     G_CALLBACK (gtk_main_quit), NULL);
71
72   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
73   gtk_box_set_homogeneous (GTK_BOX (hbox), TRUE);
74   gtk_container_add (GTK_CONTAINER (window), hbox);
75
76   for (n = 0; n < 4; n++)
77     {
78       button = gtk_button_new_with_label (label[n]);
79       g_object_set_data (G_OBJECT (button), "window", (gpointer) window);
80       g_signal_connect (G_OBJECT (button), "clicked",
81                         G_CALLBACK (show_dialog), GINT_TO_POINTER (n));
82       gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
83     }
84   button = gtk_button_new_with_label ("Quit");
85   g_signal_connect (G_OBJECT (button), "clicked",
86                     G_CALLBACK (gtk_main_quit), NULL);
87   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
88
89   gtk_widget_show_all (window);
90   gtk_main ();
91
92   return 0;
93 }
```

7.5.3　ファイル選択ダイアログ

ファイル選択ダイアログ（**GtkFileChooser**）は，ファイルの選択を行うダイアログです（図 7.23）.

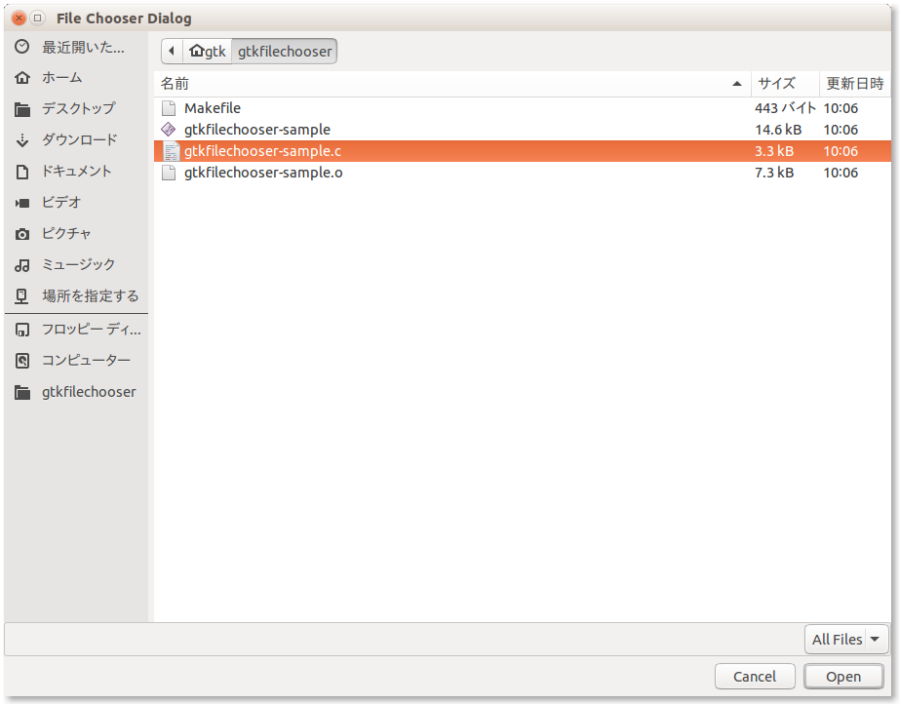


図 7.23　ファイル選択ダイアログ

オブジェクトの階層構造

```
GInterface
 +----GtkFileChooser
```

ウィジェットの作成

ファイル選択ダイアログを作成するには関数 gtk_file_chooser_dialog_new を使用します．

```
GtkWidget*
gtk_file_chooser_dialog_new (const gchar          *title,
                             GtkWidget            *parent,
                             GtkFileChooserAction action,
                             const gchar          *first_button_text,
                             ...);
```

引数に与える情報は，関数のプロトタイプ宣言だけではわかりにくいので例を挙げて説明します．まず，第 1 引数と第 2 引数には，ダイアログのタイトルとそのダイアログを呼び出す親となるウィジェットを与えます．そして第 3 引数の **GtkFileChooserAction** には，表 7.21 に示すように 4 通りの選択肢があります．これは作成するダイアログが，ファイルを開くために使用するのか，ファイルを保存するために使用するのかといったダイアログの種類を指定する引数です．

表 7.21 GtkFileChooserAction の値

値	説明
GTK_FILE_CHOOSER_ACTION_OPEN	ファイルを開くダイアログを選択する．
GTK_FILE_CHOOSER_ACTION_SAVE	ファイルを保存するダイアログを選択する．
GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER	ディレクトリを選択するダイアログを選択する．
GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER	ディレクトリを作成するダイアログを選択する．

第 4 引数からはダイアログに表示するボタンを指定します．このための引数には「ボタンのラベル」と「ボタンの種類」を組として与えます．ボタンの種類は GtkResponseType から指定します．

そして引数の最後には，終端の印として NULL を与えます．

以下の例は，Open ボタンと Cancel ボタンを持つ，ファイルを開くためのダイアログを作成するものです．

```
dialog = gtk_file_chooser_dialog_new ("File Open Dialog",
                                      GTK_WIDGET(parent),
                                      GTK_FILE_CHOOSER_ACTION_OPEN,
                                      "_Cancel",
                                      GTK_RESPONSE_CANCE,
                                      "_Open",
                                      GTK_RESPONSE_ACCEPT,
                                      NULL);
```

ウィジェットのプロパティ設定

ファイル選択ダイアログの設定項目を以下に示します．

- ファイル名
  関数 gtk_file_chooser_get_filename によって，エントリに入力されているファイル名を取得できます．また，関数 gtk_file_chooser_set_filename によって，指定したファイル名をエントリに設定できます．

```
gchar* gtk_file_chooser_get_filename (GtkFileChooser *chooser);

gboolean gtk_file_chooser_set_filename (GtkFileChooser *chooser,
                                        const gchar    *filename);
```

  また同様の関数に，gtk_file_chooser_get_uri や gtk_file_chooser_set_uri があります．これらの関数は，ローカルなファイル名の代わりに URI の形式でファイル名を取得したりします．例えばローカルなファイルであれば，“file:///” で始まる URI となります．

```
gchar* gtk_file_chooser_get_uri (GtkFileChooser *chooser);

gboolean gtk_file_chooser_set_uri (GtkFileChooser *chooser,
                                   const gchar    *filename);
```

複数のファイルを選択できる場合には，関数 gtk_file_chooser_get_filenames や関数 gtk_file_chooser_get_uris を使用すると，選択した複数のファイル名を取得できます．選択したファイル名は単方向リストとして取得されます．

```
GSList* gtk_file_chooser_get_filenames (GtkFileChooser *chooser);

GSList* gtk_file_chooser_get_uris (GtkFileChooser *chooser);
```

- フォルダ名
  現在の開いているフォルダ名を取得したり設定したりするには，関数 gtk_file_chooser_get_current_folder と関数 gtk_file_chooser_set_current_folder を使用します．ファイル名と同様に，URI 用の関数として関数 gtk_file_chooser_get_current_folder_uri や関数 gtk_file_chooser_set_current_folder_uri があります．

```
gchar*
gtk_file_chooser_get_current_folder (GtkFileChooser *chooser);

gboolean
gtk_file_chooser_set_current_folder (GtkFileChooser *chooser,
                                     gchar          *filename);

gchar*
gtk_file_chooser_get_current_folder_uri (GtkFileChooser *chooser);

gboolean
gtk_file_chooser_set_current_folder_uri (GtkFileChooser *chooser,
                                         gchar          *uri);
```

- ファイルフィルタ
  GtkFileChooser では，ダイアログに表示するファイルをフィルタリングできます．フィルタを設定することにより，ダイアログの目的に応じて必要なファイルのみを表示させることができます．
  フィルタの扱いは GtkFileFilter を使用します．ここでは GtkFileFilter の詳細については省略しますが，フィルタを作成してダイアログに登録する手順は以下のようになります．
  1. フィルタの作成
     関数 gtk_file_filter_new で新規のフィルタを作成します．

```
GtkFileFilter* gtk_file_filter_new (void);
```

  2. ダイアログに表示するフィルタ名の設定
     関数 gtk_file_filter_set_name でダイアログに表示するフィルタ名を設定します．

```
void gtk_file_filter_set_name (GtkFileFilter *filter,
                               const gchar   *name);
```

  3. フィルタリング規則の設定
     作成したフィルタがどのようなファイルを表示するのか，フィルタリング設定を行います．フィルタリング設定の関数には以下に示すような関数が用意されています．詳細は省略しますが，具体的な使用方法はソース 7–5–3 を参考にしてください．

```
void gtk_file_filter_add_pattern (GtkFileFilter *filter,
                                  const gchar   *pattern);

void gtk_file_filter_add_mime_type (GtkFileFilter *filter,
                                    const gchar   *mime_type);

void gtk_file_filter_add_pixbuf_formats (GtkFileFilter *filter);
```

  4. ダイアログへの登録
     関数 gtk_file_chooser_add_filter で，フィルタをダイアログに登録します．

```
void gtk_file_chooser_add_filter (GtkFileChooser *chooser,
                                  GtkFileFilter  *filter);
```

- 上書き保存の確認の有無
  ダイアログの種類として GTK_FILE_CHOOSER_ACTION_SAVE を指定している場合に, 選択されたファイルが既に存在するときに上書き確認のダイアログを表示するかどうかを設定します．関数 gtk_file_chooser_set_do_overwrite_confirmation の第 2 引数に TRUE を指定すると，上書き確認のダイアログが表示されるようになります．また，現在の設定を取得するには，関数 gtk_file_chooser_get_do_overwrite_confirmation を使用します．

```
void gtk_file_chooser_set_do_overwrite_confirmation
     (GtkFileChooser *chooser,
      gboolean        do_overwrite_confirmation);

gboolean gtk_file_chooser_get_do_overwrite_confirmation
         (GtkFileChooser *chooser);
```

- 隠しファイルの表示の有無
  関数 gtk_file_chooser_set_show_hidden の第 2 引数に TRUE を指定すると，ピリオドで始まる隠しファイルも表示するようになります．また，関数 gtk_file_chooser_get_show_hidden を使用することで現在の設定を取得できます．

```
void gtk_file_chooser_set_show_hidden (GtkFileChooser *chooser,
                                       gboolean        show_hidden);

gboolean gtk_file_chooser_get_show_hidden (GtkFileChooser *chooser);
```

- 複数ファイルの選択の有無
  関数 gtk_file_chooser_set_select_multiple の第 2 引数に TRUE を指定すると，ダイアログで複数のファイルを選択できるようになります．関数 gtk_file_chooser_get_select_multiple を使用することで，現在の設定を取得できます．

```
void
gtk_file_chooser_set_select_multiple (GtkFileChooser *chooser,
                                      gboolean select_multiple);

gboolean gtk_file_chooser_get_select_multiple (GtkFileChooser *chooser);
```

サンプルプログラム

ソース 7–5–3 にファイル選択ダイアログのサンプルプログラムを示します．プログラムを実行した画面で，Select ボタンを押すと図 7.23 のようなファイルを選択するダイアログが表示されます．

9 行目から 78 行目までのダイアログの動作部分の詳細について説明します．まず 28–35 行目の関数で，ファイルを開くためのファイル選択ダイアログを作成します．

36–49 行目でフィルタを設定しています．ここでは，すべてのファイルを表示するフィルタと，JPEG 画像と PNG 画像を表示するフィルタをそれぞれ設定しています．すべてのファイルを表示するフィルタの設定には，関数 gtk_file_filter_add_pattern を使用して，パターンに “*” を指定しています．また JPEG 画像フォーマットのためのフィルタ設定には，関数 gtk_file_filter_add_mime_type を使用しています．

さらにこのサンプルプログラムでは何もしていませんが，51–52 行目でフィルタが選択されたときに発生するシグナル notify::filter に対するコールバック関数を設定しています．

ソース 7–5–3 ファイル選択ダイアログのサンプルプログラム：gtkfilechooser-sample.c

```
 1 #include <gtk/gtk.h>
 2
 3 static void
 4 filter_changed (GtkFileChooserDialog *dialog, gpointer user_data)
 5 {
 6   g_print ("File␣filter␣changed.\n");
 7 }
 8
 9 static void
10 cb_button (GtkButton *button, gpointer user_data)
11  {
```

```
12   GtkWidget            *dialog;
13   GtkWidget            *parent;
14   GtkEntry             *entry;
15   GtkFileFilter        *filter;
16   GtkFileChooserAction action[] = {GTK_FILE_CHOOSER_ACTION_OPEN,
17                                    GTK_FILE_CHOOSER_ACTION_SAVE,
18                                    GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER,
19                                    GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER};
20   gint  response;
21   gchar *filename;
22   gchar *folder;
23
24   parent
25     = GTK_WIDGET (g_object_get_data (G_OBJECT (user_data), "parent"));
26   entry = GTK_ENTRY (user_data);
27
28   dialog = gtk_file_chooser_dialog_new ("File␣Chooser␣Dialog",
29                                         GTK_WINDOW (parent),
30                                         action[0],
31                                         "_Cancel",
32                                         GTK_RESPONSE_CANCEL,
33                                         "_Open",
34                                         GTK_RESPONSE_ACCEPT,
35                                         NULL);
36   filter = gtk_file_filter_new ();
37   gtk_file_filter_set_name (filter, "All␣Files");
38   gtk_file_filter_add_pattern (filter, "*");
39   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dialog), filter);
40
41   filter = gtk_file_filter_new ();
42   gtk_file_filter_set_name (filter, "JPEG");
43   gtk_file_filter_add_mime_type (filter, "image/jpeg");
44   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dialog), filter);
45
46   filter = gtk_file_filter_new ();
47   gtk_file_filter_set_name (filter, "PNG");
48   gtk_file_filter_add_mime_type (filter, "image/png");
49   gtk_file_chooser_add_filter (GTK_FILE_CHOOSER (dialog), filter);
50
51   g_signal_connect (dialog, "notify::filter",
52                     G_CALLBACK (filter_changed), NULL);
53
54   gtk_widget_show_all (dialog);
55
56   response = gtk_dialog_run (GTK_DIALOG (dialog));
57   if (response == GTK_RESPONSE_ACCEPT)
58     {
59       filename
60         = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
61       folder
62         = gtk_file_chooser_get_current_folder (GTK_FILE_CHOOSER (dialog));
63       g_print ("%s\n", folder);
64       g_free (folder);
65
66       gtk_entry_set_text (entry, filename);
67       g_free (filename);
68     }
69   else if (response == GTK_RESPONSE_CANCEL)
70     {
71       g_print ("Cancel␣button␣was␣pressed.\n");
72     }
73   else
74     {
75       g_print ("Another␣response␣was␣recieved.\n");
76     }
77   gtk_widget_destroy (dialog);
78  }
79
80 int
81 main (int argc, char *argv[])
82 {
83   GtkWidget *window;
84   GtkWidget *hbox;
85   GtkWidget *label;
86   GtkWidget *entry;
87   GtkWidget *button;
```

```
88
89    gtk_init (&argc, &argv);
90
91    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
92    gtk_window_set_title  (GTK_WINDOW (window), "GtkFileChooser␣Sample");
93    gtk_container_set_border_width (GTK_CONTAINER (window), 5);
94    g_signal_connect (G_OBJECT (window), "destroy",
95                      G_CALLBACK (gtk_main_quit), NULL);
96
97    hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
98    gtk_container_add (GTK_CONTAINER (window), hbox);
99
100   label = gtk_label_new ("File␣:");
101   gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);
102
103   entry = gtk_entry_new ();
104   gtk_box_pack_start (GTK_BOX (hbox), entry, TRUE, TRUE, 0);
105   g_object_set_data (G_OBJECT (entry), "parent", (gpointer) window);
106
107   button = gtk_button_new_with_label ("Select");
108   g_signal_connect (G_OBJECT (button), "clicked",
109                     G_CALLBACK (cb_button), (gpointer) entry);
110   gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
111
112   gtk_widget_show_all (window);
113   gtk_main ();
114
115   return 0;
116 }
```

7.5.4 アバウトダイアログ

アバウトダイアログ（**GtkAboutDialog**）は，バージョン 2.6 から追加されたウィジェットです（図 7.24）．GtkAboutDialog はプログラムの名前やバージョン，作者等の情報を表示するダイアログです．

図 7.24 アバウトダイアログ

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkBin
                         +----GtkWindow
                               +----GtkDialog
                                     +----GtkAboutDialog
```

ウィジェットの作成

GtkAboutDialog ウィジェットを作成するには次の関数を使用します．

```
GtkWidget* gtk_about_dialog_new (void);
```

ウィジェットの設定

アバウトダイアログに表示可能な代表的な項目と，関連する関数を以下に示します．

- プログラム名

```
void gtk_about_dialog_set_program_name (GtkAboutDialog *about,
                                        const gchar    *name);

const gchar* gtk_about_dialog_get_program_name (GtkAboutDialog *about);
```

- バージョン番号

```
void gtk_about_dialog_set_version (GtkAboutDialog *about,
                                   const gchar    *version);

const gchar* gtk_about_dialog_get_version (GtkAboutDialog *about);
```

- プログラム作成者

```
void gtk_about_dialog_set_authors (GtkAboutDialog *about,
                                   const gchar    **authors);

const gchar* const *
gtk_about_dialog_get_authors (GtkAboutDialog *about);
```

- ドキュメント作成者

```
void
gtk_about_dialog_set_documenters (GtkAboutDialog *about,
                                  const gchar    **documenters);

const gchar* const *
gtk_about_dialog_get_documenters (GtkAboutDialog *about);
```

- メッセージ翻訳者

```
void
gtk_about_dialog_set_translator_credits (GtkAboutDialog *about,
                                         const gchar *translator_credits);

const gchar*
gtk_about_dialog_get_translator_credits (GtkAboutDialog *about);
```

- プログラムのコメント

```
void gtk_about_dialog_set_comments (GtkAboutDialog *about,
                                    const gchar    *comments);

const gchar* gtk_about_dialog_get_comments (GtkAboutDialog *about);
```

- コピーライト

```
void gtk_about_dialog_set_copyright (GtkAboutDialog *about,
                                     const gchar    *copyright);

const gchar*
gtk_about_dialog_get_copyright (GtkAboutDialog *about);
```

- ライセンス

```
void gtk_about_dialog_set_license_type (GtkAboutDialog *about,
                                        GtkLicense     license_type);

GtkLicense gtk_about_dialog_get_license_type (GtkAboutDialog *about);
```

表 7.22 GtkLicenseType の値

値	説明
GTK_LICENSE_UNKONW	指定しません
GTK_LICENSE_CUSTOM	ユーザーが独自に指定します
GTK_LICENSE_GPL_2_0	The GNU General Public License バージョン 2.0 以降
GTK_LICENSE_GPL_3_0	The GNU General Public License バージョン 3.0 以降
GTK_LICENSE_LGPL_2_1	The GNU Lesser General Public License バージョン 2.1 以降
GTK_LICENSE_LGPL_3_0	The GNU Lesser General Public License バージョン 3.0 以降
GTK_LICENSE_BSD	The BSD standard license
GTK_LICENSE_MIT_X11	The MIT/X11 standard license
GTK_LICENSE_ARTISTIC	The Artistic License, version 2.0
GTK_LICENSE_GPL_2_0_ONLY	The GNU General Public License バージョン 2.0
GTK_LICENSE_GPL_3_0_ONLY	The GNU General Public License バージョン 3.0
GTK_LICENSE_LGPL_2_1_ONLY	The GNU Lesser General Public License バージョン 2.1
GTK_LICENSE_LGPL_3_0_ONLY	The GNU Lesser General Public License バージョン 3.0

- Web サイト

```
void gtk_about_dialog_set_website (GtkAboutDialog *about,
                                   const gchar    *website);

const gchar* gtk_about_dialog_get_website (GtkAboutDialog *about);
```

- ロゴ

```
void gtk_about_dialog_set_logo (GtkAboutDialog *about,
                                GdkPixbuf      *logo);

GdkPixbuf* gtk_about_dialog_get_logo (GtkAboutDialog *about);
```

サンプルプログラム

ソース 7–5–4 にアバウトダイアログのサンプルプログラムを示します.

ソース 7–5–4 アバウトダイアログのサンプルプログラム：gtkaboutdialog-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_show_dialog (GtkWidget *widget, gpointer user_data)
5  {
6    GtkWidget      *dialog;
7    GtkAboutDialog *about;
8    GdkPixbuf      *pixbuf;
9    const gchar    *authors[] = {"GTK", NULL};
10   const gchar    *documenters[] = {"GTK", NULL};
11   const gchar    *translators = "GTK";
12
13   dialog = gtk_about_dialog_new ();
14   about = GTK_ABOUT_DIALOG (dialog);
15   gtk_about_dialog_set_program_name (about, "GtkAboutDialog-Sample");
16   gtk_about_dialog_set_authors (about, authors);
17   gtk_about_dialog_set_documenters (about, documenters);
18   gtk_about_dialog_set_translator_credits (about, translators);
19   gtk_about_dialog_set_version (about, "1.0.0");
20   gtk_about_dialog_set_copyright (about, "Copyright␣(C)␣2016");
21   gtk_about_dialog_set_license_type (about, GTK_LICENSE_GPL_3_0);
22   gtk_about_dialog_set_comments (about,
23                                  "This␣is␣a␣GtkAboutDialog␣sample␣program.");
24   gtk_about_dialog_set_website (about, "file:///...");
25
```

```
26   pixbuf = gdk_pixbuf_new_from_file ("gnome-tigert.png", NULL);
27   gtk_about_dialog_set_logo (about, pixbuf);
28   g_object_unref (pixbuf);
29
30   gtk_container_set_border_width (GTK_CONTAINER (dialog), 5);
31
32   gtk_widget_show_all (dialog);
33 }
34
35 int
36 main (int argc, char *argv[])
37 {
38   GtkWidget *window;
39   GtkWidget *button;
40
41   gtk_init (&argc, &argv);
42
43   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
44   gtk_window_set_title (GTK_WINDOW (window), "GtkAboutDialog-Sample");
45   gtk_widget_set_size_request (window, 300, -1);
46   g_signal_connect (G_OBJECT (window), "destroy",
47                     G_CALLBACK (gtk_main_quit), NULL);
48
49   button = gtk_button_new_with_label ("Show␣About␣Dialog");
50   gtk_container_add (GTK_CONTAINER (window), button);
51   g_signal_connect (G_OBJECT (button), "clicked",
52                     G_CALLBACK (cb_show_dialog), NULL);
53
54   gtk_widget_show_all (window);
55   gtk_main ();
56
57   return 0;
58 }
```

7.6 ツリービュー

ツリービューウィジェット（GtkTreeView）には，データの表示に 2 種類の使い方があります．1 つ目は，リストデータを表示する使い方です．2 つ目は，ツリーデータを表示する使い方です．ツリービューウィジェットは若干複雑な設計になっているので，ここではあまり深い部分には立ち入らずに，具体的な使用方法に注目して具体例を挙げながら説明していくことにします．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkTreeView
```

ウィジェットの作成

ツリービューウィジェットを作成するには，関数 gtk_tree_view_new を使用します．表示するデータ形式によらず，基本となるツリービューウィジェットはこの関数によって作成します．

```
GtkWidget* gtk_tree_view_new (void);
```

作成したウィジェットに対して，モデル（GtkTreeModel）を設定することで，表示するデータ形式を決定します．モデルが先に作成されている場合は，関数 gtk_tree_view_new_with_model を使って，ウィジェットを作成すると同時にモデルをセットできます．

```
GtkWidget* gtk_tree_view_new_with_model (GtkTreeModel *model);
```

関数 gtk_tree_view_new でウィジェットを作成した場合には，関数 gtk_tree_view_set_model によりモデルをセットします．

```
void gtk_tree_view_set_model (GtkTreeView  *tree_view,
                              GtkTreeModel *model);
```

このモデルに GtkListStore を用いるか GtkTreeStore を用いるかで，リストデータを表示するのかツリーデータを表示するのかが決まります．

7.6.1 簡単なリスト表示

リストデータを表示する手順は次のようになります．この流れを簡単な例（図 7.25）をもとに解説します．ソースコードはソース 7–6–1 のようになります．

1. リストモデルの作成
2. 列の作成と属性の割り当て
3. リストデータの追加

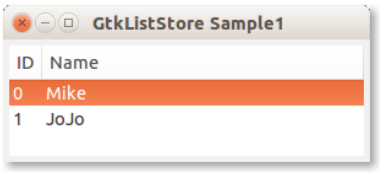


図 7.25 リストの作成

リストモデルの作成（44–45 行目）

リスト表示のためのモデルには GtkListStore を使用します．モデル作成には，関数 gtk_list_store_new を使います．

```
GtkListStore* gtk_list_store_new (gint n_columns, ...);
```

まず第 1 引数にはデータの項目数を指定し，残りの引数にはそれぞれの項目のデータ型を G_TYPE_BOOLEAN, G_TYPE_INT，G_TYPE_STRING，GDK_TYPE_PIXBUF などのマクロで指定します．データの項目数は表示するデータ数だけではなく，セルの色などの属性なども含めた項目数を表します．これについては，次の例で詳しく説明します．

列の作成と属性の設定（49–54 行目）

モデルを作成したら，次は各列の作成と属性の割り当てを行います．列を扱う場合には，GtkTreeViewColumn と GtkCellRenderer を組み合わせて使用します．GtkTreeViewColumn は列全体を扱う変数で，GtkCellRenderer は列データの描画を扱う変数と考えてください．

GtkTreeViewColumn の作成には，関数 gtk_tree_view_column_new を使用します．

```
GtkTreeViewColumn* gtk_tree_view_column_new (void);
```

GtkCellRenderer の作成は，列データにどんな種類のデータを描画するかによって，以下の関数を選択的に使用します．

- gtk_cell_renderer_text_new
  数値や文字列を表示する場合にこの関数を使用します．

  ```
  GtkCellRenderer* gtk_cell_renderer_text_new (void);
  ```

- gtk_cell_renderer_toggle_new
  TRUE，FALSE のデータをチェックボタンで表示する場合にこの関数を使用します．

  ```
  GtkCellRenderer* gtk_cell_renderer_toggle_new (void);
  ```

- gtk_cell_renderer_pixbuf_new
  アイコンデータを表示する場合にこの関数を使用します．

  ```
  GtkCellRenderer* gtk_cell_renderer_pixbuf_new (void);
  ```

関数 gtk_tree_view_column_new で GtkTreeViewColumn を作成した場合には，関数 gtk_tree_view_column_set_title，gtk_tree_view_column_pack_start，gtk_tree_view_column_set_attributes を呼び出す必要があります．

1. 列のタイトル設定（51 行目）

   ```
   void
   gtk_tree_view_column_set_title (GtkTreeViewColumn *tree_column,
                                   const gchar       *title);
   ```

2. GtkTreeViewColumn への GtkCellRenderer の配置（52 行目）

```
void
gtk_tree_view_column_pack_start (GtkTreeViewColumn *tree_column,
                                 GtkCellRenderer   *cell,
                                 gboolean          expand);
```

3. 列属性の割り当て（53–54 行目）
   関数 gtk_tree_view_column_set_attributes で，列に対して複数の属性を割り当てることができます．第 3 引数からは属性と列番号を対にして与えて，引数の最後には NULL を与えます．また，これまでの設定を保持したまま新しい設定を追加したい場合には，関数 gtk_tree_view_column_add_attribute を使用します．

```
void
gtk_tree_view_column_set_attributes(GtkTreeViewColumn *tree_column,
                                    GtkCellRenderer   *cell_renderer,
                                    const gchar       *attribute,
                                    gint              column,
                                    ...);
```

第 1 引数：　列
第 2 引数：　セルレンダラー
第 3 引数：　属性（表 7.23 を参照）
第 4 引数：　列番号

```
void
gtk_tree_view_column_add_attribute (GtkTreeViewColumn *tree_column,
                                    GtkCellRenderer *cell_renderer,
                                    const gchar     *attribute,
                                    gint            column);
```

リストデータを表示するために最低限使用する属性を表 7.23 に示します．このほかにも列に表示するデータの種類によってさまざまな属性があります．

表 7.23　列の属性

属性	説明
text	数値や文字列を表示する列に対して設定する．
active	ブール値をチェックボタンで表示する列に対して設定する．
pixbuf	アイコンを表示する列に対して設定する．

関数 gtk_tree_view_column_new を使用して列を作成した場合は，上記で説明したようにいくつかの関数を呼び出す必要がありますが，関数 gtk_tree_view_column_new_with_attributes を使用すると上記の手順を一括して行うことができます（58–60 行目）．

```
GtkTreeViewColumn*
gtk_tree_view_column_new_with_attributes (const gchar     *title,
                                          GtkCellRenderer *cell,
                                          const gchar     *attribute,
                                          gint            column,
                                          ...);
```

列のツリービューへの追加（55，61 行目）

作成した列（GtkTreeViewColumn）をツリービューに追加するには，関数 gtk_tree_view_append_column を使います．関数の戻り値として新しい列を追加した後の列数が返ります．

```
gint gtk_tree_view_append_column (GtkTreeView       *tree_view,
                                  GtkTreeViewColumn *column);
```

リストデータの追加（27–33 行目）

リストデータの追加は次の手順で行います．

1. 行の追加（29 行目）
   行の追加は関数 gtk_list_store_append を用います．

```
void gtk_list_store_append (GtkListStore *list_store,
                            GtkTreeIter  *iter);
```

2. 追加した行へのデータ登録（30–32 行目）
   関数 gtk_list_store_append で取得した GtkTreeIter は，行データへのアクセスポイントとなります．この GtkTreeIter を関数 gtk_list_store_set の引数に与えて，データを登録します．第 3 引数から列番号とデータを対にして指定し，最後の引数は −1 で終わります．

```
void gtk_list_store_set (GtkListStore *list_store,
                         GtkTreeIter  *iter,
                         ...);
```

ソース 7–6–1 リストの作成とデータの登録：gtkliststore-sample1.c

```
1  #include <gtk/gtk.h>
2
3  enum
4  {
5    COLUMN_ID,
6    COLUMN_NAME,
7    N_COLUMNS
8  };
9
10 typedef struct _ListData
11 {
12   guint id;
13   gchar *name;
14 } ListData;
15
16 static ListData data[] = { {0, "Mike"}, {1, "JoJo"} };
17
18 static void
19 add_data (GtkTreeView *treeview)
20 {
21   GtkListStore *store;
22   GtkTreeIter  iter;
23   int          n;
24
25   store = GTK_LIST_STORE (gtk_tree_view_get_model (treeview));
26
27   for (n = 0; n < sizeof (data) / sizeof (data[0]); n++)
28     {
29       gtk_list_store_append (store, &iter);
30       gtk_list_store_set (store, &iter,
31                           COLUMN_ID,   data[n].id,
32                           COLUMN_NAME, data[n].name, -1);
33     }
34 }
35
36 static GtkWidget*
37 create_list_model (void)
38 {
39   GtkWidget         *treeview;
40   GtkListStore      *liststore;
41   GtkCellRenderer   *renderer;
42   GtkTreeViewColumn *column;
43
44   liststore = gtk_list_store_new (N_COLUMNS,
45                                   G_TYPE_UINT, G_TYPE_STRING);
46   treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL (liststore));
47   g_object_unref (liststore);
48
49   renderer = gtk_cell_renderer_text_new ();
50   column = gtk_tree_view_column_new ();
51   gtk_tree_view_column_set_title (column, "ID");
```

```
52   gtk_tree_view_column_pack_start (column, renderer, FALSE);
53   gtk_tree_view_column_set_attributes (column, renderer,
54                                        "text", COLUMN_ID, NULL);
55   gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
56
57   renderer = gtk_cell_renderer_text_new ();
58   column =
59     gtk_tree_view_column_new_with_attributes ("Name", renderer,
60                                               "text", COLUMN_NAME, NULL);
61   gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
62
63   return treeview;
64 }
65
66 int
67 main (int argc, char *argv[])
68 {
69   GtkWidget *window;
70   GtkWidget *treeview;
71
72   gtk_init (&argc, &argv);
73
74   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
75   gtk_window_set_title (GTK_WINDOW (window), "GtkListStore␣Sample1");
76   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
77   g_signal_connect (G_OBJECT (window), "destroy",
78                     G_CALLBACK (gtk_main_quit), NULL);
79   gtk_widget_set_size_request (window, 300, 100);
80
81   treeview = create_list_model ();
82   gtk_container_add (GTK_CONTAINER (window), treeview);
83
84   add_data (GTK_TREE_VIEW (treeview));
85
86   gtk_widget_show_all (window);
87   gtk_main ();
88
89   return 0;
90 }
```

7.6.2　さらに細かな列属性の設定

ここでは列属性のさらに細かな設定方法について紹介します．先の例では表示するデータに対する最低限の属性しか設定しませんでしたが，実際には前景色（文字の色）や背景色，表示位置，アンダーライン等さまざまな属性を設定することが可能です．

次の例（ソース 7–6–2）では，先ほどの例に対して背景色とアンダーラインの属性を追加する方法を紹介します（図 7.26）．

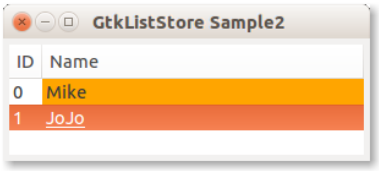


図 7.26　さまざまな列属性の設定

モデルの作成（35–43 行目）

今回は，表示する ID と名前データのほかに，列の背景色とアンダーラインの種類の 2 つの属性値をデータとして持つので，G_TYPE_STRING と G_TYPE_UINT のデータを追加しています．

属性の設定（74–77 行目）

関数 gtk_tree_view_column_add_attribute を使って，2 つ目の列に対して **background 属性**と **underline 属性**を追加しています．devhelp によるとそれぞれの属性は表 7.24 のようになっています．また，PangoUnderline は表 7.25 のように定義されています．

表 7.24 文字列データに対する属性

属性	データ型	アクセス
background	gchar*	Write
underline	PangoUnderline	Read / Write

表 7.25 PangoUnderline の種類

値	説明
PANGO_UNDERLINE_NONE	アンダーラインなし
PANGO_UNDERLINE_SINGLE	1 本線のアンダーライン
PANGO_UNDERLINE_DOUBLE	2 本線のアンダーライン
PANGO_UNDERLINE_LOW	1 本線のアンダーライン．キーボードアクセラレータのような 1 文字のキャラクターに使う
PANGO_UNDERLINE_ERROR	波線のアンダーライン

ソース 7–6–2 列属性の設定：gtkliststore-sample2.c

```
1  #include <gtk/gtk.h>
2
3  enum
4  {
5    COLUMN_ID,
6    COLUMN_NAME,
7    COLUMN_NAME_COLOR,
8    COLUMN_NAME_LINE,
9    N_COLUMNS
10 };
11
12 typedef struct _ListData
13 {
14   guint          id;
15   gchar          *name;
16   gchar          *color;
17   PangoUnderline linestyle;
18 } ListData;
19
20 static ListData data[] =
21 {
22   {0, "Mike", "Orange", PANGO_UNDERLINE_NONE},
23   {1, "JoJo", "Red",    PANGO_UNDERLINE_SINGLE}
24 };
25
26 static void
27 add_data (GtkTreeView *treeview)
28 {
29   GtkListStore *store;
30   GtkTreeIter  iter;
31   int          n;
32
33   store = GTK_LIST_STORE (gtk_tree_view_get_model (treeview));
34
35   for (n = 0; n < sizeof (data) / sizeof (data[0]); n++)
36     {
37       gtk_list_store_append (store, &iter);
38       gtk_list_store_set (store, &iter,
39                           COLUMN_ID,          data[n].id,
40                           COLUMN_NAME,        data[n].name,
41                           COLUMN_NAME_COLOR, data[n].color,
42                           COLUMN_NAME_LINE,  data[n].linestyle, -1);
43     }
44 }
45
46 static GtkWidget*
47 create_list_model (void)
48 {
```

```
49   GtkWidget          *treeview;
50   GtkListStore       *liststore;
51   GtkCellRenderer    *renderer;
52   GtkTreeViewColumn *column;
53
54   liststore = gtk_list_store_new (N_COLUMNS,
55                                   G_TYPE_UINT,
56                                   G_TYPE_STRING,
57                                   G_TYPE_STRING,
58                                   G_TYPE_UINT);
59   treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL (liststore));
60   g_object_unref (liststore);
61
62   renderer = gtk_cell_renderer_text_new ();
63   column =
64     gtk_tree_view_column_new_with_attributes ("ID", renderer,
65                                               "text", COLUMN_ID, NULL);
66   gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
67
68   renderer = gtk_cell_renderer_text_new ();
69   column =
70     gtk_tree_view_column_new_with_attributes ("Name", renderer,
71                                               "text", COLUMN_NAME, NULL);
72   gtk_tree_view_column_add_attribute (column, renderer,
73                                       "background", COLUMN_NAME_COLOR);
74   gtk_tree_view_column_add_attribute (column, renderer,
75                                       "underline", COLUMN_NAME_LINE);
76
77   gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
78
79   return treeview;
80 }
81
82 int
83 main (int argc, char *argv[])
84 {
85   GtkWidget *window;
86   GtkWidget *treeview;
87
88   gtk_init (&argc, &argv);
89
90   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
91   gtk_window_set_title (GTK_WINDOW (window), "GtkListStore␣Sample2");
92   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
93   g_signal_connect (G_OBJECT (window), "destroy",
94                     G_CALLBACK (gtk_main_quit), NULL);
95   gtk_widget_set_size_request (window, 300, 100);
96
97   treeview = create_list_model ();
98   gtk_container_add (GTK_CONTAINER (window), treeview);
99
100  add_data (GTK_TREE_VIEW (treeview));
101
102  gtk_widget_show_all (window);
103  gtk_main ();
104
105  return 0;
106 }
```

7.6.3　リストデータの削除

選択されているデータを削除する方法を，例を使って説明します（ソース 7–6–3）．次の手順で，選択されているデータを削除します．

1. 選択データ情報の取得
2. 選択された行の取得
3. 行の削除

選択データ情報の取得（9 行目）

選択された行のリストを取得するには，関数 gtk_tree_view_get_selection を使用します．戻り値は GtkTreeSelection 型へのポインタです．選択された行がない場合には NULL が返ります．

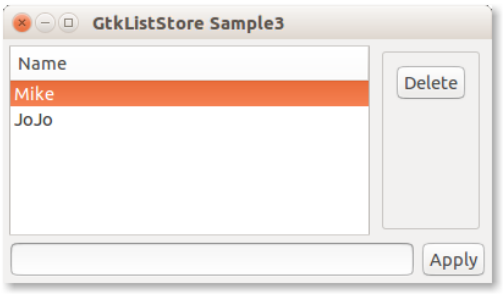


図 7.27　リストデータの削除

```
GtkTreeSelection* gtk_tree_view_get_selection (GtkTreeView *tree_view);
```

選択された行の取得（13 行目）

手順 1 で取得した GtkTreeSelection データを引数として，関数 gtk_tree_selection_get_selected により，選択されている行（GtkTreeIter）を取得します．選択された行がない場合には，関数の戻り値として FALSE が返ります．

```
gboolean
gtk_tree_selection_get_selected (GtkTreeSelection *selection,
                                 GtkTreeModel     **model,
                                 GtkTreeIter      *iter);
```

行の削除（14 行目）

選択されている行を削除するには，関数 gtk_list_store_remove を使います．正常に行が削除された場合には，関数の戻り値として TRUE が返ります．

```
gboolean gtk_list_store_remove (GtkListStore *list_store,
                                GtkTreeIter  *iter);
```

ソース 7–6–3 列データの削除：gtkliststore-sample3.c から一部抜粋

```
 1 static void
 2 delete_list (GtkTreeView *treeview)
 3 {
 4   GtkListStore     *store;
 5   GtkTreeSelection *selection;
 6   GtkTreeIter      iter;
 7   gboolean         success;
 8
 9   selection = gtk_tree_view_get_selection (treeview);
10   if (!selection) return;
11
12   store = GTK_LIST_STORE(gtk_tree_view_get_model (treeview));
13   success = gtk_tree_selection_get_selected (selection, NULL, &iter);
14   if (success) gtk_list_store_remove (store, &iter);
15 }
```

7.6.4　リストデータへの一括アクセス

先ほどの例を利用すれば，選択された行に対して何か処理を行うことが可能です．ここでは，リストに登録されたすべてのデータに対して順番にアクセスする方法を，例を使って 2 種類紹介します．図 7.28 は，印刷ボタンを押すと登録されたリストデータをターミナルに表示する例です（ソース 7–6–4）．

関数 gtk_tree_model_iter_next を使用する方法（29–34 行目）

関数 gtk_tree_model_iter_next は，現在の行（GtkTreeIter）の次の行を取得する関数です．

```
gboolean gtk_tree_model_iter_next (GtkTreeModel *tree_model,
                                   GtkTreeIter  *iter);
```

関数 gtk_tree_model_get_iter_first を使用すると，リストデータの先頭行を取得できます．後はループ文を用いて順に行データを取得すれば OK です．

```
gboolean gtk_tree_model_get_iter_first (GtkTreeModel *tree_model,
                                        GtkTreeIter  *iter);
```

関数 gtk_tree_model_foreach を使用する方法（1–14，36 行目）

関数 gtk_tree_model_foreach を使用すると，リストデータの各行に対して設定した関数を順番に実行できます．

```
void gtk_tree_model_foreach (GtkTreeModel            *model,
                             GtkTreeModelForeachFunc func,
                             gpointer                user_data);
```

GtkTreeModelForeachFunc のプロトタイプ宣言は次のようになります．

```
gboolean (*GtkTreeModelForeachFunc) (GtkTreeModel *model,
                                     GtkTreePath  *path,
                                     GtkTreeIter  *iter,
                                     gpointer     data);
```

行（GtkTreeIter）からそれぞれの列のデータを取得するには，関数 gtk_tree_model_get を使用します．第 3 引数から，取得するデータの列番号と取得したデータを格納する変数領域のアドレスを対にして与えます．引数の最後は −1 で終わります．

```
void gtk_tree_model_get (GtkTreeModel *tree_model,
                         GtkTreeIter  *iter,
                         ...);
```

ソース 7-6-4 リストデータへの一括アクセス：gtkliststore-sample4.c から一部抜粋

```
1  static gboolean
2  list_print_func (GtkTreeModel *model,
3                   GtkTreePath  *path,
4                   GtkTreeIter  *iter,
5                   gpointer     user_data)
6  {
7    gchar *name;
8
9    gtk_tree_model_get (model, iter, COLUMN_NAME, &name, -1);
10   g_print ("%s\n", name);
11   g_free (name);
12
13   return FALSE;
14 }
15
16 static void
17 cb_button_print (GtkButton *button, gpointer user_data)
18 {
19   GtkTreeView  *treeview;
20   GtkTreeModel *model;
21   GtkTreeIter  iter;
22   gboolean     success;
23   gchar        *name;
24
```

