入門GTK+

菅谷 保之 著

2012年10月10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はありません．そこで，本書では GTK+ を使った簡単なプログラムから本格的なプログラムまで，具体的な例を挙げながらわかりやすく解説します．本書が読者の GTK+ による GUI アプリケーション作成の学習に役立てば幸いです．

2009 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．

- **第 1 章 GTK+ ってなんだろう**
  GTK+ に関する簡単な紹介と，Ubuntu 9.04 日本語 Remix 版での GTK+ の開発環境の設定方法を説明します．
- **第 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．
- **第 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．
- **第 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．
- **第 5 章 GdkPixbuf を使った画像アプリケーションの作成**
  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，簡単な画像処理アプリケーションを作成しながら，GdkPixbuf について解説します．
- **第 6 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．
- **第 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．
- **第 8 章 拡張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．もちろん，既存のウィジェットを組み合わせて便利な発展ウィジェットを作ることもできます．この章では拡張ウィジェットの作成方法を具体例を挙げて解説します．
- **第 9 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．
- **第 10 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．
- **付録 A GTK+ の開発環境の構築**
  GTK+ での開発に必要な環境の構築方法を説明します．
- **付録 B Visual Studio 2008 Express Edition でのビルド方法**
  Windows 上で Visual Studio 2008 Express Edition 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 C GtkStockItem 一覧**
  GTK+ であらかじめ定義されている，GtkStockItem と呼ばれるアイコンの一覧です．

- **付録 D 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 9.04 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（巻末の付録 A（p. 277）を参照）．

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 aptitude install libgtk2.0-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
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);
```

サポート

本書で解説を行っているソースコードや正誤表などは，次の Web ページで公開しています．

http://www.iim.cs.tut.ac.jp/~sugaya/books/GUI-ApplicationProgramming/

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

第 1 章

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．

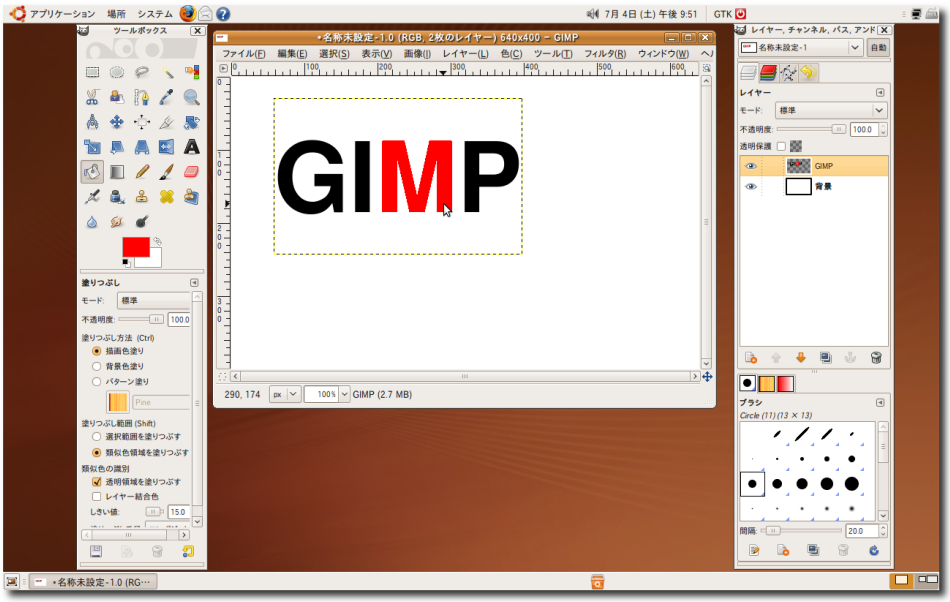


図 1.1　GIMP の実行画面

[*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/

- 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.
- 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

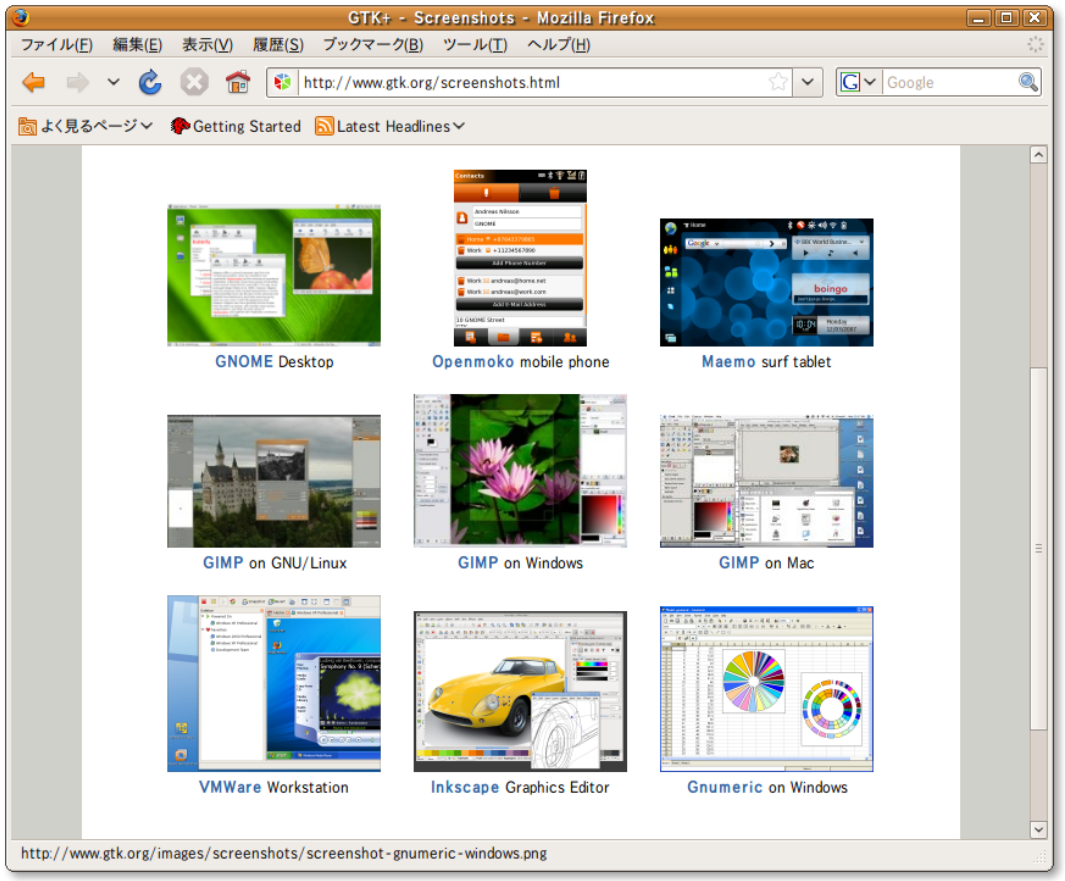
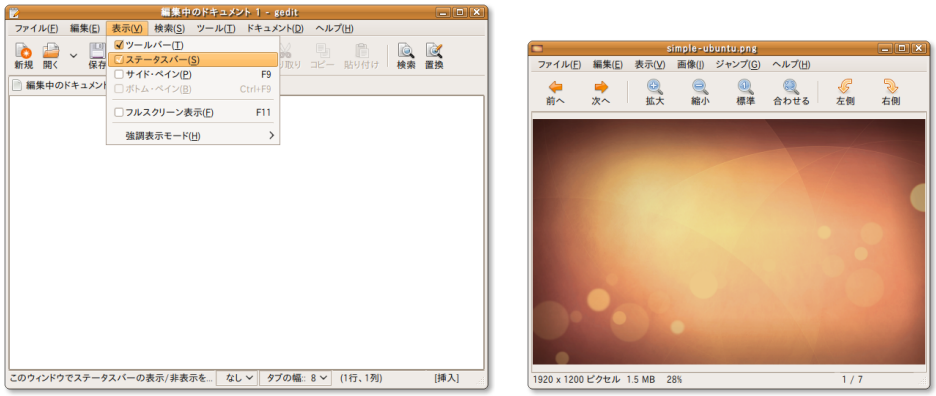


図 1.2　gtk.org では GTK+ を使って作られたアプリケーションがたくさん紹介されている



(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

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.

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 9.04 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 9.04 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，それ以外に以下に挙げた環境での設定方法を付録 A （p. 277）に付けたので，お使いの環境に合わせて開発環境を整えてください.

- Fedora 11
- OpenSUSE 11.1
- Windows Vista

Ubuntu 9.04 の日本語 Remix 版は，Ubuntu Japanese Local Community Team の Web サイト（http://www.ubuntulinux.

[*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.

jp/）で公開されています．ここでは Ubuntu 9.04 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　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．インストールするパッケージ名があらかじめわかっている場合には，次節で紹介する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方法が簡単です．

Synaptic パッケージマネージャを用いて GTK+ の開発環境をインストールするには，次のようにします．

1. メニューバーから「システム」→「システム管理」→「Synaptic パッケージマネージャ」を選択します（図 1.4）．

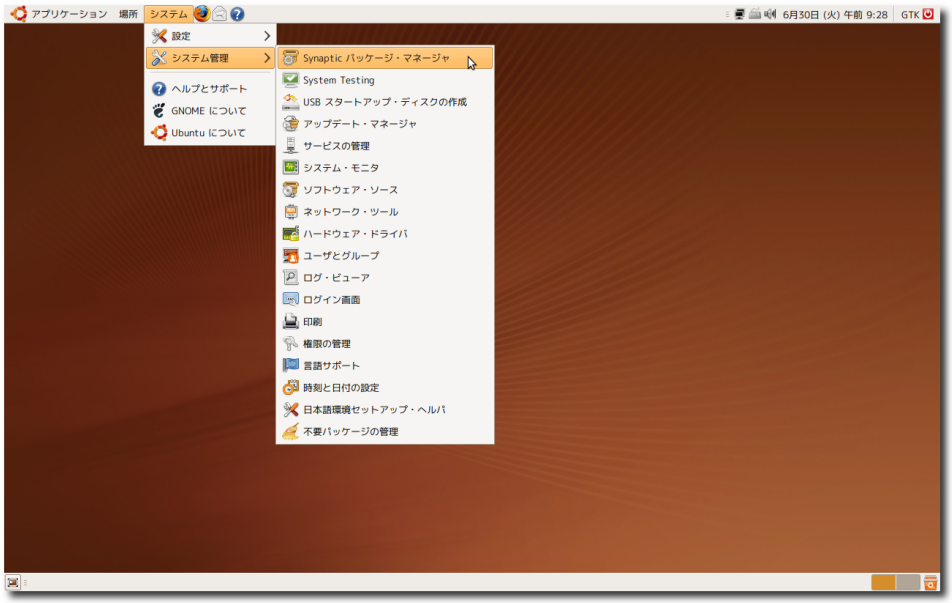
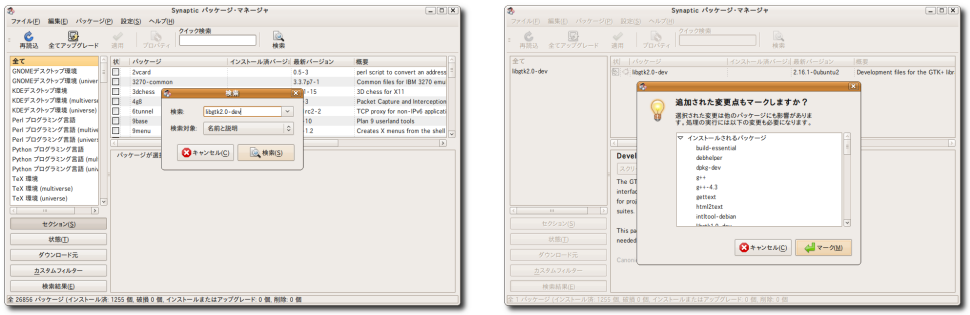


図 1.4　Synaptic パッケージマネージャの起動

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 “libgtk2.0-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 1.5(a)）．
3. 文字列 “libgtk2.0-dev” を含むパッケージの一覧が表示されるのでダブルクリックします．選択したパッケージと，それに関連してインストールされる依存関係に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 1.5(b)）．



(a) パッケージの検索　　(b) インストールされるパッケージの確認

図 1.5　パッケージの検索

4. Synaptic パッケージマネージャの「適用」ボタンをクリックします（図 1.6）．確認のために図 1.7(a) のような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．

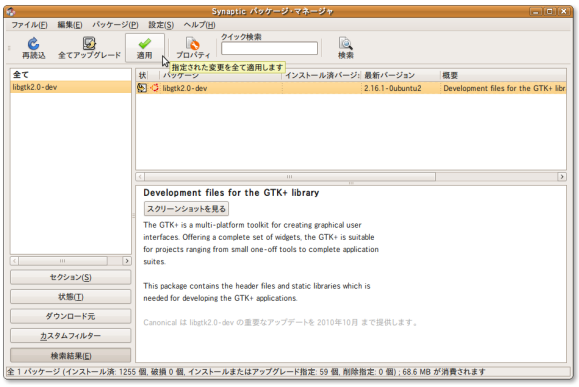
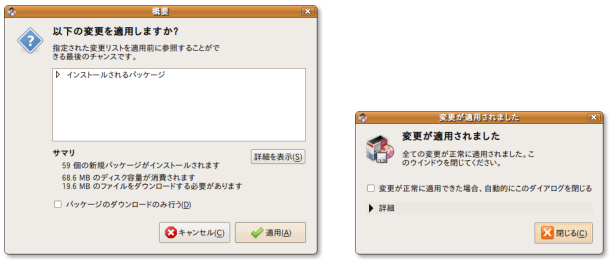


図 1.6　インストールされるパッケージ一覧



(a) 適用の確認画面　　(b) インストール終了

図 1.7　選択したインストールパッケージの適用

5. インストールが完了すると図 1.7(b) のようなダイアログが表示されるので，「閉じる (C)」ボタンを押してください．

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．第 9 章で必要となる Anjuta や gtranslator，2.2.5 で紹介する devhelp なども同じ手順でインストールできます．

1.4.2　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，aptitude 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 aptitude コマンドを使うと簡単にインストールできます．

前項で GUI によってインストールした libgtk2.0-dev を，aptitude でインストールするには，次のようにします．

```
$ sudo aptitude install libgtk2.0-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
拡張状態情報を読み込んでいます
パッケージの状態を初期化しています... 完了
以下の新規パッケージがインストールされます:
...
20.1MB のアーカイブを取得する必要があります。 展開後に 68.6MB のディスク領域が新
```

```
たに消費されます。
先に進みますか? [Y/n/?] y
```

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

```
$ sudo aptitude install anjuta devhelp gtranslator ↵
```

第 2 章

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. 277）および付録 B （p. 283）も参考にしてください．

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 2.1　チュートリアルで作成する画像ビューワ

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);
```

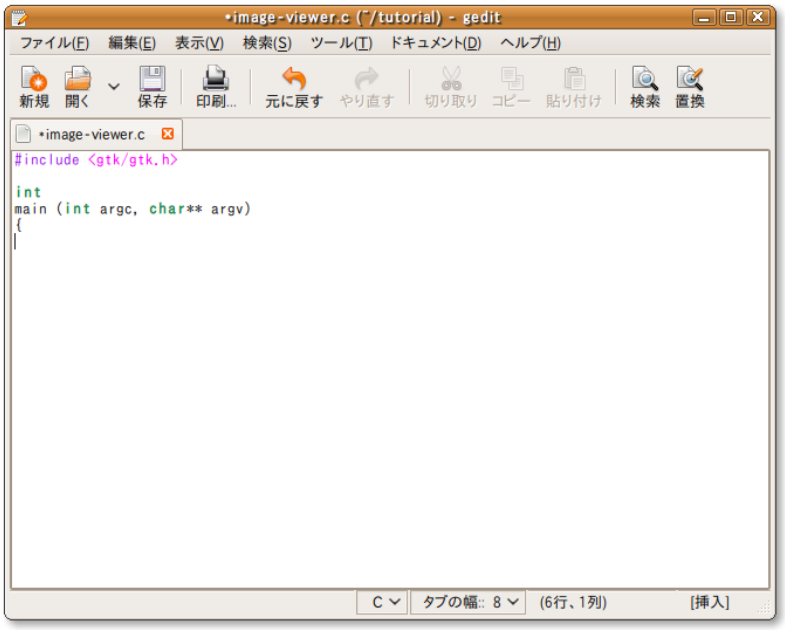


図 2.2　gedit でソースコードを入力する

```
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+-2.0 ↵
-D_REENTRANT -I/usr/include/gtk-2.0 -I/usr/lib/gtk-2.0/include -I/usr/include/at
k-1.0 -I/usr/include/cairo -I/usr/include/pango-1.0 -I/usr/include/pixman-1 -I/u
sr/include/freetype2 -I/usr/include/directfb -I/usr/include/libpng12 -I/usr/incl
ude/glib-2.0 -I/usr/lib/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+-2.0` ↵
```

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．

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．

図 2.3　ウィンドウの表示

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．

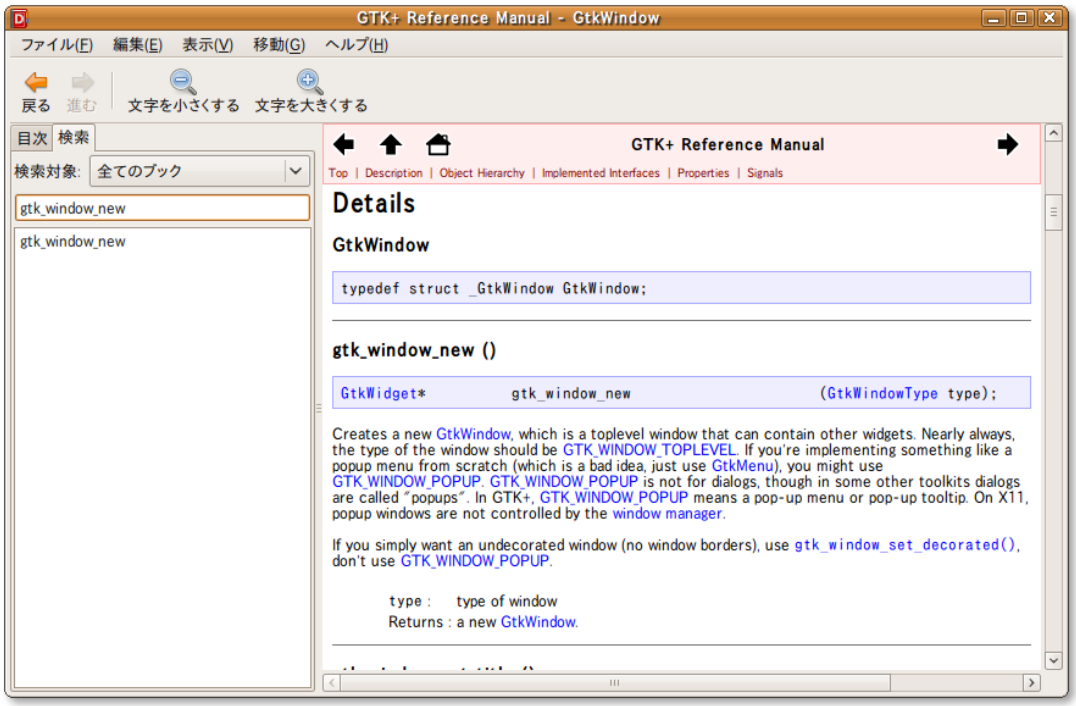


図 2.4 devhelp による関数検索

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．

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．

図 2.5　アプリケーション作成の流れ

```
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+-2.0‘ ↵
```

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．

図 2.6　ボタンのついたウィンドウ

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．

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 5.3 節（p. 69）参照）．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 5.1 （p. 66）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” では，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_vbox_new (FALSE, 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_vbox_new を使用します．

```
GtkWidget* gtk_vbox_new (gboolean homogeneous, gint spacing);
```

第 1 引数　配置される子ウィジェットに同じスペースを割り当てるかどうかを指定する．同じスペースを割り当てる場合は TRUE を，そうでない場合は FALSE 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（p. 34）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_vbox_new (FALSE, 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 （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/bird.png
```

図 2.7(a) のように，指定した画像が表示されたでしょうか．

一見問題なく表示されているように見えますが，実は図 2.7(a) では画像全体が表示されていません．全体を表示するには，図 2.7(b) のようにウィンドウを大きくしないといけません．なんだかアプリケーションとしてはあまり格好良くありませんが，この問題は次節で解決することにします．

(a) (b)

図 2.7 画像の表示，(a) 初期画面，(b) ウィンドウを縦に伸ばした画面

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，専用の関数 gtk_scrolled_window_add_with_viewport を使用します．