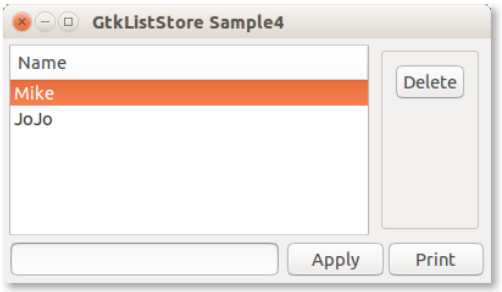


図 7.28　リストデータへの一括アクセス

```
25   treeview =
26     GTK_TREE_VIEW (g_object_get_data (G_OBJECT (user_data), "treeview"));
27   model = GTK_TREE_MODEL (gtk_tree_view_get_model (treeview));
28 #ifdef USE_LOOP
29   success = gtk_tree_model_get_iter_first (model, &iter);
30   while (success) {
31     gtk_tree_model_get (model, &iter, COLUMN_NAME, &name, -1);
32     g_print ("%s\n", name);
33     success = gtk_tree_model_iter_next (model, &iter);
34   }
35 #else
36   gtk_tree_model_foreach (model, list_print_func, NULL);
37 #endif
38 }
```

7.6.5　行の入れ換え

選択された行と前後の行を入れ換える方法について，例を使って説明します（ソース 7–6–5）．

行の入れ換えでは，選択された行の前後の行を取得して，データを入れ換えます．ポイントは選択された行の前後の行をどうやって取得するかです．選択された行の次の行を取得するには，選択行を関数 gtk_tree_selection_get_selected で取得し（40 行目），その次の行を関数 gtk_tree_model_iter_next で取得します（46 行目）．

反対に選択された行の前の行を取得する関数がありません．そこで，まず 19 行目で現在の行に対する GtkTreePath 情報を取得して，次に 20 行目で前の行の GtkTreePath 情報を取得しています．そして，21 行目でその行を取得しています．

入れ換える行それぞれの GtkTreeIter が取得できたら，関数 gtk_list_store_swap を使用して 2 つの行を入れ換えます（23, 48 行目）．

```
void gtk_list_store_swap (GtkListStore *store,
                          GtkTreeIter  *a,
                          GtkTreeIter  *b);
```

ソース 7-6-5　行の入れ換え：gtkliststore-sample5.c から一部抜粋

```
1  static void
2  up_list (GtkTreeView *treeview)
3  {
4    GtkListStore     *store;
5    GtkTreeSelection *selection;
6    GtkTreeIter      iter;
7    gboolean         success;
8
9    selection = gtk_tree_view_get_selection (treeview);
10   if (!selection) return;
11
12   store = GTK_LIST_STORE (gtk_tree_view_get_model (treeview));
13   success = gtk_tree_selection_get_selected (selection, NULL, &iter);
14   if (success)
15     {
16       GtkTreePath *path;
17       GtkTreeIter pre_iter;
18
19       path = gtk_tree_model_get_path (GTK_TREE_MODEL (store), &iter);
20       if (gtk_tree_path_prev (path) &&
```

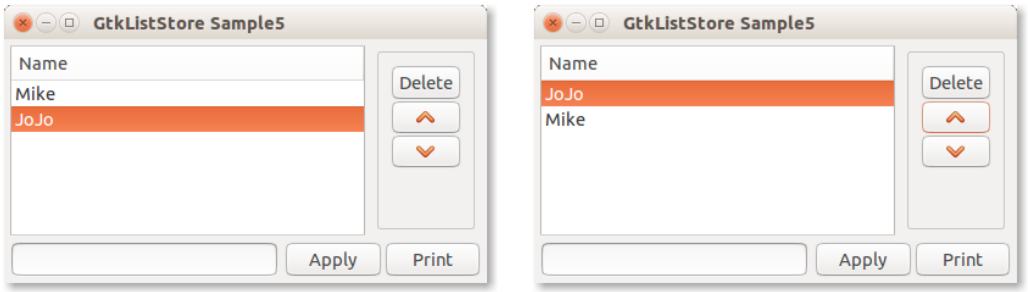


図 7.29　行の入れ換え

```
21            gtk_tree_model_get_iter (GTK_TREE_MODEL (store), &pre_iter, path))
22          {
23            gtk_list_store_swap (store, &pre_iter, &iter);
24          }
25      }
26  }
27
28  static void
29  down_list (GtkTreeView *treeview)
30  {
31    GtkListStore     *store;
32    GtkTreeSelection *selection;
33    GtkTreeIter      iter;
34    gboolean         success;
35
36    selection = gtk_tree_view_get_selection (treeview);
37    if (!selection) return;
38
39    store = GTK_LIST_STORE (gtk_tree_view_get_model (treeview));
40    success = gtk_tree_selection_get_selected (selection, NULL, &iter);
41    if (success)
42      {
43        GtkTreeIter next_iter;
44
45        next_iter = iter;
46        if (gtk_tree_model_iter_next (GTK_TREE_MODEL (store), &next_iter))
47          {
48            gtk_list_store_swap (store, &iter, &next_iter);
49          }
50      }
51  }
```

7.6.6　GtkCellRenderer に対するシグナルとコールバック関数

GtkCellRenderer には，扱うデータによってテキスト（GtkCellRendererText），トグルボタン（GtkCellRendererToggle），アイコン（GtkCellRendererPixbuf）があることは先に紹介しました．その中で GtkCellRendererText と GtkCellRendererToggle には，それぞれ独自のシグナルとそれに対するコールバック関数が定義されています．表 7.26 と表 7.27 にそれぞれのシグナルを示します．

それぞれのシグナルに対するコールバック関数のプロトタイプ宣言は次のようになっています．

```
void user_function (GtkCellRendererToggle *cellrenderertoggle,
                    gchar                 *path_string,
                    gpointer              user_data);

void user_function (GtkCellRendererText *cellrenderertext,
                    gchar               *path_string,
                    gchar               *new_text,
                    gpointer            user_data);
```

ソース 7-6-6 に GtkCellRendererText と GtkCellRendererToggle のコールバック関数の例を示します．

GtkCellRendererText のコールバック関数

まず 65–66 行目でコールバック関数の設定をしていますが，関数内で GtkTreeView からデータを取得したいので，関数へ引き渡すユーザデータとして変数 treeview を指定しています．

コールバック関数は 21–39 行目になります．関数の引数にユーザが編集した新しい文字列が自動的に渡されるので，関数 gtk_list_store_set を使用して更新された文字列を登録し直します．

表 7.26　GtkCellRendererText のシグナル

シグナル	説明
edited	セル内のデータが編集可能な場合に，データの編集が行われたときに発生するシグナル

表 7.27　GtkCellRendererToggle のシグナル

シグナル	説明
toggled	トグルボタンがクリックされたときに発生するシグナル

GtkCellRendererToggle のコールバック関数

先ほどと同様に，関数内で GtkTreeView からデータを取得したいので，関数へ引き渡すユーザデータとして変数 treeview を指定しています．

コールバック関数は 1–19 行目になります．15 行目で現在の状態を取得し，16 行目で状態を反転しています．このようにユーザがコールバック関数内で状態の更新を行わなければ，いくらマウスでチェックボタンをクリックしても状態は変更されないことに注意してください．

ソース 7–6–6 GtkCellRenderer のシグナルとコールバック関数：gtkliststore-sample6.c から一部抜粋

```
1  static void
2  cb_status_toggled (GtkCellRendererToggle *renderer,
3                     gchar                 *path_string,
4                     gpointer              user_data)
5  {
6    GtkTreeModel  *model;
7    GtkTreeIter   iter;
8    GtkTreePath   *path;
9    gboolean      status;
10
11   model = gtk_tree_view_get_model (GTK_TREE_VIEW(user_data));
12   path  = gtk_tree_path_new_from_string (path_string);
13
14   gtk_tree_model_get_iter (model, &iter, path);
15   gtk_tree_model_get (model, &iter, COLUMN_STATUS, &status, -1);
16   gtk_list_store_set (GTK_LIST_STORE(model), &iter, COLUMN_STATUS, !status, -1);
17   gtk_tree_path_free (path);
18 }
19
20 static void
21 cb_name_edited (GtkCellRendererText *renderer,
22                 const gchar         *path_string,
23                 const gchar         *new_text,
24                 gpointer            user_data)
25 {
26   GtkTreeModel *model;
27   GtkTreeIter  iter;
28   GtkTreePath  *path;
29   gboolean     status;
30
31   model = gtk_tree_view_get_model (GTK_TREE_VIEW (user_data));
32   path  = gtk_tree_path_new_from_string (path_string);
33
34   gtk_tree_model_get_iter (model, &iter, path);
35   gtk_list_store_set (GTK_LIST_STORE (model), &iter, COLUMN_NAME, new_text, -1);
36   gtk_tree_path_free (path);
37 }
38
39 static GtkWidget*
```

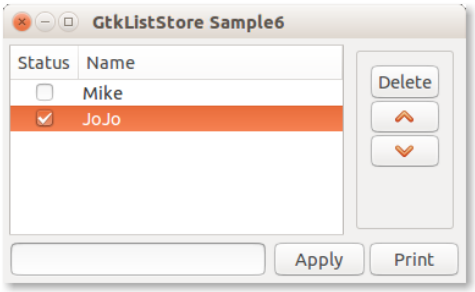


図 7.30　GtkCellRenderer に対するコールバック関数の例

```
40 list_new (void)
41 {
42   GtkWidget          *treeview;
43   GtkListStore       *liststore;
44   GtkCellRenderer    *renderer;
45   GtkTreeViewColumn *column;
46
47   liststore = gtk_list_store_new (N_COLUMNS,
48                                   G_TYPE_BOOLEAN,
49                                   G_TYPE_STRING,
50                                   G_TYPE_BOOLEAN);
51   treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL (liststore));
52
53   renderer = gtk_cell_renderer_toggle_new ();
54   g_signal_connect (renderer, "toggled",
55                     G_CALLBACK (cb_status_toggled), treeview);
56   column =
57     gtk_tree_view_column_new_with_attributes ("Status", renderer,
58                                               "active", COLUMN_STATUS,
59                                               NULL);
60   gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
61
62   renderer = gtk_cell_renderer_text_new ();
63   g_signal_connect (renderer, "edited",
64                     G_CALLBACK (cb_name_edited), treeview);
65   column =
66     gtk_tree_view_column_new_with_attributes ("Name", renderer,
67                                               "text", COLUMN_NAME,
68                                               "editable", COLUMN_EDITABLE,
69                                               NULL);
70   gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
71
72   return treeview;
73 }
```

7.6.7　データのソート

リストに登録されたデータを自動的にソートできるように，関数 gtk_tree_view_column_set_sort_column_id を使用します．関数の引数には GtkTreeViewColumn 変数と，どのデータでソートするかをその ID でソートするかを指定します．ソース 7–6–7 では COLUMN_NAME，つまり名前の項目でソートするように設定しています（11 行目）．

```
void
gtk_tree_view_column_set_sort_column_id(GtkTreeViewColumn *tree_column,
                                        gint               sort_column_id);
```

ソース 7-6-7 データのソート：gtkliststore-sample7.c から一部抜粋

```
1   renderer = gtk_cell_renderer_text_new ();
2   g_signal_connect (renderer, "edited",
3                     G_CALLBACK (cb_name_edited), treeview);
4   column =
5     gtk_tree_view_column_new_with_attributes ("Name", renderer,
6                                               "text", COLUMN_NAME,
7                                               "editable",
```

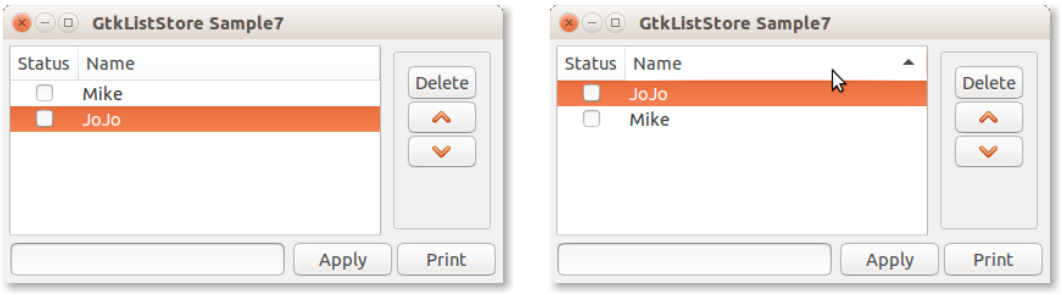


図 7.31　データのソート

```
8                                                   COLUMN_EDITABLE,
9                                                   NULL);
10  gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
11  gtk_tree_view_column_set_sort_column_id (column, COLUMN_NAME);
```

7.6.8 ツリーデータの表示

ソース 7–6–8 を例にして，ツリーデータの表示方法について解説します．

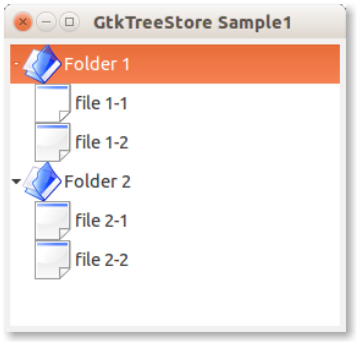


図 7.32 ツリーデータの表示

ツリーモデルの作成（83–84 行目）

ツリー表示のためのモデルには GtkTreeStore を使用します．モデル作成には，関数 gtk_tree_store_new を使います．引数は関数 gtk_list_store_new と同様です．

```
GtkTreeStore* gtk_tree_store_new (gint n_columns, ...);
```

アイコンセルの作成（90–94 行目）

テキストやチェックボタンをセルとする列を作成する方法は先に説明しました．ここではアイコンを表示する列を作成する方法を解説します．アイコン表示用のセルを作成するには，関数 gtk_cell_renderer_pixbuf_new を使用します．

セルにアイコンデータ（GdkPixbuf）を表示するためには，作成したセルに **pixbuf 属性**を与えなければいけません．

アイコンデータの登録（56–59 行目）

ツリーデータの追加は次の手順で行います．

1. 行の追加（56 行目）
   行の追加は関数 gtk_tree_store_append を用います．第 2 引数には新しく追加する GtkTreeIter を，第 3 引数には 1 つ上のレベルの GtkTreeIter のアドレスを指定します．ツリーデータを一番上のレベルに追加する場合には，第 3 引数は NULL にします．

   ```
   void gtk_tree_store_append (GtkTreeStore *tree_store,
                               GtkTreeIter  *iter,
                               GtkTreeIter  *parent);
   ```

2. 追加した行へのデータ登録（57–59 行目）
   関数 gtk_tree_store_append で取得した GtkTreeIter は，行データへのアクセスポイントとなります．この GtkTreeIter を関数 gtk_tree_store_set の引数に与えて，データを登録します．第 3 引数から列番号とデータを対にして指定し，最後の引数は −1 で終わります．

   ```
   void gtk_tree_store_set (GtkTreeStore *tree_store,
                            GtkTreeIter  *iter,
                            ...);
   ```

   アイコンデータを登録するには，GdkPixbuf 型のデータを使用します．この例では関数 gdk_pixbuf_new_from_file を使ってファイルから GdkPixbuf データを作成しています．

ヘッダの表示設定（122 行目）

ヘッダの表示設定をする関数は gtk_tree_view_set_headers_visible を使用します．第 2 引数を TRUE にするとヘッダを表示，FALSE にするとヘッダを表示しません．

```
void gtk_tree_view_set_headers_visible (GtkTreeView *tree_view,
                                        gboolean    headers_visible);
```

ソース 7-6-8 ツリーデータの表示：gtktreestore-sample1.c

```
1  #include <gtk/gtk.h>
2
3  enum
4  {
5    COLUMN_ICON,
6    COLUMN_LABEL,
7    N_COLUMNS
8  };
9
10 typedef struct _TreeData TreeData;
11 struct _TreeData
12 {
13   gchar    *iconname;
14   gchar    *label;
15   TreeData *child;
16 };
17
18 static TreeData sublevel1[] =
19 {
20   {"file.png", "file␣1-1", NULL},
21   {"file.png", "file␣1-2", NULL},
22   NULL
23 };
24
25 static TreeData sublevel2[] =
26 {
27   {"file.png", "file␣2-1", NULL},
28   {"file.png", "file␣2-2", NULL},
29   NULL
30 };
31
32 static TreeData toplevel[] =
33 {
34   {"folder.png", "Folder␣1", sublevel1},
35   {"folder.png", "Folder␣2", sublevel2},
36   NULL
37 };
38
39 static void
40 add_data (GtkTreeView *treeview)
41 {
42   GtkTreeStore *store;
43   TreeData     *top;
44
45   store = GTK_TREE_STORE(gtk_tree_view_get_model (treeview));
46
47   top = toplevel;
48   while (top->iconname)
49     {
50       TreeData    *child;
51       GdkPixbuf   *pixbuf;
52       GtkTreeIter iter;
53
54       child = top->child;
55       pixbuf = gdk_pixbuf_new_from_file (top->iconname, NULL);
56       gtk_tree_store_append (store, &iter, NULL);
57       gtk_tree_store_set (store, &iter,
58                           COLUMN_ICON,  pixbuf,
59                           COLUMN_LABEL, top->label, -1);
60       while (child->iconname)
61         {
62           GtkTreeIter child_iter;
63
64           pixbuf = gdk_pixbuf_new_from_file (child->iconname, NULL);
65           gtk_tree_store_append (store, &child_iter, &iter);
```

```
66            gtk_tree_store_set (store, &child_iter,
67                                COLUMN_ICON,  pixbuf,
68                                COLUMN_LABEL, child->label, -1);
69            child++;
70          }
71        top++;
72      }
73  }
74
75  static GtkWidget*
76  create_tree_model (void)
77  {
78    GtkWidget         *treeview;
79    GtkTreeStore      *treestore;
80    GtkCellRenderer   *renderer;
81    GtkTreeViewColumn *column;
82
83    treestore = gtk_tree_store_new (N_COLUMNS,
84                                    GDK_TYPE_PIXBUF, G_TYPE_STRING);
85    treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL (treestore));
86    g_object_unref (treestore);
87
88    column = gtk_tree_view_column_new ();
89
90    renderer = gtk_cell_renderer_pixbuf_new ();
91    gtk_tree_view_column_set_title (column, "Folder");
92    gtk_tree_view_column_pack_start (column, renderer, FALSE);
93    gtk_tree_view_column_add_attribute (column, renderer, "pixbuf", COLUMN_ICON);
94
95    renderer = gtk_cell_renderer_text_new ();
96    gtk_tree_view_column_pack_start (column, renderer, TRUE);
97    gtk_tree_view_column_add_attribute (column, renderer, "text", COLUMN_LABEL);
98
99    gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
100
101   return treeview;
102 }
103
104 int
105 main (int argc, char *argv[])
106 {
107   GtkWidget *window;
108   GtkWidget *treeview;
109
110   gtk_init (&argc, &argv);
111
112   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
113   gtk_window_set_title (GTK_WINDOW (window), "GtkTreeStore␣Sample1");
114   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
115   g_signal_connect (G_OBJECT (window), "destroy",
116                     G_CALLBACK (gtk_main_quit), NULL);
117   gtk_widget_set_size_request (window, 280, 240);
118
119   treeview = create_tree_model ();
120   gtk_tree_view_set_headers_visible (GTK_TREE_VIEW (treeview), FALSE);
121   gtk_container_add (GTK_CONTAINER (window), treeview);
122
123   add_data (GTK_TREE_VIEW (treeview));
124
125   gtk_widget_show_all (window);
126   gtk_main ();
127
128   return 0;
129 }
```

7.6.9 ダブルクリックによるツリーの展開

行データをダブルクリックすることによってツリーを展開する例を，ソース 7–6–9 に示します．

行データをダブルクリックすると **row-activated シグナル**が発生します．関数 g_signal_connect を用いて，このシグナルに対するコールバック関数を設定します（84–85 行目）．row-activated シグナルに対するコールバック関数のプロトタイプ宣言は次のようになっています．

```
void user_function (GtkTreeView       *treeview,
```

```
                        GtkTreePath       *path,
                        GtkTreeViewColumn *column,
                        gpointer          user_data);
```

この例ではダブルクリックした行が展開されているかどうかを調べて，展開されていない場合は，関数 gtk_tree_view_expand_row によって行を展開し，展開されている場合は，関数 gtk_tree_view_collapse_row によって行を閉じています．関数 gtk_tree_view_expand_row の第 3 引数は，指定した行以下を再帰的にすべて展開するかどうかを指定します．

```
gboolean gtk_tree_view_expand_row (GtkTreeView *tree_view,
                                   GtkTreePath *path,
                                   gboolean    open_all);

gboolean gtk_tree_view_collapse_row (GtkTreeView *tree_view,
                                     GtkTreePath *path);
```

ダブルクリックした行が既に展開されているかどうかを調べる関数は提供されていません．この例では展開されている行のリストを取得し，クリックされた行のラベルがリスト中に存在するかどうかで，クリックした行が展開されているかどうかを調べています（20–40 行目）．

ソース 7-6-9 ダブルクリックによるツリーの展開：gtktreestore-sample2.c から一部抜粋

```
 1 static void
 2 get_expanded_path (GtkTreeView *treeview,
 3                    GtkTreePath *path,
 4                    gpointer    user_data)
 5 {
 6   GtkTreeModel *model;
 7   GtkTreeIter  iter;
 8   GList        **list;
 9   gchar        *label;
10
11   list  = user_data;
12   model = gtk_tree_view_get_model (treeview);
13
14   gtk_tree_model_get_iter (model, &iter, path);
15   gtk_tree_model_get (model, &iter, COLUMN_LABEL, &label, -1);
16
17   *list = g_list_append (*list, label);
18 }
19
20 static gboolean
21 is_expanded (GtkTreeView *treeview,
22              const gchar *label)
23 {
24   GList    *list = NULL, *node;
25   gboolean success = FALSE;
26
27   gtk_tree_view_map_expanded_rows (treeview, get_expanded_path, &list);
28   for (node = list; node; node = g_list_next (node))
29     {
30       if (strcmp ((char *) node->data, label) == 0)
31         {
32           success = TRUE;
33           break;
34         }
35     }
36   g_list_foreach (list, (GFunc) g_free, NULL);
37   g_list_free (list);
38
39   return success;
40 }
41
42 static void
43 cb_double_clicked (GtkTreeView       *treeview,
44                    GtkTreePath       *path,
45                    GtkTreeViewColumn *column,
46                    gpointer          user_data)
47 {
48   GtkTreeModel *model;
49   GtkTreeIter  iter;
```

```
50   gchar        *label;
51
52   model = gtk_tree_view_get_model (treeview);
53   if (gtk_tree_model_get_iter (model, &iter, path))
54     {
55       gtk_tree_model_get (model, &iter, COLUMN_LABEL, &label, -1);
56       if (is_expanded (treeview, label))
57         {
58           gtk_tree_view_collapse_row (treeview, path);
59         }
60       else
61         {
62           gtk_tree_view_expand_row (treeview, path, FALSE);
63         }
64       g_free (label);
65     }
66 }
67
68 int
69 main (int argc, char **argv)
70 {
71   GtkWidget *window;
72   GtkWidget *treeview;
73
74   gtk_init (&argc, &argv);
75
76   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
77   gtk_window_set_title (GTK_WINDOW (window), "GtkTreeStore␣Sample2");
78   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
79   g_signal_connect (G_OBJECT (window), "destroy",
80                     G_CALLBACK (gtk_main_quit), NULL);
81   gtk_widget_set_size_request (window, 280, 240);
82
83   treeview = create_tree_model ();
84   g_signal_connect (treeview, "row-activated",
85                     G_CALLBACK (cb_double_clicked), NULL);
86   gtk_tree_view_set_headers_visible (GTK_TREE_VIEW (treeview), FALSE);
87   gtk_container_add (GTK_CONTAINER (window), treeview);
88
89   add_data (GTK_TREE_VIEW (treeview));
90
91   gtk_widget_show_all (window);
92   gtk_main ();
93
94   return 0;
95 }
```

7.6.10 ツリーストアに関するその他の関数

ツリーストアに関する関数はほとんどリストストアに関する関数と同様ですが，一部ツリーストアに特有な関数も存在します．ここではそれらの関数を紹介します．詳細や具体的な使用例などは省略します．

- gtk_tree_store_is_ancestor
  第 2 引数に指定したノード（iter）が第 3 引数に与えたノード（descendant）の親（再帰的にまたはその親）なら TRUE を返します．

```
gboolean gtk_tree_store_is_ancestor (GtkTreeStore *tree_store,
                                     GtkTreeIter  *iter,
                                     GtkTreeIter  *descendant);
```

- gtk_tree_store_iter_depth
  指定したノードのツリーの深さを返します．一番上のレベルは 0 となります．

```
gint gtk_tree_store_iter_depth (GtkTreeStore *tree_store,
                                GtkTreeIter  *iter);
```

7.7　その他の特殊なウィジェット

7.7.1　ツールチップ

ツールチップウィジェット（**GtkTooltip**）は，ボタン等のウィジェット上にマウスカーソルが一定時間以上存在するときに表示される説明です．ツールバーに配置するウィジェットには，このツールチップを表示する機能が備わっています．GtkTooltip を利用すると，さまざまなウィジェットにツールチップを表示させることができます（図 7.33）．

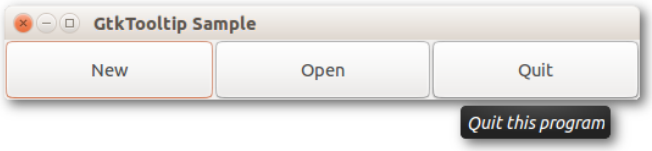


図 7.33　ツールチップ

オブジェクトの階層構造

```
GObject
 +----GtkTooltip
```

ツールチップの設定

ウィジェットにツールチップを設定するには関数 gtk_widget_set_tooltip_text を使用します．

```
void gtk_widget_set_tooltip_text (GtkWidget   *widget,
                                  const gchar *text);
```

第 1 引数にはツールチップを設定するウィジェットを，第 2 引数にはツールチップの文字列を与えます．

サンプルプログラム

ソース 7–7–1 では，普通のボタンウィジェットに対して，ツールチップを設定しています．図 7.33 のように一定時間ボタンの上にカーソルを置いておくと，チップが表示されます．

ソース 7–7–1 ツールチップのサンプルプログラム：gtktooltip-sample.c

```
1  #include <gtk/gtk.h>
2
3  int
4  main (int argc, char *argv[])
5  {
6    GtkWidget   *window;
7    GtkWidget   *hbox;
8    GtkWidget   *button;
9
10   gtk_init (&argc, &argv);
11
12   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
13   gtk_window_set_title (GTK_WINDOW (window), "GtkTooltip␣Sample");
14   g_signal_connect (G_OBJECT (window), "destroy",
15                     G_CALLBACK (gtk_main_quit), NULL);
16
17   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
18   gtk_container_add (GTK_CONTAINER (window), hbox);
19
20   button = gtk_button_new_with_label ("New");
21   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
22   gtk_widget_set_tooltip_text (button, "Run␣new␣program");
23
24   button = gtk_button_new_with_label ("Open");
25   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
26   gtk_widget_set_tooltip_text (button, "Open␣a␣file");
27
28   button = gtk_button_new_with_label ("Quit");
```

```
29   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
30   gtk_widget_set_tooltip_markup (button, "<i>Quit this program</i>");
31
32   g_signal_connect (G_OBJECT (button), "clicked",
33                     G_CALLBACK (gtk_main_quit), NULL);
34
35   gtk_widget_show_all (window);
36   gtk_main ();
37
38   return 0;
39 }
```

7.7.2　プログレスバー

プログレスバーウィジェット（**GtkProgressBar**）は，動作の進行状況などを視覚的に表示するためのウィジェットです．

表示形式には，図 7.34 のように 3 つあります．上段左のように，バーが左右を行ったり来たりして，動作中であることを示すもの（これをアクティビティモードと呼ぶことにします）と，上段右のように進行状況をバーの長さで表現するもの（これをプログレッシブモードと呼ぶことにします）があります．後者のタイプには，図の下段のようにバーが右から左へ伸びていくものもあります．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkProgress
                   +----GtkProgressBar
```

ウィジェットの作成

プログレスバーを作成するには，関数 gtk_progress_bar_new を使用します．

```
GtkWidget* gtk_progress_bar_new (void);
```

プログレスバーの更新

プログレスバーの更新の方法は，プログレスバーの表示モードによって異なります．

- アクティビティモードの場合
  プログレスバーがアクティビティモードで表示されている場合には，関数 gtk_progress_bar_pulse を呼び出すことでプログレスバーの状態を更新できます．

  ```
  void gtk_progress_bar_pulse (GtkProgressBar *pbar);
  ```

  関数 gtk_progress_bar_set_pulse_step を使用すると，プログレスバーのステップ幅を変更できます．この値は 0.0 から 1.0 までの範囲で与えます．

  ```
  void gtk_progress_bar_set_pulse_step (GtkProgressBar *pbar,
                                        gdouble        fraction);
  ```

- プログレッシブモードの場合
  プログレスバーがプログレッシブモードで表示されている場合には，関数 gtk_progress_bar_set_fraction を使用します．この関数では，プログレスバーの長さを 0.0 から 1.0 までの範囲の実数で指定します．

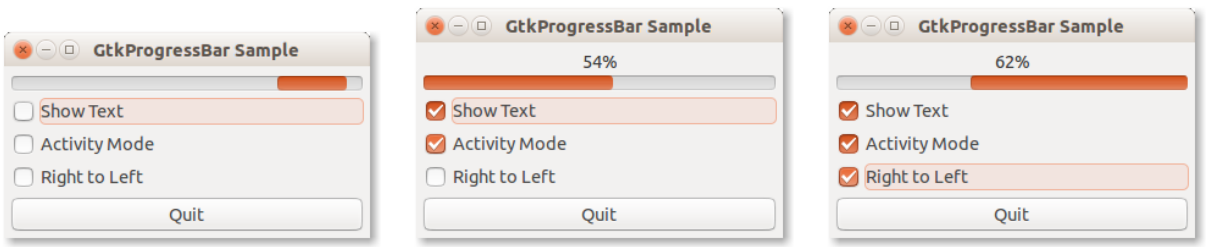


図 7.34　プログレスバー

```
void gtk_progress_bar_set_fraction (GtkProgressBar *pbar,
                                    gdouble        fraction);
```

ウィジェットのプロパティ設定

プログレスバーのプロパティには以下のものがあります.

- プログレスバーの現在の値
  プログレスバーの現在の値を取得するには，関数 gtk_progress_bar_get_fraction を使用します．また，先ほどの関数 gtk_progress_bar_set_fraction によって，値を更新できます.

```
gdouble gtk_progress_bar_get_fraction (GtkProgressBar *pbar);
```

- プログレスバーの表示方向
  プログレスバーの表示方向を変更するには, 関数 gtk_orientable_set_orientation を使用します．指定する方向は, 表 7.8 を参照してください．また，関数 gtk_orientable_get_orientation によって，現在の設定を取得できます.

```
void
gtk_orientable_set_orientation (GtkOrientable  *orientable,
                                GtkOrientation orientation);
```

- プログレスバーのテキスト表示
  関数 gtk_progress_bar_set_show_text を使うとテキストを表示することができます．プログレスバーの表示と同時に処理の進行具合をパーセンテージで示す場合に適しています.

```
void gtk_progress_bar_set_show_text (GtkProgressBar *pbar,
                                     gboolean       show_text);
```

- プログレスバーに表示するテキスト
  表示するテキストは，関数 gtk_progress_bar_set_text を使用して設定します．反対に現在表示されているテキストを取得するには，関数 gtk_progress_bar_get_text を使用します.

```
void gtk_progress_bar_set_text (GtkProgressBar *pbar,
                                const gchar    *text);

const gchar * gtk_progress_bar_get_text (GtkProgressBar *pbar);
```

サンプルプログラム

ソース 7–7–2 に，プログレスバーのサンプルプログラムのソースコードを示します．実行すると図 7.34 のようなウィンドウが表示され，プログレスバーの動作を確認できます.

このプログラムでは，一定時間ごとにプログレスバーを更新するために，GLib のタイマー機能（g_timeout_add ）を使用しています（4.5 節，p. 66）.

ソース 7-7-2 プログレスバーのサンプルプログラム：gtkprogressbar-sample.c

```
1  #include <gtk/gtk.h>
2
3  static gboolean activity_mode = 0;
4  static gboolean show_text = 0;
5  static gint     timer = 0;
6
7  static gboolean
8  progressbar_update (gpointer user_data)
9  {
10   GtkProgressBar *progressbar = GTK_PROGRESS_BAR (user_data);
11   gdouble        new_val;
12   const gchar    *text;
13   gchar          label[256];
14
15   if (!activity_mode)
16     {
17       gtk_progress_bar_pulse (progressbar);
18     }
19   else
20     {
```

```
21        new_val = gtk_progress_bar_get_fraction (progressbar) + 0.01;
22        if (new_val > 1.0) new_val = 0.0;
23        gtk_progress_bar_set_fraction (progressbar, new_val);
24        if (show_text)
25          {
26            sprintf (label, "%3d%s", (int) (new_val * 100.0), "%");
27            gtk_progress_bar_set_text (progressbar, label);
28          }
29      }
30    return TRUE;
31  }
32
33  static void
34  cb_show_text (GtkToggleButton *widget, gpointer user_data)
35  {
36    GtkProgressBar *progressbar = GTK_PROGRESS_BAR (user_data);
37
38    show_text = !show_text;
39    gtk_progress_bar_set_show_text (GTK_PROGRESS_BAR (user_data),
40                                    activity_mode && show_text);
41  }
42
43  static void
44  cb_activity_mode (GtkToggleButton *widget, gpointer user_data)
45  {
46    GtkProgressBar *progressbar = GTK_PROGRESS_BAR (user_data);
47
48    activity_mode = gtk_toggle_button_get_active (widget);
49    if (activity_mode)
50      {
51        gtk_progress_bar_set_fraction (progressbar, 0.0);
52      }
53    else
54      {
55        gtk_progress_bar_pulse (progressbar);
56      }
57    gtk_progress_bar_set_show_text (progressbar,
58                                    activity_mode && show_text);
59  }
60
61  static void
62  cb_orientation (GtkToggleButton *widget, gpointer user_data)
63  {
64    gboolean inverted;
65
66    inverted = gtk_progress_bar_get_inverted (GTK_PROGRESS_BAR (user_data));
67    gtk_progress_bar_set_inverted (GTK_PROGRESS_BAR (user_data), !inverted);
68  }
69
70  static void
71  cb_quit (GtkButton *widget, gpointer user_data)
72  {
73    g_source_remove (timer);
74    gtk_main_quit ();
75  }
76
77  int
78  main (int argc, char *argv[])
79  {
80    GtkWidget *window;
81    GtkWidget *vbox;
82    GtkWidget *progressbar;
83    GtkWidget *button;
84
85    gtk_init (&argc, &argv);
86
87    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
88    gtk_window_set_title  (GTK_WINDOW (window), "GtkProgressBar␣Sample");
89    gtk_widget_set_size_request (window, 300, -1);
90    gtk_container_set_border_width (GTK_CONTAINER (window), 5);
91    g_signal_connect (G_OBJECT (window), "destroy",
92                      G_CALLBACK (cb_quit), NULL);
93
94    vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 5);
95    gtk_container_add (GTK_CONTAINER (window), vbox);
96
```

```
97    progressbar = gtk_progress_bar_new ();
98    gtk_box_pack_start (GTK_BOX (vbox), progressbar, FALSE, FALSE, 0);
99
100
101   button = gtk_check_button_new_with_label ("Show␣Text");
102   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
103   g_signal_connect (G_OBJECT (button), "clicked",
104                     G_CALLBACK (cb_show_text), progressbar);
105
106   button = gtk_check_button_new_with_label ("Activity␣Mode");
107   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
108   g_signal_connect (G_OBJECT (button), "clicked",
109                     G_CALLBACK (cb_activity_mode), progressbar);
110
111   button = gtk_check_button_new_with_label ("Right␣to␣Left");
112   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
113   g_signal_connect (G_OBJECT (button), "clicked",
114                     G_CALLBACK (cb_orientation), progressbar);
115
116   button = gtk_button_new_with_label ("Quit");
117   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
118   g_signal_connect (G_OBJECT (button), "clicked",
119                     G_CALLBACK (gtk_main_quit), NULL);
120
121   timer = g_timeout_add (100, progressbar_update, progressbar);
122
123   gtk_widget_show_all (window);
124   gtk_main ();
125
126   return 0;
127 }
```

7.7.3　セパレータ

セパレータウィジェット（**GtkSeparator**）は何の動作もしませんが，GUI レイアウトにメリハリをつけるためには重要なウィジェットです．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
        +----GtkWidget
               +----GtkSeparator
                      +----GtkHSeparator
                      +----GtkVSeparator
```

ウィジェットの作成

セパレータウィジェットを作成するには関数 gtk_separator_new を使用します．セパレータの方向は引数の GtkOrientation により指定します．

```
GtkWidget* gtk_separator_new (GtkOrientation orientation);
```

7.7.4　スケール

スケールウィジェット（**GtkScale**）は，スピンボタンと同じように，数値を GUI でコントロールするためのウィジェットです．スピンボタンがエントリとボタンによって数値のコントロールを行うのに対して，スケールはバーをマウスでドラッグすることによって数値のコントロールを行います（図 7.35）．

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
        +----GtkWidget
               +----GtkRange
                      +----GtkScale
                             +----GtkHScale
                             +----GtkVScale
```

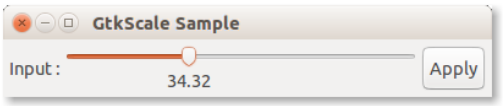


図 7.35　スケール

ウィジェットの作成

スケールウィジェットには横方向のスケールと縦方向のスケールがあり，それぞれ次の関数によって作成します．

- gtk_scale_new
  GtkAdjustment で数値の範囲等を設定し，関数の引数に指定して水平スケールを作成します．

```
GtkWidget* gtk_scale_new (GtkOrientation *orientation, GtkAdjustment *adjustment);
```

- gtk_scale_new_with_range
  数値の範囲とステップ幅を引数に指定して，水平スケールを作成します．

```
GtkWidget* gtk_scale_new_with_range (gdouble min, gdouble max, gdouble step);
```

シグナルとコールバック関数

スケールウィジェットで最もよく使用されるシグナルは **value-changed シグナル**です．このシグナルは，スケールの値が変化したときに発生します．

value-changed シグナル に対するコールバック関数のプロトタイプ宣言は次のようになります．第 1 引数の型を見てもわかるように GtkScale は GtkRange のサブクラスで，シグナルも GtkRange のシグナルを継承しています．

```
gboolean user_function (GtkRange *range, gpointer user_data);
```

ウィジェットのプロパティ設定

スケールウィジェットのプロパティには次の項目が存在します．

- 現在の値
  スケールの現在の値は**レンジウィジェット**の関数 gtk_range_get_value を使って取得できます．反対に関数 gtk_range_set_value を使ってスケールの値を設定することもできます．

```
gdouble gtk_range_get_value (GtkRange *range);

void gtk_range_set_value (GtkRange *range, gdouble value);
```

- 数値の表示
  スケールはバーを動かすことで数値をコントロールしますが，バーの位置だけでは厳密な数値を知ることができません．スケールウィジェットには現在の値を表示する機能があります．数値を表示するかどうかの設定は関数 gtk_scale_set_draw_value で行います．現在の設定を取得するには関数 gtk_scale_get_draw_value を使用します．

```
void
gtk_scale_set_draw_value (GtkScale *scale, gboolean draw_value);

gboolean gtk_scale_get_draw_value (GtkScale *scale);
```

- 数値の表示位置
  数値を表示する場合，その表示位置をスケールの上下左右のどこに表示するかを設定できます．表示位置の設定と現在の設定取得には，それぞれ関数 gtk_scale_set_value_pos と関数 gtk_scale_get_value_pos を使用します．

```
void
gtk_scale_set_value_pos (GtkScale *scale, GtkPositionType pos);

GtkPositionType gtk_scale_get_value_pos (GtkScale *scale);
```

- 小数値の表示桁数
  以下の関数を使って，表示する数値の小数部分の表示桁数を設定したり，現在の設定を取得したりできます．

```
void gtk_scale_set_digits (GtkScale *scale, gint digits);

gint gtk_scale_get_digits (GtkScale *scale);
```

サンプルプログラム

ソース 7–7–3 にスケールのサンプルプログラムのソースコードを示します．

ソース 7–7–3 スケールウィジェットのサンプルプログラム：gtkscale-sample.c

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

static void
cb_value_changed (GtkScale *scale, gpointer user_data)
{
  g_print ("value␣=␣%f\n", gtk_range_get_value (GTK_RANGE (scale)));
}

static void
cb_button (GtkButton *button, gpointer user_data)
{
  g_print ("value␣=␣%f\n", gtk_range_get_value (GTK_RANGE (user_data)));
}

int
main (int argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *hbox;
  GtkWidget *label;
  GtkWidget *scale;
  GtkWidget *button;
  gdouble   min = 0.0, max = 100.0, step = 0.1;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "GtkScale␣Sample");
  gtk_widget_set_size_request (window, 400, -1);
  gtk_container_set_border_width (GTK_CONTAINER (window), 5);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 5);
  gtk_container_add (GTK_CONTAINER (window), hbox);

  label = gtk_label_new ("Input␣:");
  gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);

  scale = gtk_scale_new_with_range (GTK_ORIENTATION_HORIZONTAL, min, max, step);
  gtk_scale_set_digits (GTK_SCALE (scale), 2);
  gtk_scale_set_draw_value (GTK_SCALE (scale), TRUE);
  gtk_scale_set_value_pos (GTK_SCALE (scale), GTK_POS_BOTTOM);

  g_signal_connect (G_OBJECT (scale), "value-changed",
                    G_CALLBACK (cb_value_changed), NULL);
  gtk_box_pack_start (GTK_BOX (hbox), scale, TRUE, TRUE, 0);

```

```
49    button = gtk_button_new_with_label ("Apply");
50    g_signal_connect (G_OBJECT (button), "clicked",
51                      G_CALLBACK (cb_button), (gpointer) scale);
52    gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
53
54    gtk_widget_show_all (window);
55    gtk_main ();
56
57    return 0;
58  }
```

7.7.5　アイコンビュー

アイコンビューウィジェット（**GtkIconView**）は，画像をサムネイル化したものやアイコンなどを一括して閲覧できるようにするウィジェットです（図 7.36）．ファイルブラウザや画像ブラウザなどの作成に適しています．

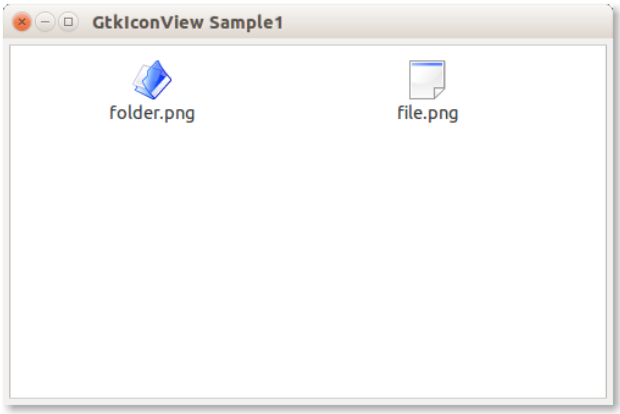


図 7.36　アイコンビューウィジェットの例

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +----GtkWidget
             +----GtkContainer
                   +----GtkIconView
```

ウィジェットの作成

アイコンビューウィジェットの作成には，関数 gtk_icon_view_new もしくは gtk_icon_view_new_with_model を使用します．