```
gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW (scroll_window), image);
```

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_vbox_new (FALSE, 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
52                                        (scroll_window),
53                                        GTK_POLICY_AUTOMATIC,
54                                        GTK_POLICY_AUTOMATIC);
55        {
56          GtkWidget *image;
57
58          /* ファイルから画像を読み込んでイメージの作成 */
59          image = gtk_image_new_from_file (argv[1]);
60          /* イメージをスクロールバー付きウィンドウに配置 */
61          gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW
62                                                 (scroll_window), image);
63        }
64        /* ボタンの作成 */
65        button = gtk_button_new_with_label ("Quit");
66        /* 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 */
67        g_signal_connect (G_OBJECT (button), "clicked",
68                          G_CALLBACK (cb_button_clicked), NULL);
69        /* ボタンをボックスに配置 */
70        gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
71      }
72    }
73    /* ウィンドウの表示 */
74    gtk_widget_show_all (window);
75    /* メインループ */
76    gtk_main ();
77
78    return 0;
79  }
```

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．

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

```
$ ./image-viewer ~/images/bird.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　メニューバーの追加

画像ビューワ作成の最後のステップとして，メニューバーを追加します．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 つを表示します．
メニューバーに関する拡張は，本節と次節で実装します．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

2.6.1　メニュー作成の手順

ここでは，メニューを作成するために **GtkUIManager** を使用します．GtkUIManager は，これまで扱ってきたような GUI を構成するウィジェットではなく，メニュー情報を扱う枠組みだと思ってください．GtkUIManager を利用してメニューを作成する手順は，次の通りです．

1. メニュー構成の作成
2. メニューアイテムの詳細設定
3. メニュー構成とメニューアイテム情報の登録
4. メニューバーの取得

2.6.2 メニュー構成の作成

メニュー構成は，基本的には 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．

```
static const gchar *menu_info =
  "<ui>"
  "  <menubar name='Menubar'>"
  "    <menu name='File'>"
  "      <menuitem name='Open'/>"
  "      <separator/>"
  "      <menuitem name='Quit'/>"
  "    </menu>"
  "  </menubar>"
  "</ui>";
```

メニューの構成は，<ui></ui> タグの中に記述します．さらに今回はメニューバーを作成するので，<menubar></menubar> タグ内に <menu></menu> タグや <menuitem></menuitem> タグを使用して，メニュー構成を記述していきます．

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．また，このメニューでは，Open と Quit の間に仕切り（セパレータ）を配置しています．

ここで，File，Open，Quit は実際にメニューとして表示されるラベルではなく，そのメニューおよびメニューアイテムを識別する ID だと思ってください．表示されるラベルやそのメニューアイテムに対するショートカットキー，コールバック関数等の設定については，次の項で説明します．

2.6.3 メニューアイテムの詳細設定

メニューアイテムの詳細設定は，**GtkActionEntry** で行います．主に，以下の 6 項目を設定します．

1. メニューもしくはメニューアイテムの ID
2. メニューアイコン
3. メニューラベル
4. ショートカットキー
5. メニューアイテムの説明
6. 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 GtkActionEntry entries[] = {
  {"File", NULL, "_File"},
  {"Open", GTK_STOCK_OPEN, "_Open", "<control>O", "Open an image",
    G_CALLBACK (cb_open)},
  {"Quit", GTK_STOCK_QUIT, "_Quit", "<control>Q", "Quit this program",
    G_CALLBACK (cb_quit)}
};
```

メニューアイテムの 2 番目の項目で指定している GTK_STOCK_OPEN や GTK_STOCK_QUIT は，GTK+ にあらかじめ用意されているアイコンを表す文字列で，**ストックアイテム**と呼ばれます．ストックアイテムを利用することで，メニューアイテムにアイコンを表示したり，ボタン上にアイコンを表示することが，簡単に実現できるようになっています．

それぞれの項目を設定する詳細は，ここでは省略します．詳しく知りたい場合は，第 7 章の 7.4.3 （p. 160）を参照してください．

2.6.4　メニュー情報の登録

メニュー構成等が定義できたら，以下に示す手順で GtkUIManager にそれらの情報を登録します．

1. UI マネージャの作成
2. アクショングループの作成
3. アクショングループにメニューアイテムを追加
4. UI マネージャにアクショングループを追加
5. UI マネージャにメニュー構成を追加

まず，次のようにして UI マネージャを作成します．

```
GtkUIManager *ui;
ui = gtk_ui_manager_new ();
```

次に，アクショングループ（**GtkActionGroup**）を作成して，前節で定義したメニューアイテムの詳細情報を，関数 gtk_action_group_add_actions で追加します．

```
GtkActionGroup *actions;
actions = gtk_action_group_new ("filemenu");
gtk_action_group_add_actions (actions, entries,
                              sizeof (entries) / sizeof (entries[0]),
                              parent);
```

アクショングループを作成する関数 gtk_action_group_new の引数には，UI マネージャからそのアクショングループを取得する際の ID となる文字列を与えます．

```
void gtk_action_group_add_actions (GtkActionGroup       *action_group,
                                   const GtkActionEntry *entries,
                                   guint                n_entries,
                                   gpointer             user_data);
```

第 1 引数　アクショングループ
第 2 引数　メニューアイテムの定義情報
第 3 引数　メニューアイテム数
第 4 引数　コールバック関数に渡すデータ

アクショングループにメニューアイテム情報を登録したら，今度はそのアクショングループを UI マネージャに登録します．

```
gtk_ui_manager_insert_action_group (ui, actions, 0);
```

関数 gtk_ui_manager_insert_action_group は，UI マネージャにアクショングループを登録する関数です．第 3 引数にはそのアクショングループを登録する位置を指定しますが，0（先頭位置）を指定しておけばよいでしょう．

```
void gtk_ui_manager_insert_action_group (GtkUIManager   *self,
                                         GtkActionGroup *action_group,
                                         gint           pos);
```

第 1 引数　UI マネージャ
第 2 引数　登録するアクショングループ
第 3 引数　登録位置

最後に，メニュー構成情報を UI マネージャに登録します．

```
gtk_ui_manager_add_ui_from_string (ui, menu_info, -1, NULL);
```

関数 gtk_ui_manager_add_ui_from_string は，文字列からメニュー構成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_ui_manager_add_ui_from_string (GtkUIManager *self,
                                         const gchar  *buffer,
                                         gssize       length,
                                         GError       **error);
```

第 1 引数　UI マネージャ
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　　追加した UI の ID

2.6.5 ショートカットキーの関連付け

GtkActionEntry でメニューアイテムの詳細を定義したとき，そのメニューアイテムに対するショートカットキーも定義していました．しかしそれだけでは，ウィンドウ上でそのショートカットキーを押しても反応してくれません．ウィンドウ上でメニューアイテムのショートカットキーを有効にするために，関数 gtk_window_add_accel_group で設定します．

```
gtk_window_add_accel_group (GTK_WINDOW (parent),
                            gtk_ui_manager_get_accel_group (ui));
```

この関数の第 2 引数には **GtkAccelGroup** 型のショートカットキー情報を指定する必要がありますが，これを関数 gtk_ui_manager_get_accel_group を使用して取得しています．

```
void gtk_window_add_accel_group (GtkWindow     *window,
                                 GtkAccelGroup *accel_group);
```

第 1 引数　ウィンドウウィジェット
第 2 引数　ショートカットキー情報

2.6.6 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 UI マネージャに登録したら，関数 gtk_ui_manager_get_widget を使用して，メニューバーウィジェットを取得します．

ウィジェットを取得する際に指定するパスは，メニューを構成する際に指定した name に対応する文字列を ‘/’ で継ぎ合わせた形で指定します．メニューバーは今回作成したメニューの最上位に位置するので，“/Menubar” と先頭に ‘/’ を付けて指定していることに注意してください．

```
GtkWidget* gtk_ui_manager_get_widget (GtkUIManager *self,
                                      const gchar  *path);
```

第 1 引数　UI マネージャ
第 2 引数　メニューアイテムパス
戻り値　　取得したウィジェット

2.6.7 ウィジェットを取得するテクニック

アプリケーションを作成していると，コールバック関数内でさまざまなウィジェットを参照したいときがあります．今回の例では，アプリケーションを終了する際に UI マネージャのメモリ領域を解放するために，コールバック関数 cb_quit 内で UI マネージャを指す変数を参照する必要があります．

コールバック関数にはデータを 1 つだけ渡すことができるので，参照したいウィジェットが 1 つだけであれば問題ありません．それではコールバック関数で，複数のウィジェットを参照したい場合はどうしたらよいでしょうか．1 つの解決方法としてはウィジェットをグローバル変数として定義する方法がありますが，ここではそれ以外の方法として，関数 g_object_set_data と関数 g_object_get_data を利用した方法を説明します．

以前に解説したように，すべてのウィジェットは GObject から派生したものです．この GObject 型の変数に関数 g_object_set_data を用いることで，さまざまなデータを登録できます．今回のプログラムでは，ソースコードの 110 行目で次のように利用しています．

```
g_object_set_data (G_OBJECT (window), "ui", (gpointer) ui);
```

この例では，ウィンドウに “ui” という識別子で UI マネージャを登録しています．こうすることで，ウィンドウが参照できる場所であれば，次のように関数 g_object_get_data を利用して，登録したウィジェットを取得できます．

```
GtkUIManager *ui;
ui = (GtkUIManager *) g_object_get_data (G_OBJECT (window), "ui");
```

2.6.8　コンパイルと動作確認

本節では，UI マネージャを利用したメニュー作成部分を，1 つの関数として分離しました．ソース 2–5 の 53–78 行目がその関数です．この関数の戻り値（UI マネージャへのポインタ）をメイン関数の 108 行目で受け取り，112 行目で UI マネージャからメニューバーを取得しています．

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/bird.png
```

図 2.9 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．

図 2.9　メニューバーを追加した画像ビューワ

ソース 2–5 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 (GtkAction *action, gpointer user_data)
9  {
10   g_print ("This function is not implemented yet.\n");
```

```
11 }
12
13 /*
14   Quitメニューが選択されたときに呼び出される関数
15 */
16 static void
17 cb_quit (GtkAction *action, gpointer user_data)
18 {
19   GObject *window = G_OBJECT (user_data);
20
21   g_object_unref (g_object_get_data (window, "ui"));
22   gtk_main_quit ();
23 }
24
25 /*
26  * メニューの構造
27  */
28 static const gchar *menu_info =
29   "<ui>"
30   "␣␣<menubar␣name='Menubar'>"
31   "␣␣␣␣<menu␣name='File'␣action='File'>"
32   "␣␣␣␣␣␣<menuitem␣name='Open'␣action='Open'/>"
33   "␣␣␣␣␣␣<separator/>"
34   "␣␣␣␣␣␣<menuitem␣name='Quit'␣action='Quit'/>"
35   "␣␣␣␣</menu>"
36   "␣␣</menubar>"
37   "</ui>";
38
39 /*
40  * メニューアイテムの詳細
41  */
42 static GtkActionEntry entries[] = {
43   {"File", NULL, "_File"},
44   {"Open", GTK_STOCK_OPEN, "_Open", "<control>O", "Open␣an␣image",
45    G_CALLBACK (cb_open)},
46   {"Quit", GTK_STOCK_QUIT, "_Quit", "<control>Q", "Quit␣this␣program",
47    G_CALLBACK (cb_quit)}
48 };
49
50 /*
51  * メニュー作成関数
52  */
53 static GtkUIManager*
54 create_menu (GtkWidget *parent)
55 {
56   GtkUIManager   *ui;
57   GtkActionGroup *actions;
58
59   /* UIマネージャの作成 */
60   ui = gtk_ui_manager_new ();
61   /* アクショングループの作成 */
62   actions = gtk_action_group_new ("menu");
63   /* アクショングループにメニューアイテムを追加 */
64   gtk_action_group_add_actions (actions, entries,
65                                 sizeof (entries) / sizeof (entries[0]),
66                                 parent);
67   /* UIマネージャにアクショングループを追加 */
68   gtk_ui_manager_insert_action_group (ui, actions, 0);
69   /* UIマネージャにメニュー構成を追加 */
70   gtk_ui_manager_add_ui_from_string (ui, menu_info, -1, NULL);
71   /*
72    * メニューアイテムに定義されたショートカットをウィンドウ上でも有効
73    * になるように設定
74    */
75   gtk_window_add_accel_group (GTK_WINDOW (parent),
76                               gtk_ui_manager_get_accel_group (ui));
77   return ui;
78 }
79
80 /*
81   メイン関数
82 */
83 int
84 main (int argc, char** argv)
85 {
```

```
86    GtkWidget *window;
87
88    /* GTK+の初期化およびオプション解析 */
89    gtk_init (&argc, &argv);
90
91    /* ウィンドウの作成 */
92    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
93    /* ウィンドウの大きさの設定 */
94    gtk_widget_set_size_request (window, 400, 300);
95    {
96      GtkWidget *vbox;
97
98      /* 縦にウィジェットを配置するボックスの作成 */
99      vbox = gtk_vbox_new (FALSE, 2);
100     /* ボックスをウィンドウに配置 */
101     gtk_container_add (GTK_CONTAINER (window), vbox);
102     {
103       GtkUIManager *ui;
104       GtkWidget    *menubar;
105       GtkWidget    *scroll_window;
106
107       /* メニューの作成 */
108       ui = create_menu (window);
109       /* ウィンドウにUIマネージャをデータとしてセット */
110       g_object_set_data (G_OBJECT (window), "ui", (gpointer) ui);
111       /* メニューバーを取得 */
112       menubar = gtk_ui_manager_get_widget (ui, "/Menubar");
113       /* メニューをボックスに配置 */
114       gtk_box_pack_start (GTK_BOX (vbox), menubar, FALSE, FALSE, 0);
115
116       /* スクロールバー付きウィンドウの作成 */
117       scroll_window = gtk_scrolled_window_new (NULL, NULL);
118       /* スクロールバーの表示設定 */
119       gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW
120                                       (scroll_window),
121                                       GTK_POLICY_AUTOMATIC,
122                                       GTK_POLICY_AUTOMATIC);
123       /* スクロールバー付きウィンドウをボックスに配置 */
124       gtk_box_pack_start (GTK_BOX (vbox), scroll_window, TRUE, TRUE, 0);
125       {
126         GtkWidget *image;
127
128         /* イメージの作成 */
129         image = gtk_image_new ();
130         /* イメージをスクロールバー付きウィンドウに配置 */
131         gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW
132                                                (scroll_window), image);
133       }
134     }
135   }
136   /* ウィンドウの表示 */
137   gtk_widget_show_all (window);
138   /* メインループ */
139   gtk_main ();
140
141   return 0;
142 }
```

2.6.9　まとめ

GtkUIManager 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. メニュー構成の作成
   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. メニューアイテムの詳細設定
3. メニュー構成とメニューアイテム情報の登録
   以下の手順は多少前後しても問題ありません．例えば，(a) で UI マネージャを作成した後に，(e) を先に実行して UI マネージャにメニュー構成を追加しても OK です．
   (a) UI マネージャの作成

(b) アクショングループの作成
(c) アクショングループにメニューアイテムを追加
(d) UI マネージャにアクショングループを追加
(e) UI マネージャにメニュー構成を追加

4. メニューバーの取得

2.7　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．

2.7.1　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,
                                      GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
                                      GTK_STOCK_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に対するストックアイテムと，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（GTK_STOCK_CANCEL）と「開く」ボタン（GTK_STOCK_OPEN）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．

2.7.2　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);
```

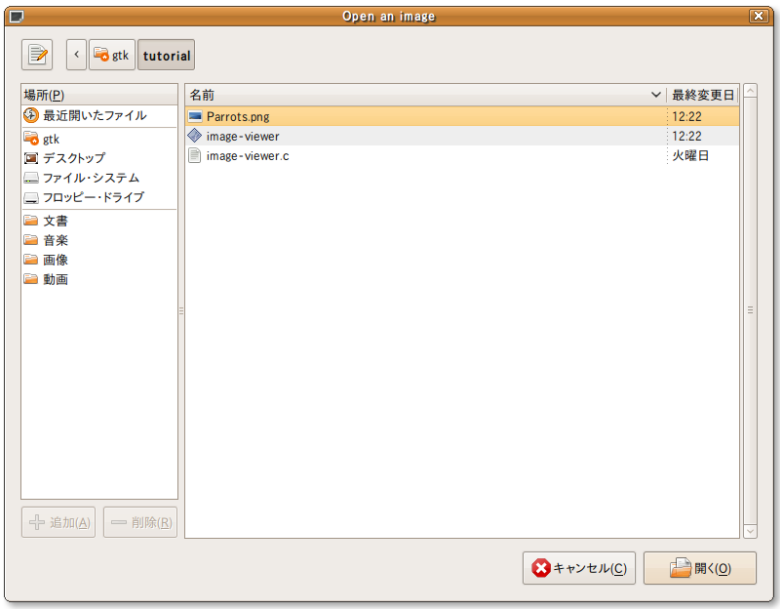
最後に，関数 gtk_file_chooser_get_filename によって取得した文字列のためのメモリ領域を，関数 g_free で解放します．

2.7.3　コンパイルと動作確認

ソース 2–6 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 164 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 2.10 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 2–6　画像ビューワ完成版：image-viewer.c

```
1  #include <gtk/gtk.h>
2
3  /*
4    Openメニューが選択されたときに呼び出される関数
5  */
6  static void
7  cb_open (GtkAction *action, gpointer user_data)
8  {
9    GtkWidget* window;
10   GtkWidget* dialog;
11   gint       response;
12
13   /* ウィンドウの取得 */
14   window = GTK_WIDGET (user_data);
15   /* ファイル選択ダイアログの作成 */
```



(a) ファイル選択ダイアログ

(b) 選択した画像の表示

図 2.10　完成した画像ビューワ

```
16  dialog = gtk_file_chooser_dialog_new ("Open␣an␣image",
17                                        GTK_WINDOW (window),
18                                        GTK_FILE_CHOOSER_ACTION_OPEN,
19                                        GTK_STOCK_CANCEL,
20                                        GTK_RESPONSE_CANCEL,
21                                        GTK_STOCK_OPEN,
22                                        GTK_RESPONSE_ACCEPT,
23                                        NULL);
24  /* ダイアログの表示 */
25  gtk_widget_show_all (dialog);
26  /* ダイアログによるファイル選択処理 */
27  response = gtk_dialog_run (GTK_DIALOG (dialog));
28  if (response == GTK_RESPONSE_ACCEPT)
29    {
30      gchar      *filename;
31      GtkWidget *image;
32
33      /* 選択したファイル名の取得 */
34      filename
35        = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER (dialog));
36      /* イメージの取得 */
37      image
38        = GTK_WIDGET (g_object_get_data (G_OBJECT (window), "image"));
39      /* ファイルから画像を読み込んでイメージにセット */
40      gtk_image_set_from_file (GTK_IMAGE (image), filename);
41      /* 文字列領域の解放 */
42      g_free (filename);
```

```
43      }
44    /* ダイアログの破棄 */
45    gtk_widget_destroy (dialog);
46  }
47
48  /*
49    Quitメニューが選択されたときに呼び出される関数
50  */
51  static void
52  cb_quit (GtkAction *action, gpointer user_data)
53  {
54    GObject *window = G_OBJECT (user_data);
55
56    g_object_unref (g_object_get_data (window, "ui"));
57    gtk_main_quit ();
58  }
59
60  /*
61   * メニューの構造
62   */
63  static const gchar* menu_info =
64    "<ui>"
65    "␣␣<menubar␣name='Menubar'>"
66    "␣␣␣␣<menu␣name='File'␣action='File'>"
67    "␣␣␣␣␣␣<menuitem␣name='Open'␣action='Open'/>"
68    "␣␣␣␣␣␣<separator/>"
69    "␣␣␣␣␣␣<menuitem␣name='Quit'␣action='Quit'/>"
70    "␣␣␣␣</menu>"
71    "␣␣</menubar>"
72    "</ui>";
73
74  /*
75   * メニューアイテムの詳細
76   */
77  static GtkActionEntry entries[] = {
78    {"File", NULL, "_File"},
79    {"Open", GTK_STOCK_OPEN, "_Open", "<control>O", "Open␣an␣image",
80     G_CALLBACK(cb_open)},
81    {"Quit", GTK_STOCK_QUIT, "_Quit", "<control>Q", "Quit␣this␣program",
82     gtk_main_quit}
83  };
84
85  /*
86   * メニュー作成関数
87   */
88  static GtkUIManager*
89  create_menu (GtkWidget *parent)
90  {
91    GtkUIManager   *ui;
92    GtkActionGroup *actions;
93
94    /* UIマネージャの作成 */
95    ui = gtk_ui_manager_new ();
96    /* アクショングループの作成 */
97    actions = gtk_action_group_new ("menu");
98    /* アクショングループにメニューアイテムを追加 */
99    gtk_action_group_add_actions (actions, entries,
100                                  sizeof (entries) / sizeof (entries[0]),
101                                  parent);
102   /* UIマネージャにアクショングループを追加 */
103   gtk_ui_manager_insert_action_group (ui, actions, 0);
104   /* UIマネージャにメニュー構成を追加 */
105   gtk_ui_manager_add_ui_from_string (ui, menu_info, -1, NULL);
106   /*
107    * メニューアイテムに定義されたショートカットをウィンドウ上でも有効
108    * になるように設定
109    */
110   gtk_window_add_accel_group (GTK_WINDOW (parent),
111                               gtk_ui_manager_get_accel_group (ui));
112   return ui;
113 }
114
115 /*
116   メイン関数
117 */
```

```
118 int
119 main (int argc, char** argv)
120 {
121   GtkWidget *window;
122
123   /* GTK+の初期化およびオプション解析 */
124   gtk_init (&argc, &argv);
125
126   /* ウィンドウの作成 */
127   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
128   /* ウィンドウの大きさの設定 */
129   gtk_widget_set_size_request (window, 400, 300);
130   {
131     GtkWidget *vbox;
132
133     /* 縦にウィジェットを配置するボックスの作成 */
134     vbox = gtk_vbox_new (FALSE, 2);
135     /* ボックスをウィンドウに配置 */
136     gtk_container_add (GTK_CONTAINER (window), vbox);
137     {
138       GtkUIManager *ui;
139       GtkWidget    *menubar;
140       GtkWidget    *scroll_window;
141
142       /* メニューの作成 */
143       ui = create_menu (window);
144       /* ウィンドウにUIマネージャをデータとしてセット */
145       g_object_set_data (G_OBJECT (window), "ui", (gpointer) ui);
146       /* メニューバーを取得 */
147       menubar = gtk_ui_manager_get_widget (ui, "/Menubar");
148       /* メニューをボックスに配置 */
149       gtk_box_pack_start (GTK_BOX (vbox), menubar, FALSE, FALSE, 0);
150
151       /* スクロールバー付きウィンドウの作成 */
152       scroll_window = gtk_scrolled_window_new (NULL, NULL);
153       /* スクロールバーの表示設定 */
154       gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW
155                                       (scroll_window),
156                                       GTK_POLICY_AUTOMATIC,
157                                       GTK_POLICY_AUTOMATIC);
158       /* スクロールバー付きウィンドウをボックスに配置 */
159       gtk_box_pack_start (GTK_BOX (vbox), scroll_window, TRUE, TRUE, 0);
160       {
161         GtkWidget *image;
162
163         /* イメージの作成 */
164         image = gtk_image_new ();
165         /* イメージをスクロールバー付きウィンドウに配置 */
166         gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW
167                                                (scroll_window), image);
168         /* ウィンドウにイメージをデータとしてセット */
169         g_object_set_data (G_OBJECT (window), "image", (gpointer) image);
170       }
171     }
172   }
173   /* ウィンドウの表示 */
174   gtk_widget_show_all (window);
175   /* メインループ */
176   gtk_main ();
177
178   return 0;
179 }
```