```
GtkWidget* gtk_icon_view_new (void);

GtkWidget* gtk_icon_view_new_with_model (GtkTreeModel *model);
```

2 つ目の関数からもわかるように，アイコンビューウィジェットは GtkTreeModel を用いて表示するデータを管理します．

モデルの作成

モデルの作成には関数 gtk_list_store_new を使用します．用途によってさまざまなデータを設定できますが，最低限表示するアイテムのラベルと画像データ（GdkPixbuf データ）を設定する必要があります．ここでは画像とラベルを組み合わせたものをアイテムと呼ぶことにします．以下は最低限のデータのみを設定したモデル作成の例です．

```
GtkListStore *store = gtk_list_store_new (2, G_TYPE_STRING, GDK_TYPE_PIXBUF);
```

関数 gtk_icon_view_new でウィジェットを作成した場合には，関数 gtk_icon_view_set_model を呼び出して後で作成したモデルを登録する必要があります．

表 7.28　アイコンビューウィジェットのシグナル

シグナル	説明
item-activated	表示されているアイテムがダブルクリックされたときに発生するシグナル
select-all	表示されているアイテムがすべて選択状態になったときに発生するシグナル
unselect-all	表示されているアイテムがすべて非選択状態になったときに発生するシグナル

```
void gtk_icon_view_set_model (GtkIconView  *icon_view,
                              GtkTreeModel *model);
```

表示項目の設定

モデルの作成と登録が終了したら，次に，登録したモデルに含まれる項目のなかで，どの項目がウィジェットに表示するラベルと画像かを指定する必要があります．項目の設定にはそれぞれ以下に示す関数を使用します．

```
void gtk_icon_view_set_text_column (GtkIconView *icon_view, gint column);

void gtk_icon_view_set_pixbuf_column (GtkIconView *icon_view, gint column);
```

関数の第 2 番目の引数には，作成したモデルの対応する項目番号を指定します．番号は 0 から始まることに注意してください．

データの追加

データの追加はツリービューウィジェットの節でも説明した，関数 gtk_list_store_append と関数 gtk_list_store_set を組み合わせて使います．詳しい説明は 7.6.1 （p. 195）を参照してください．

シグナルとコールバック関数

表 7.28 にアイコンビューウィジェットのシグナルの一部を示します．この中で一番使用頻度が高いのは **item-activated シグナル**でしょう．このシグナルは表示されているアイテムがダブルクリックされたときに発生するシグナルなので，このシグナルに合わせてファイルを開いたり，プログラムを実行したりということが実現できます．

item-activated シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．

```
void user_function (GtkIconView *iconview, GtkTreePath *arg1, gpointer user_data);
```

この関数の第 2 引数から関数 gtk_tree_model_get_iter を使って，クリックされたアイテムに対する GtkTreeIter の値を取得できます．

ウィジェットのプロパティ設定

アイコンビューウィジェットの代表的なプロパティを以下に示します．また図 7.37 にアイコンビューウィジェットの空白に関するプロパティを図示したので参考にしてください．

- 列数
  アイテムを横に何個表示するかを設定します．この値を −1 とした場合，領域の大きさに応じて適切に設定されます．

```
void gtk_icon_view_set_columns (GtkIconView *icon_view, gint columns);

gint gtk_icon_view_get_columns (GtkIconView *icon_view);
```

- マージン（margin）
  アイコンビューウィジェットの上下左右の空白を設定します．

```
void gtk_icon_view_set_margin (GtkIconView *icon_view, gint margin);

gint gtk_icon_view_get_margin (GtkIconView *icon_view);
```

- アイテムの幅（item width）
  アイテムの幅を設定します．この値を −1 とした場合，画像の大きさに応じて適切に設定されます．

```
void gtk_icon_view_set_item_width (GtkIconView *icon_view,
                                   gint         item_width);

gint gtk_icon_view_get_item_width (GtkIconView *icon_view);
```

- 列の空白（column spacing）
  アイテム間の横の空白を設定します．

```
void gtk_icon_view_set_column_spacing (GtkIconView *icon_view,
                                       gint         column_spacing);

gint gtk_icon_view_get_column_spacing (GtkIconView *icon_view);
```

- 行の空白（row spacing）
  アイテム間の縦の空白を設定します．

```
void gtk_icon_view_set_row_spacing (GtkIconView *icon_view,
                                    gint         row_spacing);

gint gtk_icon_view_get_row_spacing (GtkIconView *icon_view);
```

- 画像とラベルの間の空白（spacing）
  画像とラベル間の空白を設定します．

```
void gtk_icon_view_set_spacing (GtkIconView *icon_view, gint spacing);

gint gtk_icon_view_get_spacing (GtkIconView *icon_view);
```

- ラベルの表示位置
  ラベルを画像の下に表示するか右に表示するかを設定します．表示位置は GtkOrientation の値で指定します．

```
void gtk_icon_view_set_orientation (GtkIconView     *icon_view,
                                    GtkOrientation orientation);

GtkOrientation
gtk_icon_view_get_orientation (GtkIconView *icon_view);
```

- 並べ替えの可/不可
  アイテムが追加されたときに表示しているアイテム全体を並べ替えするかどうかを設定します．

```
void gtk_icon_view_set_reorderable (GtkIconView *icon_view,
                                    gboolean     reorderable);

gboolean gtk_icon_view_get_reorderable (GtkIconView *icon_view);
```

- 選択モード
  アイテムの選択モードを設定します．選択モードは GtkSelectionMode の値（表 7.29 を参照）で指定します．

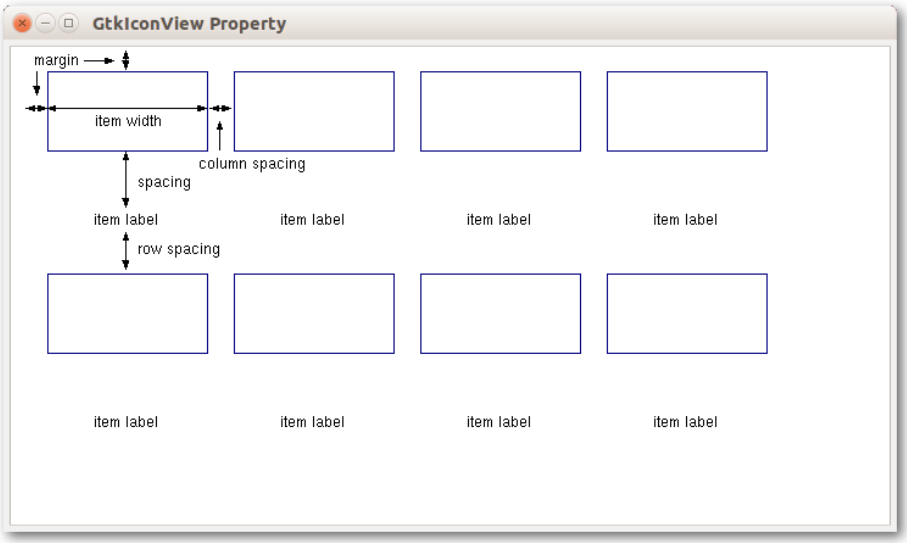


図 7.37　アイコンビューウィジェットのプロパティ

```
void gtk_icon_view_set_selection_mode (GtkIconView *icon_view,
                                       GtkSelectionMode mode);

GtkSelectionMode
gtk_icon_view_get_selection_mode (GtkIconView *icon_view);
```

表 7.29　GtkSelectionMode の種類

値	説明
GTK_SELECTION_NONE	何も選択しない
GTK_SELECTION_SINGLE	0 もしくは一つのアイテムを選択する
GTK_SELECTION_BROWSE	一つのアイテムを選択する
GTK_SELECTION_MULTIPLE	複数のアイテムを選択します

サンプルプログラム

ソース 7–7–4 に，アイコンビューウィジェットのサンプルプログラムのソースコードを示します．このプログラムは画像ファイルを読み込んでアイコンとして表示するだけの簡単なものです．アイコン表示の実際の例を確認してみてください．

ソース 7-7-4 アイコンビューウィジェットのサンプルプログラム その 1：gtkiconview-sample1.c

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

enum
{
  COLUMN_NAME,
  COLUMN_PIXBUF,
  N_COLUMNS
};

static void
add_data (GtkIconView *iconview)
{
  GdkPixbuf    *folder_pixbuf;
  GdkPixbuf    *file_pixbuf;
  GtkListStore *store;
  GtkTreeIter  iter;

  file_pixbuf   = gdk_pixbuf_new_from_file ("./file.png", NULL);
  folder_pixbuf = gdk_pixbuf_new_from_file ("./folder.png", NULL);

  store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));

  gtk_list_store_clear (store);

  gtk_list_store_append (store, &iter);
  gtk_list_store_set (store, &iter,
                      COLUMN_NAME, "folder.png",
                      COLUMN_PIXBUF, folder_pixbuf, -1);
  g_object_unref (folder_pixbuf);

  gtk_list_store_append (store, &iter);
  gtk_list_store_set (store, &iter,
                      COLUMN_NAME, "file.png",
                      COLUMN_PIXBUF, file_pixbuf, -1);
  g_object_unref (file_pixbuf);
}

static GtkWidget*
create_icon_view_widget (void)
{
  GtkWidget    *iconview;
  GtkListStore *store;

  store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING, GDK_TYPE_PIXBUF);
  iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL (store));
  g_object_unref (store);

```

```
48   return iconview;
49 }
50
51 static void
52 cb_item_activated (GtkIconView *iconview,
53                    GtkTreePath *treepath,
54                    gpointer    user_data)
55 {
56   GtkListStore *store;
57   GtkTreeIter  iter;
58   gchar        *name;
59
60   store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));
61   gtk_tree_model_get_iter (GTK_TREE_MODEL (store), &iter, treepath);
62   gtk_tree_model_get (GTK_TREE_MODEL (store), &iter, COLUMN_NAME, &name, -1);
63   g_print ("item␣'%s'␣is␣clicked.\n", name);
64   g_free (name);
65 }
66
67 int
68 main (int argc, char *argv[])
69 {
70   GtkWidget *window;
71   GtkWidget *scroll_window;
72   GtkWidget *iconview;
73
74   gtk_init (&argc, &argv);
75
76   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
77   gtk_window_set_title (GTK_WINDOW (window), "GtkIconView␣Sample1");
78   gtk_widget_set_size_request (window, 500, 300);
79   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
80   g_signal_connect (G_OBJECT (window), "destroy",
81                     G_CALLBACK (gtk_main_quit), NULL);
82
83   scroll_window = gtk_scrolled_window_new (NULL, NULL);
84   gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW
85                                        (scroll_window),
86                                        GTK_SHADOW_ETCHED_IN);
87   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
88                                   GTK_POLICY_AUTOMATIC,
89                                   GTK_POLICY_AUTOMATIC);
90   gtk_container_add (GTK_CONTAINER (window), scroll_window);
91
92   iconview = create_icon_view_widget ();
93   gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview),
94                                  COLUMN_NAME);
95   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW(iconview),
96                                    COLUMN_PIXBUF);
97   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 128);
98   g_signal_connect (G_OBJECT (iconview), "item-activated",
99                     G_CALLBACK (cb_item_activated), NULL);
100
101  gtk_container_add (GTK_CONTAINER (scroll_window), iconview);
102
103  add_data (GTK_ICON_VIEW(iconview));
104
105  gtk_widget_show_all (window);
106  gtk_main ();
107
108  return 0;
109 }
```

サンプルプログラム その 2 ファイルブラウザ

次の例は，アイコンビューウィジェットを用いたファイルブラウザです．基本的な部分はさきほどの例と同様です．今回の例では表示用のラベルとアイコンだけではなく，そのファイルの絶対パスやそのファイルがディレクトリであるかどうかといったデータも保持するようにしています．

この例では，表示するファイルをディレクトリ，ファイルの順に表示して，さらにそれぞれをアルファベット順に並び替えるための設定を行っています．ソースコード中では 124 行目から 128 行目でソートに関する設定を行い，79 行目から 110 行目で実際にソートを行う関数を定義しています．

標準ソート関数の設定（124–125 行目）

関数 gtk_tree_sortable_set_default_sort_func で独自で定義するソート関数を設定します．

```
void gtk_tree_sortable_set_default_sort_func
                                    (GtkTreeSortable        *sortable,
                                     GtkTreeIterCompareFunc sort_func,
                                     gpointer               user_data,
                                     GtkDestroyNotify       destroy);
```

ソートに使用する項目の設定（126–128 行目）

関数 gtk_tree_sortable_set_sort_column_id でどの項目でソートを行うかを設定します．関数では何番目の項目でソートを行うか，また昇順もしくは降順でソートするかを設定します．ソートの昇順，降順の指定は GtkSortType で指定します（表 7.30 を参照）．第 2 引数に GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID を指定すると，関数 gtk_tree_sortable_set_default_sort_func に指定した関数を用いてソートを行います．

```
void gtk_tree_sortable_set_sort_column_id (GtkTreeSortable *sortable,
                                           gint             sort_column_id,
                                           GtkSortType      order);
```

表 7.30　GtkSortType の種類

値	説明
GTK_SORT_ASCENDING	昇順にソート
GTK_SORT_DESCENDING	降順にソート

ソース 7–7–5 アイコンビューウィジェットのサンプルプログラム その 2：gtkiconview-sample2.c

```
1  #include <gtk/gtk.h>
2
3  enum
4  {
5    COLUMN_PATH,
6    COLUMN_DISPLAY_NAME,
7    COLUMN_PIXBUF,
8    COLUMN_IS_DIRECTORY,
9    N_COLUMNS
10 };
11
12 static gchar *currentdir = NULL;
13
14 static void
15 cb_quit (GtkWidget *widget, gpointer user_data)
16 {
17   g_free (currentdir);
```

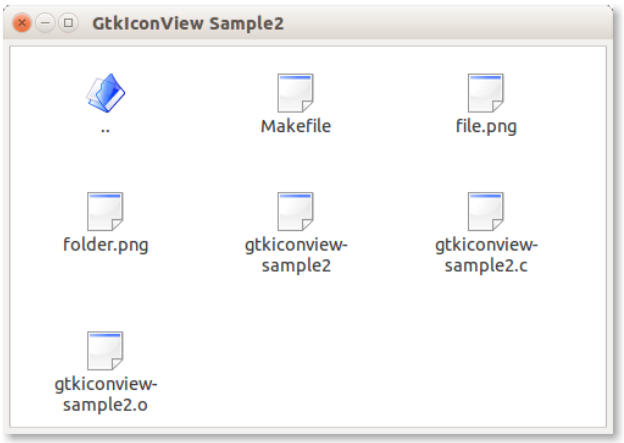


図 7.38　簡易ファイルブラウザ

```
18   gtk_main_quit ();
19 }
20
21 static void
22 add_data (GtkIconView *iconview)
23 {
24   GdkPixbuf     *folder_pixbuf;
25   GdkPixbuf     *file_pixbuf;
26   GtkListStore  *store;
27   GDir          *dir;
28   const gchar   *name;
29   GtkTreeIter   iter;
30   gchar         *path;
31   gchar         *display_name;
32   gboolean      is_dir;
33
34   file_pixbuf   = gdk_pixbuf_new_from_file ("./file.png", NULL);
35   folder_pixbuf = gdk_pixbuf_new_from_file ("./folder.png", NULL);
36
37   store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));
38
39   gtk_list_store_clear (store);
40
41   dir = g_dir_open (currentdir, 0, NULL);
42   if (!dir) return;
43
44   while (name = g_dir_read_name (dir))
45     {
46       if (name[0] != '.')
47         {
48           path = g_build_filename (currentdir, name, NULL);
49           is_dir = g_file_test (path, G_FILE_TEST_IS_DIR);
50           display_name = g_filename_to_utf8 (name, -1, NULL, NULL, NULL);
51
52           gtk_list_store_append (store, &iter);
53           gtk_list_store_set (store, &iter,
54                               COLUMN_PATH, path,
55                               COLUMN_DISPLAY_NAME, display_name,
56                               COLUMN_IS_DIRECTORY, is_dir,
57                               COLUMN_PIXBUF,
58                               (is_dir) ? folder_pixbuf : file_pixbuf,
59                               -1);
60           g_free (path);
61           g_free (display_name);
62         }
63     }
64   g_dir_close (dir);
65
66   if (g_utf8_collate (currentdir, "/") != 0)
67     {
68       gtk_list_store_append (store, &iter);
69       gtk_list_store_set (store, &iter,
70                           COLUMN_PATH, g_path_get_dirname (currentdir),
71                           COLUMN_DISPLAY_NAME, "..",
72                           COLUMN_IS_DIRECTORY, TRUE,
73                           COLUMN_PIXBUF, folder_pixbuf, -1);
74     }
75   g_object_unref (folder_pixbuf);
76   g_object_unref (file_pixbuf);
77 }
78
79 static gint
80 sort_func (GtkTreeModel *model,
81            GtkTreeIter  *a,
82            GtkTreeIter  *b,
83            gpointer     user_data)
84 {
85   gboolean is_dir_a, is_dir_b;
86   gchar    *name_a, *name_b;
87   int      result;
88
89   gtk_tree_model_get (model, a,
90                       COLUMN_IS_DIRECTORY, &is_dir_a,
91                       COLUMN_DISPLAY_NAME, &name_a,
92                       -1);
93   gtk_tree_model_get (model, b,
```

```
94                      COLUMN_IS_DIRECTORY, &is_dir_b,
95                      COLUMN_DISPLAY_NAME, &name_b,
96                      -1);
97    if (!is_dir_a && is_dir_b)
98      {
99        result = 1;
100     }
101   else if (is_dir_a && !is_dir_b)
102     {
103       result = -1;
104     }
105   else
106     {
107       result = g_utf8_collate (name_a, name_b);
108     }
109   return result;
110 }
111
112 static GtkWidget*
113 create_icon_view_widget (void)
114 {
115   GtkWidget    *iconview;
116   GtkListStore *store;
117
118   currentdir = g_get_current_dir ();
119   store = gtk_list_store_new (N_COLUMNS,
120                               G_TYPE_STRING,
121                               G_TYPE_STRING,
122                               GDK_TYPE_PIXBUF,
123                               G_TYPE_BOOLEAN);
124   gtk_tree_sortable_set_default_sort_func (GTK_TREE_SORTABLE (store),
125                                            sort_func, NULL, NULL);
126   gtk_tree_sortable_set_sort_column_id
127     (GTK_TREE_SORTABLE (store),
128      GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID, GTK_SORT_ASCENDING);
129
130   iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL (store));
131   g_object_unref (store);
132
133   return iconview;
134 }
135
136 static void
137 cb_item_activated (GtkIconView *iconview,
138                    GtkTreePath *treepath,
139                    gpointer    user_data)
140 {
141   GtkListStore *store;
142   GtkTreeIter  iter;
143   gchar        *path;
144   gboolean     is_dir;
145
146   store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));
147   gtk_tree_model_get_iter (GTK_TREE_MODEL (store), &iter, treepath);
148   gtk_tree_model_get (GTK_TREE_MODEL (store), &iter,
149                       COLUMN_PATH, &path,
150                       COLUMN_IS_DIRECTORY, &is_dir,
151                       -1);
152   if (is_dir)
153     {
154       g_free (currentdir);
155       currentdir = g_strdup (path);
156       add_data (iconview);
157     }
158   g_free (path);
159 }
160
161 int
162 main (int argc, char *argv[])
163 {
164   GtkWidget *window;
165   GtkWidget *scroll_window;
166   GtkWidget *iconview;
167
168   gtk_init (&argc, &argv);
169
```

```
170   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
171   gtk_window_set_title (GTK_WINDOW (window), "GtkIconView␣Property");
172   gtk_widget_set_size_request (window, 500, 300);
173   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
174   g_signal_connect (G_OBJECT (window), "destroy",
175                     G_CALLBACK (cb_quit), NULL);
176
177   scroll_window = gtk_scrolled_window_new (NULL, NULL);
178   gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW
179                                        (scroll_window),
180                                        GTK_SHADOW_ETCHED_IN);
181   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
182                                   GTK_POLICY_AUTOMATIC,
183                                   GTK_POLICY_AUTOMATIC);
184   gtk_container_add (GTK_CONTAINER (window), scroll_window);
185
186   iconview = create_icon_view_widget ();
187   gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview),
188                                  COLUMN_DISPLAY_NAME);
189   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW (iconview),
190                                    COLUMN_PIXBUF);
191   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 80);
192   g_signal_connect (G_OBJECT (iconview), "item-activated",
193                     G_CALLBACK (cb_item_activated), NULL);
194
195   gtk_icon_view_set_margin (GTK_ICON_VIEW (iconview), 16);
196   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 80);
197   gtk_icon_view_set_spacing (GTK_ICON_VIEW (iconview), 16);
198   gtk_icon_view_set_spacing (GTK_ICON_VIEW (iconview), 16);
199   gtk_icon_view_set_column_spacing (GTK_ICON_VIEW (iconview), 32);
200   gtk_icon_view_set_row_spacing (GTK_ICON_VIEW (iconview), 32);
201
202   gtk_container_add (GTK_CONTAINER (scroll_window), iconview);
203
204   add_data (GTK_ICON_VIEW (iconview));
205
206   gtk_widget_show_all (window);
207   gtk_main ();
208
209   return 0;
210 }
```

7.7.6 エントリ補完ウィジェット

エントリ補完ウィジェット（**GtkEntryCompletion**）は，エントリに入力を行う際に入力文字列の補完を行うウィジェットです．よく利用される文字列補完の例は，ファイル選択ダイアログで入力した文字列と一致するファイル名を自動的に補完するものです．

オブジェクトの階層構造

```
GObject
 +----GtkEntryCompletion
```

ウィジェットの作成

エントリ補完ウィジェットの作成には関数 gtk_entry_completion_new を使用します．

```
GtkEntryCompletion* gtk_entry_completion_new (void);
```

モデルの作成

モデルの作成には関数 gtk_list_store_new を使用します．最低限の動作には G_TYPE_STRING を設定する必要があります．そして作成したモデルを，関数 gtk_entry_completion_set_model を使用して登録します．実際の使用例は，サンプルプログラムのソースコードを参照してください．

```
void gtk_entry_completion_set_model (GtkEntryCompletion *completion,
                                     GtkTreeModel       *model);

void
```

```
gtk_entry_completion_set_text_column (GtkEntryCompletion *completion,
                                      gint                column);
```

サンプルプログラム

ソース 7–7–6 にエントリ補完ウィジェットのサンプルプログラムのソースコードを示します．この例では”a”, ”ai”, ”aiu”という 3 つの文字列を補完用の文字列としてあらかじめ登録しています．入力する文字列によって図 7.39 のように補完文字列の候補が表示されます．

ソース 7–7–6 エントリ補完ウィジェットのサンプルプログラム その 1：gtkientrycompletion-sample1.c

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

enum
{
  COLUMN_COMPLETION_TEXT,
  N_COLUMNS
};

static GtkEntryCompletion*
create_completion_widget (void)
{
  GtkEntryCompletion *completion;
  GtkListStore       *store;
  GtkTreeIter        iter;

  completion = gtk_entry_completion_new ();
  store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING);
  gtk_entry_completion_set_model (completion, GTK_TREE_MODEL (store));
  g_object_unref (store);
  gtk_entry_completion_set_text_column (completion, 0);

  gtk_list_store_append (store, &iter);
  gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "a", -1);

  gtk_list_store_append (store, &iter);
  gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "ai", -1);

  gtk_list_store_append (store, &iter);
  gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "aiu", -1);

  return completion;
}

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

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

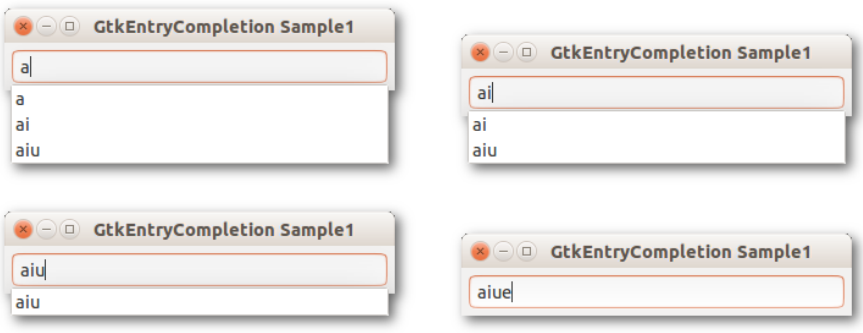


図 7.39　エントリ補完ウィジェットのサンプルプログラム 1

```
44   gtk_window_set_title (GTK_WINDOW (window), "GtkEntryCompletion␣Sample1");
45   gtk_widget_set_size_request (window, 320, -1);
46   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
47   g_signal_connect (G_OBJECT (window), "destroy",
48                     G_CALLBACK (gtk_main_quit), NULL);
49
50   entry = gtk_entry_new ();
51   gtk_container_add (GTK_CONTAINER (window), entry);
52
53   completion = create_completion_widget ();
54   gtk_entry_set_completion (GTK_ENTRY (entry), completion);
55   g_object_unref (completion);
56
57   gtk_widget_show_all (window);
58   gtk_main ();
59
60   return 0;
61 }
```

サンプルプログラム その 2 ファイルブラウザ

次の例（ソース 7–7–7）は，アイコンビューウィジェットのサンプルプログラム 2 （ソース 7–7–5）のファイルブラウザに，エントリ補完ウィジェットを追加したものです．

87–88 行目でそのディレクトリ内のファイル名をエントリ補完ウィジェットに登録しているので，エントリウィジェットにキーボードから入力を行うと，入力した文字列に応じてディレクトリ内のファイル名の候補が表示されます．このとき，31–33 行目で関数 gtk_tree_sortable_set_sort_column_id で補完文字列でソートするように設定しているので，補完候補の文字列は自動的にソートされて表示されます（図 7.40）．

ソース 7-7-7 エントリ補完ウィジェットのサンプルプログラム その 2：gtkientrycompletion-sample2.c

```
1  #include <gtk/gtk.h>
2
3  enum
4  {
5    COLUMN_COMPLETION_TEXT,
6    N_COMPLETION_COLUMNS
7  };
8
9  enum
10 {
11   COLUMN_PATH,
12   COLUMN_DISPLAY_NAME,
13   COLUMN_PIXBUF,
14   COLUMN_IS_DIRECTORY,
15   N_ICONVIEW_COLUMNS
16 };
17
18 static GdkPixbuf *file_pixbuf, *folder_pixbuf;
19 static gchar* currentdir;
20
```

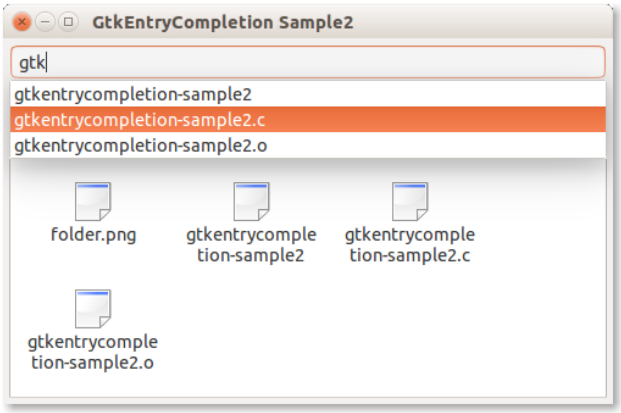


図 7.40　エントリ補完ウィジェットのサンプルプログラム 2

```
21 static GtkEntryCompletion*
22 create_completion_widget (void)
23 {
24   GtkEntryCompletion *completion;
25   GtkTreeModel       *completion_model;
26
27   completion = gtk_entry_completion_new ();
28   completion_model =
29     GTK_TREE_MODEL (gtk_list_store_new (N_COMPLETION_COLUMNS,
30                                         G_TYPE_STRING));
31   gtk_tree_sortable_set_sort_column_id
32     (GTK_TREE_SORTABLE (completion_model),
33      COLUMN_COMPLETION_TEXT, GTK_SORT_ASCENDING);
34   gtk_entry_completion_set_model (completion, completion_model);
35   g_object_unref (completion_model);
36   gtk_entry_completion_set_text_column (completion, 0);
37
38   return completion;
39 }
40
41 static void
42 load_pixbuf (void)
43 {
44   file_pixbuf   = gdk_pixbuf_new_from_file ("file.png", NULL);
45   folder_pixbuf = gdk_pixbuf_new_from_file ("folder.png", NULL);
46 }
47
48 static void
49 add_data (GtkIconView        *iconview,
50           GtkEntryCompletion *completion)
51 {
52   GtkListStore  *iconview_store, *completion_store;
53   GDir          *dir;
54   const gchar   *name;
55   GtkTreeIter   iter;
56   gchar         *path, *display_name;
57   gboolean      is_dir;
58
59   iconview_store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));
60   completion_store
61     = GTK_LIST_STORE (gtk_entry_completion_get_model (completion));
62
63   gtk_list_store_clear (iconview_store);
64   gtk_list_store_clear (completion_store);
65
66   dir = g_dir_open (currentdir, 0, NULL);
67   if (!dir) return;
68
69   while (name = g_dir_read_name (dir))
70     {
71       if (name[0] != '.')
72         {
73           path = g_build_filename (currentdir, name, NULL);
74           is_dir = g_file_test (path, G_FILE_TEST_IS_DIR);
75           display_name = g_filename_to_utf8 (name, -1, NULL, NULL, NULL);
76
77           gtk_list_store_append (iconview_store, &iter);
78           gtk_list_store_set (iconview_store, &iter,
79                               COLUMN_PATH, path,
80                               COLUMN_DISPLAY_NAME, display_name,
81                               COLUMN_IS_DIRECTORY, is_dir,
82                               COLUMN_PIXBUF,
83                               (is_dir) ? folder_pixbuf : file_pixbuf,
84                               -1);
85
86           gtk_list_store_append (completion_store, &iter);
87           gtk_list_store_set (completion_store, &iter,
88                               COLUMN_COMPLETION_TEXT, display_name, -1);
89           g_free (path);
90           g_free (display_name);
91         }
92     }
93   g_dir_close (dir);
94
95   if (g_utf8_collate (currentdir, "/") != 0)
96     {
```

```
97        gtk_list_store_append (iconview_store, &iter);
98        gtk_list_store_set (iconview_store, &iter,
99                            COLUMN_PATH, g_path_get_dirname (currentdir),
100                           COLUMN_DISPLAY_NAME, "..",
101                           COLUMN_IS_DIRECTORY, TRUE,
102                           COLUMN_PIXBUF, folder_pixbuf, -1);
103     }
104 }
105
106 static gint
107 sort_func (GtkTreeModel *model,
108            GtkTreeIter  *a,
109            GtkTreeIter  *b,
110            gpointer     data)
111 {
112   gboolean is_dir_a, is_dir_b;
113   gchar    *name_a, *name_b;
114   int      result;
115
116   gtk_tree_model_get (model, a,
117                       COLUMN_IS_DIRECTORY, &is_dir_a,
118                       COLUMN_DISPLAY_NAME, &name_a, -1);
119   gtk_tree_model_get (model, b,
120                       COLUMN_IS_DIRECTORY, &is_dir_b,
121                       COLUMN_DISPLAY_NAME, &name_b, -1);
122   if (!is_dir_a && is_dir_b)
123     {
124       result = 1;
125     }
126   else if (is_dir_a && !is_dir_b)
127     {
128       result = -1;
129     }
130   else
131     {
132       result = g_utf8_collate (name_a, name_b);
133     }
134   return result;
135 }
136
137 static GtkWidget*
138 create_icon_view_widget (void)
139 {
140   GtkWidget    *iconview;
141   GtkListStore *store;
142
143   currentdir = g_get_current_dir ();
144   store = gtk_list_store_new (N_ICONVIEW_COLUMNS,
145                               G_TYPE_STRING,
146                               G_TYPE_STRING,
147                               GDK_TYPE_PIXBUF,
148                               G_TYPE_BOOLEAN);
149   gtk_tree_sortable_set_default_sort_func (GTK_TREE_SORTABLE (store),
150                                            sort_func, NULL, NULL);
151   gtk_tree_sortable_set_sort_column_id
152     (GTK_TREE_SORTABLE (store),
153      GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID, GTK_SORT_ASCENDING);
154
155   iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL (store));
156   g_object_unref (store);
157
158   return iconview;
159 }
160
161 static void
162 cb_item_activated (GtkIconView *iconview,
163                    GtkTreePath *treepath,
164                    gpointer    data)
165 {
166   GtkListStore          *store;
167   GtkTreeIter           iter;
168   gchar                 *path;
169   gboolean              is_dir;
170   GtkEntryCompletion    *completion;
171
172   store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));
173   gtk_tree_model_get_iter (GTK_TREE_MODEL (store), &iter, treepath);
```

```
174   gtk_tree_model_get (GTK_TREE_MODEL (store), &iter,
175                       COLUMN_PATH, &path,
176                       COLUMN_IS_DIRECTORY, &is_dir,
177                       -1);
178   if (is_dir)
179     {
180       g_free (currentdir);
181       currentdir = g_strdup (path);
182       add_data (iconview, GTK_ENTRY_COMPLETION (data));
183     }
184   g_free (path);
185 }
186
187 int
188 main (int argc, char *argv[])
189 {
190   GtkWidget          *window;
191   GtkWidget          *vbox;
192   GtkWidget          *entry;
193   GtkWidget          *scrolled_window;
194   GtkWidget          *iconview;
195   GtkEntryCompletion *completion;
196
197   gtk_init (&argc, &argv);
198
199   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
200   gtk_window_set_title (GTK_WINDOW (window),
201                         "GtkEntryCompletion␣Sample2");
202   gtk_widget_set_size_request (window, 500, 300);
203   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
204   g_signal_connect (G_OBJECT (window), "destroy",
205                     G_CALLBACK (gtk_main_quit), NULL);
206
207   vbox = gtk_vbox_new (FALSE, 0);
208   gtk_container_add (GTK_CONTAINER (window), vbox);
209
210   entry = gtk_entry_new ();
211   gtk_box_pack_start (GTK_BOX (vbox), entry, FALSE, FALSE, 0);
212
213   completion = create_completion_widget ();
214   gtk_entry_set_completion (GTK_ENTRY (entry), completion);
215   g_object_unref (completion);
216
217   scrolled_window = gtk_scrolled_window_new (NULL, NULL);
218   gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW
219                                        (scrolled_window),
220                                        GTK_SHADOW_ETCHED_IN);
221   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),
222                                   GTK_POLICY_AUTOMATIC,
223                                   GTK_POLICY_AUTOMATIC);
224   gtk_box_pack_start (GTK_BOX (vbox), scrolled_window, TRUE, TRUE, 0);
225
226   load_pixbuf ();
227
228   iconview = create_icon_view_widget ();
229   gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview),
230                                  COLUMN_DISPLAY_NAME);
231   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW (iconview),
232                                    COLUMN_PIXBUF);
233   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 80);
234   g_signal_connect (G_OBJECT (iconview), "item_activated",
235                     G_CALLBACK (cb_item_activated), completion);
236   gtk_container_add (GTK_CONTAINER (scrolled_window), iconview);
237
238   add_data (GTK_ICON_VIEW (iconview), completion);
239
240   gtk_widget_show_all (window);
241   gtk_main ();
242
243   return 0;
244 }
```

7.7.7　オーバーレイ

オーバーレイウィジェット (**GtkOverlay**) は，ウィジェットの上に別のウィジェットを重ねて表示することを可能にするウィジェットです（図 7.41）．

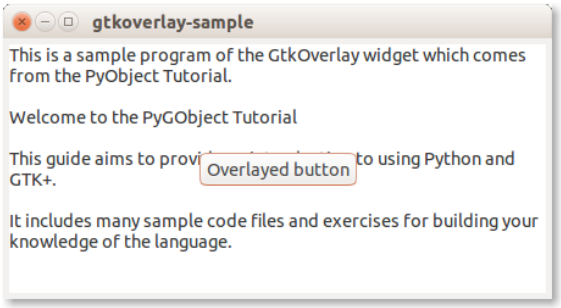


図 7.41　オーバーレイウィジェット

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +---GtkWidget
            +--- GtkContainer
                  +--- GtkBin
                        +--- GtkOverlay
```

ウィジェットの作成

オーバレイウィジェットを作成するには，関数 gtk_overlay_new を使用します．

```
GtkWidget* gtk_overlay_new (void);
```

オーバーレイの設定

ウィジェットのオーバーレイを実現するには，オーバーレイウィジェットに対して，オーバーレイされるウィジェットとオーバレイするウィジェットの二つのウィジェットを設定する必要があります．これらのウィジェットの関係を図 7.42 に示します．

オーバーレイされるウィジェットは，関数 gtk_container_add を使用して，オーバーレイウィジェット内に配置します．また，オーバーレイするウィジェットは，関数 gtk_overlay_add_overlay を使用して登録します．

```
void gtk_overlay_add_overlay (GtkOverlay *overlay, GtkWidget *widget);
```

第 1 引数：　オーバーレイウィジェット
第 2 引数：　オーバーレイするウィジェット

サンプルプログラム

ソース 7–7–8 にオーバーレイウィジェットのサンプルプログラムを示します[*1]．このプログラムは，テキストビューウィジェットの上にボタンウィジェットをオーバーレイしています．

ソース 7–7–8　オーバーレイウィジェットのサンプルプログラム：gtkoverlay-sample.c

```
1 #include <gtk/gtk.h>
2
3 int
4 main (int argc, char *argv[])
5 {
```

[*1] このサンプルプログラムは，Python GTK+3 Tutorial を参照させていただきました．

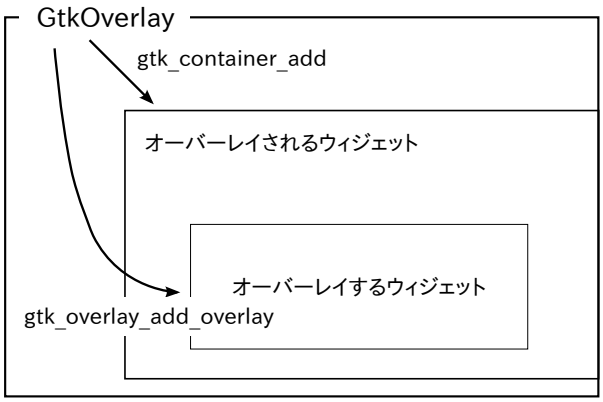


図 7.42　オーバーレイウィジェットに対するウィジェット設定

```
 6   GtkWidget     *window;
 7   GtkWidget     *overlay;
 8   GtkWidget     *textview;
 9   GtkTextBuffer *buffer;
10   GtkButton     *button;
11
12   gtk_init (&argc, &argv);
13
14   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
15   gtk_widget_set_size_request (window, 300, 200);
16   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
17
18   overlay = gtk_overlay_new ();
19   gtk_container_add (GTK_CONTAINER (window), overlay);
20
21   textview = gtk_text_view_new ();
22   gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW(textview), GTK_WRAP_WORD_CHAR);
23   buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW(textview));
24   gtk_text_buffer_set_text (buffer,
25                             "This␣is␣a␣sample␣program␣of␣the␣GtkOverlay␣widget␣which␣
                                 comes␣from␣the␣PyObject␣Tutorial.\n\nWelcome␣to␣the␣
                                 PyGObject␣Tutorial\n\nThis␣guide␣aims␣to␣provide␣an␣
                                 introduction␣to␣using␣Python␣and␣GTK+.\n\nIt␣includes␣
                                 many␣sample␣code␣files␣and␣exercises␣for␣building␣your␣
                                 knowledge␣of␣the␣language.", -1);
26   gtk_container_add (GTK_CONTAINER (overlay), textview);
27
28   button = gtk_button_new_with_label ("Overlayed␣button");
29   gtk_widget_set_halign (button, GTK_ALIGN_CENTER);
30   gtk_widget_set_valign (button, GTK_ALIGN_CENTER);
31
32   gtk_overlay_add_overlay (GTK_OVERLAY (overlay), button);
33
34   gtk_widget_show_all (window);
35
36   gtk_main ();
37
38   return 0;
39 }
```

7.7.8　リビーラー

リビーラーウィジェット (**GtkRevealer**) は, ウィジェットの表示/非表示を切り替えることのできるウィジェットです（図 7.43）. このウィジェットの特徴はウィジェットを表示したり非表示にするときに, 上下左右からウィジェットをスライドするようにアニメーションさせることができることです.

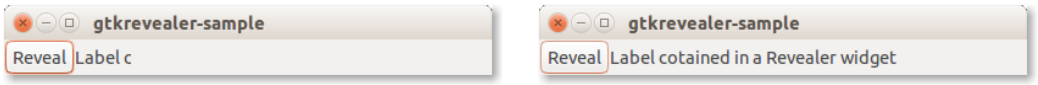


図 7.43　リビーラーウィジェット

オブジェクトの階層構造

```
GObject
 +----GInitiallyUnowned
       +---GtkWidget
            +--- GtkContainer
                  +--- GtkBin
                        +--- GtkRevealer
```

ウィジェットの作成

リビーラーウィジェットを作成するには, 関数 gtk_revealer_new を使用します.

```
GtkWidget* gtk_revealer_new (void);
```

ウィジェットの表示/非表示

ウィジェットの表示/非表示切り替えは関数 gtk_revealer_set_reveal_child によって行います.

```
void gtk_revealer_set_reveal_child (GtkRevealer *revealer,
                                    gboolean     reveal_child);
```

第 1 引数：　リビーラーウィジェット
第 2 引数：　表示 (TRUE) or 非表示 (FALSE)

ウィジェットが表示状態か非表示状態かどうかを調べるには, 関数 gtk_revealer_get_reveal_child を使用します.

```
gboolean gtk_revealer_set_reveal_child (GtkRevealer *revealer);
```

第 1 引数：　リビーラーウィジェット
戻り値：　表示 (TRUE) or 非表示 (FALSE)

ウィジェットのプロパティ設定

- アニメーション時間
  ウィジェットが表示（もしくは非表示）されるまでのアニメーション時間です. アニメーション時間の取得, 設定にはそれぞれ, 関数 gtk_revealer_get_transition_duration, 関数 gtk_revealer_set_transition_duration を使用します.

```
guint gtk_revealer_get_transition_duration (GtkRevealer *revealer);

void gtk_revealer_set_transition_duration (GtkRevealer *revealer,
                                           gunit       duration);
```

- アニメーションの方向
  アニメーションの方向の取得, 設定を関数 gtk_revealer_get_transition_type, 関数 gtk_revealer_set_transition_type によって行います.

```
GtkRevealerTransitionType
gtk_revealer_get_transition_type (GtkRevealer *revealer);
```

```
void gtk_revealer_set_transition_type (GtkRevealer *revealer,
                                       GtkRevealerTransitionType transition);
```

GtkRevealerTransitionType は表 7.31 に示した値から指定します.

表 7.31　GtkRevealerTransitionType の値

シグナル	説明
GTK_REVEALER_TRANSITION_TYPE_NONE	アニメーションなし
GTK_REVEALER_TRANSITION_TYPE_CROSSFADE	フェイドイン
GTK_REVEALER_TRANSITION_TYPE_SLIDE_RIGHT	左から右にアニメーション
GTK_REVEALER_TRANSITION_TYPE_SLIDE_LEFT	右から左にアニメーション
GTK_REVEALER_TRANSITION_TYPE_UP	下から上にアニメーション
GTK_REVEALER_TRANSITION_TYPE_DOWN	上から下にアニメーション

サンプルプログラム

ソース 7–7–9 にリビーラーウィジェットのサンプルプログラムを示します[*2]. このプログラムでは, ボタンクリックをトリガーにして, リビーラーウィジェット内に配置したラベルウィジェットの表示/非表示切り替えを行っています.

ソース 7–7–9 リビーラーウィジェットのサンプルプログラム：gtkrevealer-sample.c

```
1  #include <gtk/gtk.h>
2
3  static gboolean
4  reveal_child (GtkWidget* button,
5                gpointer   user_data)
6  {
7    GtkRevealer* revealer = (GtkRevealer *) user_data;
8
9    if (gtk_revealer_get_reveal_child (revealer))
10     {
11       gtk_revealer_set_reveal_child (GTK_REVEALER (revealer), FALSE);
12     }
13   else
14     {
15       gtk_revealer_set_reveal_child (GTK_REVEALER (revealer), TRUE);
16     }
17 }
18
19 int
20 main (int argc, char *argv[])
21 {
22   GtkWidget *window;
23   GtkWidget *grid;
24   GtkWidget *revealer;
25   GtkWidget *label;
26   GtkWidget *button;
27
28   gtk_init (&argc, &argv);
29
30   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
31   gtk_widget_set_size_request (window, 400, -1);
32   grid = gtk_grid_new ();
33   gtk_container_add (GTK_CONTAINER (window), grid);
34
35   revealer = gtk_revealer_new ();
36   gtk_revealer_set_reveal_child (GTK_REVEALER (revealer), FALSE);
37   gtk_revealer_set_transition_type (GTK_REVEALER (revealer),
38                                     GTK_REVEALER_TRANSITION_TYPE_SLIDE_LEFT);
39   gtk_grid_attach (GTK_GRID (grid), revealer, 1, 0, 1, 1);
40
41   label = gtk_label_new ("Label␣cotained␣in␣a␣Revealer␣widget");
42   gtk_container_add (GTK_CONTAINER (revealer), label);
43
44   button = gtk_button_new_with_label ("Reveal");
```

[*2] このサンプルプログラムは, Python GTK+3 Tutorial を参照させていただきました.

```
45   gtk_grid_attach (GTK_GRID (grid), button, 0, 0, 1, 1);
46   g_signal_connect (G_OBJECT (button), "clicked",
47                     G_CALLBACK (reveal_child), revealer);
48
49   gtk_widget_show_all (window);
50
51   gtk_main ();
52
53   return 0;
54 }
```

8 プログラミングの小箱

本章では以下に挙げるような，実際にアプリケーションを作成する際に役に立つちょっとしたテクニックを紹介します．

- マウスクリックの検出や座標の取得
- キープレスの検出とキーの取得
- ドラッグ&ドロップ
- プログラムオプションの解析

8.1 マウスクリックの検出と座標の取得

この節ではマウスクリックを検出したり，マウスがクリックされたときの座標を取得する方法などを説明します．

8.1.1 マウス情報検出の準備

ウィンドウ上でマウスクリックを検出したり，マウスをクリックしたときのカーソルの座標を取得するためには，そのウィジェットにマウスクリック等のイベントを検出するための設定を行う必要があります．イベント検出の設定を行うには次の関数を使用します．

```
void gtk_widget_set_events (GtkWidget *widget, gint events);
```

第 1 引数にはイベントを検出したいウィジェットを指定します．第 2 引数には列挙体 GdkEventMask で定義されたイベントを指定します．

```
typedef enum
{
  GDK_EXPOSURE_MASK             = 1 << 1,
  GDK_POINTER_MOTION_MASK       = 1 << 2,
  GDK_POINTER_MOTION_HINT_MASK  = 1 << 3,
  GDK_BUTTON_MOTION_MASK        = 1 << 4,
  GDK_BUTTON1_MOTION_MASK       = 1 << 5,
  GDK_BUTTON2_MOTION_MASK       = 1 << 6,
  GDK_BUTTON3_MOTION_MASK       = 1 << 7,
  GDK_BUTTON_PRESS_MASK         = 1 << 8,
  GDK_BUTTON_RELEASE_MASK       = 1 << 9,
  GDK_KEY_PRESS_MASK            = 1 << 10,
  GDK_KEY_RELEASE_MASK          = 1 << 11,
  GDK_ENTER_NOTIFY_MASK         = 1 << 12,
  GDK_LEAVE_NOTIFY_MASK         = 1 << 13,
  GDK_FOCUS_CHANGE_MASK         = 1 << 14,
```

```
  GDK_STRUCTURE_MASK            = 1 << 15,
  GDK_PROPERTY_CHANGE_MASK      = 1 << 16,
  GDK_VISIBILITY_NOTIFY_MASK    = 1 << 17,
  GDK_PROXIMITY_IN_MASK         = 1 << 18,
  GDK_PROXIMITY_OUT_MASK        = 1 << 19,
  GDK_SUBSTRUCTURE_MASK         = 1 << 20,
  GDK_SCROLL_MASK               = 1 << 21,
  GDK_TOUCH_MASK                = 1 << 22,
  GDK_SMOOTH_SCROLL_MASK        = 1 << 23,
  GDK_ALL_EVENTS_MASK           = 0x3FFFFE
} GdkEventMask;
```

8.1.2　マウスボタンのクリックを検出する

マウスクリックを検出するためには，前項に従って GDK_BUTTON_PRESS_MASK をウィジェットに設定します．次にウィジェットに **button-press-event シグナル**に対するコールバック関数を設定します．このシグナルに対するコールバック関数のプロトタイプ宣言は次のようになっています．

```
gboolean user_function (GtkWidget      *widget,
                        GdkEventButton *event,
                        gpointer       user_data);
```

このコールバック関数は標準で設定されているコールバック関数の後に呼び出されます．また，複数のコールバック関数が存在するときは，コールバック関数の戻り値が FALSE の場合，後に続くコールバック関数は呼び出されますが，戻り値が TRUE の場合は，後に続くコールバック関数は呼び出されません．

8.1.3　マウスの座標を取得する

マウスがクリックされたときの座標を取得するには，GdkEventButton 構造体のメンバ x と y を使用します．ウィンドウ上でマウスをクリックすると，その座標をターミナルに出力する例をソース 10–1–1 に示します．

マウスクリックを検出するためのイベントを 26 行目で設定し，27–28 行目でコールバック関数の設定を行っています．そして 8–9 行目で，マウスカーソルの座標を GdkEventButton 構造体のメンバ x と y によって出力しています．

```
typedef struct {
 GdkEventType type;
 GdkWindow    *window;
 gint8        send_event;
 guint32      time;
 gdouble      x;
 gdouble      y;
 gdouble      *axes;
 guint        state;
 guint        button;
 GdkDevice    *device;
 gdouble      x_root, y_root;
} GdkEventButton;
```

ソース 10–1–1 マウスクリックの検出と座標の取得のサンプルプログラム：mouse-tips1.c

```
1  #include <gtk/gtk.h>
2
3  static gboolean
4  cb_mouse_press (GtkWidget      *widget,
5                  GdkEventButton *event,
6                  gpointer       user_data)
7  {
8    g_print ("The␣mouse␣was␣clicked␣on␣the␣main␣window␣at␣(%3d,␣%3d).\n",
9             (int) event->x, (int) event->y);
10   return FALSE;
```

```
11 }
12
13 int
14 main (int argc, char *argv[])
15 {
16   GtkWidget *window;
17
18   gtk_init (&argc, &argv);
19
20   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
21   gtk_window_set_title (GTK_WINDOW (window), "Mouse␣Tips1");
22   gtk_widget_set_size_request (window, 300, 200);
23   g_signal_connect (G_OBJECT (window), "destroy",
24                     G_CALLBACK (gtk_main_quit), NULL);
25
26   gtk_widget_add_events (window, GDK_BUTTON_PRESS_MASK);
27   g_signal_connect (G_OBJECT (window), "button-press-event",
28                     G_CALLBACK (cb_mouse_press), NULL);
29
30   gtk_widget_show_all (window);
31   gtk_main ();
32
33   return 0;
34 }
```

8.1.4 マウスの移動を検出する

マウスの移動を検出するためには，GDK_POINTER_MOTION_MASK をウィジェットに設定し，次にウィジェットに **motion-notify-event シグナル** に対するコールバック関数を設定します．このシグナルに対するコールバック関数のプロトタイプ宣言は次のようになっています．

```
gboolean user_function (GtkWidget      *widget,
                        GdkEventMotion *event,
                        gpointer       user_data);
```

ソース 10–1–2 は，マウスの移動を検出してマウスカーソルの座標を出力するサンプルです．マウスボタンを押したままでマウスを移動しても検出できるように，GDK_BUTTON_PRESS_MASK も設定していることに注意してください．

ソース 10–1–2 マウス移動検出のサンプルプログラム：mouse-tips2.c

```
1  #include <gtk/gtk.h>
2
3  static gboolean
4  cb_mouse_move (GtkWidget      *widget,
5                 GdkEventMotion *event,
6                 gpointer       user_data)
7  {
8    g_print ("(%3d,␣%3d)\r", (int) event->x, (int) event->y);
9
10   return FALSE;
11 }
12
13 int
14 main (int argc, char *argv[])
15 {
16   GtkWidget *window;
17
18   gtk_init (&argc, &argv);
19
20   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
21   gtk_window_set_title (GTK_WINDOW (window), "Mouse␣Tips2");
22   gtk_widget_set_size_request (window, 300, 200);
23   g_signal_connect (G_OBJECT (window), "destroy",
24                     G_CALLBACK (gtk_main_quit), NULL);
25   gtk_widget_set_events (window,
26                          GDK_BUTTON_PRESS_MASK |
27                          GDK_POINTER_MOTION_MASK);
28   g_signal_connect (G_OBJECT (window), "motion-notify-event",
```

```
29                      G_CALLBACK (cb_mouse_move), NULL);
30
31    gtk_widget_show_all (window);
32    gtk_main ();
33
34    return 0;
35  }
```

8.1.5　マウスボタンの種類を判定する

マウスをクリックしたときにどのボタンが押されたのかを知るには，GdkEventButton 構造体メンバの button を使用します．また GdkEventButton 構造体メンバの state を使用すると，同時にコントロールキーやシフトキーが押されているかどうかを知ることができます．

button の値は，左ボタンを押したときに 1，中ボタンで 2，右ボタンで 3 となります．また state の値は，列挙体 GdkModifierType で定義された値になります．

ソース 10–1–3 は，マウスをクリックしたときに GdkEventButton の button の値と state の値を表示するプログラムです．

ソース 10–1–3 マウスボタンの種類検出のサンプルプログラム：mouse-tips3.c

```
1  #include <gtk/gtk.h>
2
3  static gboolean
4  cb_mouse_press (GtkWidget      *widget,
5                  GdkEventButton *event,
6                  gpointer       user_data)
7  {
8    g_print ("Button␣type=%d,␣State=%d\n", event->button, event->state);
9    return FALSE;
10 }
11
12 int
13 main (int argc, char *argv[])
14 {
15   GtkWidget *window;
16
17   gtk_init (&argc, &argv);
18
19   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
20   gtk_window_set_title (GTK_WINDOW (window), "Mouse␣Tips3");
21   gtk_widget_set_size_request (window, 300, 200);
22   g_signal_connect (G_OBJECT (window), "destroy",
23                     G_CALLBACK (gtk_main_quit), NULL);
24
25   gtk_widget_add_events (window, GDK_BUTTON_PRESS_MASK);
26   g_signal_connect (G_OBJECT (window), "button-press-event",
27                     G_CALLBACK (cb_mouse_press), NULL);
28
29   gtk_widget_show_all (window);
30   gtk_main ();
31
32   return 0;
33 }
```

8.1.6　ダブルクリックを検出する

マウスのダブルクリックを検出するには，GdkEventButton 構造体メンバの type を使用します．ダブルクリックは GdkEventType で，GDK_2BUTTON_PRESS = 5 と定義されています．

8.1.7　マウスカーソルを変更する

マウスカーソルを変更する手順は「カーソルデータの生成」と「カーソルの設定」の 2 段階からなります．
カーソルデータを生成するにはいくつかの方法がありますが，ここでは次の方法を説明します．

1. 既存のカーソルを使用する方法
2. 画像データなどからカーソルを生成する方法

既存のカーソルを使用する方法

既存のカーソルを使用するには関数 gdk_cursor_new_for_display を使用します．

```
GdkCursor* gdk_cursor_new_for_display (GdkDisplay     *display,
                                       GdkCursorType cursor_type);
```

この関数は 2 つの引数を取ります．1 つ目の引数には GdkDiplay 型の変数を指定しますが，これは関数 gdk_display_get_default により取得すればいいでしょう．

```
GdkDisplay* gdk_display_get_default (void);
```

次に 2 つ目の引数で使用するカーソルを指定します．これは次の GdkCursorType で定義されています．

```
typedef enum
{
  GDK_X_CURSOR            = 0,
  GDK_ARROW               = 2,
  GDK_BASED_ARROW_DOWN    = 4,
  ...
  GDK_XTERM               = 152,
  GDK_LAST_CURSOR,
  GDK_BLANK_CURSOR        = -2,
  GDK_CURSOR_IS_PIXMAP    = -1
} GdkCursorType;
```

すべての値を確認するには，devhelp で GdkCursorType を検索するとよいでしょう．

ソース 10–1–4 に，既存のカーソルを使用してマウスカーソルを変更する例を示します．

ソース 10–1–4 マウスカーソル変更のサンプルプログラム：mouse-tips5-1.c

```
1  #include <gtk/gtk.h>
2
3  int
4  main (int argc, char *argv[])
5  {
6    GtkWidget *window;
7    GdkCursor *cursor;
8
9    gtk_init (&argc, &argv);
10
11   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
12   gtk_window_set_title (GTK_WINDOW (window), "Mouse␣Tips5-1");
13   gtk_widget_set_size_request (window, 300, 200);
14   g_signal_connect (G_OBJECT (window), "destroy",
15                     G_CALLBACK (gtk_main_quit), NULL);
16   gtk_widget_show_all (window);
17
18   cursor  = gdk_cursor_new_for_display (gdk_display_get_default(),
19                                         GDK_X_CURSOR);
20   gdk_window_set_cursor (gtk_widget_get_window (window), cursor);
21
22   gtk_main ();
23
24   return 0;
25 }
```

GdkPixbuf データからマウスカーソルを生成する方法

関数 gdk_cursor_new_from_pixbuf を使うと画像からマウスカーソルを生成することができます．

```
GdkCursor *
gdk_cursor_new_from_pixbuf (GdkDisplay *display,
                            GdkPixbuf  *pixbuf,
                            gint        x,
                            gint        y);
```

第 1 引数	GdkDisplay 型変数へのポインタ
第 2 引数	GdkPixbuf 形式の画像データ
第 3 引数	ホットスポットの X 座標
第 4 引数	ホットスポットの Y 座標
戻り値	生成したマウスカーソルデータ

関数の第 3 引数と第 4 引数の (x, y) は，マウス画像中のホットスポット（矢印カーソルの矢印の先端のような点）の座標位置を指定します．

ソース 10–1–5 は，マウスボタンを押したときと離したときにマウスカーソルを変更するプログラムです．マウスボタンが離されたときのイベントを検出するために，100 行目で GDK_BUTTON_RELEASE_MASK を設定していることに注意してください．

ソース 10–1–5 GdkPixbuf 画像からマウスカーソルを生成するサンプルプログラム：mouse-tips5-2.c

```
1  #include <gtk/gtk.h>
2
3  GdkCursor *hand_open;
4  GdkCursor *hand_close;
5
6  static gboolean
7  cb_mouse_press (GtkWidget      *widget,
8                  GdkEventButton *event,
9                  gpointer       user_data)
10 {
11   gdk_window_set_cursor (gtk_widget_get_window (widget), hand_close);
12
13   return FALSE;
14 }
15
16 static gboolean
17 cb_mouse_release (GtkWidget      *widget,
18                   GdkEventButton *event,
19                   gpointer       user_data)
20 {
21   gdk_window_set_cursor (gtk_widget_get_window (widget), hand_open);
22
23   return FALSE;
24 }
25
26 int
27 main (int argc, char *argv[])
28 {
29   GtkWidget *window;
30   GdkPixbuf *pixbuf;
31
32   gtk_init (&argc, &argv);
33
34   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
35   gtk_window_set_title (GTK_WINDOW (window), "Mouse␣Tips5-2");
36   gtk_widget_set_size_request (window, 300, 200);
37   g_signal_connect (G_OBJECT (window), "destroy",
38                     G_CALLBACK (gtk_main_quit), NULL);
39
40   gtk_widget_add_events (window,
41                          GDK_BUTTON_PRESS_MASK |
42                          GDK_BUTTON_RELEASE_MASK);
43   g_signal_connect (G_OBJECT (window), "button-press-event",
44                     G_CALLBACK (cb_mouse_press), NULL);
45   g_signal_connect (G_OBJECT (window), "button-release-event",
46                     G_CALLBACK (cb_mouse_release), NULL);
47
48   pixbuf = gdk_pixbuf_new_from_file_at_size ("./all-scroll.svg",
49                                              32, 32, NULL);
50   hand_open  = gdk_cursor_new_from_pixbuf (gdk_display_get_default(),
51                                            pixbuf, 0, 0);
52   pixbuf = gdk_pixbuf_new_from_file_at_size ("./grabbing.svg",
53                                              32, 32, NULL);
54   hand_close = gdk_cursor_new_from_pixbuf (gdk_display_get_default(),
55                                            pixbuf, 0, 0);
56   gtk_widget_show_all (window);
```

```
57
58    gdk_window_set_cursor (gtk_widget_get_window (window), hand_open);
59
60    gtk_main ();
61
62    return 0;
63 }
```

8.2 キープレスの検出とキーの取得

この節ではキープレスの検出と押されたキーの取得方法を説明します．

8.2.1 キープレス検出の準備

ウィンドウ上でキーが押されたことを検出するには，前節のマウスのときと同様に，そのウィジェットにキープレス等のイベントを検出するための設定を行う必要があります．

イベント検出の設定を行うには，関数 を使用します．キーが押されたことを検出するためには GDK_KEY_PRESS_MASK を，キーが離されたことを検出するためには GDK_KEY_RELEASE_MASK を設定します．

8.2.2 キープレスを検出する

キープレスを検出してコールバック関数を呼び出すには，ウィジェットに **key-press-event シグナル**に対するコールバック関数を設定します．このシグナルに対するコールバック関数のプロトタイプ宣言は次のようになっています．

```
gboolean user_function (GtkWidget   *widget,
                        GdkEventKey *event,
                        gpointer    user_data);
```

8.2.3 キーを取得する

キーの種類を取得するには GdkEvenKey 構造体のメンバを使用します．GdkEventKey 構造体は次のように定義されています．

```
typedef struct {
  GdkEventType type;
  GdkWindow    *window;
  gint8        send_event;
  guint32      time;
  guint        state;
  guint        keyval;
  gint         length;
  gchar        *string;
  guint16      hardware_keycode;
  guint8       group;
} GdkEventKey;
```

keyval はキーの種類，state は同時に押されているコントロールキーやシフトキーの種類，string はキーの種類の文字列を表します．ソース 8–2 は，押されたキーに対する GdkEventKey の keyval，state，string の値を表示するプログラムです．

ソース 8–2 キープレスの検出とキーの取得のサンプルプログラム：key-tips1.c

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

static gboolean
cb_key_press (GtkWidget   *widget,
              GdkEventKey *event,
              gpointer    user_data)
{
  g_print ("keyval␣=␣%d,␣state␣=␣%d,␣string␣=␣%s\n",
           event->keyval, event->state, event->string);
  return FALSE;
}

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

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "Key␣Tips1");
  gtk_widget_set_size_request (window, 300, 200);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  gtk_widget_add_events (window, GDK_KEY_PRESS_MASK);
  g_signal_connect (G_OBJECT (window), "key-press-event",
                    G_CALLBACK (cb_key_press), NULL);

  gtk_widget_show_all (window);
  gtk_main ();

  return 0;
}
```

8.3 ドラッグ＆ドロップ

ドラッグ＆ドロップは，GUI アプリケーションにはなくてはならない便利な機能です．ここではドラッグ＆ドロップの詳しい原理には触れず，実際のアプリケーションでの実装例を紹介します．

8.3.1 ドロップ機能の実装

まずウィジェットをドラッグ＆ドロップのドロップ側，つまりデータの受け手として実装する方法を説明します．ウィジェットにドロップの機能を設定するのは非常に簡単です．ドロップ機能を設定したいウィジェットに対して，関数 gtk_drag_dest_set を使ってドロップの設定を行うだけです．

```
void gtk_drag_dest_set (GtkWidget            *widget,
                        GtkDestDefaults      flags,
                        const GtkTargetEntry *targets,
                        gint                 n_targets,
                        GdkDragAction        actions);
```

第 1 引数	ドロップ機能を設定するウィジェット
第 2 引数	ドラッグ＆ドロップ動作が生じたときに，どのような動作を行うか（表 8.1 参照）
第 3 引数	ドロップを受け付けるデータ形式（GtkTargetEntry 参照）
第 4 引数	ドロップを受け付けるデータ形式の数
第 5 引数	対応するドラッグアクション（表 8.3 参照）

引数の第 1 ウィジェットには，ドロップ機能を設定したいウィジェットを指定します．第 2 引数には，列挙体 **GtkDestDefault** で定義された値（表 8.1 を参照）を指定します．大抵の場合は，GTK_DEST_DEFAULT_ALL を指定しておけばよいでしょう．

表 8.1 GtkDestFafault の値

値	説明
GTK_DEST_DEFAULT_MOTION	ドラッグ動作中にドラッグ可能なウィジェットかどうか調べる.
GTK_DEST_DEFAULT_HIGHLIGHT	ドラッグ中にハイライト表示する.
GTK_DEST_DEFAULT_DROP	ドロップ動作が起きたときにドロップ可能なウィジェットかどうか調べる.
GTK_DEST_DEFAULT_ALL	上記をすべて行う.

第 3 引数はドロップされるデータとして受け付けるデータ形式を構造体 GtkTargetEntry 型の変数で指定し，第 4 引数にはその数を指定します.

```
typedef struct {
 gchar *target;
 guint  flags;
 guint  info;
} GtkTargetEntry;
```

target にはドロップを受け付けるデータの MIME 型を与えます．そして flags には，**GtkTargetFlags** で定義された値（表 8.2 を参照）もしくは 0 を与えます．0 を指定すると，ドラッグ側のアプリケーションやウィジェットを制限しません．info にはドロップされたデータを識別するための値を任意に指定します.

表 8.2 GtkTargetFlags の値

値	説明
GTK_TARGET_SAME_APP	同一のアプリケーションでのみ操作を許可する.
GTK_TARGET_SAME_WIDGET	同一のウィジェットでのみ操作を許可する.
GTK_TARGET_OTHER_APP	他のアプリケーションでも操作を許可する.
GTK_TARGET_OTHER_WIDGET	他のウィジェットでも操作を許可する.

関数の最後の引数には，コピーや移動などのどの操作に対して反応するかを，列挙体 **GdkDragAction** で定義された値（表 8.3 を参照）の論理和で指定します.

表 8.3 GdkDragAction の値

値	説明
GDK_ACTION_DEFAULT	何の意味も持たない（指定してはいけない）.
GDK_ACTION_COPY	データをコピーする.
GDK_ACTION_MOVE	データを移動する.
GDK_ACTION_LINK	リンクを作成する.
GDK_ACTION_PRIVATE	ソース側に動作を確認する.
GDK_ACTION_ASK	ユーザに動作を確認する.

drag-data-received シグナルに対するコールバック関数

以上のように関数 gtk_drag_dest_set を使ってドロップの設定を行えば，そのウィジェットにはドロップの機能が設定され，ドラッグデータを受け付けられるようになります.

しかし，このままでは受け取ったデータに対して何も操作を行えません．そこで，**drag-data-received シグナル**に対するコールバック関数を定義して，その関数内でドラッグデータを操作することにします．drag-data-received シグナルに対するコールバック関数のプロトタイプ宣言は次のようになっています.

```
void user_function (GtkWidget        *widget,
                    GdkDragContext   *drag_context,
                    gint             x,
                    gint             y,
                    GtkSelectionData *data,
```

```
                    guint           info,
                    guint           time,
                    gpointer        user_data);
```

いろいろな引数がありますが，基本的には第 6 引数の data と第 7 引数 info からドラッグデータの情報を得ることができます．第 7 引数の info には，構造体 GtkTargetEntry のメンバ info で指定した値が入ります．構造体 GtkSelectionData は次のように定義されています．

```
typedef struct {
 GdkAtom     selection;
 GdkAtom     target;
 GdkAtom     type;
 gint        format;
 guchar      *data;
 gint        length;
 GdkDisplay *display;
} GtkSelectionData;
```

ドロップデータを表示するサンプルプログラム

ソース 10–3–1 に，ドロップデータの情報をターミナルに表示するプログラムを示します（図 8.1）．いろいろなデータをドロップして，どのような情報が表示されるかを調べてみるといいでしょう．

ここでは window ウィジェットに対して，72–74 行目で関数 gtk_drag_dest_set によってドロップ機能の設定を行っています．12–19 行目で定義した変数 target_table で，Web データもドロップの対象としています．また，75–76 行目で drag-data-received シグナルに対するコールバック関数を設定しています．コールバック関数の実体は 23–56 行目です．

ドロップ処理が終了したときには，関数 gtk_drag_finish を呼び出します．

```
void gtk_drag_finish (GdkDragContext *context,
                      gboolean       success,
                      gboolean       del,
                      guint32        time_);
```

ドラッグ側でデータを削除するように促すには，第 3 引数を TRUE にします．

ソース 10-3-1 ドロップ機能の実装：dnd-sample1.c

```
1  #include <gtk/gtk.h>
2
3  enum
4  {
5    DROP_URI_LIST,
6    DROP_X_MOZ_URL,
7    DROP_HTML,
8    DROP_TEXT_PLAIN,
9    DROP_STRING
10 };
11
12 static GtkTargetEntry target_table[] =
13 {
14   {"text/uri-list",  0, DROP_URI_LIST},
15   {"text/x-moz-url", 0, DROP_X_MOZ_URL},
16   {"text/html",      0, DROP_HTML},
```

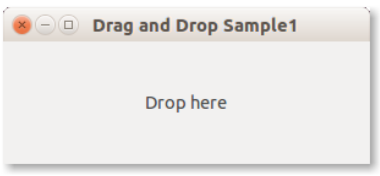


図 8.1　ドロップ操作のサンプル

```
17   {"text/plain",      0, DROP_TEXT_PLAIN},
18   {"STRING",          0, DROP_STRING}
19 };
20
21 static guint ntargets = sizeof (target_table) / sizeof (target_table[0]);
22
23 static void
24 cb_drag_data_received (GtkWidget          *widget,
25                        GdkDragContext     *context,
26                        gint               x,
27                        gint               y,
28                        GtkSelectionData   *data,
29                        guint              info,
30                        guint              time,
31                        gpointer           user_data)
32 {
33   gchar *received;
34
35   g_print ("data->target␣=␣%s\n",
36            gdk_atom_name (gtk_selection_data_get_target (data)));
37   g_print ("data->type␣␣␣=␣%s\n",
38            gdk_atom_name (gtk_selection_data_get_data_type (data)));
39   int length = gtk_selection_data_get_length (data);
40   g_print ("data->length␣=␣%d\n", length);
41   int format = gtk_selection_data_get_format (data);
42   g_print ("data->format␣=␣%d\n", format);
43
44   if (length > 0 && format == 8)
45     {
46       g_print ("Received␣string␣=␣");
47       received = g_strchomp ((gchar *) gtk_selection_data_get_data (data));
48       for (; *received != '\0'; )
49         {
50           g_print ("%c", *received++);
51         }
52       g_print ("\n");
53       gtk_drag_finish (context, TRUE, FALSE, time);
54     }
55   gtk_drag_finish (context, FALSE, FALSE, time);
56 }
57
58 int
59 main (int argc, char *argv[])
60 {
61   GtkWidget *window;
62   GtkWidget *label;
63
64   gtk_init (&argc, &argv);
65
66   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
67   gtk_window_set_title (GTK_WINDOW (window), "Drag␣and␣Drop␣Sample1");
68   gtk_widget_set_size_request (window, 300, 100);
69   g_signal_connect (G_OBJECT (window), "destroy",
70                     G_CALLBACK (gtk_main_quit), NULL);
71
72   gtk_drag_dest_set (window,
73                      GTK_DEST_DEFAULT_ALL, target_table, ntargets,
74                      GDK_ACTION_COPY | GDK_ACTION_MOVE);
75   g_signal_connect (window, "drag-data-received",
76                     G_CALLBACK (cb_drag_data_received), NULL);
77
78   label = gtk_label_new ("Drop␣here");
79   gtk_container_add (GTK_CONTAINER (window), label);
80
81   gtk_widget_show_all (window);
82   gtk_main ();
83
84   return 0;
85 }
```

8.3.2 ドラッグ機能の実装

次に，ドラッグ機能の実装について説明します．ウィジェットにドラッグ機能を設定するには，関数 gtk_drag_source_set を使用します．

```
void gtk_drag_source_set (GtkWidget            *widget,
                          GdkModifierType start_button_mask,
                          const GtkTargetEntry *targets,
                          gint                 n_targets,
                          GdkDragAction        actions);
```

第 1 引数	ドラッグ機能を設定するウィジェット
第 2 引数	装飾子（表??参照）
第 3 引数	ドロップを受け付けるデータ形式（GtkTargetEntry 参照）
第 4 引数	ドロップを受け付けるデータ形式の数
第 5 引数	対応するドラッグアクション（表 8.3 参照）

また, GtkIconView ウィジェットのそれぞれのアイコンにドラッグ機能を持たせるには, 関数 gtk_icon_view_model_drag_source を使用します.

```
void
gtk_icon_view_enable_model_drag_source (GtkIconView          *icon_view,
                                        GdkModifierType start_button_mask,
                                        const GtkTargetEntry *targets,
                                        gint                 n_targets,
                                        GdkDragAction        actions);
```

第 1 引数	アイコンビューウィジェット
第 2 引数	装飾子（表??参照）
第 3 引数	ドロップを受け付けるデータ形式（GtkTargetEntry 参照）
第 4 引数	ドロップを受け付けるデータ形式の数
第 5 引数	対応するドラッグアクション（表 8.3 参照）

この 2 つの関数は, 第 2 引数以降は全く同じです．第 2 引数ではどのボタンでドラッグを行うかを, 列挙体 GdkModifierType で定義された値で指定します．第 3 引数から第 5 引数までは，関数 gtk_drag_dest_set の引数と同様です.

drag-data-get シグナルに対するコールバック関数

そして，ドラッグが起こったときにドラッグ用のデータを設定するために，**drag-data-get シグナル**に対するコールバック関数を設定し，そのコールバック関数内でデータの設定を行います．drag-data-get シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります.

```
void user_function (GtkWidget        *widget,
                    GdkDragContext   *drag_context,
                    GtkSelectionData *data,
                    guint            info,
                    guint            time,
                    gpointer         user_data);
```

この関数の第 3 引数の data に対して，関数 gtk_selection_data_set を使用してドラッグデータを設定します.

```
void gtk_selection_data_set (GtkSelectionData *selection_data,
                             GdkAtom          type,
                             gint             format,
                             const guchar     *data,
                             gint             length);
```

第 1 引数	GtkSelectionData 型の変数
第 2 引数	データの種類
第 3 引数	データの形式
第 4 引数	データ
第 5 引数	データ長

第 1 引数には，drag-data-get シグナルに対するコールバック関数の第 3 引数 data を指定します．そして第 2 引数には，第 3 引数 data のメンバ target を指定します．第 3 引数はデータ 1 単位のビット数を指定します．通常は 8（ビット）とします．第 4 引数にはドラッグデータを指定し，第 5 引数にはその長さを指定します（ソース 10–3–2 の 107–109 行目を参照）．

ドラッグ機能を設定したサンプルプログラム

ソース 10–3–2 は，アイコンビューウィジェットの各アイコンにドラッグ機能を設定したプログラムです．図 8.2 はプログラムの実行画面です．上段のアイコンを下段のフレーム内にドラッグ&ドロップすると，アイコンのラベルをターミナルに出力します．

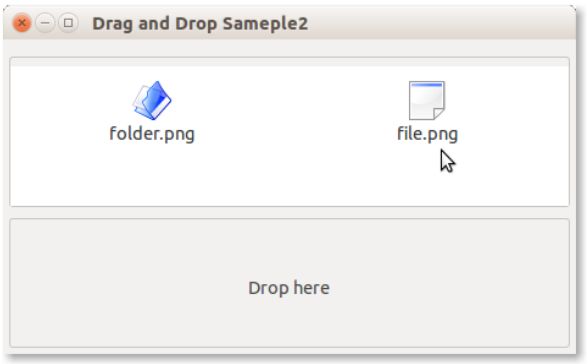


図 8.2　ドラッグ&ドロップアプリケーション

ソース 10-3-2 ドラッグ機能の実装：dnd-sample2.c

```
1  #include <gtk/gtk.h>
2  #include <string.h>
3
4  enum
5  {
6    DROP_URI_LIST,
7    DROP_X_MOZ_URL,
8    DROP_HTML,
9    DROP_TEXT_PLAIN,
10   DROP_STRING
11 };
12
13 static GtkTargetEntry target_table[] =
14 {
15   {"text/uri-list",  0, DROP_URI_LIST},
16   {"text/x-moz-url", 0, DROP_X_MOZ_URL},
17   {"text/html",      0, DROP_HTML},
18   {"text/plain",     0, DROP_TEXT_PLAIN},
19   {"STRING",         0, DROP_STRING}
20 };
21
22 static guint ntargets = sizeof (target_table) / sizeof (target_table[0]);
23
24 enum
25 {
26   COLUMN_NAME,
27   COLUMN_PIXBUF,
28   N_COLUMNS
29 };
30
31 static void
32 add_data (GtkIconView *iconview)
33 {
34   GdkPixbuf     *folder_pixbuf;
35   GdkPixbuf     *file_pixbuf;
36   GtkListStore *store;
37   GtkTreeIter   iter;
38
39   folder_pixbuf = gdk_pixbuf_new_from_file ("folder.png", NULL);
40   file_pixbuf   = gdk_pixbuf_new_from_file ("file.png", NULL);
```

```
41
42    store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));
43
44    gtk_list_store_clear (store);
45
46    gtk_list_store_append (store, &iter);
47    gtk_list_store_set (store, &iter,
48                        COLUMN_NAME, "folder.png",
49                        COLUMN_PIXBUF, folder_pixbuf, -1);
50    g_object_unref (folder_pixbuf);
51
52    gtk_list_store_append (store, &iter);
53    gtk_list_store_set (store, &iter,
54                        COLUMN_NAME, "file.png",
55                        COLUMN_PIXBUF, file_pixbuf, -1);
56    g_object_unref (file_pixbuf);
57  }
58
59  static GtkWidget*
60  create_icon_view_widget (void)
61  {
62    GtkWidget     *iconview;
63    GtkListStore  *store;
64
65    store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING, GDK_TYPE_PIXBUF);
66    iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL(store));
67    g_object_unref (store);
68
69    return iconview;
70  }
71
72  static void
73  source_drag_data_get (GtkWidget         *widget,
74                        GdkDragContext    *context,
75                        GtkSelectionData  *data,
76                        guint             info,
77                        guint             time,
78                        gpointer          user_data)
79  {
80    GList        *root, *list, *string_list = NULL;
81    GtkTreeModel *model;
82    GString      *string = NULL;
83
84    model = GTK_TREE_MODEL (gtk_icon_view_get_model (GTK_ICON_VIEW (widget)));
85    root = gtk_icon_view_get_selected_items (GTK_ICON_VIEW (widget));
86
87    for (list = root; list; list = g_list_next (list))
88      {
89        GtkTreePath *path;
90        GtkTreeIter iter;
91        gchar       *name;
92
93        path = (GtkTreePath *) list->data;
94        gtk_tree_model_get_iter (model, &iter, path);
95        gtk_tree_model_get (model, &iter, COLUMN_NAME, &name, -1);
96        string_list = g_list_append (string_list, name);
97      }
98    for (list = string_list; list; list = g_list_next (list))
99      {
100       if (!string) {
101         string = g_string_new ((const gchar *) string_list->data);
102       } else {
103         string = g_string_append (string, (const gchar *) list->data);
104       }
105       string = g_string_append (string, "\n");
106     }
107   gtk_selection_data_set (data,
108                           gtk_selection_data_get_target (data), 8,
109                           string->str, strlen (string->str));
110   g_string_free (string, FALSE);
111 }
112
113 static void
114 cb_drag_data_received (GtkWidget         *widget,
115                        GdkDragContext    *context,
116                        gint              x,
```

```
117                         gint              y,
118                         GtkSelectionData  *data,
119                         guint             info,
120                         guint             time)
121 {
122   char *received;
123   int   length, format;
124   int   n;
125
126   length = gtk_selection_data_get_length (data);
127   format = gtk_selection_data_get_format (data);
128   if ((length >= 0) && (format == 8))
129     {
130       g_print ("Received␣string␣=␣");
131       received = g_strchomp ((gchar *) gtk_selection_data_get_data (data));
132       for (; *received != '\0'; received++)
133         {
134           g_print ("%c", *received);
135         }
136       g_print ("\n");
137     }
138   gtk_drag_finish (context, FALSE, FALSE, time);
139 }
140
141 int
142 main (int argc, char *argv[])
143 {
144   GtkWidget *window;
145   GtkWidget *vbox;
146   GtkWidget *iconview;
147   GtkWidget *frame;
148   GtkWidget *label;
149
150   gtk_init (&argc, &argv);
151
152   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
153   gtk_window_set_title (GTK_WINDOW (window), "Drag␣and␣Drop␣Sameple2");
154   gtk_widget_set_size_request (window, 300, -1);
155   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
156   g_signal_connect (G_OBJECT (window), "destroy",
157                     G_CALLBACK (gtk_main_quit), NULL);
158
159   vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 0);
160   gtk_container_add (GTK_CONTAINER (window), vbox);
161
162   frame = gtk_frame_new ("");
163   gtk_box_pack_start (GTK_BOX (vbox), frame, TRUE, TRUE, 0);
164   gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_IN);
165
166   iconview = create_icon_view_widget ();
167   gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview), COLUMN_NAME);
168   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW (iconview),
169                                    COLUMN_PIXBUF);
170   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 128);
171
172   gtk_icon_view_enable_model_drag_source (GTK_ICON_VIEW (iconview),
173                                           GDK_BUTTON1_MASK,
174                                           target_table,
175                                           ntargets,
176                                           GDK_ACTION_COPY |
177                                           GDK_ACTION_MOVE);
178
179   gtk_icon_view_set_selection_mode (GTK_ICON_VIEW (iconview),
180                                     GTK_SELECTION_MULTIPLE);
181   g_signal_connect (iconview, "drag-data-get",
182                     G_CALLBACK (source_drag_data_get), NULL);
183   gtk_container_add (GTK_CONTAINER (frame), iconview);
184   add_data (GTK_ICON_VIEW (iconview));
185
186   frame = gtk_frame_new ("");
187   gtk_widget_set_size_request (frame, -1, 80);
188   gtk_box_pack_start (GTK_BOX (vbox), frame, TRUE, TRUE, 0);
189   gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_IN);
190
191   gtk_drag_dest_set (frame,
192                      GTK_DEST_DEFAULT_ALL, target_table, ntargets,
```

```
193                       GDK_ACTION_COPY | GDK_ACTION_MOVE);
194   g_signal_connect (frame, "drag-data-received",
195                     G_CALLBACK(cb_drag_data_received), NULL);
196
197   label = gtk_label_new ("Drop␣here");
198   gtk_container_add (GTK_CONTAINER (frame), label);
199
200   gtk_widget_show_all (window);
201   gtk_main ();
202
203   return 0;
204 }
```

8.3.3　GtkIconView ウィジェットでのドラッグ＆ドロップの実装

最後に，アイコンビューウィジェット間でアイコンをドラッグ＆ドロップすることにより，コピーおよび移動を行うプログラムを紹介します（ソース 10–3–3）．

図 8.3 上段はプログラムの実行初期の画面です．マウスの左ボタンでドラッグ＆ドロップを行うとアイコンをコピーし，シフトキーを押しながらマウスの左ボタンでドラッグ＆ドロップを行うとアイコンを移動します．

この例ではドラッグ＆ドロップの対象とするデータをこのアプリケーションのアイコンビューウィジェットに限定するために，7–8 行目の GtkTargetEntry のメンバ flags を GTK_TARGET_SAME_APP に設定しています．また，アクションが移動だった場合のために，**drag-data-delete シグナル**に対するコールバック関数を設定し，127–147 行目で定義されたコールバック関数内でドラッグ側のアイコンの削除を行っています．このプログラムでは 108–112 行目のように，データの受け取り側でアイコンを削除することも可能です．