2.7.4　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
- GObject 型変数へのデータの登録と取得

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．

第 3 章

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 （http://library.gnome.org/devel/gtk/stable/ch01.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．

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

```
GObject
 + -- GInitiallyUnowned
       + -- GtkObject
             + -- GtkWidget
                   + -- GtkContainer
                         + -- GtkBin
                               + -- GtkWindow
                               |     + -- GtkDialog
                               |     |     + -- GtkAboutDialog
                               |     |     + -- GtkColorSelectionDialog
                               |     |     + -- GtkFileChooserDialog
                               |     |     + -- GtkFileSelection
                               |     |     + -- GtkFontSelectionDialog
                               |     |     + -- GtkInputDialog
                               |     |     + -- GtkMessageDialog
                               |     |     + -- GtkPageSetupUnixDialog
                               |     |     + -- GtkPrintUnixDialog
                               |     |     + -- GtkRecentChooserDialog
                               |     + -- GtkAssistant
                               |     + -- GtkPlug
                               + -- GtkAlignment
                               + -- GtkFrame
                               |     + -- GtkAspectFrame
                               + -- GtkButton
                               |     + -- GtkToggleButton
                               |     |     + -- GtkCheckButton
                               |     |           + -- GtkRadioButton
                               |     + -- GtkColorButton
                               |     + -- GtkFontButton
                               |     + -- GtkLinkButton
                               |     + -- GtkOptionMenu
                               |     + -- GtkScaleButton
                               |           + -- GtkVolumeButton
                               + -- GtkItem
                               |     + -- GtkMenuItem
                               |     |     + -- GtkCheckMenuItem
                               |     |     |     + -- GtkRadioMenuItem
                               |     |     + -- GtkImageMenuItem
                               |     |     + -- GtkSeparatorMenuItem
                               |     |     + -- GtkTearoffMenuItem
                               |     + -- GtkListItem
                               |     + -- GtkTreeItem
                   ...
```

図 3.1　ウィジェットの階層構造

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

水平ボックスを作成するには，関数 gtk_hbox_new を使用します．第 1 引数には TRUE か FALSE のどちらかの値を指定します．TRUE を指定した場合，このボックス内に配置されるウィジェットの幅が均等になります．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_hbox_new (gboolean homogeneous, gint spacing);
```

第 1 引数	配置される子ウィジェットに同じスペースを割り当てるかどうか．TRUE または FALSE 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

同様に，垂直ボックスを作成するには，関数 gtk_vbox_new を使用します．

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（GtkVbox の場合は上から下）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（GtkVbox では下から上）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．

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

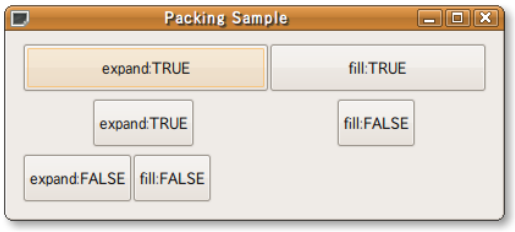


図 3.3　ウィジェットの配置方法

ソース 3–1 ウィジェットの配置：packing-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 *button;
10
11   gtk_init (&argc, &argv);
12
13   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
14   gtk_window_set_title (GTK_WINDOW (window), "Packing␣Sample");
15   gtk_container_set_border_width (GTK_CONTAINER (window), 10);
16   g_signal_connect (G_OBJECT (window), "destroy",
17                     G_CALLBACK (gtk_main_quit), NULL);
18
19   vbox = gtk_vbox_new (TRUE, 5);
20   gtk_container_add (GTK_CONTAINER (window), vbox);
21
22   hbox = gtk_hbox_new (FALSE, 0);
23   gtk_box_pack_start (GTK_BOX (vbox), hbox, TRUE, TRUE, 0);
24   button = gtk_button_new_with_label ("expand:TRUE");
25   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
26   button = gtk_button_new_with_label ("fill:TRUE");
27   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
28
29   hbox = gtk_hbox_new (FALSE, 0);
30   gtk_box_pack_start (GTK_BOX (vbox), hbox, TRUE, TRUE, 0);
31   button = gtk_button_new_with_label ("expand:TRUE");
32   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, FALSE, 0);
33   button = gtk_button_new_with_label ("fill:FALSE");
34   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, FALSE, 0);
35
36   hbox = gtk_hbox_new (FALSE, 0);
37   gtk_box_pack_start (GTK_BOX (vbox), hbox, TRUE, TRUE, 0);
38   button = gtk_button_new_with_label ("expand:FALSE");
39   gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
40   button = gtk_button_new_with_label ("fill:FALSE");
41   gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
42
43   gtk_widget_show_all (window);
44   gtk_main ();
45
46   return 0;
47 }
```

3.2.3　テーブル

テーブルの作成

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，テーブルウィジェット（GtkTable）を用いると便利です．テーブルウィジェットを作成するには，関数 gtk_table_new を使用します．第 1 引数と第 2 引数で，縦に配置するウィジェット数と横に配置するウィジェット数を指定します．

第 3 引数には，TRUE もしくは FALSE を指定します．TRUE を指定した場合は図 3.4 上段のように，すべてのセルの大きさを一番大きなウィジェットの大きさに統一します．一方，FALSE を指定した場合は図 3.4 下段のように，行と列ごとにセルの高さと幅が調整されます．

```
GtkWidget* gtk_table_new (guint    rows,
                          guint    columns,
                          gboolean homogeneous);
```

第 1 引数　縦に配置するウィジェット数
第 2 引数　横に配置するウィジェット数
第 3 引数　セルの大きさをすべて統一するかどうか．TRUE または FALSE を指定する．

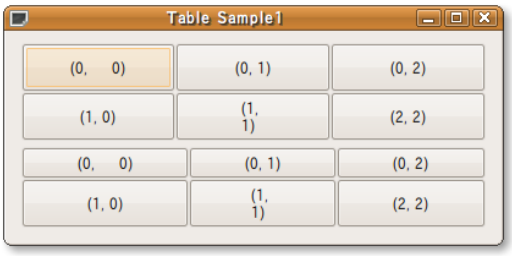


図 3.4　テーブルウィジェットへの配置 その 1

テーブルへの配置

テーブルウィジェットにウィジェットを配置するには関数 gtk_table_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_table_attach (GtkTable         *table,
                       GtkWidget        *child,
                       guint            left_attach,
                       guint            right_attach,
                       guint            top_attach,
                       guint            bottom_attach,
                       GtkAttachOptions xoptions,
                       GtkAttachOptions yoptions,
                       guint            xpadding,
                       guint            ypadding);
```

第 1 引数　テーブルウィジェット
第 2 引数　テーブル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領域の左の列番号（図 3.5 を参照）
第 4 引数　ウィジェットを配置する領域の右の列番号（図 3.5 を参照）
第 5 引数　ウィジェットを配置する領域の上の行番号（図 3.5 を参照）
第 6 引数　ウィジェットを配置する領域の下の行番号（図 3.5 を参照）
第 7 引数　横方向の配置オプション
第 8 引数　縦方向の配置オプション
第 9 引数　配置するウィジェットの左右にあけるスペース
第 10 引数　配置するウィジェットの上下にあけるスペース

第 3 引数 left_attach から第 6 引数 bottom_attach は，ウィジェットをテーブルのどの位置に配置するかを指定する引数です．図 3.5 のような 3 行 4 列のテーブルの 1 行 2 列目の斜線の位置にウィジェットを配置する場合には，left_attach，right_attach，top_attach，bottom_attach の値を 1，2，0，1 とします．left_attach，right_attach，top_attach，bottom_attach の値を 2，4，2，3 とすると，図 3.5 の右下の斜線のように，セルをまたいでウィジェットを配置することも可能です．

第 7，8 引数に指定する xoptions と yoptions は，ウィジェットをどのように配置するかを指定するオプションです．GtkAttachOptions は表 3.2 のように定義されていて，論理和演算によって複数指定できます．

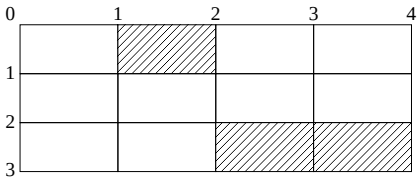


図 3.5　テーブルウィジェットへの配置指定

例えば，xoptions と yoptions に表 3.3 に示した値を与えた場合，それぞれの行に対応するテーブルは図 3.6 のようになります．図 3.6 のソースコードをソース 3–2 に示します．

ソース 3–2　テーブルへの配置：table-sample2.c

```
1  #include <gtk/gtk.h>
2
3  int
4  main (int argc, char **argv)
5  {
6    GtkWidget *window;
7    GtkWidget *vbox;
8    GtkWidget *table;
9
10   gtk_init (&argc, &argv);
11
12   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
13   gtk_window_set_title (GTK_WINDOW (window), "Table␣Sample2");
14   gtk_widget_set_size_request (window, 400, 200);
15   gtk_container_set_border_width (GTK_CONTAINER (window), 10);
16   g_signal_connect (G_OBJECT (window), "destroy",
```

表 3.2　テーブルの配置オプション（GtkAttachOptions）

値	説明
GTK_EXPAND	もしテーブルの格子があるウィジェットよりも大きい場合は，ウィジェットが広がる．
GTK_SHRINK	もしテーブル自体が要求したサイズよりも小さい空間を占めている場合，普通はウィンドウの下部へ押し出されて見えなくなる．そのような場合に GTK_SHRINK が指定されていると，ウィジェットはテーブルの中で縮む．
GTK_FILL	領域を埋めるようにテーブルを広げる．

表 3.3　GtkAttachOptions のパラメータ指定

	GtkAttachOptions
第 1 行	GTK_FILL | GTK_SHRINK | GTK_EXPAND
第 2 行	GTK_FILL | GTK_SHRINK
第 3 行	GTK_SHRINK | GTK_EXPAND

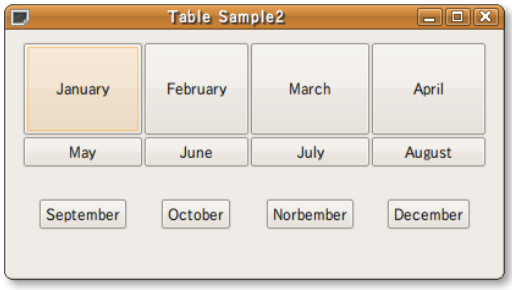


図 3.6　テーブルウィジェットへの配置 その 2

```
17                      G_CALLBACK (gtk_main_quit), NULL);
18   vbox = gtk_vbox_new (FALSE, 5);
19   gtk_container_add (GTK_CONTAINER (window), vbox);
20
21   table = gtk_table_new (3, 4, FALSE);
22   gtk_box_pack_start (GTK_BOX (vbox), table, TRUE, TRUE, 0);
23   {
24     GtkWidget *button;
25
26     button = gtk_button_new_with_label ("January");
27     gtk_table_attach (GTK_TABLE (table), button, 0, 1, 0, 1,
28                       GTK_FILL | GTK_SHRINK | GTK_EXPAND,
29                       GTK_FILL | GTK_SHRINK | GTK_EXPAND, 0, 0);
30     button = gtk_button_new_with_label ("February");
31     gtk_table_attach (GTK_TABLE (table), button, 1, 2, 0, 1,
32                       GTK_FILL | GTK_SHRINK | GTK_EXPAND,
33                       GTK_FILL | GTK_SHRINK | GTK_EXPAND, 0, 0);
34     button = gtk_button_new_with_label ("March");
35     gtk_table_attach (GTK_TABLE (table), button, 2, 3, 0, 1,
36                       GTK_FILL | GTK_SHRINK | GTK_EXPAND,
37                       GTK_FILL | GTK_SHRINK | GTK_EXPAND, 0, 0);
38     button = gtk_button_new_with_label ("April");
39     gtk_table_attach (GTK_TABLE (table), button, 3, 4, 0, 1,
40                       GTK_FILL | GTK_SHRINK | GTK_EXPAND,
41                       GTK_FILL | GTK_SHRINK | GTK_EXPAND, 0, 0);
42     button = gtk_button_new_with_label ("May");
43     gtk_table_attach (GTK_TABLE (table), button, 0, 1, 1, 2,
44                       GTK_FILL | GTK_SHRINK,
45                       GTK_FILL | GTK_SHRINK, 0, 0);
46     button = gtk_button_new_with_label ("June");
47     gtk_table_attach (GTK_TABLE (table), button, 1, 2, 1, 2,
48                       GTK_FILL | GTK_SHRINK,
49                       GTK_FILL | GTK_SHRINK, 0, 0);
50     button = gtk_button_new_with_label ("July");
51     gtk_table_attach (GTK_TABLE (table), button, 2, 3, 1, 2,
52                       GTK_FILL | GTK_SHRINK,
53                       GTK_FILL | GTK_SHRINK, 0, 0);
54     button = gtk_button_new_with_label ("August");
55     gtk_table_attach (GTK_TABLE (table), button, 3, 4, 1, 2,
56                       GTK_FILL | GTK_SHRINK,
57                       GTK_FILL | GTK_SHRINK, 0, 0);
58     button = gtk_button_new_with_label ("September");
59     gtk_table_attach (GTK_TABLE (table), button, 0, 1, 2, 3,
60                       GTK_SHRINK | GTK_EXPAND,
61                       GTK_SHRINK | GTK_EXPAND, 0, 0);
62     button = gtk_button_new_with_label ("October");
63     gtk_table_attach (GTK_TABLE (table), button, 1, 2, 2, 3,
64                       GTK_SHRINK | GTK_EXPAND,
65                       GTK_SHRINK | GTK_EXPAND, 0, 0);
66     button = gtk_button_new_with_label ("Norbember");
67     gtk_table_attach (GTK_TABLE (table), button, 2, 3, 2, 3,
68                       GTK_SHRINK | GTK_EXPAND,
69                       GTK_SHRINK | GTK_EXPAND, 0, 0);
70     button = gtk_button_new_with_label ("December");
71     gtk_table_attach (GTK_TABLE (table), button, 3, 4, 2, 3,
72                       GTK_SHRINK | GTK_EXPAND,
73                       GTK_SHRINK | GTK_EXPAND, 0, 0);
74   }
75   gtk_widget_show_all (window);
76
77   gtk_main ();
78   return 0;
79 }
```

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 9 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. 13）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．

第 4 章の表 4.1 （p. 49）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，GTK+ 2.0 Tutorial の Appendix A. GTK Signals [*1] を参照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);
```

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 3.4 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．

[*1] http://library.gnome.org/devel/gtk-tutorial/stable/a2700.html

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

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 3–3 に示します．コンパイルして実行すると，図 3.7 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 3.7 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'
```

ソース 3–3　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);
```

表 3.4　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．

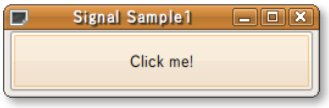


図 3.7　コールバック関数の例 1

```
 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 3–4 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 3–4 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 3–5 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 3.8 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 3–5 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
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;
```

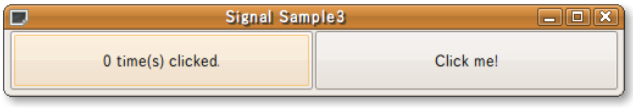


図 3.8　関数 g_signal_connect_swapped の例

```
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_hbox_new (TRUE, 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 3–6 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 3.9 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．

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

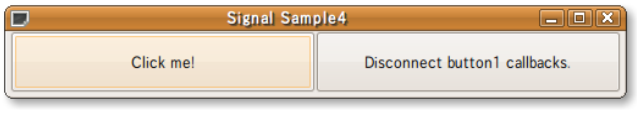


図 3.9　コールバック関数の解除

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 3–6 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     {
26       g_print ("Callback␣function␣is␣already␣diconnected.\n");
27     }
28 }
29
30 int
31 main (int argc, char **argv)
32 {
33   GtkWidget *window;
```

図 3.10　シグナルの伝搬

```
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_hbox_new (TRUE, 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 3–7 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 3.10 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，第 7 章の 7.5 節 (p. 165) 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 3–7　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 }
```

第 4 章

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と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_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．

- 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 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．

- 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．

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.

```
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 }
```

4.2.3 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.