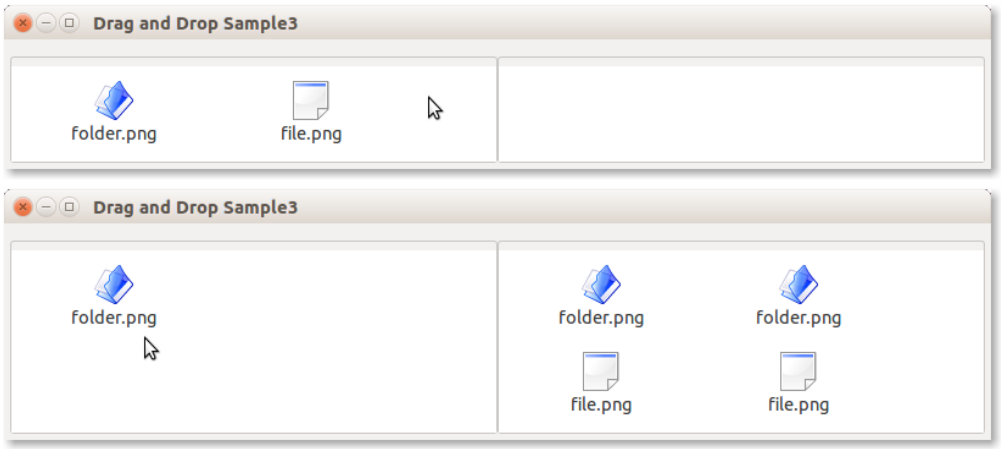


図 8.3　ドラッグ＆ドロップの実装

ソース 10-3-3 GtkIconView ウィジェットでのドラッグ機能の実装：dnd-sample3.c

```
1  #include <gtk/gtk.h>
2  #include <string.h>
3
4  static
5  GtkTargetEntry target_table[] =
6  {
7    {"STRING",  GTK_TARGET_SAME_APP, 0},
8    {"text/plain",  GTK_TARGET_SAME_APP, 0}
9  };
10
11 static guint ntargets = sizeof (target_table) / sizeof (target_table[0]);
12
13 enum
14 {
15   COLUMN_NAME,
16   COLUMN_PIXBUF,
17   N_COLUMNS
```

```
18 };
19
20 static void
21 source_drag_data_get (GtkWidget         *widget,
22                       GdkDragContext    *context,
23                       GtkSelectionData  *data,
24                       guint             info,
25                       guint             time,
26                       gpointer          user_data)
27 {
28   GList         *root, *list, *string_list = NULL;
29   GtkTreeModel  *model;
30   GString       *string;
31
32   model
33     = GTK_TREE_MODEL (gtk_icon_view_get_model (GTK_ICON_VIEW (widget)));
34   root = gtk_icon_view_get_selected_items (GTK_ICON_VIEW (widget));
35
36   for (list = root; list; list = g_list_next (list))
37     {
38       GtkTreePath *path;
39       GtkTreeIter iter;
40       gchar       *name;
41
42       path = (GtkTreePath *) list->data;
43       gtk_tree_model_get_iter (model, &iter, path);
44       name = gtk_tree_model_get_string_from_iter (model, &iter);
45       string_list = g_list_append (string_list, name);
46     }
47   if (string_list)
48     {
49       string = g_string_new ((const gchar *) string_list->data);
50       for (list = g_list_next (string_list); list;
51            list = g_list_next (list))
52         {
53           string = g_string_append (string, ",");
54           string = g_string_append (string, (const gchar *) list->data);
55         }
56       gtk_selection_data_set (data,
57                               gtk_selection_data_get_target (data),
58                               8, string->str, strlen (string->str));
59       g_string_free (string, FALSE);
60     }
61 }
62
63 static void
64 cb_drag_data_received (GtkWidget         *widget,
65                        GdkDragContext    *context,
66                        gint              x,
67                        gint              y,
68                        GtkSelectionData  *data,
69                        guint             info,
70                        guint             time)
71 {
72   GtkWidget     *source;
73
74   source = gtk_drag_get_source_widget (context);
75   if (source == widget)
76     {
77       gtk_drag_finish (context, FALSE, FALSE, time);
78       return;
79     }
80   int length = gtk_selection_data_get_length (data);
81   int format = gtk_selection_data_get_format (data);
82   if ((length >= 0) && (format == 8))
83     {
84       GtkTreeModel *src_model;
85       GtkTreeModel *dst_model;
86       gchar        **strlist;
87       gchar        *received;
88       int          n;
89
90       src_model = gtk_icon_view_get_model (GTK_ICON_VIEW (source));
91       dst_model = gtk_icon_view_get_model (GTK_ICON_VIEW (widget));
92
93       received = g_strchomp ((gchar *) gtk_selection_data_get_data (data));
```

```
94        strlist = g_strsplit (received, ",", 0);
95
96        for (n = 0; strlist[n]; n++)
97          {
98            GtkTreeIter iter;
99            GdkPixbuf   *pixbuf;
100           gchar       *name;
101
102           gtk_tree_model_get_iter_from_string (src_model,
103                                                &iter, strlist[n]);
104           gtk_tree_model_get (src_model, &iter,
105                               COLUMN_NAME, &name,
106                               COLUMN_PIXBUF, &pixbuf, -1);
107
108           if (gdk_drag_context_get_selected_action (context)
109               == GDK_ACTION_MOVE)
110             {
111               gtk_list_store_remove (GTK_LIST_STORE (src_model), &iter);
112             }
113
114           gtk_list_store_append (GTK_LIST_STORE (dst_model), &iter);
115           gtk_list_store_set (GTK_LIST_STORE (dst_model), &iter,
116                               COLUMN_NAME, name,
117                               COLUMN_PIXBUF, pixbuf, -1);
118         }
119       g_strfreev (strlist);
120       gtk_drag_finish (context, TRUE, FALSE, time);
121
122       return;
123     }
124   gtk_drag_finish (context, FALSE, FALSE, time);
125 }
126
127 static void
128 cb_drag_data_delete (GtkWidget      *widget,
129                      GdkDragContext *drag_context,
130                      gpointer       user_data)
131 {
132   GtkTreeModel  *model;
133   GList         *root, *list;
134
135   model
136     = GTK_TREE_MODEL(gtk_icon_view_get_model (GTK_ICON_VIEW(widget)));
137   root = gtk_icon_view_get_selected_items (GTK_ICON_VIEW(widget));
138   for (list = root; list; list = g_list_next(list))
139     {
140       GtkTreePath *path;
141       GtkTreeIter iter;
142
143       path = (GtkTreePath *) list->data;
144       gtk_tree_model_get_iter (model, &iter, path);
145       gtk_list_store_remove (GTK_LIST_STORE(model), &iter);
146     }
147 }
148
149 static void
150 add_data (GtkIconView *iconview)
151 {
152   GdkPixbuf     *folder_pixbuf;
153   GdkPixbuf     *file_pixbuf;
154   GtkListStore  *store;
155   GtkTreeIter   iter;
156
157   folder_pixbuf = gdk_pixbuf_new_from_file ("./folder.png", NULL);
158   file_pixbuf = gdk_pixbuf_new_from_file ("./file.png", NULL);
159
160   store = GTK_LIST_STORE(gtk_icon_view_get_model (iconview));
161
162   gtk_list_store_clear (store);
163
164   gtk_list_store_append (store, &iter);
165   gtk_list_store_set (store, &iter,
166                       COLUMN_NAME, "folder.png",
167                       COLUMN_PIXBUF, folder_pixbuf, -1);
168   g_object_unref (folder_pixbuf);
169
```

```
170   gtk_list_store_append (store, &iter);
171   gtk_list_store_set (store, &iter,
172                       COLUMN_NAME, "file.png",
173                       COLUMN_PIXBUF, file_pixbuf, -1);
174   g_object_unref (file_pixbuf);
175 }
176
177 static GtkWidget*
178 create_icon_view_widget (void)
179 {
180   GtkWidget    *iconview;
181   GtkListStore *store;
182
183   store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING, GDK_TYPE_PIXBUF);
184   iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL (store));
185   g_object_unref (store);
186
187   gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview), COLUMN_NAME);
188   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW (iconview),
189                                    COLUMN_PIXBUF);
190   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 96);
191
192   gtk_icon_view_enable_model_drag_source (GTK_ICON_VIEW (iconview),
193                                           GDK_BUTTON1_MASK |
194                                           GDK_SHIFT_MASK,
195                                           target_table,
196                                           ntargets,
197                                           GDK_ACTION_COPY |
198                                           GDK_ACTION_MOVE);
199   gtk_drag_dest_set (iconview,
200                      GTK_DEST_DEFAULT_ALL, target_table, ntargets,
201                      GDK_ACTION_COPY | GDK_ACTION_MOVE);
202   gtk_icon_view_set_selection_mode (GTK_ICON_VIEW (iconview),
203                                     GTK_SELECTION_MULTIPLE);
204   g_signal_connect (iconview, "drag-data-get",
205                     G_CALLBACK (source_drag_data_get), NULL);
206   g_signal_connect (iconview, "drag-data-received",
207                     G_CALLBACK (cb_drag_data_received), NULL);
208   g_signal_connect (iconview, "drag-data-delete",
209                     G_CALLBACK (cb_drag_data_delete), NULL);
210
211   return iconview;
212 }
213
214 int
215 main (int argc, char *argv[])
216 {
217   GtkWidget *window;
218   GtkWidget *hbox;
219   GtkWidget *iconview;
220   GtkWidget *frame;
221
222   gtk_init (&argc, &argv);
223
224   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
225   gtk_window_set_title (GTK_WINDOW (window), "Drag␣and␣Drop␣Sample3");
226   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
227   g_signal_connect (G_OBJECT (window), "destroy",
228                     G_CALLBACK (gtk_main_quit), NULL);
229
230   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
231   gtk_container_add (GTK_CONTAINER (window), hbox);
232
233   frame = gtk_frame_new ("");
234   gtk_widget_set_size_request (frame, 400, -1);
235   gtk_box_pack_start (GTK_BOX (hbox), frame, TRUE, TRUE, 0);
236   gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_IN);
237
238   iconview = create_icon_view_widget ();
239   gtk_container_add (GTK_CONTAINER (frame), iconview);
240   add_data (GTK_ICON_VIEW (iconview));
241
242   frame = gtk_frame_new ("");
243   gtk_widget_set_size_request (frame, 400, -1);
244   gtk_box_pack_start (GTK_BOX (hbox), frame, TRUE, TRUE, 0);
245   gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_IN);
```

```
246
247   iconview = create_icon_view_widget ();
248   gtk_container_add (GTK_CONTAINER (frame), iconview);
249
250   gtk_widget_show_all (window);
251   gtk_main ();
252
253   return 0;
254 }
```

8.4 プログラムオプションの解析

アプリケーションを実行する際に，最低限必要な引数のほかにオプションを設定してアプリケーションの動作をコントロールしたり，パラメータを変えたりすることがあります．この節ではそんなオプションの設定やコマンド引数の解析を簡単に行ってくれる **GOptionContext** について解説します．

作成するプログラムは, プログラムの引数に一つの数値を与えて, その数値に対して, 内部変数 iterations と step を使って, 以下のような処理を行うものとします.

```
for (n = 0; n < iterations; n++)
  {
    g_print ("%5d   %f\n", n + 1, val);
    val += step;
  }
```

このプログラムでは, iterations と step のデフォルト値をそれぞれ, 10, 1 としておき, プログラムのオプションでこれらの値を変更できるようにします.

8.4.1 コマンドラインオプションの設定

コマンドラインオプションの設定は構造体 GOptionEntry で行います．構造体 GOptionEntry は次のように定義されています.

```
typedef struct {
  const gchar *long_name;
  gchar       short_name;
  gint        flags;
  GOptionArg  arg;
  gpointer    arg_data;
  const gchar *description;
  const gchar *arg_description;
} GOptionEntry;
```

まず long_name と short_name でオプションを指定する文字列を指定します．long_name にはハイフン 2 つの後に続く文字列を，short_name にはハイフンに続く 1 文字を指定します．3 つ目のメンバ flags は，列挙体 **GOptionFlags** で定義された値（表 8.4 を参照）を指定します．特別な使い方をしない限りは，G_OPTION_FLAG_NOALIAS もしくは G_OPTION_FLAG_IN_MAIN を指定すればよいでしょう．

メンバ arg には，列挙体 **GOptionArg** で定義された値（表 8.5 を参照）を指定します．これはこのオプションに続く引数にどのようなデータ型の引数を取るかを表すものです．

メンバ arg_data には通常，オプションの引数に与えた値を格納するための変数のアドレスを指定します．残りの 2 つのメンバは，オプションに対する説明と，オプションが引数を取る場合に，その引数がどのような引数であるかの説明です．

反復回数 (iterations) と数値の増分 (step) 用のオプションとして, 以下のように GOptionEntry 構造体のデータを用意しました. 反復回数を変更するためのオプション文字列は, ”iterations”と’i’ としています. また, 整数値を入力するオプションなので, 4 番目のメンバには, G_OPTION_ARG_INT を指定しています.

表 8.4　GOptionFlags の値

値	説明
G_OPTION_FLAG_HIDDEN	–help オプションを使っても表示しない.
G_OPTION_FLAG_IN_MAIN	–help オプションのメインセクションに表示する.
G_OPTION_FLAG_REVERSE	G_OPTION_ARG_NONE 属性を持つオプションに対して，オプションの意味を反転する.
G_OPTION_FLAG_NO_ARG	G_OPTION_ARG_CALLBACK 属性を持つオプションに対して，コールバック関数が引数を持たないことを示す.
G_OPTION_FLAG_FILENAME	G_OPTION_ARG_CALLBACK 属性を持つオプションに対して，コールバック関数の引数にファイル名が与えられることを示す.
G_OPTION_FLAG_OPTION_ARG	G_OPTION_ARG_CALLBACK 属性を持つオプションに対して，コールバック関数の引数がオプション的（与えられない場合もある）に与えられることを示す.
G_OPTION_FLAG_NOALIAS	オプションの衝突を自動的に回避する.

表 8.5　GOptionArg の値

値	説明
G_OPTION_ARG_NONE	オプションは引数を持たない.
G_OPTION_ARG_STRING	文字列を引数にする.
G_OPTION_ARG_INT	整数を引数にする.
G_OPTION_ARG_CALLBACK	特殊な引数に対するコールバック関数を持つ.
G_OPTION_ARG_FILENAME	ファイル名を引数にする.
G_OPTION_ARG_STRING_ARRAY	文字列を引数にする．このオプションは複数使用できる.
G_OPTION_ARG_FILENAME_ARRAY	ファイル名を引数にする．このオプションは複数使用できる.
G_OPTION_ARG_DOUBLE	実数を引数にする.
G_OPTION_ARG_INT64	64 ビット整数を引数にする.

```
gint            iterations = 10;
gdouble         step = 1.0;
GOptionEntry    entries[] =
  {
    {"iterations", 'i', G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_INT,
     &iterations, "Number of iterations (Default 10)", NULL},
    {"step", 's', G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_DOUBLE,
     &step, "Step value (Default 1)", NULL},
    NULL
  };
```

8.4.2　オプションコンテキストの設定

オプションの管理は GOptionContext で行い，関数 g_option_context_new で実体を作成します．この関数の引数には，コマンドライン引数などの説明を文字列で与えます．

作成するプログラムは，コマンドライン引数で与えた数値に決められた反復回数だけステップ数を足した値を表示するものですので, 関数 g_option_context_new の引数には，コマンドライン引数に最初の数値を入力するように記述しています．

```
GOptionContext* g_option_context_new (const gchar *parameter_string);
```

第 1 引数	usage 表示の際のプログラムの引数に関する記述
戻り値	GOptionContext 型の変数

```
GOptionContext *context;
context = g_option_context_new ("#start");
```

次に関数 g_option_context_add_main_entries で定義したオプションを登録します．

```
void
g_option_context_add_main_entries (GOptionContext *context,
                                   const GOptionEntry *entries,
                                   const gchar *translation_domain);
```

第 1 引数	GOptionContext 型の変数
第 2 引数	GOptionEntry 構造体で定義したオプション
第 3 引数	メッセージ翻訳のドメイン

8.4.3　オプションの解析

オプションの登録とオプションコンテキストの設定が終われば，オプションの解析は，関数 g_option_context_parse を呼び出すだけです．

```
gboolean g_option_context_parse (GOptionContext *context,
                                 gint *argc,
                                 gchar ***argv,
                                 GError **error);
```

第 1 引数	メイン関数の第 1 引数
第 2 引数	メイン関数の第 2 引数
第 3 引数	エラー構造体
戻り値	解析に成功したら TRUE, そうでなければ FALSE

もし，オプション解析に失敗した場合には FALSE を返します．関数 g_option_context_get_enabled の戻り値が TRUE の状態で（デフォルトでは TRUE），–help オプションを指定した場合には，オプション一覧を表示して自動的にプログラムを終了します．また，このプログラムでは，引数を一つ入力する必要がありますので，引数解析に失敗した場合か，もしくは引数が一つ入力されていない場合には，関数 g_option_context_get_help を使ってヘルプの文字列を取得して，出力するようにしています（ソース 10–4–1 28–35 行目を参照）．

```
gchar *
g_option_context_get_help (GOptionContext *context,
                           gboolean       main_help,
                           GOptionGroup   *group);
```

第 1 引数	GOptionContext 型の変数
第 2 引数	すべてのヘルプを表示するかどうか
第 3 引数	GOptionGroup 型の変数
戻り値	ヘルプ文字列そうでなければ FALSE

ソース 10–4–1 に作成したプログラムのソースコードを示します．

ソース 10–4–1 プログラムオプションの解析： parse_option.c

```
1 #include <gtk/gtk.h>
2 #include <stdlib.h>
3
4 int
```

```
 5 main (int argc, char *argv[])
 6 {
 7   GOptionContext *context;
 8   gint            iterations = 10;
 9   gdouble         step = 1.0;
10   GOptionEntry    entries[] =
11     {
12       {"iterations", 'i',
13        G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_INT,
14        &iterations, "Number␣of␣iterations␣(Default␣10)", NULL},
15       {"step", 's',
16        G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_DOUBLE,
17        &step, "Step␣value␣(Default␣1)", NULL},
18       NULL
19     };
20   GError   *error = NULL;
21   gboolean result;
22
23   context = g_option_context_new ("#start");
24   g_option_context_add_main_entries (context, entries, NULL);
25
26   result = g_option_context_parse (context, &argc, &argv, &error);
27
28
29   if (!result || argc != 2)
30     {
31       gchar *help_message = g_option_context_get_help (context, TRUE, NULL);
32       g_print ("%s", help_message);
33       g_free (help_message);
34       g_option_context_free (context);
35       exit (1);
36     }
37
38   gdouble val = atof (argv[1]);
39   gint    n;
40   for (n = 0; n < iterations; n++)
41     {
42       g_print ("%5d␣␣␣%f\n", n + 1, val);
43       val += step;
44     }
45   g_option_context_free (context);
46
47   return 0;
48 }
```

実行結果を示します．まずは，引数に何も入力せずにプログラムを実行した結果です．引数が足りないので，関数 g_option_context_get_help で取得したヘルプ文字列が表示されます．

```
$ ./parse-option ↵
Usage:
  parse-option [OPTION...] #start
Help Options:
  -h, --help            Show help options

Application Options:
  -i, --iterations      Number of iterations (Default 10)
  -s, --step            Step value (Default 1)
```

次に，引数だけ与えて，オプションは指定せずにプログラムを実行してみます．引数に 1 を与えたので，標準のパラメータ iterations = 1, step = 1.0 で計算した結果が表示されます．

```
$ ./parse-option 1↵
    1   1.000000
    2   2.000000
    3   3.000000
```

```
   4   4.000000
   5   5.000000
   6   6.000000
   7   7.000000
   8   8.000000
   9   9.000000
  10   10.000000
```

最後に, 設定したオプションを指定してプログラムを実行した結果を示します. オプション指定にはショートオプション (-i) とロングオプション (–step) を使ってみました.

```
$ ./parse-option 1 -i 5 --step 10↵
   1   1.000000
   2   11.000000
   3   21.000000
   4   31.000000
   5   41.000000
```

8.4.4　オプショングループの追加

次にオプショングループ (**GOptionGroup**) を追加してみます. これは–help-*group-name* のようなオプションを作成するものです. オプショングループは, 関数 g_option_group_new で作成します.

```
GOptionGroup* g_option_group_new (const gchar *name,
                                  const gchar *description,
                                  const gchar *help_description,
                                  gpointer user_data,
                                  GDestroyNotify destroy);
```

第 1 引数	オプショングループの名前
第 2 引数	オプションを指定したときに表示するメッセージ
第 3 引数	オプションの説明
第 4 引数	オプショングループに対する関数に渡すユーザーデータ
第 5 引数	オプショングループが解放されるときに呼ばれるコールバック関数
戻り値	GOptionGroup 型の変数

この関数では，第 1 引数に作成するオプショングループの名前 (*group-name*) を指定し，第 2 引数にはこのオプショングループの説明を与えます．また，関数 g_option_group_add_entries によってオプションを追加すると，プログラムのオプションとして–help-name（name は関数の第 1 引数に与えた文字列）を与えると，関数 g_option_group_new の第 3 引数に指定した文字列が表示されます.

```
void g_option_group_add_entries (GOptionGroup       *group,
                                 const GOptionEntry *entries);
```

そして，関数 g_option_context_add_group で，作成したオプショングループをオプションコンテキストに登録します.

```
void g_option_context_add_group (GOptionContext *context,
                                 GOptionGroup *group);
```

ソース 10–4–1 に以下のようにオプショングループを追加します. これはオプションの使い方を表示する example というグループです. ここでは, このオプショングループに対する関数を設定しませんので, 関数の第 4, 5 引数は NULL としています.

```
GOptionGroup *option_group;
option_group
  = g_option_group_new ("example",
                        "Example: ./parse-option -s 0.1 -i 5 1",
                        "Show an example",
                        NULL, NULL);
```

オプショングループを追加したソースコードをコンパイルして，プログラムを実行した結果が次のようになります．Help Options: の項目に，–help-all と–help-example が追加されているのがわかります．また，Example:の項目も追加されて，オプションの使い方の例が表示されています．

```
$ ./parse-option ↵
Usage:
  parse-option [OPTION...] #start
Help Options:
  -h, --help            Show help options
  --help-all            Show all help options
  --help-example        Show an example

Example: ./parse-option -s 0.1 -i 5 1
  -i, --iterations      Number of iterations (Default 10)
  -s, --step            Step value (Default 1)

Application Options:
  -i, --iterations      Number of iterations (Default 10)
  -s, --step            Step value (Default 1)
```

8.4.5 オプショングループに対する関数の追加

オプショングループを作成すると，そのオプショングループに対して，オプション解析前に呼び出される関数とオプション解析後に呼び出される関数を設定することができます．この設定は，関数 g_option_group_set_parse_hooks によって行います．そして，このオプション解析の前後に呼び出す関数に渡したい変数を，関数 g_option_group_new の第 4 引数に与え，その変数のメモリ領域を解放する際に呼び出す関数を第 5 引数に与えます．

```
void g_option_group_set_parse_hooks (GOptionGroup *group,
                                     GOptionParseFunc pre_parse_func,
                                     GOptionParseFunc post_parse_func);
```

第 1 引数	GOptionGroup 型の変数
第 2 引数	オプション解析前に呼び出す関数
第 3 引数	オプション解析後に呼び出す関数

GOptionParseFunc のプロトタイプ宣言は次のようになります．この関数の第 3 引数 data に，関数 g_option_group_new の第 4 引数で与えた変数が入ります．

```
gboolean (*GOptionParseFunc) (GOptionContext *context,
                              GOptionGroup   *group,
                              gpointer        data,
                              GError         **error);
```

ここでは，表 8.5 に示した種類のオプションの例を示すため，オプション用の変数をまとめて扱う構造体 option を独自に定義して，オプション解析前に呼び出す関数内で，構造体メンバの初期値を設定するようにします．定義した構造体は以下のようになります．

```
typedef struct _Option
{
  gboolean verbose;
  gint     iterations;
  gdouble  step;
  gchar    *filename;
  gchar    **string_array;
} Option;
```

今回新しく追加するオプションは, 以下のようになります.

- 引数なしのオプション
- ファイル名を指定するオプション
- 複数回指定可能な文字列を引数とするオプション
- 特別なコールバック関数を呼び出すオプション

最後の特別なコールバック関数を呼び出すオプションでは, GOptionEntry 構造体のメンバ arg_data にコールバック関数のアドレスを指定します. この関数のプロトタイプ宣言は次のようになっています.

```
gboolean (*GOptionArgFunc) (const gchar *option_name,
                            const gchar *value,
                            gpointer    data,
                            GError      **error);
```

GLib のリファレンスマニュアルには, この関数の第 3 引数 data には, 関数 g_option_group_new の第 4 引数に与えた user_data が入ると書かれているのですが, どうもそのようにはなっていないようです. そのため, ソース 10–4–2 では, 変数 option をグローバル変数として定義しました.

また, 最後に関数 g_option_context_set_summary と関数 g_option_context_set_description で, それぞれオプション一覧を表示する前に出力するメッセージと, オプション一覧を表示した後に出力するメッセージを設定してました.

ソース 10–4–2 に作成したソースコードを示します. また, 様々なオプションを指定してプログラムを実行した結果を以下に示します.

```
$ ./parse-option -v -f test-filename.txt -a this -a is -a a -a pen -c 10:5 10
Initializing option members ...
Parsing option is done.
(This message is shown if the verbose option is specified.)

    n | number
------+----------
    1   10.000000
    2   20.000000
    3   30.000000
    4   40.000000
    5   50.000000
option filename = test-filename.txt
option string-array = this
option string-array = is
option string-array = a
option string-array = pen
```

ソース 10–4–2 プログラムオプションの解析： parse_option.c

```
1 #include <gtk/gtk.h>
2 #include <stdlib.h>
3
4 typedef struct _Option
```

```
5  {
6    gboolean verbose;
7    gint     iterations;
8    gdouble  step;
9    gchar    *filename;
10   gchar    **string_array;
11 } Option;
12
13 Option option;
14
15 static void
16 option_free (gpointer user_data)
17 {
18   Option *option = (Option *) user_data;
19
20   if (option->filename) g_free (option->filename);
21   if (option->string_array) g_strfreev (option->string_array);
22 }
23
24 static gboolean
25 special_option_parse_func (const gchar *option_name,
26                            const gchar *value,
27                            gpointer    user_data,
28                            GError      *error)
29 {
30   gchar  **option_strings = NULL;
31   int    nargs = 0;
32
33   option_strings = g_strsplit (value, ":", 0);
34   for (nargs = 0; option_strings[nargs] != NULL; nargs++);
35
36   if (nargs < 2)
37     {
38       g_print ("For␣%s␣option,␣"
39                "please␣specify␣two␣option␣arguments␣by␣separating␣"
40                "\":\".\n\n",
41                option_name);
42       return FALSE;
43     }
44   option.step = atof (option_strings[0]);
45   option.iterations = atoi (option_strings[1]);
46
47   g_strfreev (option_strings);
48
49   return TRUE;
50 }
51
52 static gboolean
53 pre_parse_func (GOptionContext  *context,
54                 GOptionGroup    *group,
55                 gpointer        user_data,
56                 GError          *error)
57 {
58   Option *option = (Option *) user_data;
59
60   g_print ("\nInitializing␣option␣members␣...\n\n");
61
62   option->verbose      = FALSE;
63   option->iterations   = 10;
64   option->step         = 1;
65   option->filename     = NULL;
66   option->string_array = NULL;
67
68   return TRUE;
69 }
70
71 static gboolean
72 post_parse_func (GOptionContext *context,
73                  GOptionGroup   *group,
74                  gpointer       user_data,
75                  GError         *error)
76 {
77   Option *option = (Option *) user_data;
78
79   if (option->verbose)
80     {
81       g_print ("Parsing␣option␣is␣done.\n"
```

```
82                "(This␣message␣is␣shown␣if␣the␣verbose␣option␣is␣"
83                "specified.)\n\n");
84      }
85    return TRUE;
86  }
87
88  int
89  main (int argc, char *argv[])
90  {
91    GOptionContext *context;
92    GOptionEntry   entries[] =
93      {
94        {"verbose", 'v', G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_NONE,
95         &option.verbose, "Verbose␣mode", NULL},
96        {"iterations", 'i', G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_INT,
97         &option.iterations, "Number␣of␣iterations␣(Default␣10)", NULL},
98        {"step", 's', G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_DOUBLE,
99         &option.step, "Step␣value␣(Default␣1)", NULL},
100       {"filename", 'f', G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_FILENAME,
101        &option.filename, "Specify␣a␣filename", NULL},
102       {"string-array", 'a', G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_STRING_ARRAY,
103        &option.string_array,
104        "Specify␣a␣string.␣This␣option␣can␣be␣used␣multiple␣times.", NULL},
105       {"callback", 'c', G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_CALLBACK,
106        special_option_parse_func, "The␣sample␣callback␣option",
107        "step:iterations"},
108       NULL
109     };
110   GError   *error = NULL;
111   gboolean result;
112
113   context = g_option_context_new ("#start");
114   g_option_context_add_main_entries (context, entries, NULL);
115
116   GOptionGroup *option_group;
117   option_group
118     = g_option_group_new ("example",
119                           "Example:␣./parse-option␣-s␣0.1␣-i␣5␣1",
120                           "Show␣an␣example",
121                           (gpointer) &option,
122                           (GDestroyNotify) option_free);
123   g_option_group_add_entries (option_group, entries);
124   g_option_group_set_parse_hooks (option_group,
125                                   (GOptionParseFunc) pre_parse_func,
126                                   (GOptionParseFunc) post_parse_func);
127   g_option_context_add_group (context, option_group);
128
129   g_option_context_set_summary (context,
130                                 "This␣program␣is␣an␣option␣parser␣"
131                                 "example.");
132   g_option_context_set_description (context,
133                                     "In␣this␣line,␣please␣describe␣detailed"
134                                     "descriptions␣about␣this␣program.\n");
135
136   result = g_option_context_parse (context, &argc, &argv, &error);
137
138   if (!result || argc != 2)
139     {
140       gchar *help_message = g_option_context_get_help (context, FALSE, NULL);
141       g_print ("%s\n", help_message);
142       g_free (help_message);
143       g_option_context_free (context);
144       exit (1);
145     }
146   gdouble val = atof (argv[1]);
147   gint    n;
148
149   if (option.verbose)
150     {
151       g_print ("␣␣␣␣n␣|␣number\n------+----------\n");
152     }
153   for (n = 0; n < option.iterations; n++)
154     {
155       g_print ("%5d␣␣␣%f\n", n + 1, val);
156       val += option.step;
157     }
```

```
158   if (option.filename)
159     {
160       g_print ("option␣filename␣=␣%s\n", option.filename);
161     }
162   if (option.string_array)
163     {
164       n = 0;
165       while (option.string_array[n])
166         {
167           g_print ("option␣string-array␣=␣%s\n",
168                    option.string_array[n++]);
169         }
170     }
171   g_option_context_free (context);
172
173   return 0;
174 }
```

9 独自ウィジェットの作成

本章では, カウントダウンウィジェットの作成を通して独自ウィジェットの具体的な作成方法を解説します.

9.1 カウントダウンウィジェットの仕様

まずは, 作成するカウントダウンウィジェット (Countdown) の仕様を以下のように定めます.

- ウィジェット名 ... Countdown
- 非 GUI ウィジェット
- 動作 ... カウントダウンの開始後, 設定した秒数が経つとシグナルを発生する.
- プロパティ
  - 残り秒数 (count)
- シグナル
  - finished シグナル
    残り秒数が 0 になったときに発生するシグナル
- 関数
  - countdown_new
  - countdown_start
  - countdown_stop
  - countdown_set_count
  - countdown_get_count

9.2 ヘッダファイルの作成

まず, ヘッダファイルを作成します. GTK+ のソースコードを参考に作成したヘッダファイルをソース 9–1 に示します.

構造体の typedef 設定

5–7 行目は構造体の typedef 設定です. CountdownPrivate 構造体は C++ のクラスでいうと, private メンバに相当するもので, プログラマから直接アクセスされたくないメンバをこの構造体のメンバとして設定します.

マクロ定義

9–13 行目はウィジェットに対するマクロ定義で, ウィジェットをキャストしたりウィジェットを判別するために使われるマクロです. これらはすべてのウィジェットに最低限必要なマクロです. この行まではどのウィジェットに対しても同様な記述になります.

ウィジェット構造体とウィジェットクラス構造体の定義

15–19 行目がウィジェット構造体の定義になります．カウントダウンウィジェットの親ウィジェットと，CountdownPrivate 構造体の変数をメンバとします．後ほど説明しますが，このウィジェットで取り扱う変数は，すべてプログラマから直接アクセスを禁止する CountdownPrivate 構造体のメンバとして設定します．

21–26 行目は，ウィジェットクラス構造体の定義です．クラス構造体のメンバには，親クラスの変数と，finished シグナルに対するコールバック関数を設定します．

ウィジェットに対する関数

最後に，28–34 行目までがウィジェットに対する関数のプロトタイプ宣言になります．

ソース 9–1 ヘッダファイル：countdown.h

```
1  #include <gtk/gtk.h>
2
3  G_BEGIN_DECLS
4
5  typedef struct _Countdown        Countdown;
6  typedef struct _CountdownPrivate CountdownPrivate;
7  typedef struct _CountdownClass   CountdownClass;
8
9  #define TYPE_COUNTDOWN  (countdown_get_type())
10 #define COUNTDOWN(obj)  (G_TYPE_CHECK_INSTANCE_CAST((obj), TYPE_COUNTDOWN, Countdown))
11 #define COUNTDOWN_CLASS(klass) (G_TYPE_CHECK_CLASS_CAST((klass), TYPE_COUNTDOWN,
       CountdownClass))
12 #define IS_COUNTDOWN(obj)       (G_TYPE_CHECK_INSTANCE_TYPE((obj), TYPE_COUNTDOWN))
13 #define COUNTDOWN_GET_CLASS(obj) (G_TYPE_INSTANCE_GET_CLASS((obj), TYPE_COUNTDOWN,
       CountdownClass))
14
15 struct _Countdown
16 {
17   GObject          parent;
18   CountdownPrivate *priv;
19 };
20
21 struct _CountdownClass
22 {
23   GObjectClass parent_class;
24
25   void (*finished) (Countdown *countdown);
26 };
27
28 GType      countdown_get_type  (void) G_GNUC_CONST;
29 Countdown* countdown_new       (guint count);
30 void       countdown_start     (Countdown *countdown);
31 void       countdown_stop      (Countdown *countdown);
32 void       countdown_set_count (Countdown *countdown,
33                                 guint     count);
34 guint      countdown_get_count (Countdown *countdown);
35
36 G_END_DECLS
```

9.3 Private 構造体とプロパティ・シグナル列挙体の定義

次に，Private 構造体，プロパティとシグナルに関する列挙体の定義を行います．これらについても，指定の方法は GTK+ のソースコードの記述に従っています．

まず Private 構造体ですが，メンバには，現在のカウント (count) と，関数 g_timeout_add の戻り値を保存する変数 (countdown_id) を設定します（ソース 9.3 の 1–5 行目）．7–11 行目と 13–16 行目は，プロパティとシグナルに対する列挙体定義です．18, 19 行目はプロパティとシグナルに対する変数宣言になります．

ソース 9–2 Private 構造体等の定義：countdown.c から一部抜粋

```
1  struct _CountdownPrivate
2  {
3    guint count;
```

```
4   guint countdown_id;
5 };
6
7 enum {
8   PROP_0,
9   PROP_COUNT,
10  LAST_PROP
11 };
12
13 enum {
14  SIGNAL_FINISHED,
15  SIGNAL_LAST_SIGNAL
16 };
17
18 static GParamSpec *props[LAST_PROP] = { NULL, };
19 static guint countdown_signals[SIGNAL_LAST_SIGNAL] = { 0 };
```

9.4 クラス初期化関数

次にクラス初期化関数 countdown_class_init を実装します（ソース 9–3）．この関数内では, 次の三つの設定を行っています.

- クラス関数の設定
- プロパティの設定
- シグナルの設定

クラス関数の設定

Countdown ウィジェットで設定すべき関数は, 以下の三つの関数です.

- 関数 dispose ... ウィジェットを dispose する際に, イベントループを止める処理を実装します.
- 関数 set_property ... プロパティ (count) を設定する処理を実装します.
- 関数 get_property ... プロパティ (count) を取得する処理を実装します.

ここでは, 関数へのポインタを設定するだけで, それぞれの関数の具体的な実装は後ほど説明します.

ソース 9–3 クラス初期化関数：countdown.c から一部抜粋

```
1 static void countdown_class_init (CountdownClass *klass)
2 {
3   GObjectClass *gobject_class;
4
5   gobject_class = G_OBJECT_CLASS (klass);
6
7   gobject_class->dispose      = countdown_dispose;
8   gobject_class->set_property = countdown_set_property;
9   gobject_class->get_property = countdown_get_property;
10
11  props[PROP_COUNT] =
12    g_param_spec_uint ("count",
13                       "Count",
14                       "Count␣down␣parameter",
15                       0,
16                       G_MAXUINT,
17                       0,
18                       G_PARAM_READWRITE|G_PARAM_CONSTRUCT|G_PARAM_EXPLICIT_NOTIFY);
19
20  g_object_class_install_properties (gobject_class, LAST_PROP, props);
21
22  countdown_signals[SIGNAL_FINISHED] =
23    g_signal_new ("finished",
24                  TYPE_COUNTDOWN,
25                  G_SIGNAL_RUN_LAST,
26                  G_STRUCT_OFFSET (CountdownClass, finished),
27                  NULL, NULL,
28                  g_cclosure_marshal_VOID__VOID,
29                  G_TYPE_NONE, 0);
30 }
```

プロパティの設定

Countdown ウィジェットで扱うプロパティ count は残り秒数を表しますので, 変数の型は guint としています. guint 型のプロパティを生成する関数は, 関数 g_param_spec_uint です.

```
GParamSpec * g_param_spec_uint (const gchar *name,
                                const gchar *nick,
                                const gchar *blurb,
                                guint        minimum,
                                guint        maximum,
                                guint        default_value,
                                GParamFlags flags);
```

第 1 引数：	名前
第 2 引数：	ニックネーム
第 3 引数：	説明
第 4 引数：	最小値
第 5 引数：	最大値
第 6 引数：	デフォルト値
第 7 引数：	フラグ
戻り値：	GParamSpec 型の変数

プロパティ count の定義で設定している GParamFlags の値を表 9.1 に示します. その他の値については, devhelp 等で調べてみてください. プロパティ count は, 値の取得と設定をプログラマに許可しますので, ソース 9–3 の 18 行目で, G_PARAM_READWRITE フラグを指定しています. プロパティの設定の後は, 関数 g_object_class_install_properties を呼び出します.

表 9.1　GParamFlags の種類

値	説明
G_PARAM_READABLE	読み込み可能
G_PARAM_WRITABLE	書き込み可能
G_PARAM_READWRITE	読み込みと書き込み可能 (G_PARAM_READABLE|G_PARAM_WRITABLE)
G_PARAM_CONSTRUCT	ウィジェット生成時にパラメータを設定する
G_PARAM_PARAM_EXPLICIT_NOTIFY	パラメータを設定したときに自動的に notify シグナルを発生させない.

シグナルの設定

シグナルの設定は, 関数 g_signal_new を使用します. 第 5, 6 引数は基本的に NULL で構いません. 第 2, 4 引数はウィジェットによって決定する値になります.

シグナル固有の設定をしないといけないのは, 第 1, 3, 7, 8, 9 引数になります. 第 1 引数はシグナル名を指定します. 第 3 引数は, コールバック関数の呼び出されるタイミングに関する値で, 基本的には, **G_SIGNAL_RUN_FIRST**（デフォルトコールバック関数を最初に呼び出す）か **G_SIGNAL_RUN_LAST**（デフォルトコールバック関数を最後に呼び出す）のどちらかを指定します. **G_SIGNAL_ACTION** はユーザが関数 g_signal_emit によってシグナルを発生させることができるように設定します.

第 7 引数はコールバック関数の引数の型を指定します. 28 行目の指定では, 引数なしという設定を行っています. 第 8 引数はコールバック関数の戻り値の型を指定します. 第 9 引数はコールバック関数の引数の数を指定し, 引数の数が 0 でなければ, それ以降の引数にその引数の型を指定します.

```
guint g_signal_new (const gchar        *signal_name,
                    GType               itype,
                    GSignalFlags        signal_flags,
                    guint               class_offset,
                    GSignalAccumulator accumulator,
                    gpointer            accu_data,
```

```
                    GSignalCMarshaller c_marshaller,
                    GType              return_type,
                    guint              n_params,
                    ...);
```

第 1 引数：	シグナル名
第 2 引数：	ウィジェットタイプ
第 3 引数：	シグナルフラグ
第 4 引数：	クラスオフセット
第 5 引数：	GSignalAccumulator
第 6 引数：	GSignalAccumulator に対するデータ
第 7 引数：	GSignalCMarshaller
第 8 引数：	コールバック関数の戻り値の型
第 9 引数：	パラメータ数
戻り値：	シグナル ID

9.5 ウィジェット生成関数

ウィジェット生成関数 countdown_new を実装します．この関数はウィジェットを使用する際に一番初めに呼び出す関数です．この関数内で関数 g_object_new を使用していますが，この関数を呼び出すことで，前の節で作成したクラス初期化関数 countdown_class_init と，この節で同じように実装する，関数 countdown_init が呼び出されます．

1 行目は，ウィジェットの型を生成するマクロで，Private 構造体を持つウィジェットを生成するものです．そして，関数 countdown_new の中で，関数 g_object_new を呼び出すことで，ウィジェットを生成します．この関数を呼び出すことで，関数 countdown_class_init，関数 countdown_init の順番で関数が呼び出されます．関数 countdonw_init 中の 5 行目の関数 countdown_get_instance_private により，Private 構造体を取得することができます．

ソース 9–4 ウィジェット生成関数：countdown.c から一部抜粋

```
1  G_DEFINE_TYPE_WITH_PRIVATE (Countdown, countdown, G_TYPE_OBJECT)
2
3  static void countdown_init (Countdown *countdown)
4  {
5    countdown->priv = countdown_get_instance_private (countdown);
6    countdown->priv->countdown_id = 0;
7    countdown->priv->count = 0;
8  }
9
10 Countdown* countdown_new (guint count)
11 {
12   Countdown *countdown;
13
14   countdown = g_object_new (TYPE_COUNTDOWN, NULL);
15   countdown->priv->count = count;
16
17   return countdown;
18 }
```

9.6 プロパティ取得 · 設定関数

ここでは，関数 countdown_class_init 内で設定する二つの関数

- countdown_set_property
- countdown_get_property

と，プログラマが呼び出すことのできる関数

- countdown_set_count

- countdown_get_count

を実装します．

関数 countdown_set_property と関数 countdown_get_property の引数や実装方法については，GTK+ のソースコードのウィジェット実装を参考にしています．また同じように関数 countdown_set_count と関数 countdown_get_count についても同様です．ソース 9–5 に実装結果を示します．

ソース 9–5 プロパティ取得 · 設定関数： countdown.c から一部抜粋

```
1  static void
2  countdown_set_property (GObject          *object,
3                          guint             prop_id,
4                          const GValue     *value,
5                          GParamSpec       *pspec)
6  {
7    Countdown *countdown = COUNTDOWN (object);
8
9    switch (prop_id)
10     {
11       case PROP_COUNT:
12         countdown_set_count (countdown, g_value_get_uint (value));
13       break;
14     }
15 }
16
17 static void
18 countdown_get_property (GObject          *object,
19                         guint             prop_id,
20                         GValue           *value,
21                         GParamSpec       *pspec)
22 {
23   Countdown *countdown = COUNTDOWN (object);
24   CountdownPrivate *priv = countdown->priv;
25
26   switch (prop_id)
27     {
28     case PROP_COUNT:
29       g_value_set_uint (value, priv->count);
30       break;
31     }
32 }
33
34 void
35 countdown_set_count (Countdown *countdown,
36                      guint      count)
37 {
38   g_return_if_fail (IS_COUNTDOWN (countdown));
39
40   countdown->priv->count = count;
41 }
42
43 guint
44 countdown_get_count (Countdown *countdown)
45 {
46   g_return_val_if_fail (IS_COUNTDOWN (countdown), 0);
47
48   return countdown->priv->count;
49 }
```

9.7　カウントダウン機能の実装

最後に，カウントダウン機能を実装します．関数 countdown_start でカウントダウンを開始し，関数 countdown_stop でカウントダウンを停止します．カウントダウン機能は 4.5 節で説明したイベントループを使用します．1 秒（1000 ミリ秒）ごとに指定した関数 (count_down) を呼び出すようにします．イベントループについては，4.5 節を参照してください．

イベントループで呼び出される関数 count_down では，プロパティ count の値を減らしていき，値が 0 になったときに，イベントループを終了し，finished シグナルを発生させます．

ソース 9–6 カウントダウンの実装：countdown.c から一部抜粋

```
1  static void
2  count_down (gpointer user_data)
3  {
4    Countdown *countdown = COUNTDOWN(user_data);
5    CountdownPrivate *priv = countdown->priv;
6
7    priv->count--;
8    if (priv->count == 0)
9      {
10       g_source_remove (priv->countdown_id);
11       priv->countdown_id = 0;
12       g_signal_emit (countdown, countdown_signals[SIGNAL_FINISHED], 0);
13     }
14 }
15
16 void
17 countdown_start (Countdown *countdown)
18 {
19   g_return_if_fail (IS_COUNTDOWN (countdown));
20
21   CountdownPrivate *priv = countdown->priv;
22
23   if (priv->countdown_id > 0)
24     {
25       g_source_remove (priv->countdown_id);
26       priv->countdown_id = 0;
27     }
28   priv->countdown_id
29     = g_timeout_add (1000, (GSourceFunc) count_down, (gpointer) countdown);
30 }
31
32 void
33 countdown_stop (Countdown *countdown)
34 {
35   g_return_if_fail (IS_COUNTDOWN (countdown));
36
37   CountdownPrivate *priv = countdown->priv;
38
39   if (priv->countdown_id > 0)
40     {
41       g_source_remove (priv->countdown_id);
42       priv->countdown_id = 0;
43     }
44 }
```

9.8 作成したウィジェットのテスト

作成したカウントダウンウィジェットのテストプログラムを作成して, 動作確認を行います. 作成したプログラムは, カウント数をスピンボタンウィジェットで設定し, Set ボタンで値を確定させます. そして, Start ボタンでカウントダウン開始, Stop ボタンでカウントダウンを一時停止し, カウントが 0 になると発生するシグナル finised に関連づけした関数内で, プログラムを終了するようにします.