4.2.4 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．

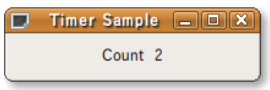


図 4.1　タイマーサンプル

ソース 4–5 タイマープログラム：timer-sample.c

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

static gint count = 10;
static guint timer_id;

static
gboolean count_down (gpointer user_data)
{
  GtkLabel *label = GTK_LABEL (user_data);
  gchar    format[] = "Count␣%2d";
  gchar            str[9];

  g_sprintf (str, format, --count);
  gtk_label_set_text (label, str);

  if (count == 0)
    {
      g_source_remove (timer_id);
      gtk_main_quit ();
    }
}

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

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title  (GTK_WINDOW (window), "Timer␣Sample");
  gtk_widget_set_size_request (window, 200, -1);
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);

  label = gtk_label_new ("Count␣10");
  gtk_container_add (GTK_CONTAINER (window), label);

  timer_id = g_timeout_add (1000, (GSourceFunc) count_down, label);

  gtk_widget_show_all (window);
  gtk_main ();

  return 0;
}
```

第 5 章

GdkPixbuf を使った画像アプリケーションの作成

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，GdkPixbuf と GTK+ を組み合わせた画像処理アプリケーションの作成方法について解説します．

5.1 画像ファイルの読み書き

5.1.1 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 5.1 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 :
```

表 5.1　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
jpeg	○	○
png	○	○
ico	○	○
ani	○	×
bmp	○	×
gif	○	×
pnm	○	×
ras	○	×
tga	○	×
tiff	○	×
xbm	○	×
xpm	○	×

```
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

$ ./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);
```

  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"
  /* width (48) */
  "\0\0\0""0"
  /* height (48) */
  "\0\0\0""0"
  /* pixel_data: */
  "\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..."
  "\0\0\12\204\0\0\0\13\233\0\0\0\14\203\0\0\0\13\6\0\0\0\12\0\..."
  "\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""..."
  "\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\..."
  "\0\0\0\0\0\0\4\0\0\0\377\340\336\334\332\352\351\346\377\357..."
  "\377\360\357\355\377\361\360\356\377\362\361\357\377\360\360..."
  ...
```

5.1.2　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,
                          ...);
```

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 のみです（表 5.1 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”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 4 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 5.2 に挙げるようなものがあります．

5.2　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);
  ```

表 5.2　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

5.3 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．

5.3.1 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 5–1 に示します．

ソース 5–1 GtkImage ウィジェットによる画像の表示：display1.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     {
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 5–1 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．

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

次の関数を使用することで，GdkPixbuf や GdkPixmap から GtkImage ウィジェットを作成することも可能です．

図 5.1 GdkImage ウィジェットによる画像の表示

```
GtkWidget* gtk_image_new_from_pixbuf (GdkPixbuf *pixbuf);
GtkWidget* gtk_image_new_from_pixmap (GdkPixmap *pixmap,
                                      GdkBitmap *mask);
```

逆に次の関数を使用することで，GtkImage ウィジェットから GdkPixbuf や GdkPixmap 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);
void gtk_image_get_pixmap (GtkImage *image,
                           GdkPixmap **pixmap, GdkBitmap **mask);
```

5.3.2　GtkDrawingArea ウィジェットによる画像の表示

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．このウィジェットは図形や画像を描画するウィジェットで，GDK ライブラリで定義された GdkDrawable（これ以降はドローアブルと呼ぶことにします）という描画領域を持ちます．なお，ウィンドウウィジェットもドローアブルを持つため，ウィンドウ上に直接描画することも可能です．

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．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 5–2（p. 72）に示します．

画像の読み込み（45 行目）

ウィンドウに表示する画像を，関数 gdk_pixbuf_new_from_file を使ってファイルから読み込みます．

GtkDrawingArea ウィジェットの設定（52–55 行目）

GtkImage ウィジェットは，画像の大きさに応じて自動的に最適な大きさに伸縮しますが，GtkDrawingArea ウィジェットを用いて画像を表示する場合には，ユーザが最適な大きさを指定する必要があります．関数 gtk_widget_set_size_request は，ウィジェットの大きさを設定します．

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

コールバック関数の設定（56–57 行目）

GtkDrawingArea ウィジェットで画像を描画する場合には，**expose-event シグナル**のコールバック関数に画像描画のコードを記述するか，画像を描画する関数を呼び出す記述をします．ユーザがドローアブル上に図形を描画する場合，図形を描画しているウィンドウが他のウィンドウで隠れると，図形を再描画しない限り，隠された領域が自動的に描画されることはありません．このような再描画が必要になったときに発生するシグナルが，expose-event シグナルです．そのため，一般的にはこのシグナルに対するコールバック関数で描画処理を行います．

画像の描画（16–23 行目）

ここでは画像を描画するために，GdkPixbuf 型のデータから GdkPixmap 型のデータ（**ピックスマップ**）を作成し，作成したピックスマップをドローアブルに描画しています．

GdkPixbuf 型のデータからピックスマップを生成するには，関数 gdk_pixbuf_render_pixmap_and_mask を使います．

```
void gdk_pixbuf_render_pixmap_and_mask (GdkPixbuf *pixbuf,
                                        GdkPixmap **pixmap_return,
                                        GdkBitmap **mask_return,
                                        int       alpha_threshold);
```

第 1 引数　GdkPixbuf 型の画像データ．
第 2 引数　生成されるピックスマップ．
第 3 引数　生成されるビットマップ（マスクデータ）．
第 4 引数　画像データがアルファチャネルを持つ場合の透明度のしきい値．0 から 255 の範囲で指定する．

画像データにアルファチャネルがある場合には，第 3 引数に与えた変数に，第 4 引数に与えたしきい値でアルファチャネルを 2 値化したマスク情報が返ります．

この方法を使っていったん画像データを背景に登録すると，画像の再描画は関数 gdk_window_clear を呼び出すのみで済むという利点があります．

gdk_pixbuf_composite_color

ソース 5–2 では，関数 gdk_pixbuf_composite_color を使って，透明部分にタイル柄を描画して，透明部分を演出しています．この関数の最後の 3 つの引数（第 15–17 引数）で，タイルの大きさ，1 つ目のタイルの色，2 つ目のタイルの色を指定しています．

```
void gdk_pixbuf_composite_color (const GdkPixbuf *src,
                                 GdkPixbuf       *dest,
                                 int              dest_x,
                                 int              dest_y,
                                 int              dest_width,
                                 int              dest_height,
                                 double           offset_x,
                                 double           offset_y,
                                 double           scale_x,
                                 double           scale_y,
                                 GdkInterpType    interp_type,
                                 int              overall_alpha,
                                 int              check_x,
                                 int              check_y,
                                 int              check_size,
                                 guint32          color1,
                                 guint32          color2);
```

第 1 引数： 入力画像.
第 2 引数： 出力画像.
第 3 引数： 入力画像の描画を開始する X 座標.
第 4 引数： 入力画像の描画を開始する Y 座標.
第 5 引数： 描画する幅.
第 6 引数： 描画する高さ.
第 7 引数： 入力画像のオフセットの X 座標.
第 8 引数： 入力画像のオフセットの Y 座標.
第 9 引数： 入力画像の X 方向の拡大率.
第 10 引数： 入力画像の Y 方向の拡大率.
第 11 引数： 画素値の内挿方法.
第 12 引数： 透明度の設定．0 から 255 の範囲で指定する．
第 13 引数： チェッカーボードのオフセットの X 座標.
第 14 引数： チェッカーボードのオフセットの Y 座標.
第 15 引数： チェッカーボードの大きさ（2 のべき乗でなければならない）.
第 16 引数： チェッカーボードの色 1.
第 17 引数： チェッカーボードの色 2.

これは，入力画像を scale_x，scale_y だけ拡大して，(offset_x, offset_y) だけ平行移動した画像を，出力画像の (dest_x, dest_y) から幅 dest_width，高さ dest_height の矩形領域に描画する関数です．透明領域は指定したチェッカーボードで塗りつぶされます．GdkInterpType は表 5.3 のように定義されています．

表 5.3 画像補間の種類

値	説明
GDK_INTERP_NEAREST	最近傍補間.
GDK_INTERP_TILES	各点を平行四辺形として扱い，各点間の輪郭をアンチエイリアシングする.
GDK_INTERP_BILINEAR	線形補間.
GDK_INTERP_HYPER	2 次補間.

最後に，関数 gdk_window_set_back_pixmap で生成したピックスマップをドローアブル（ここでは GtkDrawingArea ウィ

ジェットのメンバ window がドローアブルになる）にセットし，関数 gdk_window_clear で描画領域の更新処理を行っています．

```
void gdk_window_set_back_pixmap (GdkWindow *window,
                                 GdkPixmap *pixmap,
                                 gboolean  parent_relative);

void gdk_window_clear (GdkWindow *window);
```

ソース 5–2 GtkDrawingArea ウィジェットによる画像の表示: display2.c

```
1  #include <gtk/gtk.h>
2  #include <stdlib.h>
3
4  static gint
5  cb_expose (GtkWidget      *widget,
6             GdkEventExpose *event,
7             gpointer       user_data)
8  {
9    GdkPixbuf *pixbuf = (GdkPixbuf *) user_data;
10   GdkPixbuf *background;
11   GdkPixmap *pixmap;
12   int       w, h;
13
14   w = gdk_pixbuf_get_width (pixbuf);
15   h = gdk_pixbuf_get_height (pixbuf);
16   background = gdk_pixbuf_new (GDK_COLORSPACE_RGB, FALSE, 8, w, h);
17   gdk_pixbuf_composite_color (pixbuf, background,
18                               0, 0, w, h, 0, 0, 1.0, 1.0,
19                               GDK_INTERP_BILINEAR, 255, 0, 0, 16,
20                               0xaaaaaa, 0x555555);
21   gdk_pixbuf_render_pixmap_and_mask (background, &pixmap, NULL, 255);
22   gdk_window_set_back_pixmap (widget->window, pixmap, FALSE);
23   gdk_window_clear(widget->window);
24
25   g_object_unref (background);
26   g_object_unref (pixmap);
27
28   return TRUE;
29 }
30
31 int
32 main (int argc, char *argv[])
33 {
34   GtkWidget *window;
35   GtkWidget *canvas;
36   GdkPixbuf *pixbuf;
37
38   if (argc < 2)
39     {
40       g_print ("Usage:␣./display2␣imagefile\n");
41       exit (0);
42     }
43   gtk_init (&argc, &argv);
```

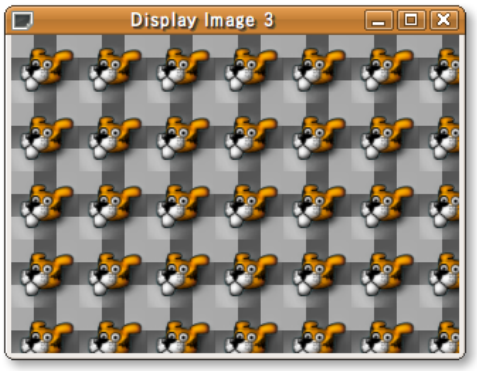


図 5.2　GtkDrawingArea ウィジェットによる画像の表示

```
44
45   pixbuf = gdk_pixbuf_new_from_file (argv[1], NULL);
46
47   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
48   gtk_window_set_title (GTK_WINDOW (window), "Display␣Image␣2");
49   g_signal_connect (G_OBJECT (window), "destroy",
50                     G_CALLBACK (gtk_main_quit), NULL);
51
52   canvas = gtk_drawing_area_new ();
53   gtk_widget_set_size_request (canvas,
54                                gdk_pixbuf_get_width (pixbuf),
55                                gdk_pixbuf_get_height (pixbuf));
56   g_signal_connect (G_OBJECT (canvas), "expose-event",
57                     G_CALLBACK (cb_expose), pixbuf);
58   gtk_container_add (GTK_CONTAINER (window), canvas);
59
60   gtk_widget_show_all (window);
61   gtk_main ();
62
63   return 0;
64 }
```

5.4　簡単な画像処理アプリケーションの作成

これまで学習した内容を利用して，簡単な画像処理アプリケーションを作成してみます．作成するアプリケーションは図 5.3 に示すような，しきい値をスピンボタンでコントロールしてインタラクティブに画像を 2 値化する GUI アプリケーションです．

5.4.1　ウィジェットの配置

図 5.3 のアプリケーションで使用したウィジェットの配置を，図 5.4 に示します．この図のポイントは，GtkDrawingArea ウィジェットの配置です．

今までは画像の大きさに合わせてウィンドウの大きさを変化させていましたが，ウィンドウが小さすぎたり大きすぎたりすると，アプリケーションとして見た目が悪くなります．ウィンドウより大きな画像を表示するのに有効なのが，GtkScrolledWindow ウィジェットです．このウィジェットは名前の通り，スクロールバーが付いたウィンドウです．スクロールバーを使って画面をスクロールすることで，ウィンドウ内に収まらない部分も表示できます．

それでは，画像がウィンドウより小さい場合はどうでしょう．図 5.2 を見てもわかるように，関数 gdk_window_set_back_pixmap で画像を表示する場合には，画像サイズよりもドローアブルのほうが大きいと，画像をタイル状に繰り返し描画します．そこで，画像がウィンドウよりも小さいときには，画像をウィンドウの中央に表示するために，アラインメントウィジェット（GtkAlignment）を使用します．

GtkAlignment ウィジェットはコンテナウィジェットで，ウィジェット作成時にパックするウィジェットの配置位置とウィジェットの拡大率を指定します．

図 5.3　画像をしきい値で 2 値化するアプリケーション

```
GtkWidget* gtk_alignment_new (gfloat xalign, gfloat yalign,
                              gfloat xscale, gfloat yscale);
```

xalign と yalign には，0 から 1 までの値を実数で指定します．0 なら左（上），1 なら右（下）となります．また xscale と yscale は，GtkAlignment ウィジェットの大きさを 1 としたときの子ウィジェットの大きさを指定します．この値を 0 とすると，GtkAlignment ウィジェットの大きさにかかわらず，子ウィジェットの大きさで表示します．

5.4.2　GUI の作成

それぞれのウィジェットを図 5.4 に示すように配置して，GUI を作成します．

ベースウィジェットの作成（14–20 行目）
ベースとなるウィンドウウィジェットと，ウィジェットを配置する垂直ボックスを作成します．

スクロールウィンドウの作成（22–29 行目）
まず，関数 gtk_scrolled_window_new でスクロールウィンドウを作成します．そして，スクロールバーの表示設定を関数 gtk_scrolled_window_set_policy で行います．スクロールバーの表示設定は，表 5.4 の中から選択します．

```
GtkWidget* gtk_scrolled_window_new (GtkAdjustment *hadjustment,
                                    GtkAdjustment *vadjustment);

void gtk_scrolled_window_set_policy (GtkScrolledWindow *scrolled_window,
                                     GtkPolicyType hscrollbar_policy,
                                     GtkPolicyType vscrollbar_policy);
```

表 5.4　スクロールバーの表示設定

種類	説明
GTK_POLICY_ALWAYS	常にスクロールバーを表示する．
GTK_POLICY_AUTOMATIC	子ウィジェットの大きさに応じて表示する．
GTK_POLICY_NEVER	表示しない．

GtkScrolledWindow の設定として，ウィンドウ内の装飾の設定があります．

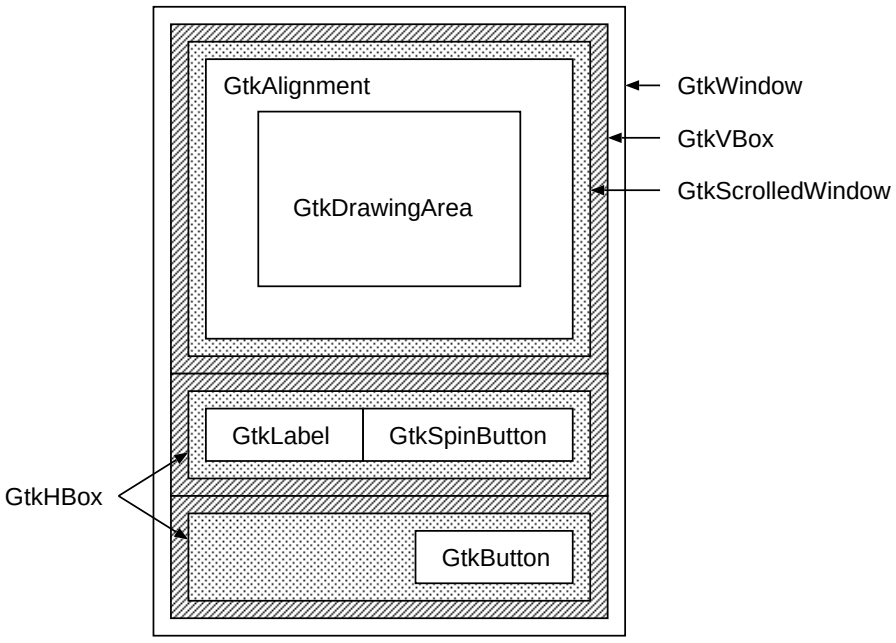


図 5.4　画像 2 値化アプリケーションのウィジェット配置

この設定は，関数 gtk_scrolled_window_set_shadow_type で行います．

```
void
gtk_scrolled_window_set_shadow_type(GtkScrolledWindow scrolled_window,
                                    GtkShadowType type);
```

GtkShadowType は，表 5.5 のように定義されています．それぞれの項目を指定した場合のウィンドウの装飾を，図 5.5 を参考にしてください．

表 5.5　スクロールバーの表示設定

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．

アラインメントウィジェットの作成（33–36 行目）

GtkAlignment ウィジェットは関数 gtk_alignment_new で作成します．今回はアラインメントの中央に GtkDrawingArea ウィジェットを配置したいので，それぞれの引数を次のように指定します．

```
alignment = gtk_alignment_new (0.5, 0.5, 0, 0);
```

スピンボタンの作成（57–61 行目）

スピンボタンウィジェットの作成には関数 gtk_spin_button_new を使用します．スピンボタンを作成する場合にはアジャストメント（GtkAdjustment）というオブジェクトを作成する必要があります．このオブジェクトは，スピンボタンで扱う数値の範囲などを扱うオブジェクトです．ここでは，初期値を 128，最小値 0，最大値 255 に設定しています． 60–61 行目では，この次の節でも説明しますが，スピンボタンの **value-changed シグナル** に対するコールバック関数を定義しています．

```
GtkWidget* gtk_spin_button_new (GtkAdjustment *adjustment,
                                gdouble       climb_rate,
                                guint         digits);

GtkObject* gtk_adjustment_new (gdouble value,
                               gdouble lower,
                               gdouble upper,
                               gdouble step_increment,
                               gdouble page_increment,
                               gdouble page_size);
```

図 5.5　ウィンドウ装飾の種類

ソース 5–3–1 GUI 作成関数： binarize.c から一部抜粋

```
1  #define WINDOW_WIDTH    400
2  #define WINDOW_HEIGHT   400
3
4  static GtkWidget*
5  make_interface (const gchar *title,
6                  GdkPixbuf   *pixbuf)
7  {
8    GtkWidget *window;
9    GtkWidget *vbox;
10   GtkWidget *scroll_window;
11   GtkWidget *hbox;
12   GtkWidget *drawingarea;
13
14   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
15   gtk_window_set_title (GTK_WINDOW (window), title);
16   gtk_widget_set_size_request (window, WINDOW_WIDTH, WINDOW_HEIGHT);
17
18   vbox = gtk_vbox_new (FALSE, 3);
19   gtk_container_add (GTK_CONTAINER (window), vbox);
20   gtk_container_set_border_width (GTK_CONTAINER (vbox), 5);
21
22   scroll_window = gtk_scrolled_window_new (NULL, NULL);
23   gtk_box_pack_start (GTK_BOX (vbox), scroll_window, TRUE, TRUE, 0);
24   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
25                                   GTK_POLICY_AUTOMATIC,
26                                   GTK_POLICY_AUTOMATIC);
27   gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW
28                                        (scroll_window),
29                                        GTK_SHADOW_ETCHED_IN);
30   {
31     GtkWidget *alignment;
32
33     alignment = gtk_alignment_new (0.5, 0.5, 0, 0);
34     gtk_scrolled_window_add_with_viewport (GTK_SCROLLED_WINDOW
35                                            (scroll_window),
36                                            alignment);
37     {
38       drawingarea = gtk_drawing_area_new ();
39       gtk_container_add (GTK_CONTAINER (alignment), drawingarea);
40       gtk_widget_set_size_request (drawingarea,
41                                    gdk_pixbuf_get_width (pixbuf),
42                                    gdk_pixbuf_get_height (pixbuf));
43       g_signal_connect (G_OBJECT (drawingarea), "expose-event",
44                         G_CALLBACK(cb_expose), (gpointer) pixbuf);
45     }
46   }
47   hbox = gtk_hbox_new (FALSE, 5);
48   gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
49   {
50     GtkWidget *label;
51     GtkObject *adjustment;
52     GtkWidget *spinbutton;
53
54     label = gtk_label_new ("Threshold␣:");
55     gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);
56
57     adjustment = gtk_adjustment_new (128, 0, 255, 1, 5, 0);
58     spinbutton = gtk_spin_button_new (GTK_ADJUSTMENT (adjustment), 1, 0);
59     gtk_box_pack_start (GTK_BOX (hbox), spinbutton, TRUE, TRUE, 0);
60     g_signal_connect (G_OBJECT (spinbutton), "value-changed",
61                       G_CALLBACK (cb_value_changed), drawingarea);
62   }
63   hbox = gtk_hbox_new (FALSE, 0);
64   gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
65   {
66     GtkWidget *button;
67     button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
68     gtk_box_pack_end (GTK_BOX (hbox), button, FALSE, FALSE, 0);
69     g_signal_connect (G_OBJECT (button), "clicked",
70                       G_CALLBACK(cb_quit), (gpointer) pixbuf);
71   }
72   g_signal_connect (G_OBJECT (window), "destroy",
```

```
73                      G_CALLBACK (cb_quit), (gpointer) pixbuf);
74
75    return window;
76  }
```

5.4.3　コールバック関数の設定

ここでは，GtkDrawingArea ウィジェットに関するコールバック関数と，GtkSpinButton ウィジェットに関するコールバック関数について説明します．

GtkDrawingArea ウィジェットに対するコールバック関数

これまで扱ったように，GtkDrawingArea ウィジェットの expose-event シグナルは，ウィジェットの再描画が必要になったときに発生します．今回は expose-event シグナルに対するコールバック関数 cb_expose で，画像の 2 値化処理と画像の描画を行うことにします．