ソース 9–7 にカウントダウンウィジェットのソースコードのすべての内容, ソース 9–8 にテストプログラムのソースコードを示します. 図 9.1 はテストプログラムの実行画面です.

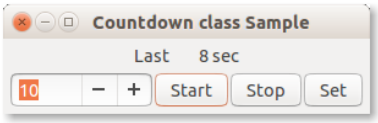


図 9.1　カウントダウンウィジェットのテストプログラム

ソース 9–7 ウィジェットのソースコード：countdown.c

```
#include <glib-object.h>
#include "countdown.h"

struct _CountdownPrivate
{
  guint count;
  guint countdown_id;
};

enum {
  PROP_0,
  PROP_COUNT,
  LAST_PROP
};

enum {
    SIGNAL_FINISHED,
    SIGNAL_LAST_SIGNAL
  };

static void countdown_dispose           (GObject *object);
static void countdown_set_property      (GObject          *object,
                                         guint             prop_id,
                                         const GValue     *value,
                                         GParamSpec       *pspec);
static void countdown_get_property      (GObject          *object,
                                         guint             prop_id,
                                         GValue           *value,
                                         GParamSpec       *pspec);

static GParamSpec *props[LAST_PROP] = { NULL, };
static guint countdown_signals[SIGNAL_LAST_SIGNAL] = { 0 };

G_DEFINE_TYPE_WITH_PRIVATE (Countdown, countdown, G_TYPE_OBJECT)

static void countdown_class_init (CountdownClass *klass)
{
  GObjectClass *gobject_class;

  gobject_class = G_OBJECT_CLASS (klass);

  gobject_class->dispose      = countdown_dispose;
  gobject_class->set_property = countdown_set_property;
  gobject_class->get_property = countdown_get_property;

  props[PROP_COUNT] =
    g_param_spec_uint ("count",
                       "Count",
                       "Count␣down␣parameter",
                       0,
                       1000,
                       0,
                       G_PARAM_READWRITE|G_PARAM_CONSTRUCT|G_PARAM_EXPLICIT_NOTIFY);

  g_object_class_install_properties (gobject_class, LAST_PROP, props);

  countdown_signals[SIGNAL_FINISHED] =
    g_signal_new ("finished",
                  TYPE_COUNTDOWN,
                  G_SIGNAL_RUN_LAST,
                  G_STRUCT_OFFSET (CountdownClass, finished),
                  NULL, NULL,
                  g_cclosure_marshal_VOID__VOID,
                  G_TYPE_NONE, 0);
}

static void countdown_dispose (GObject *object)
{
  Countdown *countdown = COUNTDOWN (object);
  CountdownPrivate *priv = countdown->priv;

  priv->count = 0;
```

```
73
74    if (priv->countdown_id > 0)
75      {
76        g_source_remove (priv->countdown_id);
77        priv->countdown_id = 0;
78      }
79    (* G_OBJECT_CLASS (countdown_parent_class)->dispose) (object);
80  }
81
82  static void
83  countdown_set_property (GObject         *object,
84                          guint            prop_id,
85                          const GValue    *value,
86                          GParamSpec      *pspec)
87  {
88    Countdown *countdown = COUNTDOWN (object);
89
90    switch (prop_id)
91      {
92        case PROP_COUNT:
93          countdown_set_count (countdown, g_value_get_uint (value));
94        break;
95      }
96  }
97
98  static void
99  countdown_get_property (GObject         *object,
100                         guint            prop_id,
101                         GValue          *value,
102                         GParamSpec      *pspec)
103 {
104   Countdown *countdown = COUNTDOWN (object);
105   CountdownPrivate *priv = countdown->priv;
106
107   switch (prop_id)
108     {
109     case PROP_COUNT:
110       g_value_set_uint (value, priv->count);
111       break;
112     }
113 }
114
115 static void countdown_init (Countdown *countdown)
116 {
117   countdown->priv = countdown_get_instance_private (countdown);
118   countdown->priv->countdown_id = 0;
119   countdown->priv->count = 0;
120 }
121
122 Countdown* countdown_new (guint count)
123 {
124   Countdown *countdown;
125
126   countdown = g_object_new (TYPE_COUNTDOWN, NULL);
127   countdown->priv->count = count;
128
129   return countdown;
130 }
131
132 static void
133 count_down (gpointer user_data)
134 {
135   Countdown *countdown = COUNTDOWN(user_data);
136   CountdownPrivate *priv = countdown->priv;
137
138   priv->count--;
139   if (priv->count == 0)
140     {
141       g_source_remove (priv->countdown_id);
142       priv->countdown_id = 0;
143       g_signal_emit (countdown, countdown_signals[SIGNAL_FINISHED], 0);
144     }
145 }
146
147 void
148 countdown_start (Countdown *countdown)
```

```
149 {
150   g_return_if_fail (IS_COUNTDOWN (countdown));
151
152   CountdownPrivate *priv = countdown->priv;
153
154   if (priv->countdown_id > 0)
155     {
156       g_source_remove (priv->countdown_id);
157       priv->countdown_id = 0;
158     }
159   priv->countdown_id
160     = g_timeout_add (1000, (GSourceFunc) count_down, (gpointer) countdown);
161 }
162
163 void
164 countdown_stop (Countdown *countdown)
165 {
166   g_return_if_fail (IS_COUNTDOWN (countdown));
167
168   CountdownPrivate *priv = countdown->priv;
169
170   if (priv->countdown_id > 0)
171     {
172       g_source_remove (priv->countdown_id);
173       priv->countdown_id = 0;
174     }
175 }
176
177 void
178 countdown_set_count (Countdown *countdown,
179                      guint      count)
180 {
181   g_return_if_fail (IS_COUNTDOWN (countdown));
182
183   countdown->priv->count = count;
184 }
185
186 guint
187 countdown_get_count (Countdown *countdown)
188 {
189   g_return_val_if_fail (IS_COUNTDOWN (countdown), 0);
190
191   return countdown->priv->count;
192 }
```

ソース 9–8 カウントダウンウィジェットのテストプログラム：countdown-test.c

```
1 #include <gtk/gtk.h>
2 #include "countdown.h"
3
4 static void shutdown (Countdown *countdown,
5                       gpointer  user_data)
6 {
7   guint id = GPOINTER_TO_UINT (user_data);
8
9   g_object_unref (countdown);
10  g_source_remove (id);
11
12  gtk_main_quit ();
13 }
14
15 static void cb_start (GtkWidget *widget,
16                       gpointer  user_data)
17 {
18   Countdown *countdown = COUNTDOWN (user_data);
19
20   countdown_start (countdown);
21 }
22
23 static void cb_stop (GtkWidget *widget,
24                      gpointer  user_data)
25 {
26   Countdown *countdown = COUNTDOWN (user_data);
27
```

```
28   countdown_stop (countdown);
29 }
30
31 static void cb_set (GtkWidget *widget,
32                     gpointer  user_data)
33 {
34   Countdown     *countdown = COUNTDOWN (user_data);
35   GtkSpinButton *spinbutton;
36
37   spinbutton = GTK_SPIN_BUTTON (g_object_get_data (G_OBJECT (countdown),
38                                                    "spinbutton"));
39   countdown_set_count (countdown,
40                        (guint) gtk_spin_button_get_value (spinbutton));
41 }
42
43 static void modify_label (gpointer user_data)
44 {
45   Countdown *countdown = COUNTDOWN (user_data);
46   GtkLabel  *label;
47   gchar     *string;
48
49   label = GTK_LABEL (g_object_get_data (G_OBJECT (countdown), "label"));
50   string = g_strdup_printf ("Last␣%8d␣sec", countdown_get_count (countdown));
51   gtk_label_set_label (label, string);
52   g_free (string);
53 }
54
55 int
56 main (int argc, char **argv)
57 {
58   GtkWidget *window;
59   GtkWidget *vbox;
60   GtkWidget *hbox;
61   GtkWidget *label;
62   GtkWidget *spinbutton;
63   GtkWidget *button_start;
64   GtkWidget *button_stop;
65   GtkWidget *button_set;
66   Countdown *countdown;
67   guint     id;
68
69   gtk_init (&argc, &argv);
70
71   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
72   gtk_window_set_title (GTK_WINDOW (window), "Countdown␣class␣Sample");
73   gtk_widget_set_size_request (window, 300, -1);
74   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
75   g_signal_connect (G_OBJECT (window),
76                     "destroy", G_CALLBACK (gtk_main_quit), NULL);
77
78   vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 5);
79   gtk_container_add (GTK_CONTAINER (window), vbox);
80
81   label = gtk_label_new ("Last␣10000␣msec");
82   g_object_set_data (G_OBJECT (window), "label", (gpointer) label);
83   gtk_box_pack_start (GTK_BOX (vbox), label, TRUE, TRUE, 0);
84
85   hbox = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
86   gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
87
88   spinbutton = gtk_spin_button_new_with_range (0, 30, 1);
89   gtk_spin_button_set_value (GTK_SPIN_BUTTON (spinbutton), 10);
90   gtk_box_pack_start (GTK_BOX (hbox), spinbutton, TRUE, TRUE, 0);
91
92   countdown = countdown_new (10);
93   id = g_timeout_add (500, (GSourceFunc) modify_label, (gpointer) countdown);
94   g_signal_connect (G_OBJECT (countdown), "finished",
95                     G_CALLBACK (shutdown), GUINT_TO_POINTER (id));
96   g_object_set_data (G_OBJECT (countdown), "label",
97                      (gpointer) label);
98   g_object_set_data (G_OBJECT (countdown), "spinbutton",
99                      (gpointer) spinbutton);
100
101  button_start = gtk_button_new_with_label ("Start");
102  g_signal_connect (G_OBJECT (button_start), "clicked",
103                    G_CALLBACK (cb_start), (gpointer) countdown);
```

```
104   gtk_box_pack_start (GTK_BOX (hbox), button_start, TRUE, TRUE, 0);
105
106   button_stop = gtk_button_new_with_label ("Stop");
107   g_signal_connect (G_OBJECT (button_stop), "clicked",
108                     G_CALLBACK (cb_stop), (gpointer) countdown);
109   gtk_box_pack_start (GTK_BOX (hbox), button_stop, TRUE, TRUE, 0);
110
111   button_set = gtk_button_new_with_label ("Set");
112   g_signal_connect (G_OBJECT (button_set), "clicked",
113                     G_CALLBACK (cb_set), (gpointer) countdown);
114   gtk_box_pack_start (GTK_BOX (hbox), button_set, TRUE, TRUE, 0);
115
116
117   gtk_widget_show_all (window);
118
119   gtk_main ();
120
121   return 0;
122 }
```

10 統合開発環境によるソフトウェア開発

本章では，GTK+/GNOME の統合開発環境 **Anjuta**（アニュータ）を使ったソフトウェア開発について解説します．Anjuta では，GTK+/GNOME の GUI を作成するために Glade を利用します．Glade は，GUI をその場で確認しながら作成できる便利なツールです．また，配布用のパッケージも簡単に作成できるので，知っておくと大変役に立ちます．

apt-get や synaptic などを使用して, anjuta をインストールしておきましょう．anjuta のパッケージをインストールしただけでは, プロジェクトのビルドに必要な libtool-bin パッケージがインストールされていないことがあります．libtool-bin もインストールするのも忘れないでください．

10.1 アプリケーションの構想

本章で作成するアプリケーションは 2 章で作成した画像ビューワを少しだけ拡張した画像処理アプリケーションとします．アプリケーションの仕様は以下のように定義します．

- 画像の読み込みと書き出し
- 読み込んだ画像に対して, 以下のような画像処理機能を実装
  - 画像の二値化
  - 画像の濃淡化
  - セピア画像の作成
- アンドゥ機能
- 情報ダイアログの実装

そして, メッセージの日本語化を行い, 最後に配布パッケージの作成までを目的とします．図 10.17 にこれから作成するアプリケーションを示します．

図 10.1　作成するアプリケーション

10.2　プロジェクトの作成

10.2.1　anjuta の起動

anjuta を起動するには，gnome ターミナルのようなターミナル上からコマンドラインで起動するか，デスクトップの検索から anjuta を検索して起動します．

10.2.2　新規プロジェクトの作成

anjuta を一番始めに起動すると，図 10.2 のようなダイアログが表れます．ここでは新しいプロジェクトを作成するので，一番上の「Create a new project」を選択します．
新規プロジェクトの作成を選択すると，作成するプロジェクトに関する設定や情報を入力する画面が表示されますので，順番に入力していきます．

10.2.3　アプリケーションの選択

まず，図 10.3(a) の画面が表示されるので，C のタブの「GTK+（アプリケーション）」を選択し，「進む (N)」ボタンを押して次に進みます．GTK+（シンプル）は，従来までの枠組みで作成するアプリケーションで，GTK+(アプリケーション）は，2.6 節で紹介した，GtkApplication を使用したアプリケーションになります．

10.2.4　基本情報の入力

次にプロジェクト名などのアプリケーションの基本的な情報を入力します（図 10.3(b)）．以下の 4 つの項目を入力したら，「進む (N)」ボタンを押して次に進みます．

- プロジェクト名
- 作者名
- メールアドレス
- バージョン

10.2.5　プロジェクトオプションの選択

次に表示されるプロジェクトのオプション設定では，標準で図 10.3(c) のようになっているので，何も変更せずに「進む (N)」ボタンを押してください．最後にサマリが表示されるので，「適用 (A)」ボタンを押して設定終了です．

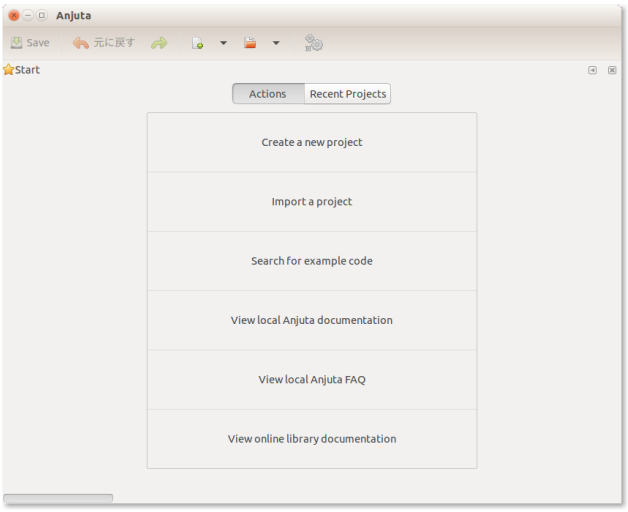
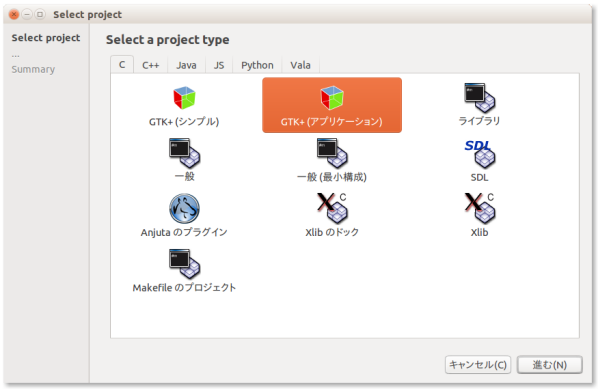


図 10.2　新規プロジェクトの作成



(a) プロジェクトの種類の選択

(b) 基本情報の入力

(c) オプション設定

(d) プロジェクトのサマリ

図 10.3　プロジェクト作成ウィザード

10.2.6　プロジェクトのビルド

プロジェクトに関する情報の入力が終了すると，アプリケーション作成に必要なファイルが自動的に作成されて，具体的にアプリケーションを作成するための画面が表示されます．この時点で，空のウィンドウを表示するソースコードが完成していますので，ファイルを編集する前にプロジェクトをビルドしてみます．プロジェクトをビルドするには，メニューから「ビルド (B)」-「プロジェクトのビルド (B)」を選択します．

プロジェクトを初めてビルドする場合は，図 10.4 のようにビルドの構成を選択するダイアログが表示されます．

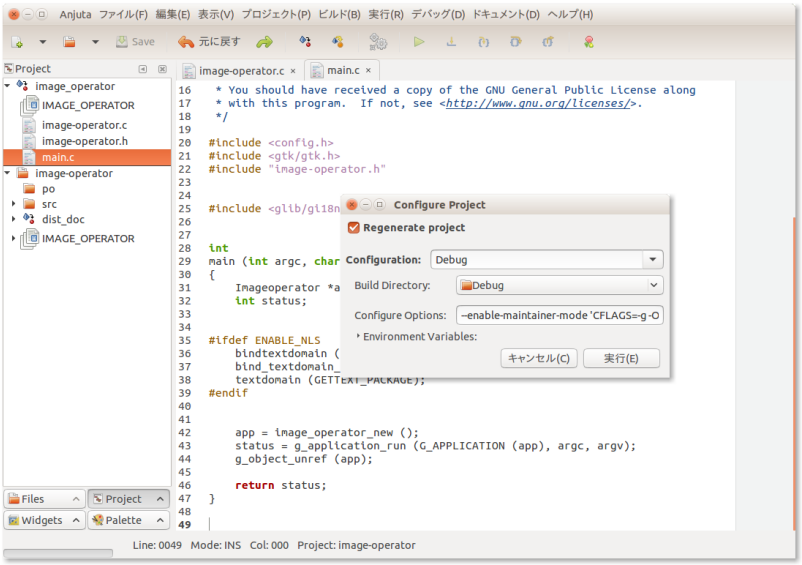


図 10.4　ビルドの構成

ここでは, 表示されている Debug のまま,「実行 (E)」ボタンを押して, プロジェクトをビルドします. ビルドが始まると, ウィンドウの下側にコンソールフレームが表示され, ビルド途中の様子が出力されます.

無事ビルドが終了したら, メニューから「実行 (R)」-「実行」を選択してビルドしたアプリケーションを実行してみてください. 図 10.5 のような画面が表示されれば成功です.

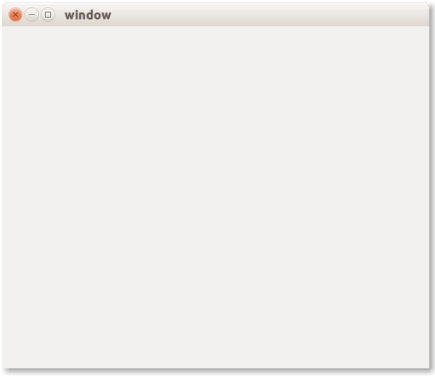


図 10.5　初期状態のアプリケーション

10.3 GUI の作成

10.3.1 glade の起動

シンプルなウィンドウが表示されることを確認したら, 次に GUI を作成してみましょう. GUI は glade というアプリケーションによって作成します. glade は anjuta からも起動できるようになっているのですが, 現在は正常に動作しないようです. そこで, 一度 anjuta を終了して, コマンドラインから glade を起動することにします. プロジェクトのディレクトリ image-operator のサブディレクトリ src 内に image_operator.ui がありますので, このファイルを glade で読み込んで起動してください. glade が起動すると図 10.6 のような画面が表示されます.

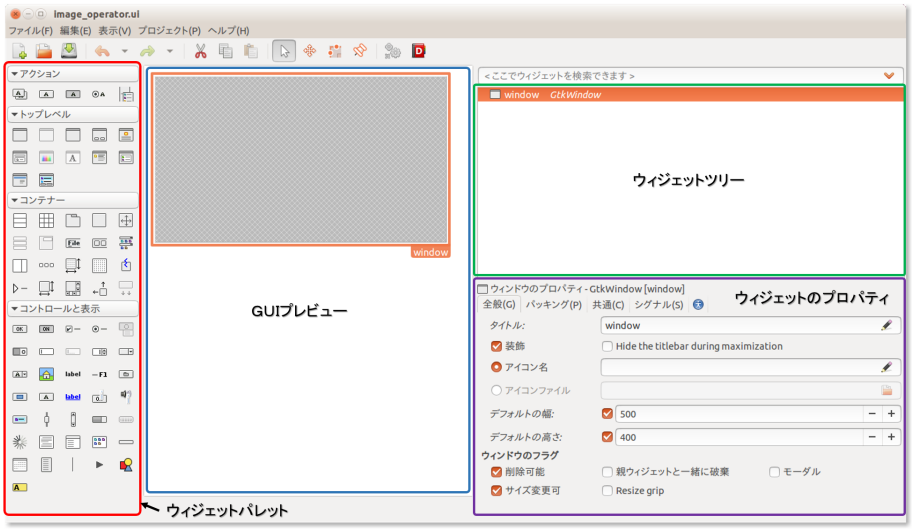


図 10.6　glade による GUI 編集画面

glade の画面は図 10.6 に色分けして示したように, 4 つの大きなフレームによって構成されています.

- ウィジェットパレット（図 10.6 の赤フレーム）
  新しく追加するウィジェットをここから選択します. パレット上で追加するウィジェットを選択して, 中央の GUI プレビューフレーム内の配置したい位置でマウスクリックすることで, ウィジェットが追加されます.
- GUI プレビュー（図 10.6 の青フレーム）
  ウィジェットの配置状況が確認できるとともに, ウィジェットパレットからウィジェットを選択して, GUI を構成していくフーレムです.

- ウィジェットツリー（図 10.6 の緑フレーム）
  現在の GUI の構成をツリー形式で表示します．
- ウィジェットのプロパティ（図 10.6 の紫フレーム）
  選択したウィジェットのプロパティを設定したり，コールバック関数を定義したりします．

10.3.2 GUI の構成

今回作成するアプリケーションは基本的には画像を表示するシンプルなものなので，ウィジェットの構成は図 10.7 に示すような簡単な構造をしています．まず，注意が必要なのは，初期状態では，GtkWindow ウィジェットがトップに配置されていることです．今回は GtkBuilder を使ってメニューを作成，配置するため，このウィンドウを削除してからアプリケーションウィンドウ (GtkApplicationWindow) をトップに配置してください．

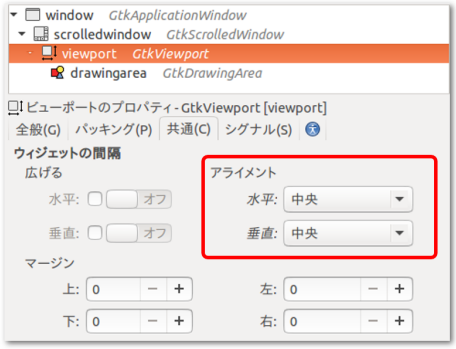


図 10.7 アプリケーションのウィジェット配置

次に，2 章で作成したアプリケーションでは，スクロールバー付きウィンドウの中に直接イメージウィジェットを配置することができましたが，glade ではスクロールバー付きウィンドウの中にウィジェットを配置しようとすると，先にビューポートウィジェットを配置するように警告がでます．仕方がないので，図 10.7 のツリー構造のように，ビューポートウィジェットを配置し，その中に画像を描画するためのドローイングエリアウィジェットを配置しています．

最後に，ドローイングエリアウィジェットをウィンドウの中央に配置するために，ビューポートウィジェットのプロパティタブで，アライメントの部分を，水平と垂直ともに中央の設定にしておきます．

10.3.3 メニューの生成

メニュー定義ファイルの作成

次にメニューを作成します．メニューは xml ファイルで定義し，GtkBuilder で読み込ませるようにします．そこで，まず anjuta の画面から新しいファイルをプロジェクトに追加します．新しいファイルを追加するには，メニューから「ファイル (F)」-「New」-「3. ファイル」を選択するか，ツールバーの一番左のプルダウンボタンをクリックし，「3. ファイル」を選択します．

ファイルの追加メニューを選択すると，図 10.8 のようなウィンドウが表示されますので，二つの項目の Type を Other に選択して，Name に menu.ui と入力します．そして，Add to project target の項目で，ui を選択して，「OK(O)」ボタンを押します．

ファイルの追加に成功すると中央の画面に menu.ui の編集画面が表示されるので，ソース 10–1 の内容を入力します．メニュー構成の記述方法については，2.7 節や 7.4.3 節を参照してください．ソース 10–1 の内容は，以下のような階層構造のメニューを定義したものです．

- File
  - Open
  - Save
  - Save as
  - Quit
- Image
  - Binalize
  - Gray level
  - Sepia
- Help
  - About

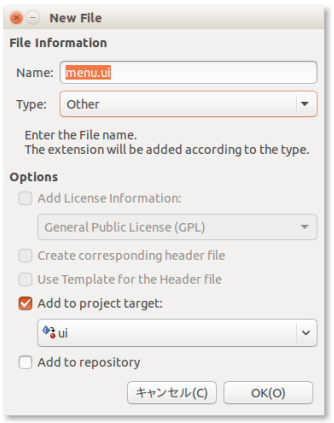


図 10.8　ファイルの追加

ソース 10–1 メニュー構成： menu.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <menu id="appmenu">
    <submenu>
      <attribute name="label"
                 translatable="yes">_File</attribute>
      <section>
        <item>
          <attribute name="label"
                     translatable="yes">_Open</attribute>
          <attribute name="action">app.open</attribute>
          <attribute name="accel">&lt;Control&gt;o</attribute>
        </item>
        <item>
          <attribute name="label"
                     translatable="yes">_Save</attribute>
          <attribute name="action">app.save</attribute>
          <attribute name="accel">&lt;Control&gt;s</attribute>
        </item>
        <item>
          <attribute name="label"
                     translatable="yes">_Save as</attribute>
          <attribute name="action">app.save-as</attribute>
          <attribute name="accel">&lt;Shift&gt;&lt;Control&gt;s</attribute>
        </item>
      </section>
      <section>
        <item>
          <attribute name="label"
                     translatable="yes">_Quit</attribute>
          <attribute name="action">app.quit</attribute>
          <attribute name="accel">&lt;Control&gt;q</attribute>
        </item>
      </section>
    </submenu>
    <submenu>
      <attribute name="label"
                 translatable="yes">_Image</attribute>
      <section>
        <item>
          <attribute name="label"
                     translatable="yes">_Binalize</attribute>
          <attribute name="action">app.bin</attribute>
          <attribute name="accel">&lt;Alt&gt;b</attribute>
        </item>
        <item>
          <attribute name="label"
                     translatable="yes">_Gray level</attribute>
          <attribute name="action">app.gray</attribute>
          <attribute name="accel">&lt;Alt&gt;g</attribute>
        </item>
        <item>
```

```
53          <attribute name="label"
54                     translatable="yes">_Sepia</attribute>
55          <attribute name="action">app.sepia</attribute>
56          <attribute name="accel">&lt;Alt&gt;s</attribute>
57        </item>
58      </section>
59    </submenu>
60    <submenu>
61      <attribute name="label"
62                 translatable="yes">_Help</attribute>
63      <section>
64        <item>
65          <attribute name="label"
66                     translatable="yes">_About</attribute>
67          <attribute name="action">app.about</attribute>
68          <attribute name="accel">&lt;Control&gt;a</attribute>
69        </item>
70      </section>
71    </submenu>
72  </menu>
73 </interface>
```

GActionEntry 構造体の設定

次に, ファイル image-operator.c に定義したメニューに対する GActionEntry 構造体の設定します. また, メニューアイテムに対するコールバック関数を実装するファイル callbacks.c と callabacks.h を追加して, コールバック関数の雛形とプロトタイプ宣言を記述しておきます.

ソース 10–2 に GActionEntry の設定, ソース 10–3 にコールバック関数のヘッダファイル, ソース 10–4 にコールバック関数の雛形の一部を示します.

ソース 10–2 GActionEntry 構造体の設定：image-operator.c（一部を抜粋）

```
1  ...
2  #include "image-operator.h"
3  #include "callbacks.h"
4  ...
5  struct _ImageoperatorPrivate
6  {
7          /* ANJUTA: Widgets declaration for image_operator.ui - DO NOT REMOVE */
8  };
9
10 static GActionEntry entries[] = {
11           {"open",    cb_open,         NULL, NULL, NULL},
12           {"save",    cb_save,         NULL, NULL, NULL},
13           {"save-as", cb_saveas,       NULL, NULL, NULL},
14           {"quit",    cb_quit,         NULL, NULL, NULL}
15           {"bin",     cb_image_bin,    NULL, NULL, NULL},
16           {"gray",    cb_image_gray,   NULL, NULL, NULL},
17           {"sepia",   cb_image_sepia,  NULL, NULL, NULL},
18           {"about",   cb_about,        NULL, NULL, NULL}
19 };
20
21 /* Create a new window loading a file */
22 static void
23 image_operator_new_window (GApplication *app,
24                            GFile        *file)
25 {
26 ...
```

ソース 10–3 callbacks.h

```
1 #ifndef __CALLBACKS_H__
2 #define __CALLBACKS_H__
3
4 #include <gtk/gtk.h>
5
6 void cb_open        (GSimpleAction *action, GVariant *parameter, gpointer user_data);
7 void cb_save        (GSimpleAction *action, GVariant *parameter, gpointer user_data);
8 void cb_saveas      (GSimpleAction *action, GVariant *parameter, gpointer user_data);
9 void cb_quit        (GSimpleAction *action, GVariant *parameter, gpointer user_data);
```

```
10 void cb_image_bin   (GSimpleAction *action, GVariant *parameter, gpointer user_data);
11 void cb_image_gray  (GSimpleAction *action, GVariant *parameter, gpointer user_data);
12 void cb_image_sepia (GSimpleAction *action, GVariant *parameter, gpointer user_data);
13 void cb_about       (GSimpleAction *action, GVariant *parameter, gpointer user_data);
14
15 #endif __CALLBACKS_H__
```

ソース 10–4 callbacks.c （一部を抜粋）

```
1  #include <gtk/gtk.h>
2
3  void cb_open (GSimpleAction *action, GVariant *parameter, gpointer user_data)
4  {
5
6  }
7   ...
8
9  void cb_about (GSimpleAction *action, GVariant *parameter, gpointer user_data)
10 {
11
12 }
```

gtkbuilder を用いたメニューの実装

最後にメニューの定義ファイルを gtkbuilder から読み込んでメニューを実装します．以下の順番で作業の説明を行います．

1. メニュー UI のファイルパス定義
   GUI のファイルパス定義と同様にメニュー UI のファイルパスを次のようにしておきます．

```
#define MENU_FILE "src/menu.ui"
```

2. ファイル image-operator.c 中の関数 image_opearator_new_window にメニュー読み込み等の記述を追加

   記述内容は, 以下のようになります．
   - 関数 g_action_map_add_action_entries によるメニューアイテムに対するコールバック関数の登録
   - 関数 gtk_builder_add_from_file によるメニュー定義ファイルの読み込み
   - 関数 gtk_application_set_menubar によるメニューバーの配置

具体的な記述内容をソース 10–5 に示します．

ソース 10–5 メニューの作成: image-operator.c （一部を抜粋）

```
1     g_action_map_add_action_entries (G_ACTION_MAP (app),
2                                      entries, G_N_ELEMENTS (entries), app);
3
4     if (!gtk_builder_add_from_file (builder, MENU_FILE, &error))
5     {
6         g_critical ("Couldn't␣load␣builder␣file:␣%s", error->message);
7         g_error_free (error);
8     }
9     GMenuModel *menubar = (GMenuModel *) gtk_builder_get_object (builder, "appmenu");
10    gtk_application_set_menubar (GTK_APPLICATION (app), menubar);
11
12    g_object_unref (builder);
```

これでアプリケーションの GUI 構成は終わったので，アプリケーションをビルドして実行してみましょう．アプリケーションの実行画面を図 10.9 に示します．

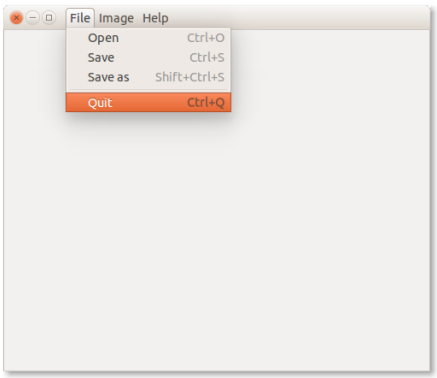


図 10.9　アプリケーションの GUI 画面

10.4　コールバック関数の実装

ここから具体的にコールバック関数を作成していきます．作成するコールバック関数を表 10.1 にまとめました．最後の関数 cb_draw はドローイングエリアウィジェットの draw シグナルに対するコールバック関数で, 画像の描画を実装します.

表 10.1　コールバック関数の一覧

関数名	説明
cb_open	画像をオープンするための関数
cb_save	画像処理を施した画像を保存するための関数
cb_saveas	画像処理を施した画像を別名で保存するための関数
cb_quit	アプリケーションを終了する関数
cb_image_bin	画像処理関数（2 値画像の作成）
cb_image_gray	画像処理関数（濃淡画像の作成）
cb_image_sepia	画像処理関数（セピア画像の作成）
cb_about	アプリケーション情報を表示する関数
cb_draw	ドローイングエリアの描画に対する関数

10.4.1　ドローイングエリアのコールバック関数の実装

画像データをドローイングエリアへ描画する処理を，コールバック関数 cb_draw 内に実装します.

プライベートメンバへのアクセス関数

まず, コールバック関数実装の準備として, 構造体 ImageoperatorPrivate のメンバに, GdkPixbuf 型の変数 pixbuf を追加し, 画像データはここからアクセスすることにします．また, 画像の保存のときに使用する現在表示している画像のファイル名のための変数 filename と, 画像処理コールバック関数から呼び出すためのドローイングエリアウィジェットをメンバとして登録します．そして, これらのメンバにアクセスするための関数をソース 10–6 のように実装します．また, これらの関数のプロトタイプ宣言を image-operator.h に記述しておきます.

ソース 10–6　ImageOperatorPrivate メンバへのアクセス関数

```
1  struct _ImageoperatorPrivate
2  {
3      /* ANJUTA: Widgets declaration for image_operator.ui - DO NOT REMOVE */
4      gchar      *filename;
5      GdkPixbuf  *pixbuf;
6      GtkWidget  *drawingarea;
7  };
8
9  gchar*
10 image_operator_get_filename (Imageoperator *app)
```

```
{
        ImageoperatorPrivate *priv = IMAGE_OPERATOR_APPLICATION(app)->priv;
        return priv->filename;
}

void
image_operator_set_filename (Imageoperator *app, gchar *filename)
{
        ImageoperatorPrivate *priv = IMAGE_OPERATOR_APPLICATION(app)->priv;
        g_return_if_fail (filename && filename[0] != '\0');

        if (priv->filename) g_free (priv->filename);
        priv->filename = filename;
}

GdkPixbuf*
image_operator_get_pixbuf (Imageoperator *app)
{
        ImageoperatorPrivate *priv = IMAGE_OPERATOR_APPLICATION(app)->priv;
        return priv->pixbuf;
}

void
image_operator_set_pixbuf (Imageoperator *app, GdkPixbuf *pixbuf)
{
        ImageoperatorPrivate *priv = IMAGE_OPERATOR_APPLICATION(app)->priv;
        g_return_if_fail (pixbuf);

        if (priv->pixbuf) g_free (priv->pixbuf);
        priv->pixbuf = pixbuf;
}

GtkWidget*
image_operator_get_drawingarea (Imageoperator *app)
{
        ImageoperatorPrivate *priv = IMAGE_OPERATOR_APPLICATION(app)->priv;
        return priv->drawingarea;
}
```

コールバック関数 cb_draw

次に, コールバック関数 cb_draw をファイル callbacks.c に実装します. ソース 10–7 で実装した関数を用いて, 画像データを取得し, 関数 gdk_cairo_set_source_pixbuf で画像データを cairo のソースとして設定して描画します. ここでは, 関数の第 3 引数にユーザーデータとして Imageoperator 型の変数が与えられるものとしています. この設定については次の節で行います.

ソース 10–7 コールバック関数 cb_draw（callbacks.c から抜粋）

```
gboolean cb_draw (GtkWidget *widget,
                  cairo_t   *cr,
                  gpointer  user_data)
{
        Imageoperator *app = IMAGE_OPERATOR_APPLICATION (user_data);
        GdkPixbuf *pixbuf = image_operator_get_pixbuf (app);

        if (pixbuf) {
                int width  = gdk_pixbuf_get_width (pixbuf);
                int height = gdk_pixbuf_get_height (pixbuf);
                gtk_widget_set_size_request (widget, width, height);
                gdk_cairo_set_source_pixbuf (cr, pixbuf, 0, 0);
                cairo_paint (cr);
        }
        return FALSE;
}
```

draw **シグナルへのコールバック関数の関連付け**

コールバック関数を実装しましたので, 最後にドローイングエリアウィジェットの draw シグナルに関連付けて呼び出す設定を行います.

関数 gtk_builder_get_object を使ってドローイングエリアウィジェットを取得し, 関数 g_signal_connect でコールバック関数の設定を行います. コールバック関数の第 3 引数に Imageoperator 型の変数を渡すために, 関数 g_signal_connect の第 4 引数

に Imageoperator 型の変数 app を指定しています.

ソース 10–8 コールバック関数 cb_draw（image-operator.c から抜粋）

```
priv->drawingarea
    = GTK_WIDGET (gtk_builder_get_object (builder, "drawingarea"));
g_signal_connect (G_OBJECT(priv->drawingarea), "draw",
                  G_CALLBACK(cb_draw), (gpointer) app);
```

プログラム起動時の画像の読み込み

アプリケーションの GUI 作成関数である image_operator_window_new の第 2 引数には, プログラム実行時の引数（ファイル名）が与えられるようになっています. そこで, プログラムの実行時にファイル名が指定された場合には, この関数内で画像を読み込みようにします. anjuta が自動生成した関数内には, 対応するソースコードを記述する部分が用意されているので, そこに画像の読み込み関数を記述します.

ソース 10–9

```
if (file != NULL)
{
    /* TODO: Add code here to open the file in the new window */
    priv->filename = g_file_get_path (file);
    priv->pixbuf = gdk_pixbuf_new_from_file (filename, NULL);
}
```

実装内容の確認

以上で, プログラム実行時に画像ファイル名を入力するとその画像を読み込み, 表示することができるようになります. 動作を確認するために, プロジェクトをビルドしてアプリケーションを実行してみましょう. アプリケーションの実行時に引数を与えるには, メニューから「実行 (R)」-「プログラムのパラメータ...」を選択して, 図 10.10(a) のダイアログを表示し, Arguments 欄に src/smoke-dance.png と入力しておきます. これは, プロジェクトディレクトリ下の src ディレクトリ内に画像ファイル smoke-dance.png が存在することを前提としています. テストする際には, 各自の環境に合わせてディレクトリ名や画像ファイル名を変更してください.

(a) プログラムの引数設定　　(b) 画像の表示までの実装結果

図 10.10　これまでのアプリケーションの実行結果

アプリケーションを実行した画面が図 10.10(b) になります. 指定した画像を読み込んで画面中央に表示されていることがわかります.

10.4.2　画像を読み込むコールバック関数の実装

画像の読み込みは，ファイル選択ダイアログから画像ファイル名を指定することにします. 表示する画像ファイル名をファイル選択ダイアログから取得するために，第 2 章で紹介した GtkFileChooser を利用します.

ファイルメニューで「Open」を選択すると，コールバック関数 cb_open が呼び出されます. この関数内でファイルの選択と，選択された画像を読み込む処理を実装します.

関数 cb_open の実装結果をソース 10–10 に示します．処理内容はほとんどチュートリアルで示したものと同様です．24–32 行目でファイル名を取得して, 画像データを読み込み, 画像データとファイル名をアプリケーションに登録します．

ソース 10–10 画像の読み込み関連の関数：callbacks.c から一部抜粋

```
1  void cb_open (GSimpleAction *action, GVariant *parameter, gpointer user_data)
2  {
3          GtkApplication *app = GTK_APPLICATION (user_data);
4          GtkWindow      *window = gtk_application_get_active_window (app);
5          GtkWidget      *dialog;
6          gint           response;
7
8          dialog = gtk_file_chooser_dialog_new ("Open␣an␣image",
9                                                GTK_WINDOW (window),
10                                               GTK_FILE_CHOOSER_ACTION_OPEN,
11                                               "_Cancel",
12                                               GTK_RESPONSE_CANCEL,
13                                               "_Open",
14                                               GTK_RESPONSE_ACCEPT,
15                                               NULL);
16         gtk_widget_show_all (dialog);
17
18         response = gtk_dialog_run (GTK_DIALOG (dialog));
19         if (response == GTK_RESPONSE_ACCEPT)
20         {
21                 Imageoperator *iapp = IMAGE_OPERATOR_APPLICATION(user_data);
22                 gchar *filename =
23                   gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
24                 GdkPixbuf *pixbuf = gdk_pixbuf_new_from_file (filename, NULL);
25
26                 image_operator_set_pixbuf (iapp, pixbuf);
27                 image_operator_set_filename (iapp, filename);
28         }
29         gtk_widget_destroy (dialog);
30 }
```

10.4.3　画像を保存するコールバック関数の実装

画像の保存を行うメニューには「save」と「save as」の 2 つがあります．「save」メニューが選択された場合は，現在表示されている画像ファイルを同じファイル名で保存します．「save as」メニューを選択した場合には，ファイル選択ダイアログを表示して，ファイル名を指定して画像を保存します．

ファイル選択ダイアログを表示してファイル名を選択する部分は，先ほどのコールバック関数 cb_open の内容を利用します．実際には画像を保存する部分は別の関数_save で行うことにし，「save」メニューのコールバック関数 cb_save では単に関数_save を呼び出します．ここでは, 画像の保存に対応する画像フォーマットは jpeg と png のみにしています．画像の保存に関するソースコードを，ソース 10–11 に示します．

ソース 10–11 画像の保存関連の関数：callbacks.c から一部抜粋