2 値化関数の呼び出し（17 行目）

まず 2 値化関数を呼び出して，入力画像（pixbuf）を 2 値化した画像（bin）を作成します．

2 値化画像の表示（19–31 行目）

ピックスマップを作成してドローアブルの背景に登録する方法は，前に紹介した通りです．

GtkSpinButton ウィジェットに対するコールバック関数

関数 cb_value_changed は，スピンボタンウィジェットの value-changed イベントに対するコールバック関数です．このイベントはスピンボタンの矢印で値を変更したときや，エントリボックスで値を変更（Enter キーを押して確定）したときに発生するイベントです．

値の変更が生じたとき，現在の値を取得して画像描画関数（cb_expose）を呼び出します．値の取得には関数 gtk_spin_button_get_value_as_int を使っています．この関数以外にも，値を取得する関数として gtk_spin_button_get_value がありますが，今回は 0 から 255 までの整数しか扱わないので，前者を使います．

そして，関数 gtk_widget_queue_draw を呼び出すことで，GtkDrawingArea ウィジェットに expose-event シグナルを発生させています．したがって，cb_expose 関数が呼び出されることになります．

```
gint    gtk_spin_button_get_value_as_int (GtkSpinButton *spin_button);
gdouble gtk_spin_button_get_value        (GtkSpinButton *spin_button);
void    gtk_widget_queue_draw            (GtkWidget *widget);
```

ソース 5–3–2　コールバック関数の設定：binarize.c から一部抜粋

```
1  #include <gtk/gtk.h>
2  #include "operation.h"
3
4  GdkPixbuf *bin;
5  gint       threshold = 128;
6
7  static gint
8  cb_expose (GtkWidget      *widget,
9             GdkEventExpose *event,
10            gpointer       data)
11 {
12   GdkPixbuf *pixbuf = (GdkPixbuf *) data;
13   GdkPixbuf *background;
14   GdkPixmap *pixmap;
15   int        w, h;
16
17   create_binarized_image (pixbuf, bin, threshold);
18
19   w = gdk_pixbuf_get_width (pixbuf);
20   h = gdk_pixbuf_get_height (pixbuf);
21   background = gdk_pixbuf_new (GDK_COLORSPACE_RGB, FALSE, 8, w, h);
22   gdk_pixbuf_composite_color (bin, background,
23                               0, 0, w, h, 0, 0, 1.0, 1.0,
```

```
24                                GDK_INTERP_BILINEAR, 255, 0, 0, 16,
25                                0xaaaaaa, 0x555555);
26    gdk_pixbuf_render_pixmap_and_mask (background, &pixmap, NULL, 255);
27    gdk_window_set_back_pixmap (widget->window, pixmap, FALSE);
28    gdk_window_clear(widget->window);
29
30    g_object_unref (background);
31    g_object_unref (pixmap);
32
33    return TRUE;
34  }
35
36  static void
37  cb_value_changed (GtkSpinButton *spinbutton,
38                    gpointer       user_data)
39  {
40    threshold =
41      gtk_spin_button_get_value_as_int (GTK_SPIN_BUTTON (spinbutton));
42    gtk_widget_queue_draw (GTK_WIDGET (user_data));
43  }
```

5.4.4　2 値化処理

最後に，画像をしきい値で 2 値化する関数の説明をします．まずソース 5–3–3 に示すように，ヘッダファイル operation.h 内で，画素アクセスマクロを定義します．画像データが無駄なくメモリ上に配置されている場合，画像 1 行分のバイト数は次の式で計算できます．

$$\text{画像の幅} \times \text{チャネル数} \tag{5.1}$$

しかし GdkPixbuf データは処理の高速化のために，画像 1 行分のデータが必ずしも式 5.1 とはなりません．そのため，GdkPixbuf データの画像 1 行分のバイト数を取得するために，関数 gdk_pixbuf_get_rowstride が用意されています．したがって，点 (x, y) の p 番目のチャネルの画素位置は式 5.2 で計算できます．

$$\text{画像 1 行分のバイト数}\ (rowstride) \times y + \text{チャネル数} \times x + p \tag{5.2}$$

ソース 5–3–2 2 値化関数のヘッダファイル：operation.h

```
1  #ifndef __OPERATION_H__
2  #define __OPERATION_H__
3
4  #define gdk_pixbuf_get_pixel(pixbuf,x,y,p) \
5  (*(gdk_pixbuf_get_pixels ((pixbuf)) + \
6   gdk_pixbuf_get_rowstride ((pixbuf)) * (y) + \
7   gdk_pixbuf_get_n_channels ((pixbuf)) * (x) + (p)))
8
9  #define gdk_pixbuf_put_pixel(pixbuf,x,y,p,val) \
10 (*(gdk_pixbuf_get_pixels ((pixbuf)) + \
11  gdk_pixbuf_get_rowstride ((pixbuf)) * (y) + \
12  gdk_pixbuf_get_n_channels ((pixbuf)) * (x) + (p)) = (val))
13
14 void create_binarized_image (const GdkPixbuf *src,
15                              GdkPixbuf       *dst,
16                              gint             threshold);
17
18 #endif /* __OPERATION_H__ */
```

実際の画像の 2 値化はソース 5–3–4 の operation.c の create_binarized_image 中で行っています．16–18 行目で点 (x, y) の RGB 値を取得して，19 行目で RGB 値を輝度値（Y 信号）に変換しています．そして，輝度値としきい値を比較して，輝度値がしきい値以上なら出力画像の画素値を 255 に，そうでなければ 0 にします．

ソース 5–3–4 2 値化関数：operation.c

```
1  #include <gtk/gtk.h>
2  #include "operation.h"
3
4  void
5  create_binarized_image (const GdkPixbuf *src,
```

```
6                           GdkPixbuf       *dst,
7                           gint            threshold)
8 {
9   int     row, col;
10  guchar r, g, b, y, val;
11
12  for (row = 0; row < gdk_pixbuf_get_height (src); row++)
13    {
14      for (col = 0; col < gdk_pixbuf_get_width (src); col++)
15        {
16          r = gdk_pixbuf_get_pixel (src, col, row, 0);
17          g = gdk_pixbuf_get_pixel (src, col, row, 1);
18          b = gdk_pixbuf_get_pixel (src, col, row, 2);
19          y = (guchar) (0.299 * r + 0.587 * g + 0.114 * b);
20          val = (y >= threshold) ? 255 : 0;
21          gdk_pixbuf_put_pixel (dst, col, row, 0, val);
22          gdk_pixbuf_put_pixel (dst, col, row, 1, val);
23          gdk_pixbuf_put_pixel (dst, col, row, 2, val);
24        }
25    }
26 }
```

5.4.5　ソースコードのコンパイルと動作テスト

これまで紹介したソースコードとソース 5–3–5 のメイン関数を合わせて，画像 2 値化アプリケーションを完成させます．

今回は，ヘッダファイルを含めて 3 つのファイルからソースコードが構成されています．これまでのようにコマンドラインでひとつひとつ gcc コマンドを実行していると煩わしいので，ここでは make コマンドでコンパイルすることにします．このソースコードの Makefile をソース 5–3–6 に示します．

作成した画像 2 値化アプリケーションの実行例を，図 5.6 に示します．図 5.6 の左側が入力画像です．この画像を，本節で作成したアプリケーションを使って，しきい値をコントロールしながら 2 値化している様子が，図 5.6 の右側になります．

```
$ make ↵
gcc -Wall `/usr/bin/pkg-config --cflags gtk+-2.0`   -c -o binarize.o binarize.c
gcc -Wall `/usr/bin/pkg-config --cflags gtk+-2.0`   -c -o operation.o operation.c
gcc binarize.o operation.o `/usr/bin/pkg-config --libs gtk+-2.0` `/usr/bin/pkg-co
nfig --libs gtk+-2.0` -o binarize-gui
$ ./binarize-gui sample.png ↵
```

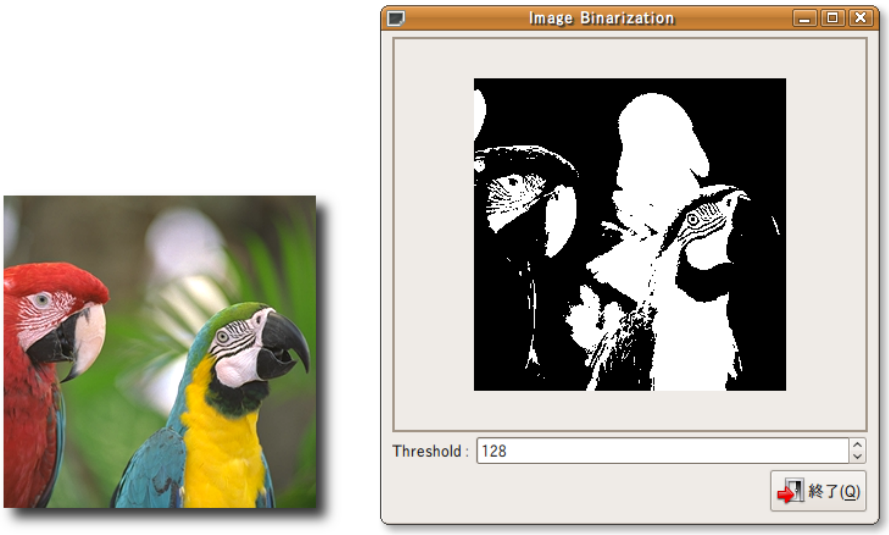


図 5.6　入力画像とアプリケーションの実行結果

ソース 5–3–5 メイン関数：binarize.c から一部抜粋

```
1  int
2  main (int argc, char *argv[])
3  {
4    GtkWidget *window;
5    GdkPixbuf *pixbuf;
6
7    if (argc < 2)
8      {
9        g_print ("Usage:␣%s␣imagefile\n", argv[0]);
10       exit (0);
11     }
12   gtk_init (&argc, &argv);
13
14   pixbuf = gdk_pixbuf_new_from_file (argv[1], NULL);
15   bin    = gdk_pixbuf_copy (pixbuf);
16   window = make_interface ("Image␣Binarization", pixbuf);
17   gtk_widget_show_all (window);
18   gtk_main ();
19
20   return 0;
21 }
```

ソース 5–3–6 画像 2 値化アプリケーションの Makefile

```
1  CC              = gcc
2
3  INSTALL         = /usr/bin/install
4  PKG_CONFIG      = /usr/bin/pkg-config
5
6  CFLAGS          = -Wall `$(PKG_CONFIG) --cflags gtk+-2.0`
7  LDFLAGS         = `$(PKG_CONFIG) --libs gtk+-2.0`
8
9  SRCS            = binarize.c operation.c
10 HDRS            = operation.h
11 OBJS            = $(SRCS:.c=.o)
12
13 DEST            = $(PREFIX)/bin
14
15 PROGRAM         = binarize-gui
16
17 all:            $(PROGRAM)
18
19 $(PROGRAM):     $(OBJS) $(HDRS)
20                 $(CC) $(OBJS) $(LDFLAGS) -o $(PROGRAM)
21
22 clean:;         rm -f *.o *~ $(PROGRAM)
23
24 install:        $(PROGRAM)
25                 $(INSTALL) -s $(PROGRAM) $(DEST)
```

第 6 章

cairo による図形の描画

6.1 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．

6.2 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 6.1 は，上記の説明を模式的に示したものです．まず，図 6.1(a) のようにソースに色を設定し，次に図 6.1(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．

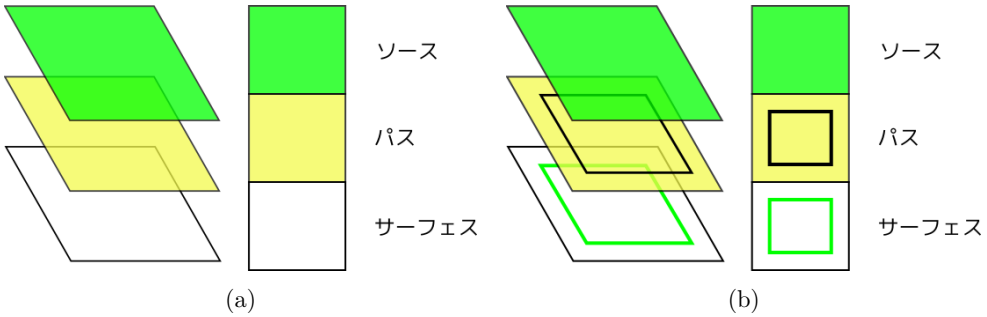


図 6.1　cairo での図形描画の概念: (a) ソース（色）の設定，(b) パスの設定と描画

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．

6.3　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

6.3.1　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 (GdkDrawable *drawable);
```

第 1 引数　GdkDrawable 型の変数．
戻り値　コンテキスト．

6.3.2　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 6.1 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 6.1　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，6.11.3 （p. 99）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 6.1 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.

6.3.3 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);
```

6.3.4　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);
```

6.4　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);
```

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．

6.5　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．6.2 節（p. 81）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);
```

6.6　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．

6.6.1　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．

6.6.2　線端の種類

線端の種類には，表 6.2 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 6.2 のようになります．

表 6.2　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 6.2 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);
```

ソース 6–1 に，図 6.2 のソースコードを示します．このソース 6–1 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

図 6.2　線端の種類

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．このため cairo では，GDK ではできないような柔軟な描画を簡単に実現できるようになっています．

ソース 6–1 線端の種類：cairo-line-cap.c

```
1  #include <gtk/gtk.h>
2
3  gboolean
4  cb_expose_event (GtkWidget      *widget,
5                   GdkEventExpose *event,
6                   gpointer       user_data)
7  {
8    GdkWindow *drawable = widget->window;
9    cairo_t   *cr;
10   double    line_width = 30.0;
11
12   cr = gdk_cairo_create (drawable);
13
14   cairo_set_line_width (cr, line_width);
15
16   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
17   cairo_set_line_cap (cr, CAIRO_LINE_CAP_BUTT);
18   cairo_move_to (cr,  50.0, 50.0);
19   cairo_line_to (cr, 350.0, 50.0);
20   cairo_stroke (cr);
21
22   cairo_set_source_rgb (cr, 0.0, 1.0, 0.0);
23   cairo_set_line_cap (cr, CAIRO_LINE_CAP_ROUND);
24   cairo_move_to (cr,  50.0, 100.0);
25   cairo_line_to (cr, 350.0, 100.0);
26   cairo_stroke (cr);
27
28   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
29   cairo_set_line_cap (cr, CAIRO_LINE_CAP_SQUARE);
30   cairo_move_to (cr,  50.0, 150.0);
31   cairo_line_to (cr, 350.0, 150.0);
32   cairo_stroke (cr);
33
34   cairo_set_line_width (cr, 1);
35   cairo_set_source_rgb (cr, 0.0, 0.0, 0.0);
36
37   cairo_move_to (cr, 50.0,  15.0);
38   cairo_line_to (cr, 50.0, 185.0);
39   cairo_stroke (cr);
40
41   cairo_move_to (cr, 50.0 - line_width / 2.0,  15.0);
42   cairo_line_to (cr, 50.0 - line_width / 2.0, 185.0);
43   cairo_stroke (cr);
44
45   cairo_destroy (cr);
46
47   return FALSE;
48 }
49
50 int
51 main (int argc, char *argv[])
52 {
53   GtkWidget *window;
```

```
54   GtkWidget *canvas;
55
56   gtk_init (&argc, &argv);
57
58   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
59   gtk_window_set_title (GTK_WINDOW (window), "cairo_line_cap␣Sample");
60   gtk_widget_set_size_request (window, 400, 200);
61   g_signal_connect (G_OBJECT (window), "destroy",
62                     G_CALLBACK (gtk_main_quit), NULL);
63
64   canvas = gtk_drawing_area_new ();
65   gtk_container_add (GTK_CONTAINER (window), canvas);
66   g_signal_connect (G_OBJECT (canvas), "expose-event",
67                     G_CALLBACK (cb_expose_event), NULL);
68
69   gtk_widget_show_all (window);
70   gtk_main ();
71
72   return 0;
73 }
```

6.6.3　接続の種類

線分と線分をつなぐ接続の種類には，表 6.3 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 6.3 のようになります．

表 6.3　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．

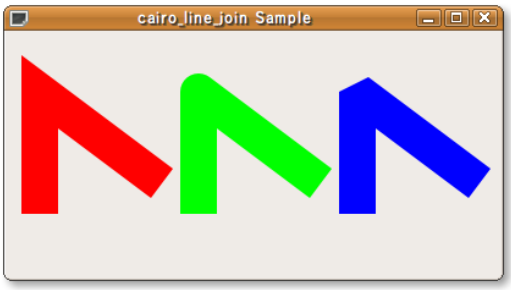


図 6.3　接続の種類

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);
```

ソース 6–2 に，図 6.3 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，6.12 節（p. 103）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 6–2　接続の種類：cairo_line_join.c（一部を抜粋）

```
1 #include <gtk/gtk.h>
2
3 gboolean
4 cb_expose_event (GtkWidget      *widget,
5                   GdkEventExpose *event,
```

```
 6                   gpointer       user_data)
 7 {
 8   GdkWindow *drawable = widget->window;
 9   cairo_t   *cr;
10   cairo_line_join_t join_type[] = {CAIRO_LINE_JOIN_MITER,
11                                    CAIRO_LINE_JOIN_ROUND,
12                                    CAIRO_LINE_JOIN_BEVEL};
13   double color[3][3] = {{1.0, 0.0, 0.0},
14                         {0.0, 1.0, 0.0},
15                         {0.0, 0.0, 1.0}};
16   int n;
17
18   cr = gdk_cairo_create (drawable);
19
20   cairo_set_line_width (cr, 30.0);
21
22   for (n = 0; n < 3; n++)
23     {
24       cairo_set_source_rgb (cr, color[n][0], color[n][1], color[n][2]);
25       cairo_set_line_join (cr, join_type[n]);
26       cairo_move_to (cr,  25.0, 150.0);
27       cairo_line_to (cr,  25.0,  50.0);
28       cairo_line_to (cr, 125.0, 125.0);
29       cairo_stroke (cr);
30       cairo_translate (cr, 130.0, 0.0);
31     }
32   cairo_destroy (cr);
33
34   return FALSE;
35 }
```

6.6.4　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);
```

この関数を使用して破線と鎖線を描画した例を図 6.4 に示します．ソースコードはソース 6–3 です．

図 6.4　線分の種類

関数 cairo_set_dash の使い方

図 6.4 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 6.4 の一番上の破線では，ソース 6–3 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 6.4 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 6–3 線分の種類：cairo_line_dash.c（一部を抜粋）

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

gboolean
cb_expose_event (GtkWidget      *widget,
                 GdkEventExpose *event,
                 gpointer       user_data)
{
  GdkWindow *drawable = widget->window;
  cairo_t   *cr;
  double    dash_pattern1[] = {40.0, 5.0};
  double    dash_pattern2[] = {40.0, 5.0, 5.0, 5.0};

  cr = gdk_cairo_create (drawable);

  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);
  cairo_set_dash (cr, dash_pattern2,
                  sizeof (dash_pattern2) / sizeof (dash_pattern2[0]),
                  -5);
  cairo_move_to (cr,  50.0, 150.0);
  cairo_line_to (cr, 350.0, 150.0);
  cairo_stroke (cr);

  cairo_destroy (cr);

  return FALSE;
}
```

6.7 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);
```

矩形描画の例を図 6.5 に，対応するソースコードをソース 6–4 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．

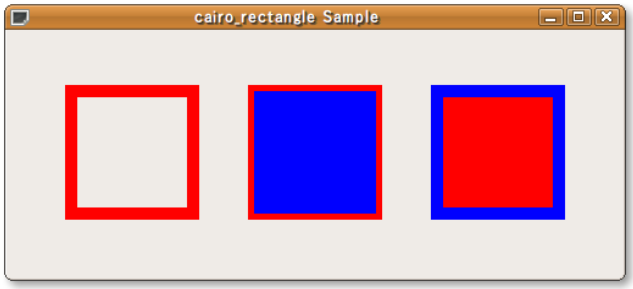


図 6.5　矩形の描画

ソース 6–4 矩形の描画：cairo_rectangle.c（一部を抜粋）

```
 1 #include <gtk/gtk.h>
 2
 3 gboolean
 4 cb_expose_event (GtkWidget      *widget,
 5                  GdkEventExpose *event,
 6                  gpointer       user_data)
 7 {
 8   GdkWindow *drawable = widget->window;
 9   cairo_t   *cr;
10
11   cr = gdk_cairo_create (drawable);
12   cairo_set_line_width (cr, 10.0);
13
14   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
15   cairo_rectangle (cr,  50.0, 50.0, 100.0, 100.0);
16   cairo_stroke (cr);
17
18   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
19   cairo_rectangle (cr, 200.0, 50.0, 100.0, 100.0);
20   cairo_stroke_preserve (cr);
21   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
22   cairo_fill (cr);
23
24   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
25   cairo_rectangle (cr, 350.0, 50.0, 100.0, 100.0);
26   cairo_fill_preserve (cr);
27   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
28   cairo_stroke (cr);
29
30   cairo_destroy (cr);
31
32   return FALSE;
33 }
```

6.8　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 6.6 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，6.5 節（p. 84）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);
```

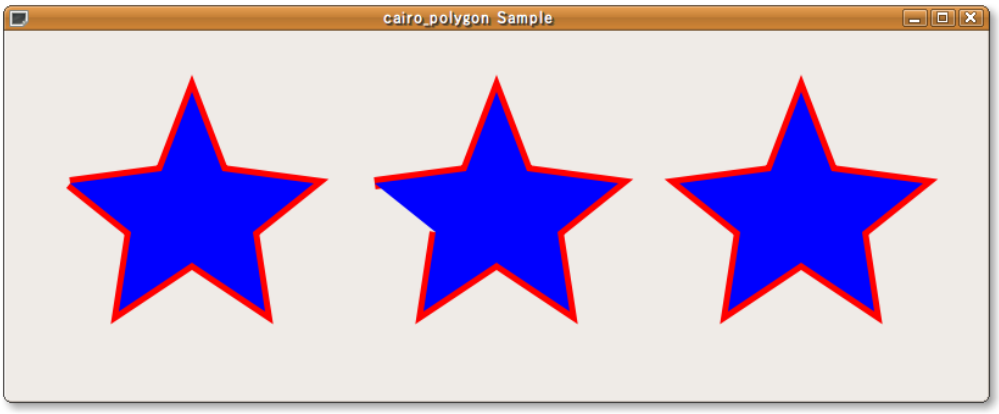


図 6.6　多角形の描画

ソース 6–5 に図 6.6 のソースコードを示します.

ソース 6–5 多角形の描画：cairo_polygon.c（一部を抜粋）

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

gboolean
cb_expose_event (GtkWidget      *widget,
                 GdkEventExpose *event,
                 gpointer       user_data)
{
  GdkWindow *drawable = widget->window;
  cairo_t   *cr;
  double    points[11][2] = {{50.0, 125.0}, {125.0, 115.0},
                             {150.0, 50.0}, {175.0, 115.0},
                             {250.0, 125.0}, {200.0, 165.0},
                             {210.0, 230.0}, {150.0, 190.0},
                             {90.0, 230.0}, {100.0, 165.0},
                             {50.0, 125.0}};
  int       n;

  cr = gdk_cairo_create (drawable);

  cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
  cairo_set_line_width (cr, 10.0);
  for (n = 0; n < 11; n++)
    {
      cairo_line_to (cr, points[n][0], points[n][1]);
    }
  cairo_stroke_preserve (cr);
  cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
  cairo_set_line_width (cr, 1.0);
  cairo_fill (cr);
  cairo_translate (cr, 250.0, 0.0);

  cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
  cairo_set_line_width (cr, 10.0);
  for (n = 0; n < 10; n++)
    {
      cairo_line_to (cr, points[n][0], points[n][1]);
    }
  cairo_stroke_preserve (cr);
  cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
  cairo_set_line_width (cr, 1.0);
  cairo_fill (cr);
  cairo_translate (cr, 250.0, 0.0);

  cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
  cairo_set_line_width (cr, 10.0);
  for (n = 0; n < 10; n++)
    {
      cairo_line_to (cr, points[n][0], points[n][1]);
    }
```

```
50   cairo_close_path (cr);
51   cairo_stroke_preserve (cr);
52   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
53   cairo_set_line_width (cr, 1.0);
54   cairo_fill (cr);
55
56   cairo_destroy (cr);
57
58   return FALSE;
59 }
```

6.9　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);
```

図 6.7 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．

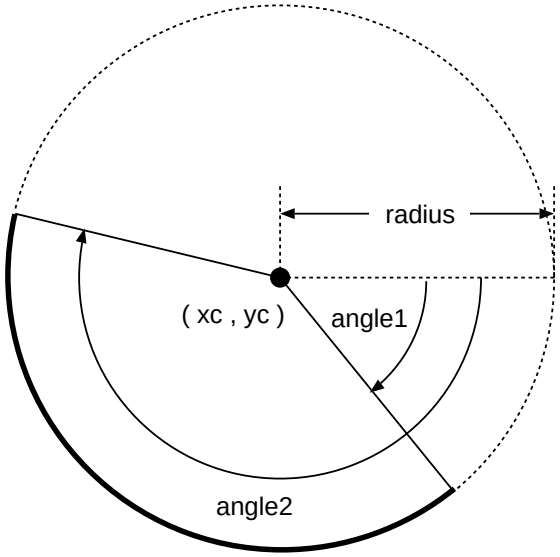


図 6.7　円弧描画のためのパラメータ

円弧の描画例を図 6.8 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 6.8 のソースコードはソース 6–6 です．

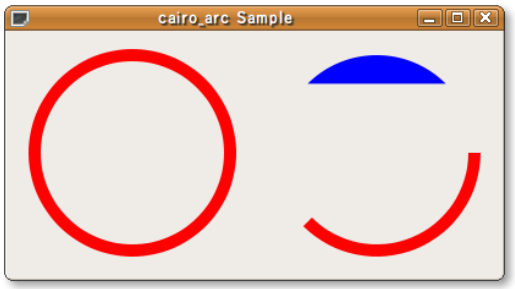


図 6.8　円弧の描画

ソース 6–6 円弧の描画：cairo_arc.c（一部を抜粋）

```
1  #include <gtk/gtk.h>
2  #include <math.h>
3
4  gboolean
5  cb_expose_event (GtkWidget      *widget,
6                   GdkEventExpose *event,
7                   gpointer       user_data)
8  {
9    GdkWindow *drawable = widget->window;
10   cairo_t   *cr;
11
12   cr = gdk_cairo_create (drawable);
13
14   cairo_set_line_width (cr, 10.0);
15   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
16
17   cairo_arc (cr, 100.0, 100.0, 80.0, 0.0, 2.0 * M_PI);
18   cairo_stroke (cr);
19
20   cairo_arc (cr, 300.0, 100.0, 80.0, 0.0, 3.0 * M_PI / 4.0);
21   cairo_stroke (cr);
22
23   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
24
25   cairo_arc (cr, 300.0, 100.0, 80.0,
26              5.0 * M_PI / 4.0, 7.0 * M_PI / 4.0);
27   cairo_fill (cr);
28
29   cairo_destroy (cr);
30
31   return FALSE;
32 }
```

6.10　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);
```

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．

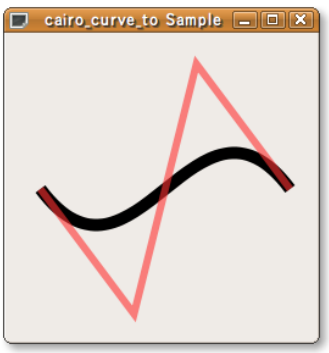


図 6.9　ベジエ曲線の描画

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 6–7 ベジエ曲線の描画：cairo_curve.c（一部を抜粋）

```
 1 #include <gtk/gtk.h>
 2
 3 gboolean
 4 cb_expose_event (GtkWidget      *widget,
 5                  GdkEventExpose *event,
 6                  gpointer       user_data)
 7 {
 8   GdkWindow *drawable = widget->window;
 9   cairo_t   *cr;
10   double    x0 = 25.6,  y0 = 128.0;
11   double    x1 = 102.4, y1 = 230.4;
12   double    x2 = 153.6, y2 = 25.6;
13   double    x3 = 230.4, y3 = 128.0;
14
15   cr = gdk_cairo_create (drawable);
16
17   cairo_set_line_width (cr, 10.0);
18   cairo_set_source_rgb (cr, 0.0, 0.0, 0.0);
19
20   cairo_move_to (cr, x0, y0);
21   cairo_curve_to (cr, x1, y1, x2, y2, x3, y3);
22   cairo_stroke (cr);
23
24   cairo_set_line_width (cr, 6.0);
25   cairo_set_source_rgba (cr, 1.0, 0.2, 0.2, 0.6);
26
27   cairo_move_to (cr, x0, y0);
28   cairo_line_to (cr, x1, y1);
29   cairo_line_to (cr, x2, y2);
30   cairo_line_to (cr, x3, y3);
31   cairo_stroke (cr);
32
33   cairo_destroy (cr);
34
35   return FALSE;
36 }
```

6.11　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

6.11.1　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);

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

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 6.10 に示します．

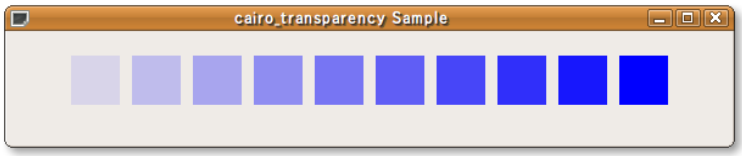


図 6.10　透明度の設定

ソース 6–8 RGBA サーフェスの設定：cairo_transparency.c（一部を抜粋）

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

gboolean
cb_expose_event (GtkWidget      *widget,
                 GdkEventExpose *event,
                 gpointer       user_data)
{
  GdkWindow *drawable = widget->window;
  cairo_t   *cr;
  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);
    }
  cairo_destroy (cr);

  return FALSE;
}
```

6.11.2　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);
```

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);
```

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，6.12 節

(p. 103) 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);

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

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 6.11 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 6–9 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 6–9 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 6.4 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 6.4　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.

図 6.11　線形グラデーションパターン

ソース 6–9 線形グラデーションパターン：cairo_pattern_linear.c（一部を抜粋）

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

gboolean
cb_expose_event (GtkWidget      *widget,
                 GdkEventExpose *event,
                 gpointer       user_data)
{
  GdkWindow *drawable = widget->window;
  cairo_t   *cr;
  cairo_pattern_t *pattern1;
  cairo_pattern_t *pattern2;
  cairo_pattern_t *pattern3;

  cr = gdk_cairo_create (drawable);

  pattern1 = cairo_pattern_create_linear (50.0, 50.0, 350.0, 50.0);
  cairo_pattern_add_color_stop_rgb (pattern1, 0.0, 1.0, 0.0, 0.0);
  cairo_pattern_add_color_stop_rgb (pattern1, 1.0, 1.0, 1.0, 1.0);
  cairo_set_source (cr, pattern1);
  cairo_rectangle (cr, 50.0, 50.0, 300.0, 50.0);
  cairo_fill (cr);

  pattern2 = cairo_pattern_create_linear (50.0, 120.0, 50.0, 170.0);
  cairo_pattern_add_color_stop_rgba (pattern2, 0.0, 0.0, 1.0, 0.0, 1.0);
  cairo_pattern_add_color_stop_rgba (pattern2, 1.0, 0.0, 1.0, 0.0, 0.0);
  cairo_set_source (cr, pattern2);
  cairo_rectangle (cr, 50.0, 120.0, 300.0, 50.0);
  cairo_fill (cr);

  pattern3 = cairo_pattern_create_linear (0, 0.0, 50.0, 50.0);
  cairo_pattern_add_color_stop_rgb (pattern3, 0.0, 0.0, 0.0, 0.0);
  cairo_pattern_add_color_stop_rgb (pattern3, 0.5, 1.0, 1.0, 0.0);
  cairo_pattern_add_color_stop_rgb (pattern3, 1.0, 0.0, 0.0, 0.0);
  cairo_pattern_set_extend (pattern3, CAIRO_EXTEND_REPEAT);
  cairo_set_source (cr, pattern3);
  cairo_translate (cr, 50.0, 190.0);
  cairo_rectangle (cr, 0.0, 0.0, 300.0, 70.0);
  cairo_fill (cr);

```

```
40   cairo_pattern_destroy (pattern1);
41   cairo_pattern_destroy (pattern2);
42   cairo_pattern_destroy (pattern3);
43   cairo_destroy (cr);
44
45   return FALSE;
46 }
```

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 6.12 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 6–10 に示します．

ソース 6–10 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_expose_event (GtkWidget      *widget,
6                  GdkEventExpose *event,
7                  gpointer       user_data)
8 {
9   GdkWindow       *drawable = widget->window;
10  cairo_t         *cr;
11  cairo_pattern_t *pattern;
12
13  cr = gdk_cairo_create (drawable);
14
15  pattern = cairo_pattern_create_radial (160.0, 160.0, 30.0,
16                                         200.0, 200.0, 200.0);
```

図 6.12　放射状グラデーションパターン

```
17   cairo_pattern_add_color_stop_rgb (pattern, 0.5, 1.0, 0.0, 0.0);
18   cairo_pattern_add_color_stop_rgb (pattern, 1.0, 1.0, 1.0, 1.0);
19   cairo_set_source (cr, pattern);
20   cairo_arc (cr, 200.0, 200.0, 160.0, 0.0, 2.0 * M_PI);
21   cairo_fill (cr);
22
23   cairo_pattern_destroy (pattern);
24   cairo_destroy (cr);
25
26   return FALSE;
27 }
```

6.11.3　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 については第 5 章で説明しました）．

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 6.13　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 6.13 に，またそのソースコードをソース 6–11 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_expose_event 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);
```

ソース 6–11 画像ソースの描画：cairo_source_image.c

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

gboolean
cb_expose_event (GtkWidget      *widget,
                 GdkEventExpose *event,
                 gpointer       user_data)
{
  GdkWindow *drawable = widget->window;
  cairo_t   *cr;

  cr = gdk_cairo_create (drawable);

  gdk_cairo_set_source_pixbuf (cr, (GdkPixbuf *) user_data, 0.0, 0.0);
  cairo_paint (cr);

  cairo_destroy (cr);

  return FALSE;
}

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

  if (argc != 2)
    {
      g_print ("%s␣imagefile\n", argv[0]);
      exit (1);
    }
  gtk_init (&argc, &argv);

  pixbuf = gdk_pixbuf_new_from_file (argv[1], NULL);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window),
                        "cairo_source_image␣Sample");
  gtk_widget_set_size_request (window,
                               gdk_pixbuf_get_width (pixbuf),
                               gdk_pixbuf_get_height (pixbuf));
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  canvas = gtk_drawing_area_new ();
  gtk_container_add (GTK_CONTAINER (window), canvas);
  g_signal_connect (G_OBJECT (canvas), "expose-event",
                    G_CALLBACK (cb_expose_event), pixbuf);

  gtk_widget_show_all (window);
  gtk_main ();

  g_object_unref (pixbuf);

  return 0;
}
```

6.11.4　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

図 6.14　画像サーフェスソースの描画

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 6.14 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 6.14 のソースコードをソース 6–12 に示します．

この例では，6.3.2 （p. 82）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_expose_event 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 6.14 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 6–12 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_expose_event (GtkWidget      *widget,
 5                  GdkEventExpose *event,
 6                  gpointer       user_data)
 7 {
 8   GdkWindow       *drawable = widget->window;
 9   cairo_t         *cr;
10   cairo_surface_t *surface;
11
12   cr = gdk_cairo_create (drawable);
13
14   surface = (cairo_surface_t *) user_data;
15   cairo_set_source_surface (cr, surface, 0.0, 0.0);
16   cairo_paint (cr);
17
18   cairo_destroy (cr);
19
20   return FALSE;
21 }
22
23 int
24 main (int argc, char *argv[])
25 {
26   GtkWidget       *window;
27   GtkWidget       *canvas;
28   cairo_surface_t *surface;
```

```
29
30   gtk_init (&argc, &argv);
31
32   surface = cairo_image_surface_create_from_png ("./Parrots.png");
33
34   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
35   gtk_window_set_title (GTK_WINDOW (window),
36                         "cairo_source_image_surface␣Sample");
37   gtk_widget_set_size_request
38     (window,
39      cairo_image_surface_get_width (surface)  * 1.2,
40      cairo_image_surface_get_height (surface) * 1.2);
41   g_signal_connect (G_OBJECT (window), "destroy",
42                     G_CALLBACK (gtk_main_quit), NULL);
43
44   canvas = gtk_drawing_area_new ();
45   gtk_container_add (GTK_CONTAINER (window), canvas);
46   g_signal_connect (G_OBJECT (canvas), "expose-event",
47                     G_CALLBACK (cb_expose_event), surface);
48
49   gtk_widget_show_all (window);
50   gtk_main ();
51
52   cairo_surface_destroy (surface);
53
54   return 0;
55 }
```

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 6.11.2 （p. 95）で紹介した方法が利用できます．

ソース 6–13 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_expose_event (GtkWidget      *widget,
5                   GdkEventExpose *event,
6                   gpointer       user_data)
7  {
8    GdkWindow       *drawable = widget->window;
9    cairo_t         *cr;
10   cairo_surface_t *surface;
11   cairo_pattern_t *pattern;
12   double          points[10][2] = {{ 50.0, 125.0}, {125.0, 115.0},
13                                    {150.0,  50.0}, {175.0, 115.0},
14                                    {250.0, 125.0}, {200.0, 165.0},
15                                    {210.0, 230.0}, {150.0, 190.0},
16                                    { 90.0, 230.0}, {100.0, 165.0}};
17   int n;
18
19   cr = gdk_cairo_create (drawable);
20
21   surface = (cairo_surface_t *) user_data;
22   pattern = cairo_pattern_create_for_surface (surface);
23   cairo_set_source (cr, pattern);
24   cairo_pattern_set_extend (cairo_get_source (cr), CAIRO_EXTEND_REPEAT);
25
26   for (n = 0; n < 10; n++)
27     {
28       cairo_line_to (cr, points[n][0], points[n][1]);
29     }
30   cairo_close_path (cr);
31   cairo_fill (cr);
32
33   cairo_pattern_destroy (pattern);
34   cairo_destroy (cr);
```

図 6.15　パターン化した画像サーフェスソースの描画

```
35
36    return FALSE;
37 }
```

6.12　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);

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

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

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);
```

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 6.16（ソース 6–14）を例に，それぞれの関数の使い方を説明します．図 6.16 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．

1. 平行移動（図 6.16 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 6.16 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 6.16 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 6.8 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 6.16 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．

[*1] Radian: 角度の単位．360 度が 2π ラジアンとなる．

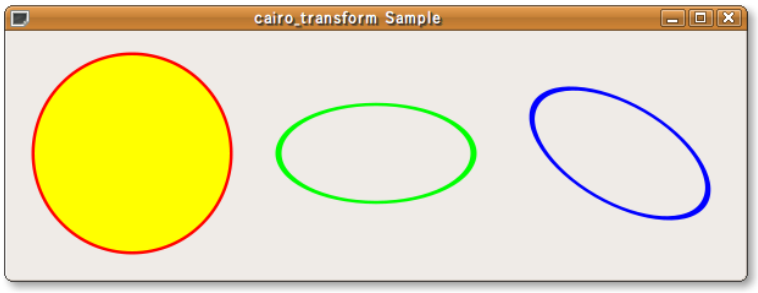


図 6.16　座標系の変換

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

cairo_status_t cairo_matrix_invert (cairo_matrix_t *matrix);
```

ソース 6–14 座標系の変換：cairo_transform.c（一部を抜粋）

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

gboolean
cb_expose_event (GtkWidget      *widget,
                 GdkEventExpose *event,
                 gpointer       user_data)
{
  GdkWindow      *drawable = widget->window;
  cairo_t        *cr;
  cairo_matrix_t mat;

  cr = gdk_cairo_create (drawable);

  cairo_set_line_width (cr, 5.0);

  cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
  cairo_translate (cr, 100.0, 100.0);
  cairo_arc (cr, 0.0, 0.0, 80.0, 0.0, 2.0 * M_PI);
  cairo_stroke (cr);

  cairo_set_source_rgb (cr, 0.0, 1.0, 0.0);
  cairo_translate (cr, 200.0, 0.0);
  cairo_scale (cr, 1.0, 0.5);
  cairo_arc (cr, 0.0, 0.0, 80.0, 0.0, 2.0 * M_PI);
  cairo_stroke (cr);

  cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
  cairo_translate (cr, 200.0, 0.0);
  cairo_scale (cr, 1.0, 2.0);
  cairo_rotate (cr, M_PI / 6.0);
  cairo_scale (cr, 1.0, 0.5);
  cairo_arc (cr, 0.0, 0.0, 80.0, 0.0, 2.0 * M_PI);
  cairo_stroke (cr);

  cairo_get_matrix (cr, &mat);
  cairo_matrix_invert (&mat);
  cairo_transform (cr, &mat);

  cairo_set_source_rgb (cr, 1.0, 1.0, 0.0);
  cairo_arc (cr, 100.0, 100.0, 80.0, 0.0, 2.0 * M_PI);
  cairo_fill (cr);

  cairo_destroy (cr);

  return FALSE;
}
```

6.13　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 6.17 （ソース 6–15）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);
```

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 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 6.5 と表 6.6 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 6.5　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

表 6.6　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

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);
```