```
1  static gboolean _save (GdkPixbuf *pixbuf, gchar *filename)
2  {
3          gchar *extensions[] = {"jpg", "png", NULL};
4          gchar *exts[] = {"jpeg", "png"};
5          gchar *ext;
6
7          ext = g_strrstr (filename, ".");
8          if (ext)
9          {
10                 ext++;
11                 for (int n = 0; extensions[n] != NULL; n++)
12                 {
13                         if (g_strcmp0 (ext, extensions[n]) == 0)
14                         {
15                                 gdk_pixbuf_save (pixbuf, filename, exts[n], NULL, NULL);
16                                 return TRUE;
17                         }
18                 }
19                 g_printerr ("This␣extension␣is␣not␣supported.\n");
```

```
20                  return FALSE;
21          }
22          else
23          {
24                  g_printerr ("This␣extension␣is␣not␣supported.\n");
25                  return FALSE;
26          }
27  }
28
29  void cb_save (GSimpleAction *action, GVariant *parameter, gpointer user_data)
30  {
31          Imageoperator *app = IMAGE_OPERATOR_APPLICATION (user_data);
32          _save (image_operator_get_pixbuf (app), image_operator_get_filename (app));
33  }
34
35  void cb_saveas (GSimpleAction *action, GVariant *parameter, gpointer user_data)
36  {
37          GtkApplication *app = GTK_APPLICATION (user_data);
38          GtkWindow      *window = gtk_application_get_active_window (app);
39          GtkWidget      *dialog;
40          gint           response;
41
42          dialog = gtk_file_chooser_dialog_new ("Save␣an␣image",
43                                                GTK_WINDOW (window),
44                                                GTK_FILE_CHOOSER_ACTION_SAVE,
45                                                "_Cancel",
46                                                GTK_RESPONSE_CANCEL,
47                                                "_Save",
48                                                GTK_RESPONSE_ACCEPT,
49                                                NULL);
50          gtk_widget_show_all (dialog);
51
52          response = gtk_dialog_run (GTK_DIALOG (dialog));
53          if (response == GTK_RESPONSE_ACCEPT)
54          {
55                  Imageoperator *iapp = IMAGE_OPERATOR_APPLICATION(user_data);
56                  gchar *filename
57                    = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
58                  GdkPixbuf *pixbuf = image_operator_get_pixbuf (iapp);
59
60                  _save (pixbuf, filename);
61                  image_operator_set_filename (iapp, filename);
62          }
63          gtk_widget_destroy (dialog);
64  }
```

10.4.4　終了コールバック関数の実装

コールバック関数 cb_quit では，画像データ等の領域を解放してアプリケーションを終了する関数を呼び出します．実装結果をソース 10–12 に示します．

ソース 10–12　関数 cb_quit：callbacks.c から一部を抜粋

```
1  void cb_quit (GSimpleAction *action, GVariant *parameter, gpointer user_data)
2  {
3          GApplication *app = G_APPLICATION (user_data);
4          Imageoperator *iapp = IMAGE_OPERATOR_APPLICATION (user_data);
5
6          if (image_operator_get_filename (iapp))
7          {
8                  g_free (image_operator_get_filename (iapp));
9          }
10         if (image_operator_get_pixbuf (iapp))
11         {
12                 g_object_unref (image_operator_get_pixbuf (iapp));
13         }
14         g_application_quit (app);
15 }
```

10.4.5　アプリケーション情報を表示するコールバック関数の実装

ここではアプリケーションの情報を表示するためにアバウトダイアログを使用します．アバウトダイアログの詳細については 7.5.4（p. 191）を参照してください．

コールバック関数 cb_about をソース 10–13 に示します．このコールバック関数によって表示されるダイアログを図 10.11 に示します．

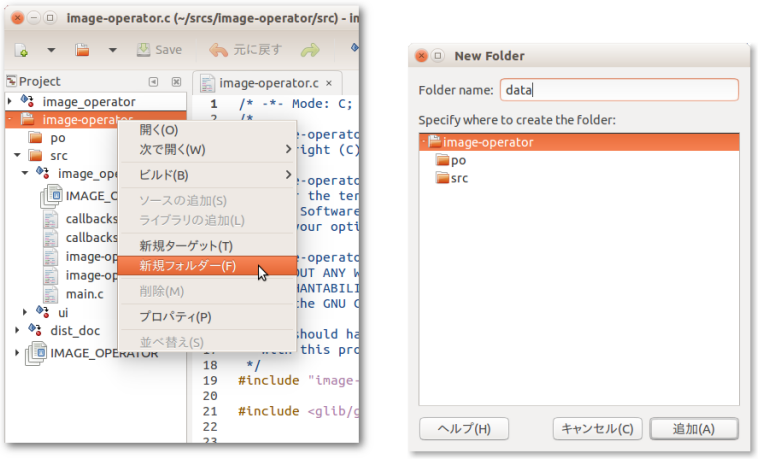
図 10.11　アプリケーション情報ダイアログ

ここでは，ダイアログで使用するアイコンデータをプロジェクトに追加する方法について説明します．具体的には，以下の項目について説明します．

- プロジェクトにアイコンデータを置くディレクトリを追加する．
- 追加したディレクトリにアイコンデータを追加して，パッケージとしてインストールする設定をする．

プロジェクトに新しいフォルダーを追加

プロジェクトに新しいフォルダーを追加するには，まず，画面左のプロジェクトフレームの image-operator という名前のディレクトリのところでマウスの右ボタンをクリックし，メニューを表示させ（10.12(a) ），「新規フォルダー (F)」を選択して，New Folder というダイアログを表示させます．そして，Folder name の欄に追加するフォルダー名を入力します．ここでは，フォルダー名を data とします（10.12(b) ）．

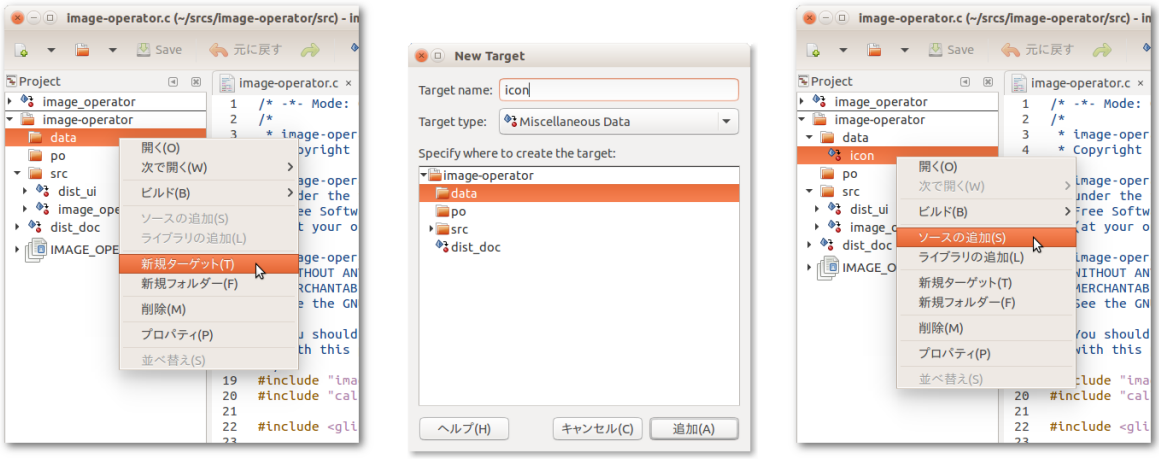


(a)　(b)

図 10.12　新規フォルダーの追加

アイコンファイルの追加

次に，作成したフォルダーにアイコンデータを追加します．まず，画面左のプロジェクトフレームの今作成したフォルダーのところでマウスの右ボタンをクリックし，メニューを表示させ（10.13(a) ），「新規ターゲット (T)」を選択して，New Target というダイアログを表示させます．そして，Target name の欄に icon と入力し，Target type を Miscellaneous Data とします（10.13(b) ）．新しいターゲットとして icon が追加されますので，そこでマウスの右ボタンをクリックし，メニューを表示させ（10.13(c) ），「ソースの追加 (S)」を選択してファイル選択ダイアログを表示させて，登録するアイコンファイルを選択して，アイコンファイルを追加します．



(a)　(b)　(c)

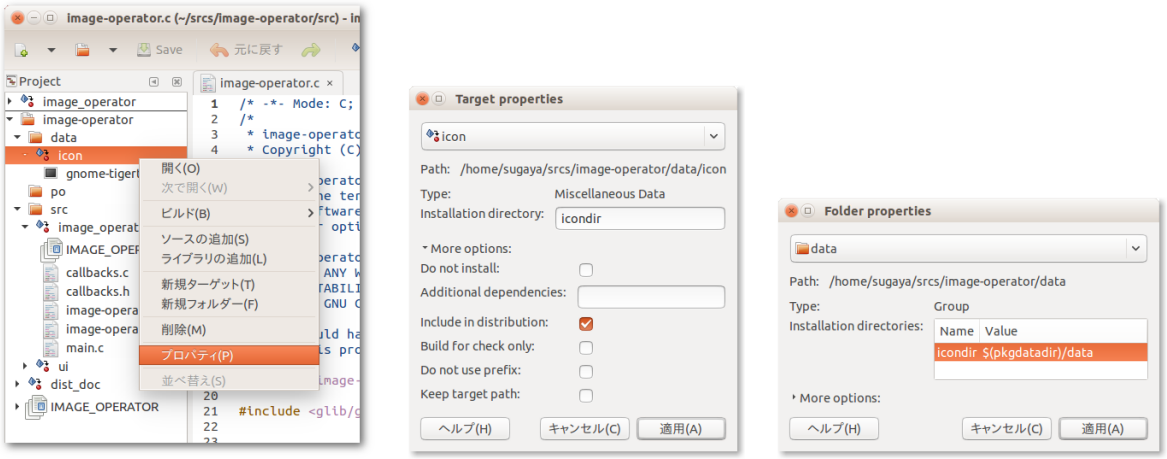
図 10.13　新規ターゲットの追加

ターゲットのプロパティ設定

追加したアイコンファイルは, このままでは配布パッケージには含まれない設定になっています. そこで, 以下の作業を行って, 配布パッケージに含める設定とインストールするディレクトリの設定をします.

まず, icon ターゲット上でマウスの右ボタンをクリックしてメニューを表示させ, 「プロパティ (P)」を選択します（10.14(a)）. 次に表示されたダイアログの「More options」をクリックし, 隠れたオプションを表示させて「Include distribution」の欄にチェックを入れ, 「適用 (A)」ボタンを押します（10.14(b)）. この設定を行うと, ターゲットの表示が icon から dist_icon に変更されます. src ディレクトリ内のターゲット ui についても同様の設定を行って, ui ファイルも配布パッケージに含まれるようにしておいてください.

最後に, data フォルダでマウスの右ボタンをクリックしてメニューを表示し, 「プロパティ (P)」を選択します. Folder property ダイアログが表示されますので, Installation directories 欄の icondir の value に $(pkgdatadir)$/data と入力して「適用 (A)」ボタンを押してください（10.14(c)）. 以上で設定は終了になります.



(a)　(b)　(c)

図 10.14　ターゲットのプロパティ設定

ソース 10–13　関数 cb_about：callbacks.c から一部抜粋

```
1 void cb_about (GSimpleAction *action, GVariant *parameter, gpointer user_data)
2 {
3         GtkApplication *app = GTK_APPLICATION (user_data);
4
5         const gchar *programname = "image-operator";
6         const gchar *authors[] = {"gtk", NULL};
```

```
7         const gchar *documenters[] = {"gtk", NULL};
8         const gchar *translator = "gtk";
9         const gchar *version = "0.1";
10        const gchar *copyright = "Copyright␣(C)␣2016";
11        const gchar *comments = "This␣is␣a␣simple␣image␣operator␣program.";
12        const gchar *website = "http://www...";
13        GdkPixbuf *logo = gdk_pixbuf_new_from_file (LOGO_FILE, NULL);
14
15        gtk_show_about_dialog (gtk_application_get_active_window (app),
16                               "program-name",       programname,
17                               "authors",            authors,
18                               "documenters",        documenters,
19                               "translator-credits", translator,
20                               "version",            version,
21                               "copyright",          copyright,
22                               "license-type",       GTK_LICENSE_GPL_3_0,
23                               "comments",           comments,
24                               "website",            website,
25                               "logo",               logo,
26                               NULL);
27        g_object_unref (logo);
28 }
```

10.4.6　画像処理関数の実装

最後に画像処理に関するコールバック関数を実装します．具体的には，GdkPixbuf データの画像座標 (x, y) の画素に対するアクセスマクロを作成し，各コールバック関数内でそれぞれの画像処理を実現します．しかし，この部分は anjuta の使い方とは関連のない部分ですので，画素データに対するアクセスマクロとコールバック関数 cb_sepia の実装結果のみを示すことにします（ソース 10–14）．

ソース 10–14　画素値の読み書きを行うマクロと関数 cb_sepia：image-operator.c から一部抜粋

```
1  #define gdk_pixbuf_get_pixel(pixbuf,x,y,p) \
2  (*(gdk_pixbuf_get_pixels((pixbuf)) + \
3   gdk_pixbuf_get_rowstride((pixbuf)) * (y) + \
4   gdk_pixbuf_get_n_channels((pixbuf)) * (x) + (p)))
5
6  #define gdk_pixbuf_put_pixel(pixbuf,x,y,p,val) \
7  (*(gdk_pixbuf_get_pixels((pixbuf)) + \
8   gdk_pixbuf_get_rowstride((pixbuf)) * (y) + \
9   gdk_pixbuf_get_n_channels((pixbuf)) * (x) + (p)) = (val))
10
11 void cb_image_sepia (GSimpleAction *action, GVariant *parameter, gpointer user_data)
12 {
13     Imageoperator *app = IMAGE_OPERATOR_APPLICATION (user_data);
14     GdkPixbuf *pixbuf = image_operator_get_pixbuf (app);
15
16     for (int y = 0; y < gdk_pixbuf_get_height (pixbuf); y++)
17     {
18         for (int x = 0; x < gdk_pixbuf_get_width (pixbuf); x++)
19         {
20                 guchar r = gdk_pixbuf_get_pixel (pixbuf, x, y, 0);
21                 guchar g = gdk_pixbuf_get_pixel (pixbuf, x, y, 1);
22                 guchar b = gdk_pixbuf_get_pixel (pixbuf, x, y, 2);
23                 guchar Y = (guchar) (0.299 * r + 0.587 * g + 0.114 * b);
24
25                 gdk_pixbuf_put_pixel (pixbuf, x, y, 0, (guchar) (Y * 240 / 255.0));
26                 gdk_pixbuf_put_pixel (pixbuf, x, y, 1, (guchar) (Y * 200 / 255.0));
27                 gdk_pixbuf_put_pixel (pixbuf, x, y, 2, (guchar) (Y * 145 / 255.0));
28         }
29     }
30     gtk_widget_queue_draw (image_operator_get_drawingarea (app));
31 }
```

10.5 配布パッケージの作成

ここでは作成したアプリケーションを配布用にパッケージングする方法を説明します.

10.5.1 ファイルマクロのの修正

まず, image-operator.c で定義した ui ファイルのマクロと callbacks.c で定義した アイコンファイルのマクロをアプリケーションを正式にインストールした後のパスに修正します.

パッケージをインストールすると，ui ファイルは PACKAGE_DATA_DIR/ui 内にインストールされます．また, アイコンファイルは, PACKAGE_DATA_DIR/data 内にインストールするように設定しましたので, それぞれ以下のようにマクロを修正します.

```
#define UI_FILE   PACKAGE_DATA_DIR"/ui/image_operator.ui"
#define MENU_FILE PACKAGE_DATA_DIR"/ui/menu.ui"

#define LOGO_FILE PACKAGE_DATA_DIR"/data/gnome-tigert.png"
```

10.5.2 配布パッケージの作成

作成したアプリケーションが正常に動作することが確認できていれば，配布パッケージを作成するのは非常に簡単です．メニューの「ビルド (B)」-「Tarball の生成 (T)」を選択してください．Tarball とは，tar というコマンドで複数のファイルを 1 つのファイルにまとめたもので，通常は gzip コマンド等でさらにファイルを圧縮しています.

Tarball を正常に生成できた場合は，プロジェクトディレクトリ内に imageoperator-0.1.tar.gz というファイルが生成されているはずなので，確認してください．プロジェクトの構成で Debug を選択している場合には, プロジェクトのディレクトリ Debug 内に Tarball が作成されています.

10.5.3 配布パッケージのコンパイル

配布パッケージを作成できたので，これを使ってアプリケーションをコンパイルしてインストールしてみましょう.

次のように配布パッケージを適当なディレクトリ（ここではホームディレクトリの tmp ディレクトリに展開します）に展開し，コンパイルを行います．コンパイルが正常終了したら make install コマンドでアプリケーションをインストールします．ここでは prefix の値を~/tmp/usr に設定して，アプリケーションを仮のディレクトリにインストールしています.

```
$ cd ↵
$ mkdir tmp ↵
$ cd tmp ↵
$ tar xvfz ~/image_operator-0.1.tar.gz ↵
...
$ cd image_operator-0.1 ↵
$ ./configure --prefix=~/tmp/usr ↵
...
$ make ↵
...
$ make install ↵
```

最後にインストールしたファイルを確認します．以下に示すように，バイナリファイルやドキュメントがそれぞれのディレクトリにインストールされていることがわかります.

```
$ cd ~/tmp/usr ↵
$ ls -R
```

```
.:
bin/ share/

./bin:
image_operator*

./share:
doc/ image_operator/

./share/doc/image_operator:
AUTHORS  COPYING  ChangeLog  INSTALL  NEWS  README

./share/image_operator:
data/ ui/

./share/image_operator/data:
gnome-tigert.png

./share/image_operator/ui:
image_operator.ui  menu.ui
```

10.6　メッセージの国際化

最近のアプリケーションは，日本語環境で実行するとメッセージが日本語で表示されて大変親切です．gettext を利用すれば，ソースコードに含まれたメッセージに対する翻訳ファイルを用意しておくことで，使用する環境に合わせた言語でメッセージを表示できます．

10.6.1　翻訳メッセージの作成

まず，ソースコード中の翻訳対象となる文字列を指定します．翻訳対象の文字列を指定するには，その文字列を_() で囲みます．今回作成したアプリケーションでは，image-operator.c と callbacks.c と menu.ui 中に翻訳対象となる文字列が含まれます．ただし，menu.ui 中の文字列については，アイテムの attribute で translatable の設定をしているため, 何もする必要はありません．また，メッセージ翻訳設定のために, callbacks.c でヘッダファイル glib/gi18n.h をインクルードしておきます．

次に，プロジェクトディレクトリ（image-operator）内の po というディレクトリ内のファイルを編集します．この後の節で説明しますが，このディレクトリ内に翻訳のためのファイルが生成されます．ここではまず, ファイル LINGUAS に，以下のように ja と入力します．これは日本語の翻訳メッセージファイルを作成することを意味します．

```
# please keep this list sorted alphabetically
#
ja
```

次に，ファイル POTFILE.in を以下のように編集します．ここには翻訳対象となる文字列が含まれるファイル名を列挙します．

```
# List of source files containing translatable strings.

src/main.c
src/image-operator.c
```

```
src/callbacks.c
src/menu.ui
```

ここで一度プロジェクトをビルドすると，上記の修正が反映されます．ここでまだ翻訳ファイルが生成されていないので，エラーメッセージが表示されますが問題ありません．翻訳ファイルを生成するために，端末上で次のコマンドを実行してください．

```
$ cd ~/srcs/image-operator ↵
$ xgettext -k_ -f po/POFILES.in -o po/image-operator.pot ↵
```

正常にコマンドが終了したら，ディレクトリ po 内に image-operator.pot というファイルが生成されます．これを ja.po という名前でコピーして，このファイルに翻訳メッセージを入力していきます．翻訳に進む前に，ja.po 内の以下の部分を修正してください．

```
"Project-Id-Version: image-operator 0.1\n"
...
"Last-Translator: gtk <gtk@>\n"
"Language-Team: gtk <gtk@>\n"
"Language: ja\n"
...
"Content-Type: text/plain; charset=UTF-8\n"
```

10.6.2　メッセージの翻訳

日本語メッセージの翻訳は Emacs や gedit のようなエディタで十分に行えますが，今回は **gtranslator** という専用のアプリケーションを使用します．gtranslator はターミナル上からコマンドラインで起動するか，メニューで検索して起動してください．

gtranslator を初めて起動した場合には，初期設定画面が表示されます．ここでは図 10.15 に示すように入力しました．

(a) 初期画面　(b) 翻訳者情報の入力

(c) 翻訳設定　(d) サマリー

図 10.15　gtranslator の初期設定画面

初期設定が終了したら，先ほど作成した ja.po を開いて翻訳を開始します．ja.po を開いた画面を図 10.16 に示します．画面左下の Original Message に翻訳前の文字列が表示されるので，その下の Translated Text」に翻訳メッセージを入力してください．

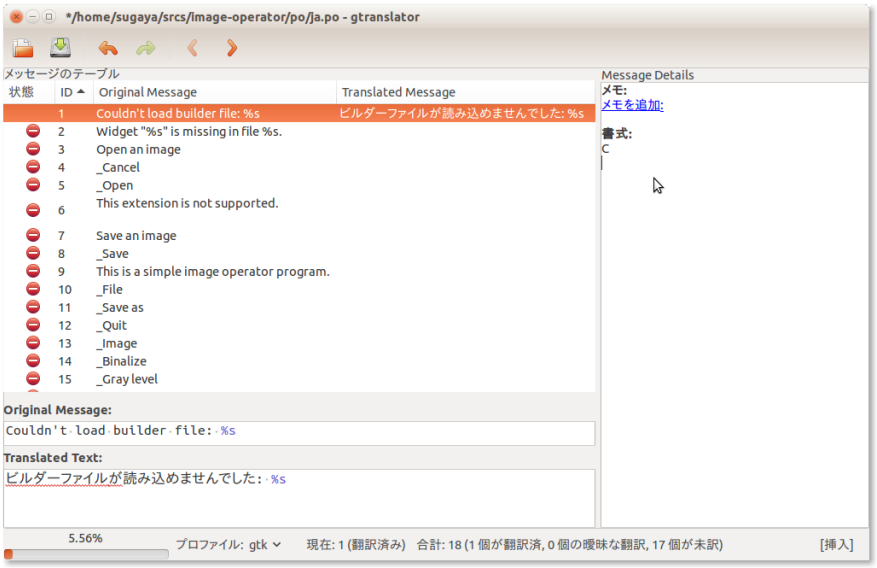


図 10.16　メッセージの翻訳画面

10.6.3　日本語メッセージの確認

翻訳したメッセージを確認するために，もう一度配布パッケージを作成して，インストールします．手順は 10.5 節（p. 297）を参照してください．インストールしたアプリケーションを実行した結果が図??です．メニューのメッセージが日本語に変わっていることが確認できると思います．

図 10.17　メッセージを日本語化したアプリケーション

10.7　発展

以上で簡単ですが，統合開発環境 anjuta を利用したソフトウェア開発の解説を終わります．今回の解説ではソースコードのデバッグなどについては触れていません．anjuta についてももっと知りたい方は，anjuta のホームページ http://anjuta.org/ を参照してください．

11 GTK+ に関連するその他のライブラリ

11.1 Pango

Pango は, html などで使用されるマークアップされた文字（例えば, ボールド体やイタリック, 下線表示など）の表示を実現したり, さらに汎用性の高い文字レイアウトを実現するライブラリです. 本節では, Pango による文字レンダリングの仕組みは解説せず, Pango を使用することで, どのような文字表示が実現できるかをサンプルプログラムを示しながら紹介します.

11.1.1 文字の装飾

まず, 文字をボールド体にしたり, イタリック体にしたりといった文字の装飾について例を示します. ここでは, ラベルウィジェットにマークアップされた文字を表示します. これは, pango ライブラリ関数をまったく意識することなく, GTK+ の関数を使用することで実現することができます.

ラベルウィジェットにマークアップされた文字を表示にするには, 関数 gtk_label_set_markup を使用します.

```
void gtk_label_set_markup (GtkLabel *label, const gchar *str);
```

第 1 引数　ラベルウィジェット
第 2 引数　マークアップした文字列

マークアップに使用できるタグの例を表 11.1 にまとめます.

表 11.1　マークアップタグの例

タグ	説明
	ボールド体
	イタリック体
<s></s>	打ち消し線
	下付き添字
	上付き添字
<u></u>	アンダーライン
	文字色の設定
	背景色の設定
	アンダーライン色の設定

関数 gtk_label_set_markup を使用したテキストの表示例を図 11.1 に示します. ソースコードはソース 11–1–1 に示します.

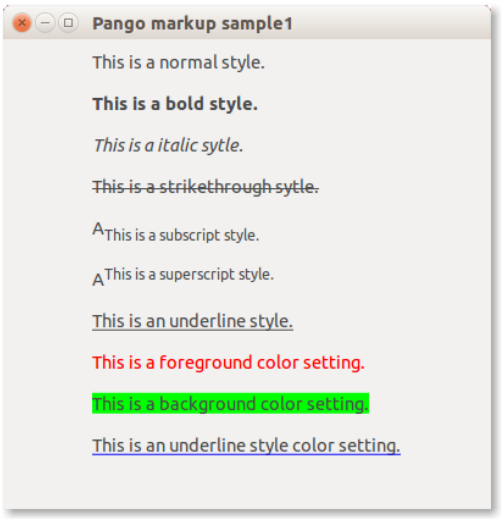


図 11.1　pango によるマークアップ文字の表示

ソース 11–1–1 文字の装飾の例: pango-markup-sample1.c

```
1  #include <gtk/gtk.h>
2
3  int
4  main (int argc, char *argv[])
5  {
6    GtkWidget *window;
7    GtkWidget *label;
8
9    gtk_init (&argc, &argv);
10
11   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
12   gtk_window_set_title (GTK_WINDOW (window), "Pango␣markup␣sample1");
13   gtk_widget_set_size_request (window, 400, -1);
14   gtk_container_set_border_width (GTK_CONTAINER (window), 10);
15
16   label = gtk_label_new ("");
17   gtk_label_set_markup (GTK_LABEL (label),
18                         "This␣is␣a␣normal␣style.\n\n"
19                         "<b>This␣is␣a␣bold␣style.</b>\n\n"
20                         "<i>This␣is␣a␣italic␣sytle.</i>\n\n"
21                         "<s>This␣is␣a␣strikethrough␣sytle.</s>\n\n"
22                         "A<sub>This␣is␣a␣subscript␣style.</sub>\n\n"
23                         "A<sup>This␣is␣a␣superscript␣style.</sup>\n\n"
24                         "<u>This␣is␣an␣underline␣style.</u>\n\n"
25                         "<span␣foreground='#ff0000'>"
26                         "This␣is␣a␣foreground␣color␣setting.</span>\n\n"
27                         "<span␣background='#00ff00'>"
28                         "This␣is␣a␣background␣color␣setting.</span>\n\n"
29                         "<span␣underline_color='#0000ff'>"
30                         "<u>This␣is␣an␣underline␣style␣color␣setting.</u></span>\n\n");
31
32   gtk_container_add (GTK_CONTAINER (window), label);
33
34   g_signal_connect (G_OBJECT (window), "destroy",
35                     G_CALLBACK (gtk_main_quit), NULL);
36
37   gtk_widget_show_all (window);
38   gtk_main ();
39
40   return 0;
41 }
```

11.1.2 多言語の表示

GTK+ では, pango の機能により unicode で文字を指定することで, 韓国語の文字などの特殊な多言語文字を表示することが可能です. 先ほど使用した関数 gtk_label_set_markup に与える文字列に表示したい文字を unicode で指定するだけでさまざまな国の文字を表示することができます.

「こんにちは」という文字をいくつかの国の文字で表示した例を図 11.2 に示します. ソースコードはソース 11–1–2 に示します.

図 11.2 pango によるさまざまな言語表示

ソース 11–1–2 pango によるさまざまな言語表示: pango-markup-sample2.c から一部を抜粋

```
1   label = gtk_label_new ("");
2   gtk_label_set_markup (GTK_LABEL (label),
3                         "英␣␣語 : ␣Hello\n\n"
4                         "日本語:␣こ ん に ち は\n\n"
5                         "韓国語:␣&#50504;&#45397;&#54616;&#49464;&#50836;\n\n"
6                         "タイ語:␣&#3626;&#3623;&#3633;&#3626;&#3604;&#3637;"
7                         "&#3588;&#3619;&#3633;&#3610;\n\n"
8                         "アラビア語:␣&#1575;&#1604;&#1587;&#1604;&#1575;&#1605;"
9                         "␣&#1593;&#1604;&#1610;&#1603;&#1605;\n\n"
10                        "ヒンドゥー語:␣&#2344;&#2350;&#2360;&#2381;&#2340;&#2375;\n");
```

11.1.3 pango 関数を用いた文字の表示

次に Pango の関数を利用してドローイングエリアに文字列を描画する例を説明します. Pango を使って文字を描画するには, **PangoLayout** 型の変数と **PangoFontDescription** 型の変数を通して, 描画に関するさまざまな設定を行います. PangoLayout は文字を描画するキャンバス, PangoFontDescription は描画する文字の設定を行います.

ここでは, cairo と連携して文字列を表示する流れを説明します. 大まかな流れを以下に示します.

1. 関数 pango_cairo_create_layout により PangoLayout 型の変数を作成する.
2. 関数 pango_layout_set_XXX 関数を使用して, 表示する文字に関する設定を行う.
3. 関数 pango_cairo_show_layout により, 設定した文字を表示する.

PangoLayout 型変数の作成

cairo と連携した PangoLayout 型の変数を作成には, 関数 pango_cairo_create_layout を使用します.

```
PangoLayout * pango_cairo_create_layout (cairo_t *cr);
```

第 1 引数	cairo_t 型の変数
戻り値	PangoLayout 型の変数

PangoLayout に関する設定

PangoLayout では文字を描画するキャンバスの設定を行います．PangoLayout で設定できる項目の例を表 11.2 に示します．PangoLayout で設定する数値は，1/PANGO_SCALE ピクセル単位で表現されています．従って，描画領域の大きさを 200 ピクセルで確保する場合には，PANGO_SCALE × 200 の値を関数の引数として与える必要があります．

表 11.2　PangoLayout の設定項目

項目	説明
width	描画領域の幅
height	描画領域の高さ
indent	インデント量
alignment	アライメント（左揃え: PANGO_ALIGH_LEFT, 中揃え: PANGO_ALIGN_CENTER, 右揃え: PANGO_ALIGN_RIGHT）
justify	両端揃えにするかどうか (TRUE of FALSE)
text	文字列
markup	マークアップされた文字列
font_description	フォント設定 (PangoFontDescription)

PangoFontDescription に関する設定

PangoFontDescription では，フォントの種類などを設定します．フォントサイズについても，先ほどを同様に，10pt のサイズで表示したい場合には，PANGO_SCALE × 10 の値を指定する必要があります．

表 11.3　PangoFontDescription の設定項目

項目	説明
family	フォントの種類
size	フォントサイズ
gravity	文字の回転方向（表 11.4 を参照）

設定した文字の表示

PangoLayout と PangoFontDescription の設定を行ったら，関数 pango_layout_set_font_description を使用して，PangoLayout に PangoFontDescription を設定します．

```
void pango_layout_set_font_description (PangoLayout                *layout,
                                        const PangoFontDescription *desc);
```

第 1 引数　PangoLayout 型の変数
第 2 引数　PangoFontDescription 型の変数

最後に，関数 pango_cairo_show_layout を用いて設定した文字を表示します．

```
void pango_cairo_show_layout (cairo_t *cr, PangoLayout *layout);
```

第 1 引数　cairo_t 型の変数
第 2 引数　PangoLayout 型の変数

ソース 11-1-3 に Pango 関数を用いた文字の表示プログラムのソースコードを示します．図 11.3 にプログラムの実行結果を示します．

ソース 11–1–3 Pango 関数を用いた文字の表示: pangolayout-sample1.c

```
1 #include <math.h>
2 #include <gtk/gtk.h>
```

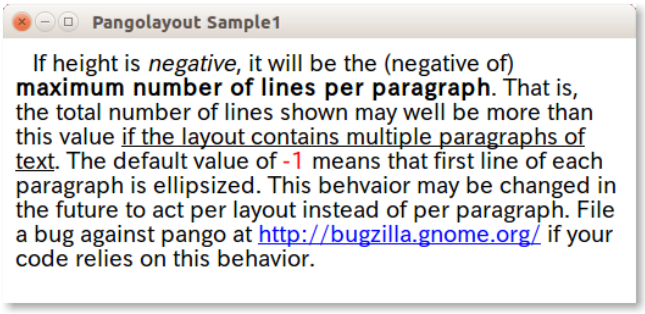


図 11.3 pango レイアウトのサンプル 1

```
3  #include <pango/pangocairo.h>
4
5  static gchar *text =
6    "If␣height␣is␣<i>negative</i>,␣it␣will␣be␣the␣(negative␣of)␣<b>maximum␣number"
7    "of␣lines␣per␣paragraph</b>.␣That␣is,␣the␣total␣number␣of␣lines␣shown␣may␣well"
8    "␣be␣more␣than␣this␣value␣<u>if␣the␣layout␣contains␣multiple␣paragraphs␣of␣text</u>.␣"
9    "The␣default␣value␣of␣<span␣fgcolor='#ff0000'>-1</span>␣means␣that␣first␣line"
10   "␣of␣each␣paragraph␣is␣ellipsized.␣This␣behvaior␣may␣be␣changed␣in␣the␣future"
11   "␣to␣act␣per␣layout␣instead␣of␣per␣paragraph.␣File␣a␣bug␣against␣pango␣at␣"
12   "<span␣underline_color='#0000ff'␣fgcolor='#0000ff'><u>http://bugzilla.gnome.org/</u>"
13   "</span>␣if␣your␣code␣relies␣on␣this␣behavior.";
14
15 gboolean
16 cb_draw (GtkWidget *widget,
17          cairo_t   *cr,
18          gpointer  user_data)
19 {
20   PangoLayout          *layout;
21   PangoFontDescription *desc;
22   const int            font_size = 14;
23   const int            margin = 10;
24   int                  width;
25   int                  height;
26   int                  n;
27
28   cairo_set_source_rgb (cr, 1.0, 1.0, 1.0);
29   cairo_paint (cr);
30
31   width  = gtk_widget_get_allocated_width (widget);
32   height = gtk_widget_get_allocated_height (widget);
33
34   layout = pango_cairo_create_layout (cr);
35
36   pango_layout_set_width (layout, PANGO_SCALE * (width - margin * 2));
37   pango_layout_set_height (layout, PANGO_SCALE * (height - margin * 2));
38   pango_layout_set_indent (layout, PANGO_SCALE * font_size);
39   pango_layout_set_alignment (layout, PANGO_ALIGN_LEFT);
40   pango_layout_set_justify (layout, TRUE);
41   pango_layout_set_markup (layout, text, -1);
42
43   desc = pango_font_description_from_string ("Sanf␣Serif");
44   pango_font_description_set_size (desc, PANGO_SCALE * font_size);
45
46   pango_layout_set_font_description (layout, desc);
47   pango_font_description_free (desc);
48
49   cairo_move_to (cr, margin, margin);
50   cairo_set_source_rgb (cr, 0.0, 0.0, 0.0);
51
52   pango_cairo_show_layout (cr, layout);
53
54   g_object_unref (layout);
55 }
56
57 int
58 main (int argc, char *argv[])
59 {
60   GtkWidget *window;
```

```
61   GtkWidget *canvas;
62
63   gtk_init (&argc, &argv);
64
65   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
66   gtk_window_set_title (GTK_WINDOW (window), "Pangolayout␣Sample1");
67   gtk_widget_set_size_request (window, 200, 100);
68   g_signal_connect (G_OBJECT (window), "destroy", G_CALLBACK (gtk_main_quit), NULL);
69
70   canvas = gtk_drawing_area_new ();
71   gtk_container_add (GTK_CONTAINER (window), canvas);
72   g_signal_connect (G_OBJECT (canvas), "draw", G_CALLBACK (cb_draw), NULL);
73
74   gtk_widget_show_all (window);
75   gtk_main ();
76
77   return 0;
78 }
```

11.1.4　縦書き表示

関数 pango_font_description_set_gravity による文字の回転と, cairo 関数を使った座標系変換を組み合わせることで, 日本語文字の縦書きも実現することができます（図 11.4）.

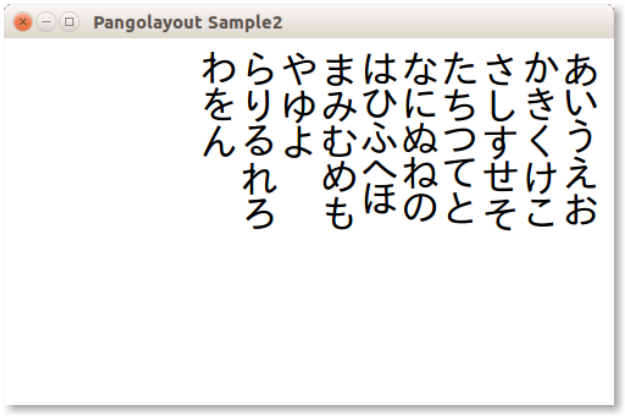


図 11.4　pango レイアウトのサンプル 2

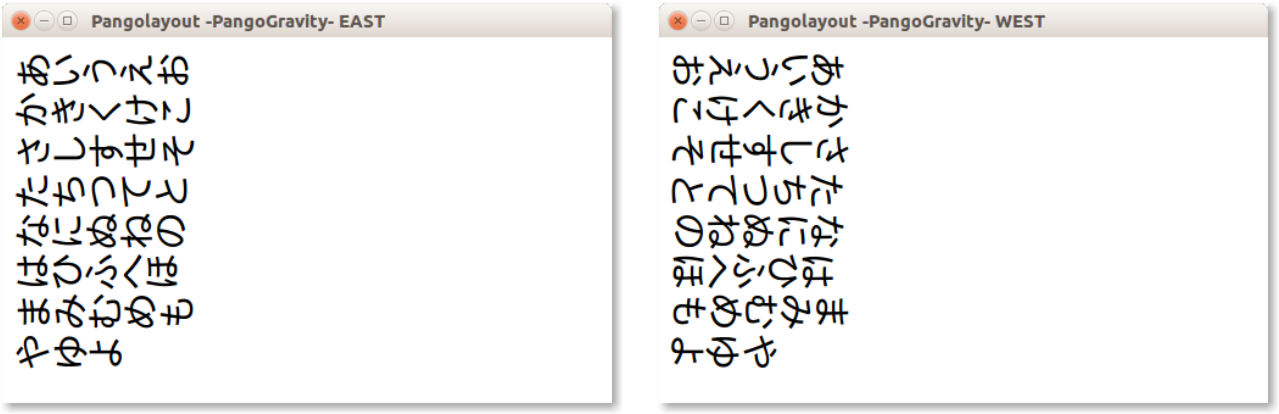
```
void pango_font_description_set_gravity (PangoFontDescription *desc,
                                         PangoGravity         gravity);
```

第 1 引数　PangoFontDescription 型の変数
第 2 引数　文字の回転方向（表 11.4 を参照）

関数 pango_font_description_set_gravity の第 2 引数に与える値を表 11.4 に示します.
また, PANGO_GRAVITY_EAST と PANGO_GRAVITY_WEST を指定して文字を回転した例を図 11.5 に示します. 単純に 1 文字ずつ回転ではなく, 行単位での回転になっているようです. さらに, 時計回りの回転 (PANGO_GRAVITY_EAST) と半時計回りの回転 (PANGO_GRAVITY_WEST) の説明と, 実際の表示との関係がよくわかりません.

表 11.4　PangoGravity の値

値	説明
PANGO_GRAVITY_SOUTH	回転なし
PANGO_GRAVITY_EAST	時計回りに 90 度回転
PANGO_GRAVITY_NORTH	上下反転
PANGO_GRAVITY_WEST	反時計回りに 90 度回転
PANGO_GRAVITY_AUTO	自動判定



(a) PANGO_GRAVITY_EAST での回転　　(b) PANGO_GRAVITY_WEAT での回転

図 11.5　PangoGravity による文字の回転の例

図 11.5(a) を見ると, これに対して原点の平行移動と軸の回転を施すことによって, 図 11.4 のような縦書き表示が実現できそうです. そこで, PANGO_GRAVITY_EAST に設定した状態で, 関数 cairo_translate によって, 原点を右端まで移動させて, 関数 cairo_rotate によって, 座標軸を 90 度回転する処理を行ったのちに, 文字の表示を行った結果が図 11.4 です. ソースコードをソース 11–1–4 に示します.

ソース 11–1–4 縦書き表示: pangolayout–sample2.c から一部を抜粋

```
1  static gchar *text =
2    "あいうえお\nかきくけこ\nさしすせそ\nたちつてと\nなにぬねの\n"
3    "はひふへほ\nまみむめも\nやゆよ\nらりるれろ\nわをん\n";
4
5  gboolean
6  cb_draw (GtkWidget *widget,
7           cairo_t   *cr,
8           gpointer  user_data)
9  {
10   PangoLayout          *layout;
11   PangoFontDescription *desc;
12   const int            font_size = 24;
13   const int            margin = 10;
14   int                  width;
15   int                  height;
16   int                  n;
17
18   cairo_set_source_rgb (cr, 1.0, 1.0, 1.0);
19   cairo_paint (cr);
20
21   width  = gtk_widget_get_allocated_width (widget);
22   height = gtk_widget_get_allocated_height (widget);
23
24   layout = pango_cairo_create_layout (cr);
25
26   pango_layout_set_width (layout, PANGO_SCALE * (height - margin * 2));
27   pango_layout_set_height (layout, PANGO_SCALE * (width - margin * 2));
28   pango_layout_set_text (layout, text, -1);
29
30   desc = pango_font_description_from_string ("Sanf␣Serif");
31   pango_font_description_set_size (desc, PANGO_SCALE * font_size);
32   pango_font_description_set_gravity (desc, PANGO_GRAVITY_EAST);
33
34   pango_layout_set_font_description (layout, desc);
35   pango_font_description_free (desc);
36
37   cairo_translate (cr, width, 0);
38   cairo_rotate (cr, 0.5 * G_PI);
39
40   cairo_move_to (cr, margin, margin);
41   cairo_set_source_rgb (cr, 0.0, 0.0, 0.0);
```

```
42
43   pango_cairo_show_layout (cr, layout);
44
45   g_object_unref (layout);
46 }
```

11.1.5　複雑な文字のレイアウト

Pango のレイアウト機能と cairo の描画機能を組み合わせるとより複雑な文字のレイアウトが実現できます．基本的には，先ほどの縦書きと同様に，cairo 関数による座標系の変換機能を使用します．その例が図 11.6 になります．一文字ずつ座標系を変換しながら文字を表示することで，例のような幾何学的な文字のレイアウトが実現できます．ソースコードをソース 11–1–5 に示します．

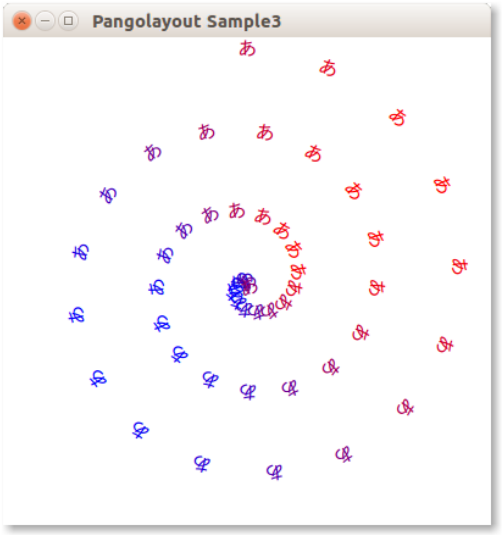


図 11.6　pango レイアウトのサンプル 3

ソース 11–1–5 pango による複雑なレイアウトの例: pangolayout-sample3.c から一部を抜粋

```
1  gboolean
2  cb_draw (GtkWidget *widget,
3           cairo_t   *cr,
4           gpointer  user_data)
5  {
6    const double         radius = 200.0;
7    const int            n_words = 50;
8    PangoLayout          *layout;
9    PangoFontDescription *desc;
10   int                  n;
11
12   cairo_set_source_rgb (cr, 1.0, 1.0, 1.0);
13   cairo_paint (cr);
14
15   cairo_translate (cr, radius, radius);
16
17   layout = pango_cairo_create_layout (cr);
18   pango_layout_set_text (layout, "あ", -1);
19
20   desc = pango_font_description_from_string ("Sanf␣Serif");
21   pango_font_description_set_size (desc, 12 * PANGO_SCALE);
22
23   pango_layout_set_font_description (layout, desc);
24   pango_font_description_free (desc);
25
26   for (n = 0; n < n_words; n++)
27     {
28       int     width;
29       int     height;
```

```
30        double angle;
31        double red;
32
33        angle = (360 * 3 * n / n_words) % 360;
34        cairo_save (cr);
35        {
36          red = (1.0 + cos ((angle - 60.0) * G_PI / 180.0)) / 2.0;
37          cairo_set_source_rgb (cr, red, 0, 1.0 - red);
38          cairo_rotate (cr, angle * G_PI / 180.0);
39
40          pango_cairo_update_layout (cr, layout);
41          pango_layout_get_size (layout, &width, &height);
42
43          cairo_move_to (cr,
44                         -(((double) width / PANGO_SCALE) / 2.0),
45                         (double) n * radius / n_words - radius);
46          pango_cairo_show_layout (cr, layout);
47        }
48        cairo_restore (cr);
49      }
50    g_object_unref (layout);
51  }
```

11.2 gio

gio は, glib の中で実装されたデータの入出力などを扱うライブラリです. データの入出力以外にも, 2 節で扱った GApplication なども gio の中で定義されています. 本章では, GApplication でのコマンドライン引数の扱いと Gzip で圧縮されたテキストデータの読み込みについて紹介します.

11.2.1 GApplication のコマンドライン引数

ソース 11–2–1 は glib のソースコードに含まれている gio のテストコードの一つです. 関数 g_application_new で作成した GApplication 変数に対して, 関数 g_signal_connect を用いて, **command-line シグナル**に対するコールバック関数を設定しています.

コールバック関数内では, g_application_command_line_get_arguments を使用してコマンドライン引数の数 (argc) とコマンドライン引数の文字列 (argv) を取得しています. この例では, コマンドライン引数を順番に出力しているだけですが, 8.4 節で紹介したオプション解析などと組み合わせることで, コマンドライン引数による, 柔軟なプログラムの動作が実現できます.