ソース 6–15　テキストの描画：cairo_text.c（一部を抜粋）

```
1 #include <gtk/gtk.h>
2
3 gboolean
4 cb_expose_event (GtkWidget      *widget,
5                  GdkEventExpose *event,
6                  gpointer       user_data)
7 {
8   GdkWindow *drawable = widget->window;
```

図 6.17　テキストの描画

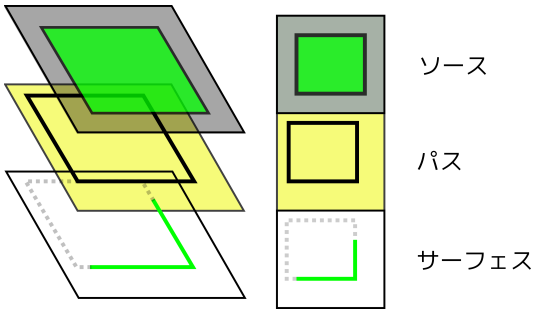


図 6.18　クリッピングの概念図

```
 9   cairo_t   *cr;
10
11   cr = gdk_cairo_create (drawable);
12
13   cairo_set_source_rgb (cr, 1.0, 0.0, 0.0);
14   cairo_select_font_face (cr, "FreeSans",
15                           CAIRO_FONT_SLANT_ITALIC,
16                           CAIRO_FONT_WEIGHT_NORMAL);
17   cairo_set_font_size (cr, 24.0);
18   cairo_move_to (cr, 50.0, 50.0);
19   cairo_show_text (cr, "This␣is␣a␣text␣drawing␣sample.");
20
21   cairo_set_source_rgb (cr, 0.0, 1.0, 0.0);
22
23   cairo_select_font_face (cr, "VL␣ゴシック",
24                           CAIRO_FONT_SLANT_NORMAL,
25                           CAIRO_FONT_WEIGHT_NORMAL);
26   cairo_move_to (cr, 50.0, 100.0);
27   cairo_show_text (cr, "これはテキスト描画のサンプルです。");
28
29   cairo_set_source_rgb (cr, 0.0, 0.0, 1.0);
30   cairo_select_font_face (cr, "さざなみ明朝",
31                           CAIRO_FONT_SLANT_NORMAL,
32                           CAIRO_FONT_WEIGHT_BOLD);
33   cairo_move_to (cr, 50.0, 150.0);
34   cairo_show_text (cr, "これはテキスト描画のサンプルです。");
35
36   cairo_destroy (cr);
37
38   return FALSE;
39 }
```

6.14　クリッピング

クリッピングの概念図を，図 6.18 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 6.18 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 6.19 （ソース 6–16）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．

ソース 6–16 ソース領域のクリッピング：cairo_clipping.c（一部を抜粋）

```
1 #include <gtk/gtk.h>
2 #include <math.h>
3
4 gboolean
5 cb_expose_event (GtkWidget      *widget,
6                  GdkEventExpose *event,
7                  gpointer       user_data)
```

図 6.19 クリッピング

```
 8 {
 9   GdkWindow       *drawable = widget->window;
10   cairo_t         *cr;
11   cairo_surface_t *surface;
12   double          x, y, r;
13
14   cr = gdk_cairo_create (drawable);
15
16   surface = (cairo_surface_t *) user_data;
17   cairo_set_source_surface (cr, surface, 0.0, 0.0);
18
19   x = cairo_image_surface_get_width (surface) / 2.0;
20   y = cairo_image_surface_get_height (surface) / 2.0;
21   r = x < y ? x : y;
22   cairo_arc (cr, x, y, r, 0.0, 2.0 * M_PI);
23   cairo_clip (cr);
24   cairo_paint (cr);
25
26   cairo_destroy (cr);
27
28   return FALSE;
29 }
```

6.15 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);
```

6.11.2 （p. 98）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 6.20 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 6–17 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 6.20　マスク

ソース 6–17 ソース領域のマスク：cairo_mask.c（一部を抜粋）

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

gboolean
cb_expose_event (GtkWidget      *widget,
                 GdkEventExpose *event,
                 gpointer       user_data)
{
  GdkWindow       *drawable = widget->window;
  cairo_t         *cr;
  cairo_pattern_t *pattern;
  cairo_surface_t *surface;
  double          x, y, r;

  cr = gdk_cairo_create (drawable);

  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;

  pattern = cairo_pattern_create_radial (x, y, 0.2 * r, x, y, r);
  cairo_pattern_add_color_stop_rgba (pattern, 0.0, 1.0, 1.0, 1.0, 1.0);
  cairo_pattern_add_color_stop_rgba (pattern, 1.0, 1.0, 1.0, 1.0, 0.0);

  cairo_mask (cr, pattern);

  cairo_pattern_destroy (pattern);
  cairo_destroy (cr);

  return FALSE;
}
```

6.16　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 6.7 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 6.21 に示します．ソースコードはソース 6–18 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);
```

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 6.7 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 6–18 図形の合成：cairo_composite.c（一部を抜粋）

```
1  #include <gtk/gtk.h>
2
3  static void
4  draw (cairo_t          *cr,
5        double           x,
6        double           y,
7        double           w,
8        double           h,
9        cairo_operator_t op,
10       gchar            *operator_str) {
11   cairo_t         *source_cr, *dest_cr;
```

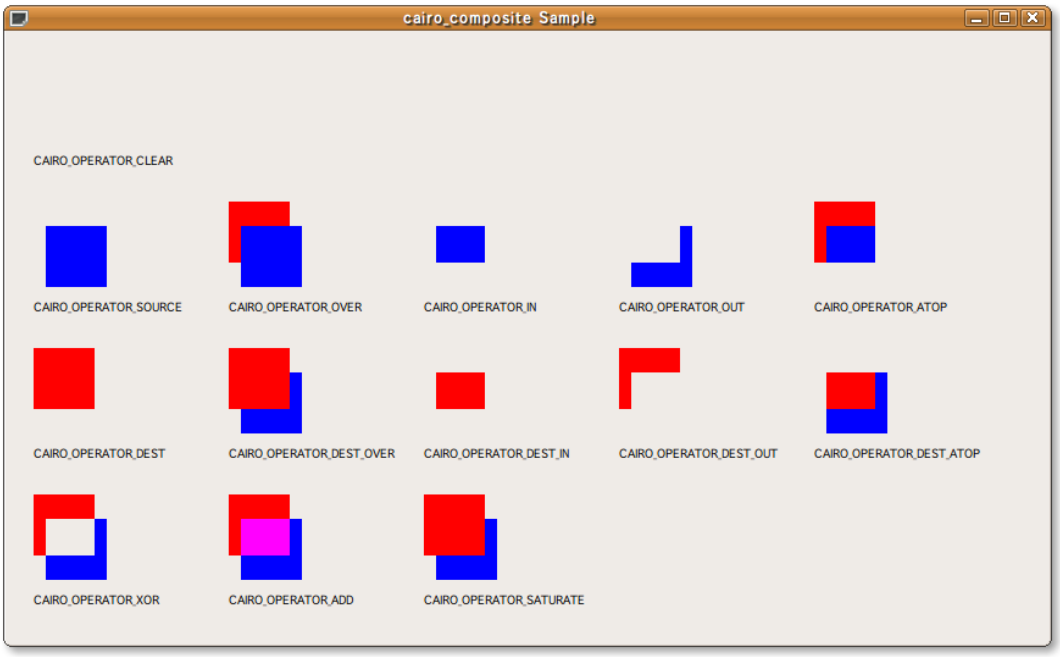


図 6.21　図形の合成

```
12   cairo_surface_t *source_surface, *dest_surface;
13
14   source_surface =
15     cairo_surface_create_similar (cairo_get_target (cr),
16                                   CAIRO_CONTENT_COLOR_ALPHA, w, h);
17   dest_surface =
18     cairo_surface_create_similar (cairo_get_target (cr),
19                                   CAIRO_CONTENT_COLOR_ALPHA, w, h);
20   source_cr = cairo_create (source_surface);
21   cairo_set_source_rgb (source_cr, 0.0, 0.0, 1.0);
22   cairo_rectangle (source_cr, x + 10.0, y + 20.0, 50.0, 50.0);
23   cairo_fill (source_cr);
24
25   dest_cr  = cairo_create (dest_surface);
26   cairo_set_source_rgb (dest_cr, 1.0, 0.0, 0.0);
27   cairo_rectangle (dest_cr, x, y, 50.0, 50.0);
28   cairo_fill (dest_cr);
29
30   cairo_set_operator (dest_cr, op);
31   cairo_set_source_surface (dest_cr, source_surface, 0.0, 0.0);
32   cairo_paint (dest_cr);
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_expose_event (GtkWidget      *widget,
47                  GdkEventExpose *event,
48                  gpointer       user_data)
49 {
50   GdkWindow *drawable = widget->window;
51   cairo_t   *cr;
52   cairo_operator_t operator[] = {CAIRO_OPERATOR_CLEAR,
53                                  CAIRO_OPERATOR_SOURCE,
54                                  CAIRO_OPERATOR_OVER,
55                                  CAIRO_OPERATOR_IN,
```

```
56                                  CAIRO_OPERATOR_OUT,
57                                  CAIRO_OPERATOR_ATOP,
58                                  CAIRO_OPERATOR_DEST,
59                                  CAIRO_OPERATOR_DEST_OVER,
60                                  CAIRO_OPERATOR_DEST_IN,
61                                  CAIRO_OPERATOR_DEST_OUT,
62                                  CAIRO_OPERATOR_DEST_ATOP,
63                                  CAIRO_OPERATOR_XOR,
64                                  CAIRO_OPERATOR_ADD,
65                                  CAIRO_OPERATOR_SATURATE};
66   gchar *operator_strs[] = {"CAIRO_OPERATOR_CLEAR",
67                             "CAIRO_OPERATOR_SOURCE",
68                             "CAIRO_OPERATOR_OVER",
69                             "CAIRO_OPERATOR_IN",
70                             "CAIRO_OPERATOR_OUT",
71                             "CAIRO_OPERATOR_ATOP",
72                             "CAIRO_OPERATOR_DEST",
73                             "CAIRO_OPERATOR_DEST_OVER",
74                             "CAIRO_OPERATOR_DEST_IN",
75                             "CAIRO_OPERATOR_DEST_OUT",
76                             "CAIRO_OPERATOR_DEST_ATOP",
77                             "CAIRO_OPERATOR_XOR",
78                             "CAIRO_OPERATOR_ADD",
79                             "CAIRO_OPERATOR_SATURATE"};
80   gint x,y, w, h, n;
81
82   cr = gdk_cairo_create (drawable);
83
84   gdk_window_get_geometry (drawable, NULL, NULL, &w, &h, NULL);
85
86   x = 20;
87   y = 20;
88   draw (cr, x, y, w, h, operator[0], operator_strs[0]);
89
90   for (x = 20, y = 140, n = 1; n <= 5; x += 160, n++)
91     {
92       draw (cr, x, y, w, h, operator[n], operator_strs[n]);
93     }
94   for (x = 20, y = 260, n = 6; n <= 10; x += 160, n++)
95     {
96       draw (cr, x, y, w, h, operator[n], operator_strs[n]);
97     }
98   for (x = 20, y = 380, n = 11; n < 14; x += 160, n++)
99     {
100      draw (cr, x, y, w, h, operator[n], operator_strs[n]);
101    }
102  cairo_destroy (cr);
103
104  return FALSE;
105 }
```

第 7 章

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，次の 3 つの種類が存在します．

- 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．

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----GtkWidget
                      +----GtkContainer
                            +----GtkBin
                                  +----GtkButton
```

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．

図 7.1　普通のボタン

表 7.1　ボタンウィジェットのシグナル

シグナル	説明
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.
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.

- 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_stock
  画像入りのボタン（図 7.1 下段）を作成する関数です．画像データはストックアイテム（GtkStockItem）から取得します．ストックアイテムには決められた文字列が ID（例えば，GTK_STOCK_OK や GTK_STOCK_OPEN）として登録してあり，この ID を関数の引数に指定します.

```
GtkWidget* gtk_button_new_from_stock (const gchar *stock_id);
```

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);
```

ウィジェットのプロパティ設定

ボタンのプロパティとしてよく使われるのはラベルです．ボタンのラベルを取得したり，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);
```

- 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.

図 7.2　ボタンウィジェットのサンプルプログラム

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
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_vbox_new (TRUE, 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．

オブジェクトの階層構造

```
GObject
  +----GInitiallyUnowned
         +----GtkObject
                +----GtkWidget
                       +----GtkContainer
                              +----GtkBin
                                     +----GtkButton
                                            +----GtkToggleButton
                                                   +----GtkCheckButton
```

表 7.2　チェック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.

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);
```

シグナルとコールバック関数

表 7.2 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.

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);
```

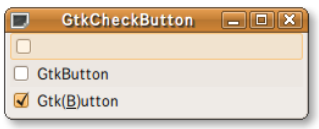


図 7.3　チェックボタン

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

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

static void
cb_button_toggled (GtkToggleButton *widget,
                   gpointer         user_data)
{
  gboolean active;
  gchar    *str[] = {"FALSE", "TRUE"};

  active = gtk_toggle_button_get_active (widget);
  g_print ("Check␣button␣state␣:␣%s\n", str[active]);
}

int
main (int argc, char **argv)
{
  GtkWidget *window;
  GtkWidget *box;
  GtkWidget *button;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "GtkCheckButton␣Sample2");
  gtk_widget_set_size_request (window, 300, -1);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  box = gtk_vbox_new (TRUE, 0);
  gtk_container_add (GTK_CONTAINER (window), box);

  button = gtk_check_button_new_with_label ("Please␣click␣me.");
  gtk_box_pack_start (GTK_BOX (box), button, TRUE, TRUE, 0);
  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button), TRUE);
  g_signal_connect (G_OBJECT (button), "toggled",
                    G_CALLBACK (cb_button_toggled), NULL);

  gtk_widget_show_all (window);
  gtk_main ();

  return 0;
}
```

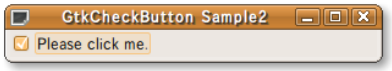


図 7.4　チェックボタンウィジェットのサンプルプログラム

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．

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----GtkWidget
                      +----GtkContainer
                            +----GtkBin
                                  +----GtkButton
                                        +----GtkToggleButton
                                              +----GtkCheckButton
                                                    +----GtkRadioButton
```

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*
```

図 7.5　ラジオボタン

表 7.3　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.

```
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]));
```

シグナルとコールバック関数

表 7.3 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．

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．

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

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

static void
cb_button_toggled (GtkRadioButton *widget, gpointer user_data)
{
  g_print ("Catch␣the␣toggled␣signal␣from␣the␣radio␣button%d.\n",
           GPOINTER_TO_INT (user_data));
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (widget)))
    {
      g_print ("The␣radio␣button%d␣is␣selected.\n",
               GPOINTER_TO_INT (user_data));
    }
}

int
main (int argc, char **argv)
{
  GtkWidget *window;
  GtkWidget *box;
  GtkWidget *button[3];
  GSList    *group = NULL;

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "GtkRadioButton␣Sample2");
  gtk_widget_set_size_request (window, 300, -1);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  box = gtk_vbox_new (TRUE, 0);
  gtk_container_add (GTK_CONTAINER (window), box);

  button[0] = gtk_radio_button_new_with_label (group, "Red");
  gtk_box_pack_start (GTK_BOX (box), button[0], TRUE, TRUE, 0);

  button[1] = gtk_radio_button_new_with_label_from_widget
    (GTK_RADIO_BUTTON (button[0]), "Green");
  gtk_box_pack_start (GTK_BOX (box), button[1], TRUE, TRUE, 0);

  button[2] = gtk_radio_button_new_with_label_from_widget
    (GTK_RADIO_BUTTON (button[0]), "Blue");
  gtk_box_pack_start (GTK_BOX (box), button[2], TRUE, TRUE, 0);

  gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (button[1]), TRUE);

  g_signal_connect (G_OBJECT (button[0]), "toggled",
                    G_CALLBACK (cb_button_toggled), GINT_TO_POINTER (0));
  g_signal_connect (G_OBJECT (button[1]), "toggled",
                    G_CALLBACK (cb_button_toggled), GINT_TO_POINTER (1));
  g_signal_connect (G_OBJECT (button[2]), "toggled",
                    G_CALLBACK (cb_button_toggled), GINT_TO_POINTER (2));

  gtk_widget_show_all (window);
  gtk_main ();

  return 0;
}
```

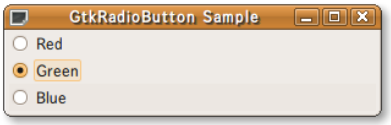


図 7.6　ラジオボタンウィジェットのサンプルプログラム

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
          +----GtkObject
                +----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 7.4 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 7.4 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．

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----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);
```

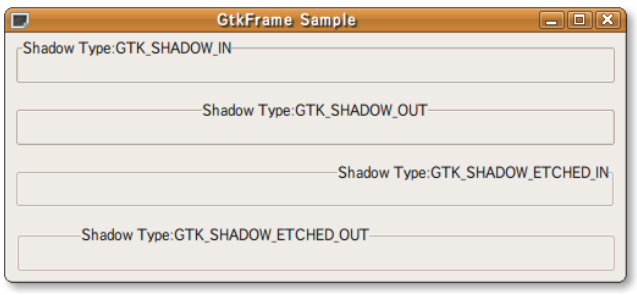


図 7.7 フレーム表示の例

```
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 5.5（p. 75）で紹介した GtkShadowType で指定します．フレームの装飾の種類は図 7.7 を参考にしてください（テーマによってはフレームの装飾が変化しない場合もあります）．

```
GtkShadowType gtk_frame_get_shadow_type (GtkFrame *frame);

void gtk_frame_set_shadow_type (GtkFrame       *frame,
                                GtkShadowType type);
```

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
          +----GtkObject
                +----GtkWidget
                      +----GtkContainer
                            +----GtkPaned
                                  +----GtkHPaned
```

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 GtkHPaned と，領域を縦に 2 つに分割する GtkVPaned の 2 種類があります．それぞれのペインは次の関数 gtk_hpaned_new と gtk_vpaned_new で作成できます．

図 7.8　ペインによる領域の分割

```
GtkWidget* gtk_hpaned_new (void);

GtkWidget* gtk_vpaned_new (void);
```

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);
```

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
           +----GtkObject
                  +----GtkWidget
                         +----GtkContainer
                                +----GtkToolbar
```

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);
```

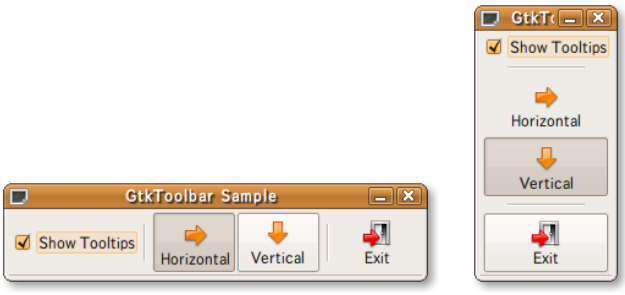


図 7.9 ツールバー

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);
```

この関数は独自のアイコンを使う場合に使用し，使いたいアイコンがストックアイテムにある場合は関数 gtk_tool_button_new_from_stock を使用すればよいでしょう．

```
GtkToolItem* gtk_tool_button_new_from_stock (const gchar *stock_id);
```

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 もしくは gtk_toggle_tool_button_new_from_stock を使用します．

```
GtkToolItem* gtk_toggle_tool_button_new (void);

GtkToolItem*
gtk_toggle_tool_button_new_from_stock (const gchar *stock_id);
```

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 もしくは gtk_radio_tool_button_new_from_stock を使用します．

```
GtkToolItem* gtk_radio_tool_button_new (GSList *group);

GtkToolItem*
gtk_radio_tool_button_new_from_stock (GSList      *group,
                                      const gchar *stock_id);
```

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_raido_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);
```

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 7.5 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．

```
typedef enum
{
  GTK_ORIENTATION_HORIZONTAL,
  GTK_ORIENTATION_VERTICAL
} GtkOrientation;
```

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);
```

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．

```
typedef enum
{
  GTK_TOOLBAR_ICONS,
  GTK_TOOLBAR_TEXT,
  GTK_TOOLBAR_BOTH,
  GTK_TOOLBAR_BOTH_HORIZ
} GtkToolbarStyle;
```

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);
```

表 7.5 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．

```
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 で定義された値で扱います．

```
void gtk_toolbar_set_icon_size (GtkToolbar  *toolbar,
                                GtkIconSize icon_size);

GtkIconSize gtk_toolbar_get_icon_size (GtkToolbar *toolbar);
```

GtkIconSize は次のように定義されています．