```
gchar **
g_application_command_line_get_arguments (GApplicationCommandLine *cmdline,
                                          int *argc);
```

第 1 引数	GApplicationCommandLine 型の変数
第 2 引数	コマンドライン引数の数を取得するための gint 型の変数のポインタ
戻り値	コマンドライン引数の文字列

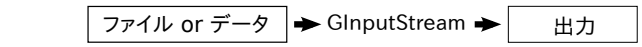
ソース 11–2–1 GApplication のコマンドライン引数: gapplication-command-line-sample.c

```
1  #include <gio/gio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  static int
6  command_line (GApplication            *application,
7                GApplicationCommandLine *cmdline)
8  {
9    gchar **argv;
10   gint argc;
11   gint i;
12
13   argv = g_application_command_line_get_arguments (cmdline, &argc);
14
15   g_application_command_line_print (cmdline, "This␣is␣a␣command␣line␣sample.\n");
16
```

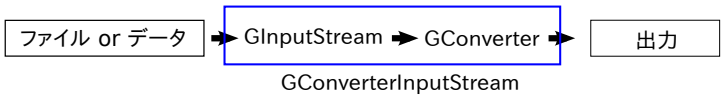
```
17   for (i = 0; i < argc; i++) g_print ("argument␣%d:␣%s\n", i, argv[i]);
18
19   g_strfreev (argv);
20
21   return 0;
22 }
23
24 int
25 main (int argc, char *argv[])
26 {
27   GApplication *app;
28   int status;
29
30   app = g_application_new ("test.gtk.application_command_list",
31                            G_APPLICATION_HANDLES_COMMAND_LINE);
32   g_signal_connect (app, "command-line", G_CALLBACK (command_line), NULL);
33
34   status = g_application_run (app, argc, argv);
35
36   g_object_unref (app);
37
38   return status;
39 }
```

11.2.2　圧縮ファイルの読み込み

gio の関数には, gzip により圧縮されたファイルを展開して読み込んだり, 反対にファイルに圧縮して書き出したりできるものがあります. これは, 図 11.7(a) のような gio で用意されている GInputStream や GOutputStream などの入出力ストリームとファイルもしくはデータ間の処理の流れに対して, 図 11.7(b) のように, **GConverter** という, データ変換インターフェースを挟むことにより, 入出力データのさまざまな変換を実現する機能によるものです.



(a) 通常の入力処理の流れ



(b) GConveter によるデータ変換を行う入力処理の流れ

図 11.7　GConverter による入力データの変換

ここでは, **GZlibDecompressor** を用いて, gzip で圧縮した C 言語のソースコードを読み込んで標準出力に出力する例を紹介します. 圧縮ファイルを展開して読み込むための手順は以下のようになります.

1. 関数 g_file_read によるファイルのオープン
2. 関数 g_zlib_decompressor_new による圧縮データ展開インターフェースの作成
3. 関数 g_converter_input_stream_new による入力データとコンバーターの関連付け

上記の手順に対応する実際のソースコードがソース 11–2–2 の 17 行目から 22 行目になります. この後は, 通常のファイルからのデータ読み込みと同様に処理を行うだけです. ソース 11–2–2 では, 圧縮ファイルからデータをすべて読み込み, 先頭から 50 文字を標準出力に出力しています.

これは, ソース 11–2–2 のテキストファイルを gzip で圧縮したファイルをプログラムの引数として与えた実行結果になります.

```
$ ./read-compressed-file read-compressed-file.c.gz ↵
$ #include <glib.h>
$ #include <gio/gio.h>
$ #include <s
$
```

ソース 11–2–2 GZlibDecompressor による圧縮ファイルの読み込み: read_compressed_data.c

```
#include <glib.h>
#include <gio/gio.h>
#include <string.h>

int
main (int argc, char *argv[])
{
  GFile*             file;
  GZlibDecompressor* converter;
  GFileInputStream*  stream;
  GInputStream*      uncompress;
  guchar             *buffer = NULL;
  guchar             buf[1024];
  gssize             nreads = -1;
  gssize             total = 0;

  file = g_file_new_for_path (argv[1]);
  stream = g_file_read (file, NULL, NULL);

  converter = g_zlib_decompressor_new (G_ZLIB_COMPRESSOR_FORMAT_GZIP);
  uncompress = g_converter_input_stream_new ((GInputStream *) stream,
                                             (GConverter *) converter);

  while (nreads != 0) {
    nreads = g_input_stream_read (uncompress, &buf, 1023, NULL, NULL);
    buffer = g_renew (guchar, buffer, total + nreads);
    memcpy (buffer + total, buf, nreads);
    total += nreads;
  }
  for (int n = 0; n < 50; n++) g_print ("%c", buffer[n]);
  g_print ("\n");

  g_object_unref (file);
  g_object_unref (stream);
  g_object_unref (uncompress);
  g_free (buffer);

  return 0;
}
```

A Visual Studio でのビルド方法

ここでは, 第 2 章（p. 7）のチュートリアルプログラムを例に, Visual Studio 用いたプログラム作成について解説します. 著者の環境では OS が Windows 10, 開発環境は Visual Studio Professional 2013 を使用していますが, OS や Visual Studio のバージョンが異なっても設定方法などに大きな違いはないと思われます.

A.1 Windows への GTK+ 開発環境のインストール

まずは, Windows 用にコンパイルされたライブラリをインストールします. Visual Studio 用のコンパイル済みライブラリは以前は, GTK+ のホームページのダウンロードページで公開されていたのですが, 現在は公開されなくなっています. 平成 28 年 6 月 29 日現在で最新と思われる Visual Studio で使用できるバイナリファイルは `http://win32builder.gnome.org/gtk+-bundle_3.10.4-20131202_win32.zip` です. これは 32 ビット版ですが, 64 ビット版も存在します.

まずはこのファイルをダウンロードして展開します. 展開すると, gtk+-bundle_3.10.4-20131202_win32 というフォルダが作成されます（著者の環境ではデスクトップ上にフォルダが作成されました）. このフォルダの名前を gtk3 に変更して, C ドライブの直下に移動もしくはコピーしてください.

展開したフォルダを置く場所は C ドライブの直下でなくても問題ありません. Visual studio を使用する際の設定時にフォルダを置いた場所を指定するように変更してください.

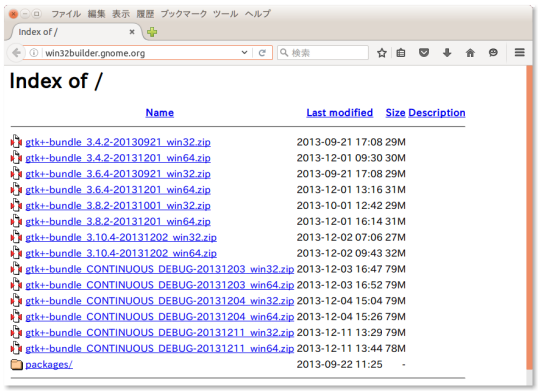


図 A.1　Windows 用のコンパイル済みライブラリのダウンロードサイト

A.2　環境変数の設定

次に Windows の環境変数 Path に C:\gtk3\bin を追加します．この設定をしていないと作成したプログラムを実行する際にエラーとなります．環境変数の設定手順を以下に示します．

1. まず，コントロールパネルのシステムパネルを開きます．そして，パネルの左に表示されたシステムの環境設定をクリックします（図 A.2）．

図 A.2　環境変数の設定: コントロールパネルのシステム

2. 次に，システムのプロパティパネルの下にある環境変数ボタンをクリックします（図 A.3(a)）．
3. 環境変数パネルが表示されますので，下の段のシステム環境変数から Path を選択して，編集ボタンを押してください（図 A.3(b)）．
4. 環境変数名の編集パネル画面で新規ボタンを押して，GTK+ 用のパスを入力できるようにします（図 A.3(c)）．
5. 最後に入力欄に C:\gtk3\bin（ダウンロードしたファイルを展開したフォルダを置いた場所に bin を追加したもの）を入力して OK ボタンを押してください（図 A.3(d)）．

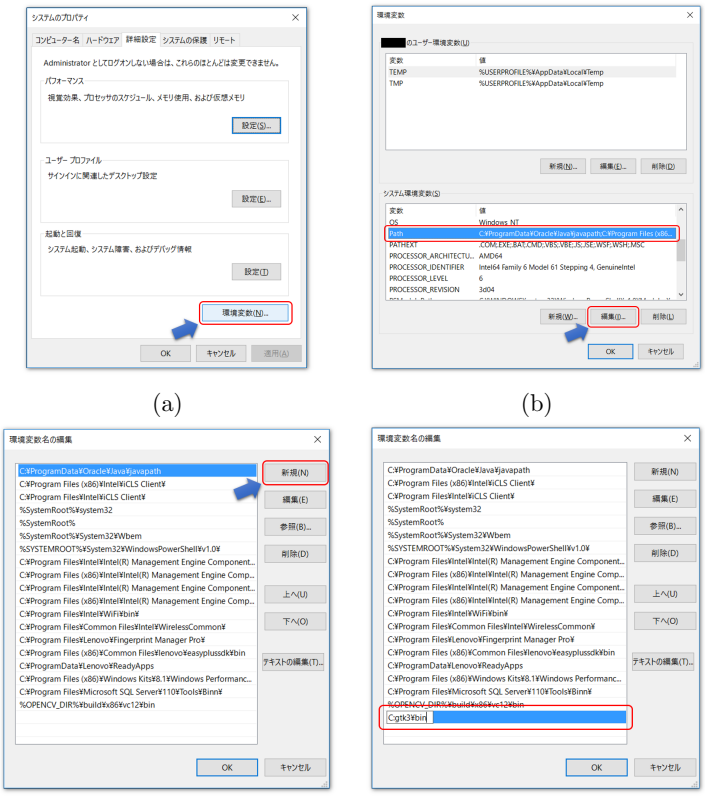


図 A.3　環境変数の設定

A.3　Visual Studio によるプログラムのビルド

ここからは, 第 2 章（p. 7）のチュートリアルプログラムを Visual Studio でコンパイルして実行するまでの流れについて説明します.

A.3.1　プロジェクトの作成

まずは, Visual Studio を起動して新規プロジェクトを作成します. そして, 「ファイル」メニューから「新規作成」–「プロジェクト」を選択します（図 A.4）.

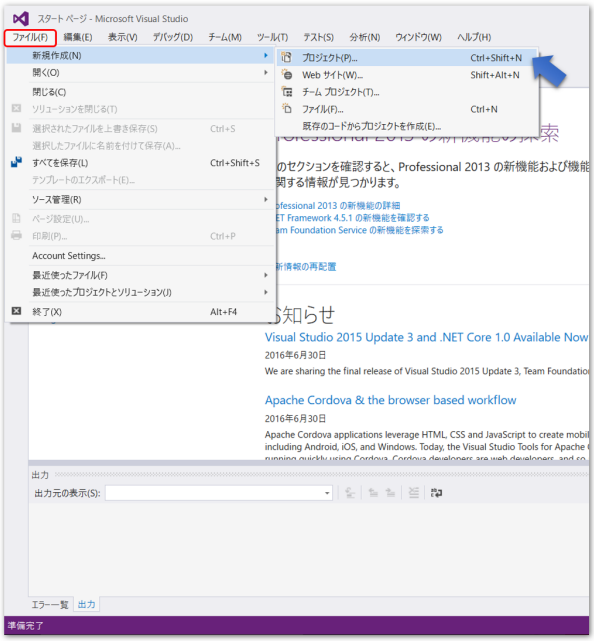


図 A.4　新規プロジェクトの作成

新しいプロジェクトウィンドウが表示されますので, 左のパネルで Visual C++ を選択, 中央のパネルで空のプロジェクトを選択して, 下の名前の入力欄に imageviewer と入力して OK ボタンをクリックします（図 A.5）.

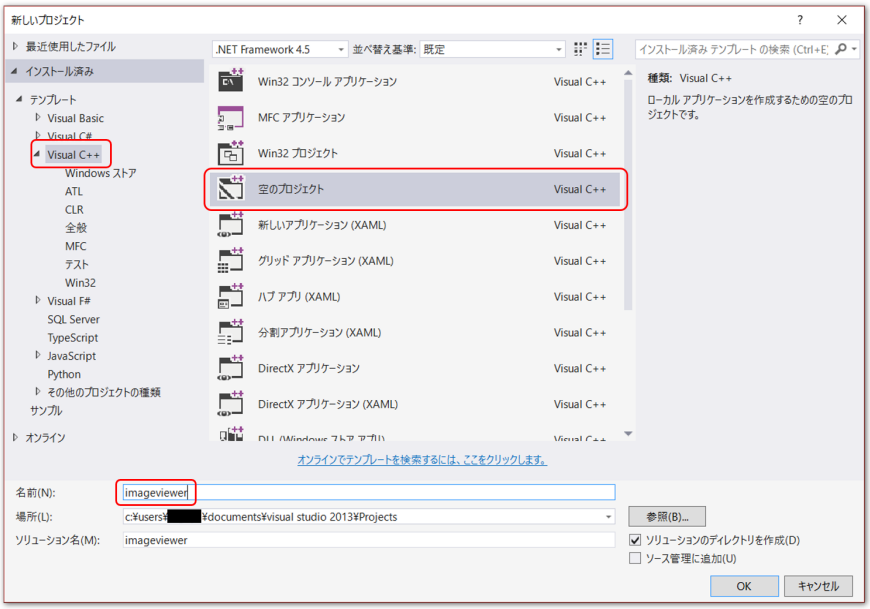


図 A.5　プロジェクト名の入力

A.3.2　GTK+ ライブラリ用のプロパティシートの作成

新しいプロジェクトが開けたら，ソースコードを入力する前に GTK+ ライブラリを使用するための設定を行います．「表示」メニューから「その他のウィンドウ」リック ー「プロパティマネージャー」を選択します．ウィンドウの右側のパネルにプロパティマネージャーが表示されたことを確認してください（図 A.6）．

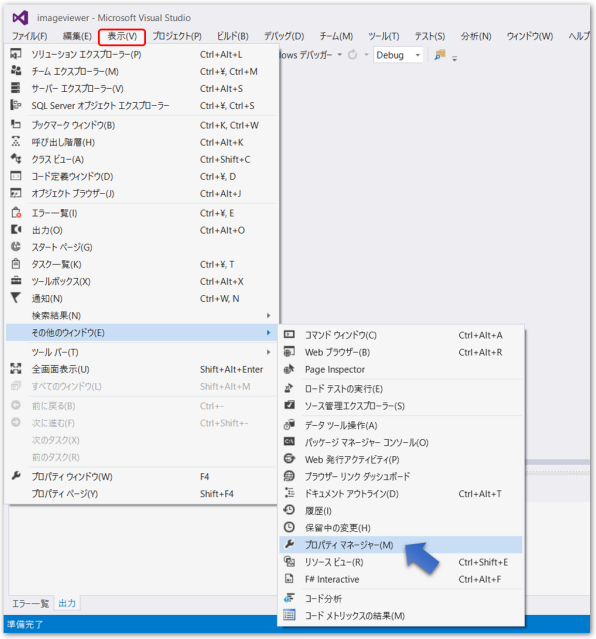


図 A.6　プロパティマネージャーの表示

ウインドウの右側パネルに表示されたプロパティマネージャーで imageviewer と表示されているラベルにマウスカーソルを合わあせて，マウスの右ボタンをクリックして，ポップアップメニューを表示させます．そして，一番上の「新しいプロジェクトプロパティシートの追加」を選択します（図 A.7）．

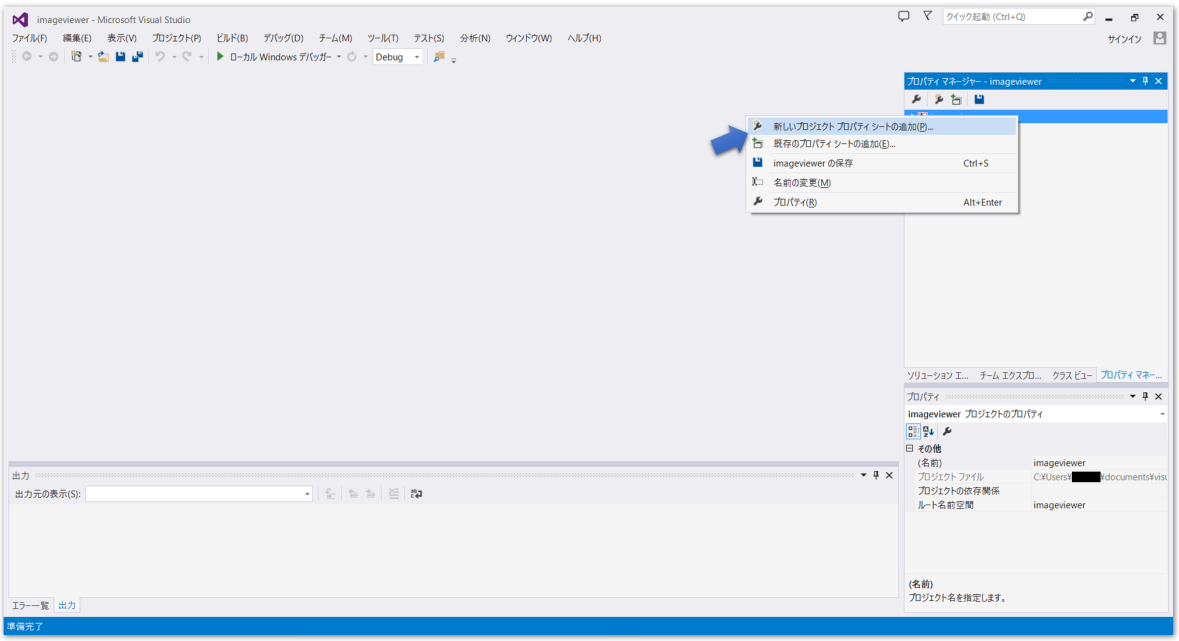


図 A.7　プロパティシートの追加

新しい項目の追加というダイアログが表示されますので，名前の欄にファイル名を入力します．ここでは，GTK+-3 用のライブラリ設定のためのプロパティシートという意味で gtk3.props という名前にすることにします（図 A.8）．

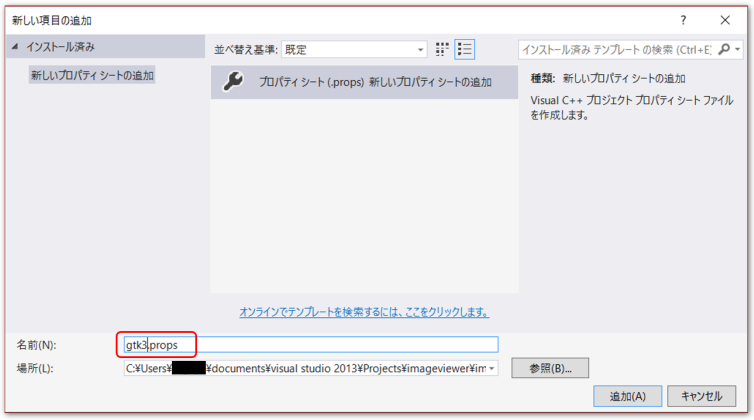


図 A.8　プロパティシートの追加

プロパティマネージャーの imageviewer のラベルの左側の三角印をクリックして，ツリー表示を展開して図 A.9 のように Debug|Win32 のラベルの下に gtk3 という名前のプロパティシートが追加されたことを確認してください．gtk3 というラベルが確認できたら，そのラベルをダブルクリックして，gtk3 のプロパティページのダイアログを表示します．設定する必要がある項目は以下の三つです．

- GTK+ ライブラリのインクルードファイルが置かれた場所
- GTK+ ライブラリのライブラリファイルが置かれた場所
- プログラムが使用するライブラリ

まずは，GTK+ ライブラリのインクルードファイルが置かれた場所の設定を行います．gtk3 のプロパティページのダイアログの左パネルの C/C++ を選択すると，中央パネルに図 A.9 のような画面が表示されますので，追加のインクルードディレの欄に，以下のディレクトリを入力します．

- c:\gtk3\include\gtk-3.0
- c:\gtk3\include\glib-2.0
- c:\gtk3\lib\glib-2.0\include
- c:\gtk3\include\pango-1.0
- c:\gtk3\include\cairo
- c:\gtk3\include\gdk-pixbuf-2.0
- c:\gtk3\include\atk-1.0

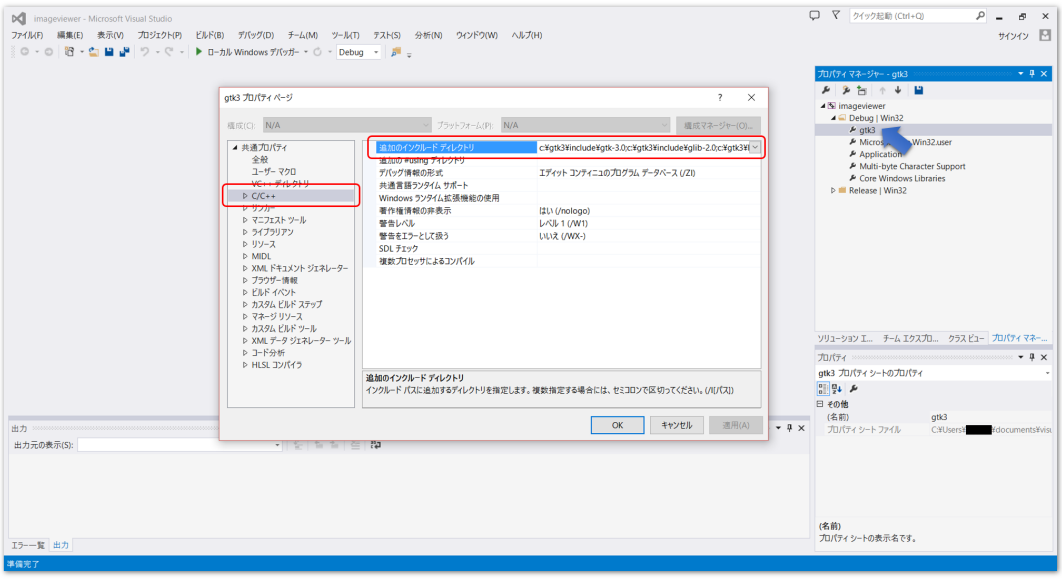


図 A.9　追加のインクルードディレクトリの設定

次に, 使用する GTK+ ライブラリが置かれた場所と実際に使用するライブラリの指定を行います. 使用する GTK+ ライブラリが置かれた場所の指定は, gtk3 プロパティページダイアログの左パネルで, 「リンカ」–「全般」を選択して, 中央のパネルの追加のライブラリディレクトリ欄に c:\gtk3\lib を入力します（図 A.10(a)）.

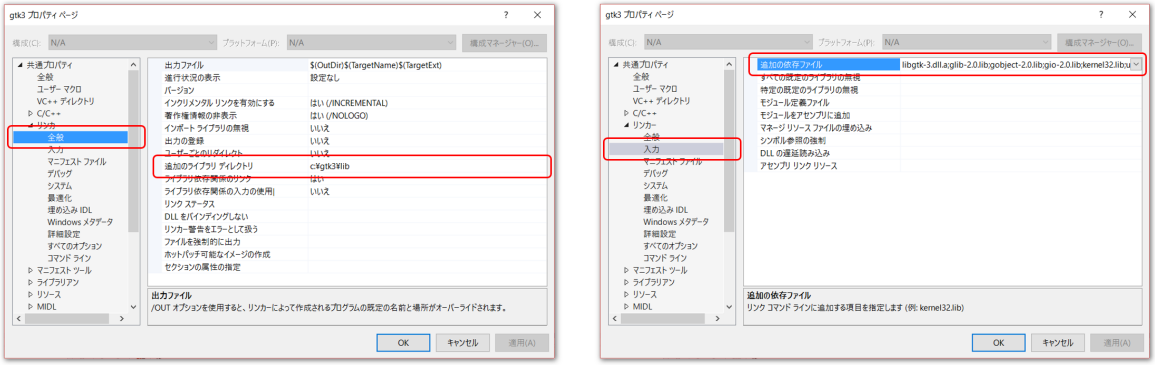
最後に, 「リンカ」–「入力」を選択して, 中央のパネルの追加の依存ファイル欄に以下のライブラリを入力します（図 A.10(b)）.

- libgtk-3.dll.a
- glib-2.0.lib
- gobject-2.0.lib
- gio-2.0.lib

ここで, ダウンロードしたライブラリ（gtk+-bundle_3.10.4-20131202_win32.zip）にはなぜか gtk の共有ライブラリ（動的リンクライブラリ）が含まれていませんので, gtk のライブラリのみ, 静的ライブラリ（libgtk-3.dll.a）を指定しています.

プログラム内で使用する関数によっては, gdk_pixbuf-2.0.lib や cairo.lib などのライブラリを必要に応じて入力するようにしてください.

今回作成したプロパティシート gtk3.props をコピーして別のフォルダーに保存しておけば, 別のプロジェクトを作成したときに, 既存のプロパティシートの追加でこのファイルをプロジェクトに追加すればよく, 上記の設定をプロジェクトごとに行う必要はありません.



(a)　　(b)

図 A.10　使用するライブラリの設定

A.3.3　プログラムのビルド

ライブラリの設定さえしてしまえば, 後はソースコードを入力してプログラムをビルドするだけです. Visual Studio でプログラムを作成したことのある方なら説明の必要はないと思いますが, 新規にソースファイルを追加して, プログラムをビルド, 実行する流れを説明します.

まず, ソリューションエクスプローラーが表示されていない場合には, 「表示」メニューから「ソリューションエクスプローラー」を選択します（図 A.11）.

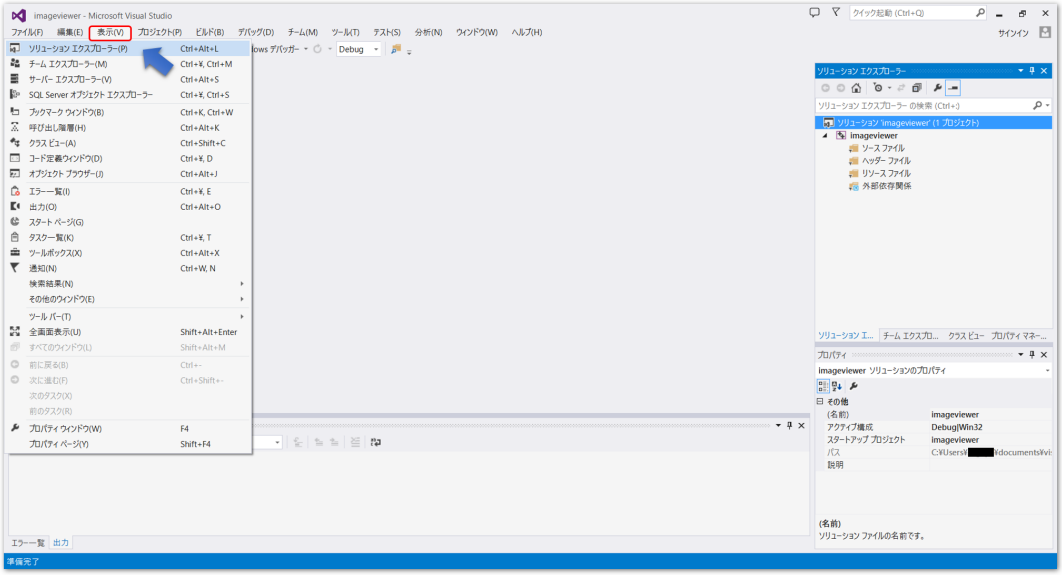


図 A.11　ソリューションエクスプローラーの表示

ウィンドウの右側のパネルにソリューションエクスプローラーが表示されたら, ソースファイルのラベルにマウスカーソルを合わせて, マウスの右ボタンをクリックして, ポップアップメニューを表示させ, 「追加」–「新しい項目」を選択します（図 A.12）.

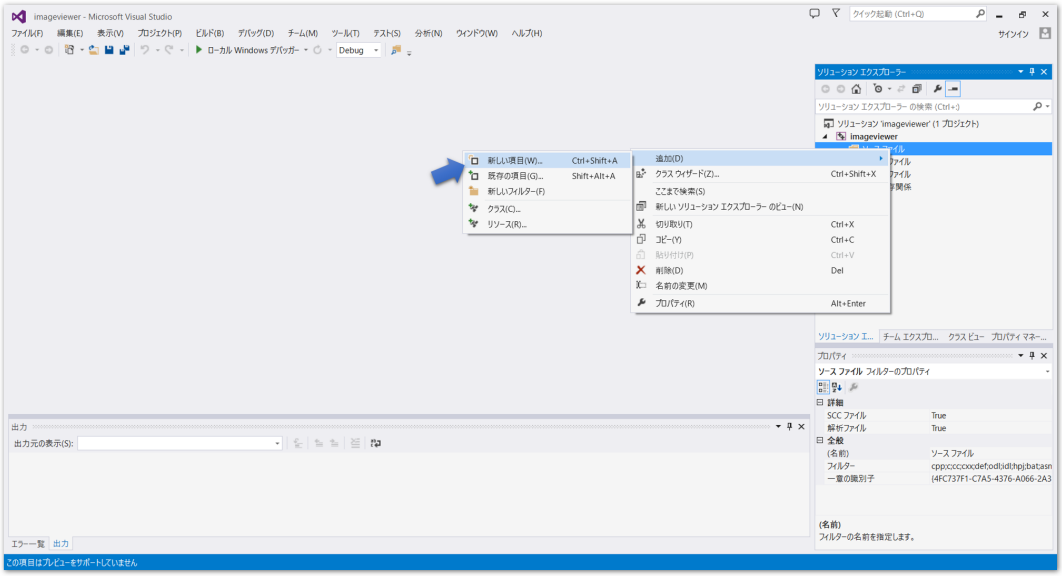


図 A.12　新しいソースファイルの追加

図 A.13 のように新しい項目の追加ダイアログが表示されますので, C++ ファイルを選択してファイル名を imageviewr.cpp と入力して追加ボタンをクリックします.

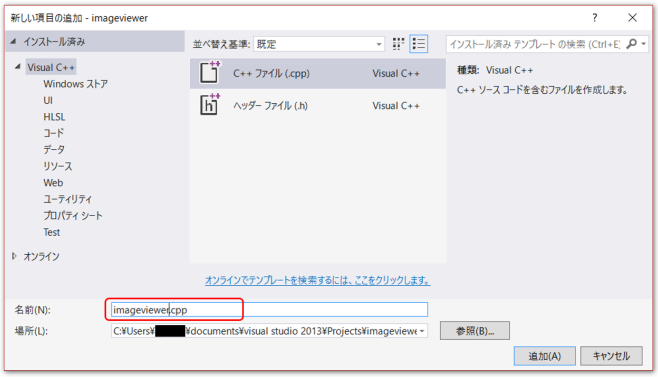
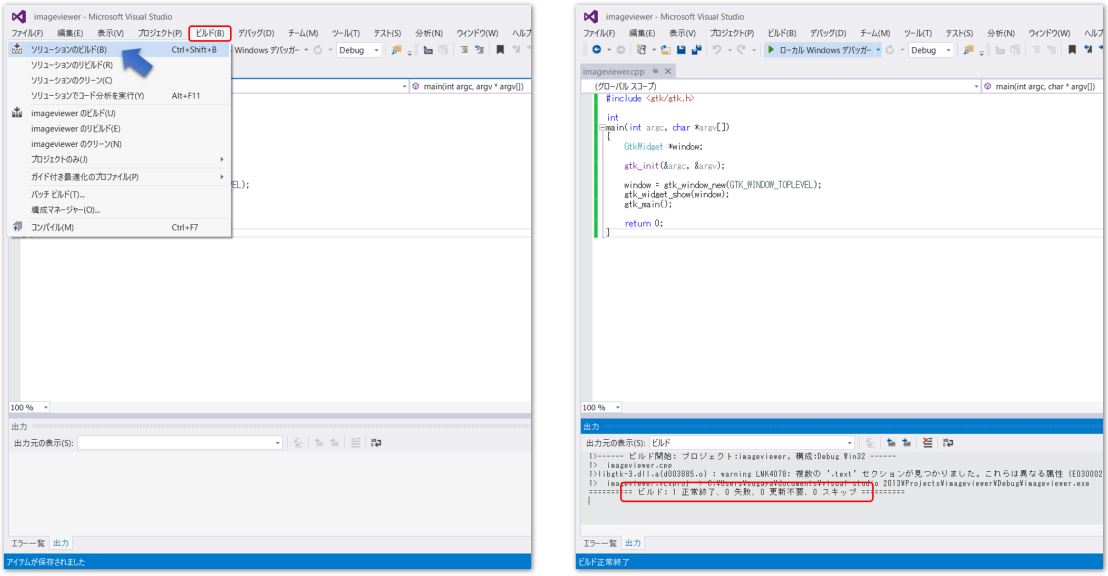


図 A.13　新しいソースファイルの追加

新規ファイルを追加したらソース 2–1（p. 8）を入力して，ソリューションをビルドします．ビルドするには，「ビルド」メニューから「ソリューションのビルド」を選択します（図 A.14(a)）．ライブラリの設定やソースコードの入力に間違いがなければ, 図 A.14(b) のようにビルドが終了します.



(a)　(b)

図 A.14　プログラムのビルド

ビルドが成功したら，「デバッグ」メニューから「デバッグ開始」もしくは「デバッグなしで開始」を選択してビルドしたプログラムを実行してみます．図 A.15 のようなウィンドウが表示されれば成功です.

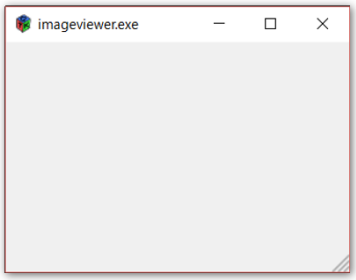


図 A.15　プログラムの実行結果

B アイコンブラウザ

GTK+ では, IconTheme というアイコンのセットがあり, IconTheme を変更することで表示するアイコンを変更することができるようになっています. GTK+ ではアプリケーションからこれらのアイコンを使用するために, アイコンに固有の名前が付けられています.

このアイコン画像と名前を確認できるのが, gtk3-icon-browser というアプリケーションです. 図 B.1 は, gtk3-icon-browser の実行画面です（ただし，アイコンは IconTheme によって異なります）. このアイコンを自分で作成するアプリケーションに使いたいんだけど名前は？ という場合には, gtk3-icon-browser で確認するとよいでしょう.

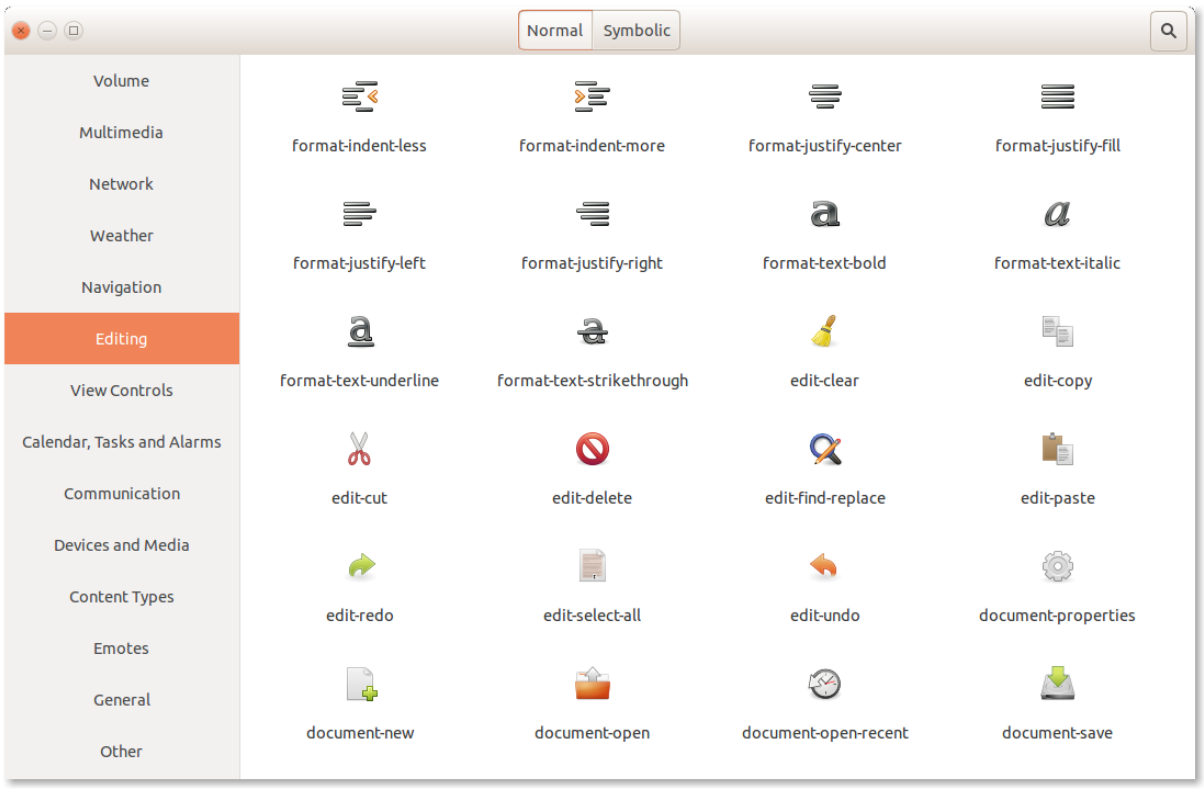


図 B.1　gtk3-icon-browser の画面

C サンプルプログラムのソースコードリスト

本書で扱ったプログラムのソースコードは，次の URL からダウンロードできます．
http://iim.cs.tut.ac.jp/member/sugaya/GTK+/
　以下はそのソースコードの構成です．なお，本文中に掲載したソースコードは，公開している実際のソースコードから説明とは関連のないエラー処理などを省略してある場合があるのでご了承ください．

```
sources/
  +--- cairo/
         +--- arc/
         +--- clipping/
         +--- composite/
         +--- curve/
         +--- line_cap/
         +--- line_dash/
         +--- line_join/
         +--- mask/
         +--- pattern_image/
         +--- pattern_linear/
         +--- pattern_radial/
         +--- polygon/
         +--- rectangle/
         +--- source_image/
         +--- source_image_surface/
         +--- text
         +--- transform/
         +--- transparency/
  +--- custom_widget
         +--- gtkiconbutton/
  +--- gdkpixbuf
         +--- display_by_gtkdrawingarea/
         +--- display_by_gtkimage/
         +--- loader/
         +--- read/
  +--- glib
         +--- file/
         +--- ghashtable/
         +--- glist/
                +--- operation/
                +--- sort/
         +--- timer/
  +--- gtk
         +--- grid/
                +--- sample1/
                +--- sample1/
         +--- hello_world/
         +--- packing/
         +--- signal/
                +--- sample1/
```

```
                +--- sample2/
                +--- sample3/
                +--- sample4/
                +--- sample5/
 +--- imageoperator
 +--- tips
        +--- customstockitem/
        +--- dnd/
        +--- keyboard/
        +--- mouse/
        +--- option/
 +--- tutorial
        +--- step1/
        +--- step2/
        +--- step3/
        +--- step4/
        +--- step5/
        +--- step6/
+--- widget
        +--- gtkaboutdialog/
        +--- gtkbutton/
        +--- gtkcheckbutton/
        +--- gtkcomboboxentry/
        +--- gtkdialog/
        +--- gtkentry/
        +--- gtkentrycompletion/
        +--- gtkexpander/
        +--- gtkfilechooser/
        +--- gtkfileselection/
        +--- gtkframe/
        +--- gtkhandlebox/
        +--- gtkiconview/
        +--- gtkliststore/
        +--- gtkmenubar/
        +--- gtkmessagedialog/
        +--- gtknotebook/
        +--- gtkpaned/
        +--- gtkpopupmenu/
        +--- gtkprogressbar/
        +--- gtkradiobutton/
        +--- gtkscale/
        +--- gtkspinbutton/
        +--- gtktextview/
        +--- gtktoolbar/
        +--- gtktooltips/
        +--- gtktreestore/
        +--- gtkuimanager/
```

索引