```
typedef enum
 {
   GTK_ICON_SIZE_INVALID,
   GTK_ICON_SIZE_MENU,
   GTK_ICON_SIZE_SMALL_TOOLBAR,
   GTK_ICON_SIZE_LARGE_TOOLBAR,
   GTK_ICON_SIZE_BUTTON,
   GTK_ICON_SIZE_DND,
   GTK_ICON_SIZE_DIALOG
 } GtkIconSize;
```

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. 125）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), 100, 300);
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    GSList      *group;
56
57    gtk_init (&argc, &argv);
58
59    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
60    gtk_window_set_title (GTK_WINDOW (window), "GtkToolbar␣Sample");
61    gtk_window_set_resizable (GTK_WINDOW (window), FALSE);
62    g_signal_connect (G_OBJECT (window), "destroy",
63                      G_CALLBACK (gtk_main_quit), NULL);
64    gtk_widget_set_size_request (window, 400, -1);
65
66    toolbar = gtk_toolbar_new ();
67    gtk_container_add (GTK_CONTAINER (window), toolbar);
68    gtk_toolbar_set_style (GTK_TOOLBAR (toolbar), GTK_TOOLBAR_BOTH);
69    gtk_toolbar_set_orientation (GTK_TOOLBAR (toolbar),
70                                 GTK_ORIENTATION_HORIZONTAL);
71    g_object_set_data (G_OBJECT (window), "toolbar", (gpointer) toolbar);
72
73    item = gtk_tool_item_new ();
74    widget = gtk_check_button_new_with_label ("Show␣text");
75    gtk_container_add (GTK_CONTAINER (item), widget);
76    gtk_tool_item_set_tooltip_text (item, "Toggle␣whether␣show␣text");
77    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (widget), TRUE);
78    g_signal_connect (G_OBJECT (widget), "toggled",
79                      G_CALLBACK (cb_show_text), (gpointer) toolbar);
80    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
81
82    item = gtk_separator_tool_item_new ();
83    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
84
85    item
86      = gtk_radio_tool_button_new_from_stock (NULL, GTK_STOCK_GO_FORWARD);
87    gtk_tool_button_set_label (GTK_TOOL_BUTTON (item), "Horizontal");
88    gtk_toggle_tool_button_set_active (GTK_TOGGLE_TOOL_BUTTON (item),
89                                       TRUE);
90    group = gtk_radio_tool_button_get_group (GTK_RADIO_TOOL_BUTTON (item));
91    gtk_tool_item_set_tooltip_text (item, "Set␣the␣toolbar␣to␣horizontal");
92    g_signal_connect (G_OBJECT (item), "toggled",
93                      G_CALLBACK (cb_set_horizontal), (gpointer) window);
94    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
95
96    item
97      = gtk_radio_tool_button_new_from_stock (group, GTK_STOCK_GO_DOWN);
98    gtk_tool_button_set_label (GTK_TOOL_BUTTON (item), "Vertical");
99    gtk_tool_item_set_tooltip_text (item, "Set␣the␣toolbar␣to␣vertical");
100   g_signal_connect (G_OBJECT (item), "toggled",
101                     G_CALLBACK (cb_set_vertical), (gpointer) window);
102   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
103
104   item = gtk_separator_tool_item_new ();
105   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
106
107   widget = gtk_image_new_from_stock (GTK_STOCK_QUIT,
108                                      gtk_toolbar_get_icon_size
109                                      (GTK_TOOLBAR (toolbar)));
110   item = gtk_tool_button_new (widget, "Quit");
111   gtk_tool_item_set_tooltip_text (item, "Exit␣this␣program");
112   g_signal_connect_swapped (G_OBJECT (item), "clicked",
113                             G_CALLBACK (gtk_main_quit),
114                             (gpointer) window);
```

```
115   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
116
117   gtk_widget_show_all (window);
118   gtk_main ();
119
120   return 0;
121 }
```

7.2.5　ハンドルボックス

ハンドルボックスウィジェット（GtkHandleBox）は，その中に 1 つのウィジェットを配置できるウィジェットで，ハンドルウィジェットを配置した親ウィジェットから取り外して扱うことが可能です．図 7.10 上段はウィンドウウィジェットに配置したハンドルボックスで，図 7.10 下段はウィンドウウィジェットから取り外した状態です．

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----GtkWidget
                      +----GtkContainer
                            +----GtkBin
                                  +----GtkHandleBox
```

ウィジェットの作成

ハンドルボックスを作成する関数は gtk_handle_box_new です．

```
GtkWidget* gtk_handle_box_new (void);
```

子ウィジェットの配置

ハンドルボックス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．

シグナルとコールバック関数

表 7.6 にハンドルボックスウィジェットのシグナルを示します．
2 つ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 (GtkHandleBox *handlebox,
                    GtkWidget    *widget, gpointer user_data);
```

ウィジェットのプロパティ設定

ハンドルックスのプロパティには次の 2 つの項目が存在します．

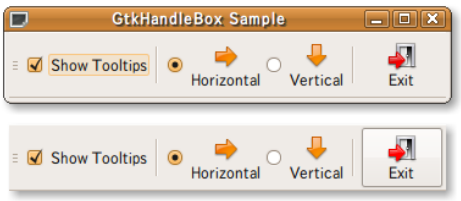


図 7.10　ハンドルボックス

表 7.6　ハンドルボックスウィジェットのシグナル

シグナル	説明
child-attached	ハンドルボックスの中身が再びハンドルボックスのもとの位置に戻ったときに発生するシグナル．
child-detached	ハンドルボックスの中身がハンドルボックスのもとの位置から取り外されたときに発生するシグナル．

- フレームの装飾
  フレームの装飾の種類は表 5.5（p. 75）に紹介した GtkShadowType で指定します．

```
void gtk_handle_box_set_shadow_type (GtkHandleBox  *handle_box,
                                     GtkShadowType type);

GtkShadowType
gtk_handle_box_get_shadow_type (GtkHandleBox *handle_box);
```

- ハンドルの位置
  以下の関数を使ってハンドルの位置を設定，取得します．

```
void
gtk_handle_box_set_handle_position (GtkHandleBox *handle_box,
                                    GtkPositionType position);

GtkPositionType
gtk_handle_box_get_handle_position (GtkHandleBox *handle_box);
```

サンプルプログラム

ハンドルボックスウィジェットのサンプルプログラムをソース 7–2–2 に示します．これはソース 7–2–1 で作成したツールバーをハンドルボックスに配置して取り外せるようにしたものです．

ソース 7-2-2 ハンドルボックスウィジェットのサンプルプログラム

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

static void
cb_show_text (GtkWidget *widget, gpointer user_data)
{
  if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (widget)))
    {
      gtk_toolbar_set_style (GTK_TOOLBAR (user_data), GTK_TOOLBAR_BOTH);
    }
  else
    {
      gtk_toolbar_set_style (GTK_TOOLBAR (user_data), GTK_TOOLBAR_ICONS);
    }
}

static void
cb_set_horizontal (GtkToggleToolButton *widget, gpointer user_data)
{
  if (gtk_toggle_tool_button_get_active (widget))
    {
      GtkOrientable *orientable;

      orientable
        = GTK_ORIENTABLE (g_object_get_data (G_OBJECT (user_data),
                                             "toolbar"));
      gtk_orientable_set_orientation (orientable,
                                      GTK_ORIENTATION_HORIZONTAL);
      gtk_widget_set_size_request (GTK_WIDGET (user_data), 400, -1);
    }
}

static void
cb_set_vertical (GtkToggleToolButton *widget, gpointer user_data)
{
  if (gtk_toggle_tool_button_get_active (widget))
    {
      GtkOrientable *orientable;

      orientable
        = GTK_ORIENTABLE (g_object_get_data (G_OBJECT (user_data),
                                             "toolbar"));
      gtk_orientable_set_orientation (orientable,
                                      GTK_ORIENTATION_VERTICAL);
```

```
44        gtk_widget_set_size_request (GTK_WIDGET (user_data), 100, 300);
45      }
46  }
47
48  int
49  main (int argc, char **argv)
50  {
51    GtkWidget   *window;
52    GtkWidget   *handlebox;
53    GtkWidget   *toolbar;
54    GtkWidget   *widget;
55    GtkToolItem *item;
56    GSList      *group;
57
58    gtk_init (&argc, &argv);
59
60    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
61    gtk_window_set_title (GTK_WINDOW (window), "GtkHandleBox␣Sample");
62    gtk_window_set_resizable (GTK_WINDOW (window), TRUE);
63    g_signal_connect (G_OBJECT (window), "destroy",
64                      G_CALLBACK (gtk_main_quit), NULL);
65    gtk_widget_set_size_request (window, 400, -1);
66
67    handlebox = gtk_handle_box_new ();
68    gtk_handle_box_set_handle_position (GTK_HANDLE_BOX (handlebox),
69                                        GTK_POS_LEFT);
70    gtk_container_add (GTK_CONTAINER (window), handlebox);
71
72    toolbar = gtk_toolbar_new ();
73    gtk_container_add (GTK_CONTAINER (handlebox), toolbar);
74    gtk_toolbar_set_style (GTK_TOOLBAR (toolbar), GTK_TOOLBAR_BOTH);
75    gtk_toolbar_set_orientation (GTK_TOOLBAR (toolbar),
76                                 GTK_ORIENTATION_HORIZONTAL);
77    g_object_set_data (G_OBJECT (window), "toolbar", (gpointer) toolbar);
78
79    widget = gtk_check_button_new_with_label ("Show␣text");
80    item = gtk_tool_item_new ();
81    gtk_container_add (GTK_CONTAINER (item), widget);
82    gtk_tool_item_set_tooltip_text (item, "Toggle␣whether␣show␣text");
83    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (widget), TRUE);
84    g_signal_connect (G_OBJECT (widget), "toggled",
85                      G_CALLBACK (cb_show_text), (gpointer) toolbar);
86    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
87
88    item = gtk_separator_tool_item_new ();
89    gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
90
91    item
92      = gtk_radio_tool_button_new_from_stock (NULL, GTK_STOCK_GO_FORWARD);
93    gtk_tool_button_set_label (GTK_TOOL_BUTTON (item), "Horizontal");
94    gtk_toggle_tool_button_set_active (GTK_TOGGLE_TOOL_BUTTON (item),
95                                       TRUE);
96    group = gtk_radio_tool_button_get_group (GTK_RADIO_TOOL_BUTTON (item));
97    gtk_tool_item_set_tooltip_text (item, "Set␣the␣toolbar␣to␣horizontal");
98    g_signal_connect (G_OBJECT (item), "toggled",
99                      G_CALLBACK (cb_set_horizontal), (gpointer) window);
100   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
101
102   item
103     = gtk_radio_tool_button_new_from_stock (group, GTK_STOCK_GO_DOWN);
104   gtk_tool_button_set_label (GTK_TOOL_BUTTON (item), "Vertical");
105   gtk_tool_item_set_tooltip_text (item, "Set␣the␣toolbar␣to␣vertical");
106   g_signal_connect (G_OBJECT (item), "toggled",
107                     G_CALLBACK (cb_set_vertical), (gpointer) window);
108   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
109
110   item = gtk_separator_tool_item_new ();
111   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
112
113   widget = gtk_image_new_from_stock (GTK_STOCK_QUIT,
114                                      gtk_toolbar_get_icon_size
115                                      (GTK_TOOLBAR (toolbar)));
116   item = gtk_tool_button_new (widget, "Quit");
117   gtk_tool_item_set_tooltip_text (item, "Exit␣this␣program");
118   g_signal_connect_swapped (G_OBJECT (item), "clicked",
119                             G_CALLBACK (gtk_main_quit),
```

```
120                                (gpointer) window);
121   gtk_toolbar_insert (GTK_TOOLBAR (toolbar), item, -1);
122
123   gtk_widget_show_all (window);
124   gtk_main ();
125
126   return 0;
127 }
```

7.2.6 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 7.11）です．

オブジェクトの階層構造

```
GObject
   +----GInitiallyUnowned
          +----GtkObject
                 +----GtkWidget
                        +----GtkContainer
                               +----GtkNotebook
```

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．

```
gint gtk_notebook_append_page (GtkNotebook *notebook,
                               GtkWidget   *child,
                               GtkWidget   *tab_label);
```

  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．

- 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);
```

図 7.11 ノートブックウィジェット

表 7.7　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.

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);
```

シグナルとコールバック関数

表 7.7 にノートブックウィジェットの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,
                    GtkNotebookPage *page,
                    guint            page_num,
                    gpointer         user_data);
```

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 (GtkWidget       *widget,
25              GtkNotebookPage *page,
26              gint            page_num)
27 {
28   GtkNotebook *notebook = GTK_NOTEBOOK (widget);
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 data)
42 {
43   gtk_notebook_set_tab_pos (GTK_NOTEBOOK (data), GTK_POS_LEFT);
44 }
45
46 static void
47 cb_tab_position_right (GtkWidget *widget, gpointer data)
48 {
49   gtk_notebook_set_tab_pos (GTK_NOTEBOOK (data), GTK_POS_RIGHT);
```

```
50 }
51
52 static void
53 cb_tab_position_top (GtkWidget *widget, gpointer data)
54 {
55   gtk_notebook_set_tab_pos (GTK_NOTEBOOK (data), GTK_POS_TOP);
56 }
57
58 static void
59 cb_tab_position_bottom (GtkWidget *widget, gpointer data)
60 {
61   gtk_notebook_set_tab_pos (GTK_NOTEBOOK (data), GTK_POS_BOTTOM);
62 }
63
64 static GtkWidget*
65 icon_button_new (const gchar *stock_id)
66 {
67   GtkWidget *button;
68   GtkWidget *icon;
69
70   icon = gtk_image_new_from_stock (stock_id, GTK_ICON_SIZE_BUTTON);
71   button = gtk_button_new ();
72   gtk_container_add (GTK_CONTAINER (button), icon);
73
74   return button;
75 }
76
77 int
78 main (int argc, char **argv)
79 {
80   GtkWidget *window;
81   GtkWidget *vbox;
82   GtkWidget *hbox;
83   GtkWidget *notebook;
84   GtkWidget *box;
85   GtkWidget *label;
86   GtkWidget *label_box;
87   GtkWidget *menu_box;
88   GtkWidget *icon;
89   GtkWidget *button;
90   gchar     buf[1024];
91   int       n;
92
93   gtk_init (&argc, &argv);
94
95   book_open  = gdk_pixbuf_new_from_file ("stock_book_open.png",  NULL);
96   book_close = gdk_pixbuf_new_from_file ("stock_book_close.png", NULL);
97
98   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
99   gtk_window_set_title  (GTK_WINDOW (window), "GtkNotebook␣Sample");
100  gtk_window_set_resizable (GTK_WINDOW (window), TRUE);
101  gtk_widget_set_size_request (window, 300, 200);
102  gtk_container_set_border_width (GTK_CONTAINER (window), 5);
103  g_signal_connect (G_OBJECT (window), "destroy",
104                    G_CALLBACK (gtk_main_quit), NULL);
105
106  vbox = gtk_vbox_new (FALSE, 5);
107  gtk_container_add (GTK_CONTAINER (window), vbox);
108
109  notebook = gtk_notebook_new ();
110  gtk_notebook_set_scrollable (GTK_NOTEBOOK (notebook), TRUE);
111  gtk_notebook_popup_enable (GTK_NOTEBOOK(notebook));
112  g_signal_connect (G_OBJECT (notebook), "switch_page",
113                    G_CALLBACK (page_switch), NULL);
114  gtk_box_pack_start (GTK_BOX (vbox), notebook, TRUE, TRUE, 0);
115
116  for (n = 1; n <= 5; n++)
117    {
118      box   = gtk_vbox_new (FALSE, 0);
119
120      sprintf (buf, "Contents␣%d", n);
121      label = gtk_label_new (buf);
122      gtk_box_pack_start (GTK_BOX (box), label, TRUE, TRUE, 0);
123
124      label_box = gtk_hbox_new (FALSE, 0);
125      icon = gtk_image_new_from_pixbuf (book_close);
```

```
126       g_object_set_data (G_OBJECT (box), "tab_icon", icon);
127       gtk_box_pack_start (GTK_BOX (label_box), icon, FALSE, TRUE, 0);
128       gtk_misc_set_padding (GTK_MISC (icon), 5, 1);
129
130       sprintf (buf, "Page␣%d", n);
131       label = gtk_label_new (buf);
132       gtk_box_pack_start (GTK_BOX (label_box), label, FALSE, TRUE, 0);
133
134       gtk_widget_show_all (label_box);
135
136       menu_box = gtk_hbox_new (FALSE, 0);
137       icon = gtk_image_new_from_pixbuf (book_close);
138       g_object_set_data (G_OBJECT (box), "menu_icon", icon);
139       gtk_box_pack_start (GTK_BOX (menu_box), icon, FALSE, TRUE, 0);
140       gtk_misc_set_padding (GTK_MISC (icon), 5, 1);
141
142       label = gtk_label_new (buf);
143       gtk_box_pack_start (GTK_BOX (menu_box), label, FALSE, TRUE, 0);
144
145       gtk_widget_show_all (menu_box);
146
147       gtk_notebook_append_page_menu (GTK_NOTEBOOK (notebook),
148                                      box, label_box, menu_box);
149     }
150   hbox = gtk_hbox_new (TRUE, 0);
151   gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
152   {
153     button = icon_button_new (GTK_STOCK_GO_BACK);
154     g_signal_connect (G_OBJECT (button), "clicked",
155                       G_CALLBACK (cb_tab_position_left), notebook);
156     gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
157
158     button = icon_button_new (GTK_STOCK_GO_FORWARD);
159     g_signal_connect (G_OBJECT (button), "clicked",
160                       G_CALLBACK (cb_tab_position_right), notebook);
161     gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
162
163     button = icon_button_new (GTK_STOCK_GO_UP);
164     g_signal_connect (G_OBJECT (button), "clicked",
165                       G_CALLBACK (cb_tab_position_top), notebook);
166     gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
167
168     button = icon_button_new (GTK_STOCK_GO_DOWN);
169     g_signal_connect (G_OBJECT (button), "clicked",
170                       G_CALLBACK (cb_tab_position_bottom), notebook);
171     gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
172   }
173   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
174   g_signal_connect (G_OBJECT (button), "clicked",
175                     G_CALLBACK (gtk_main_quit), NULL);
176   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
177
178   gtk_widget_show_all (window);
179   gtk_main ();
180
181   return 0;
182 }
```

7.2.7　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
          +----GtkObject
                +----GtkWidget
                      +----GtkContainer
                            +----GtkBin
                                  +----GtkExpander
```

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．

シグナルとコールバック関数

表 7.8 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);
```

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．

- 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．

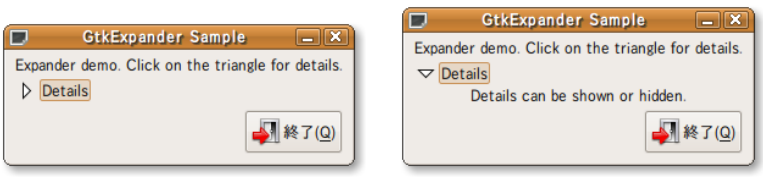


図 7.12　エクスパンダウィジェット

表 7.8 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.

```
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   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
15   gtk_window_set_title  (GTK_WINDOW (window), "GtkExpander␣Sample");
16   gtk_window_set_resizable (GTK_WINDOW (window), FALSE);
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_vbox_new (FALSE, 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_hbox_new (FALSE, 5);
36   gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
37
38   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
39   g_signal_connect (G_OBJECT (button), "clicked",
40                     G_CALLBACK (gtk_main_quit), NULL);
41
42   gtk_box_pack_end (GTK_BOX (hbox), button, FALSE, FALSE, 0);
43
44   gtk_widget_show_all (window);
45   gtk_main ();
46
47   return 0;
48 }
```

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
          +----GtkObject
                +----GtkWidget
                      +----GtkEntry
```

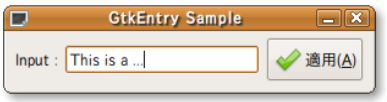


図 7.13　エントリウィジェット

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);
```

シグナルとコールバック関数

表 7.9 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);
```

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．

  ```
  void gtk_entry_set_editable (GtkEntry *entry, gboolean 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．

表 7.9　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．

```
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 data)
5  {
6    g_print ("%s\n", gtk_entry_get_text (entry));
7  }
8
9  static void
10 cb_button (GtkButton *button, gpointer data)
11 {
12   g_print ("%s\n", gtk_entry_get_text (GTK_ENTRY (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   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
26   gtk_window_set_title  (GTK_WINDOW (window), "GtkEntry␣Sample");
27   gtk_window_set_resizable (GTK_WINDOW (window), FALSE);
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_hbox_new (FALSE, 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 = gtk_button_new_from_stock (GTK_STOCK_APPLY);
44   g_signal_connect (G_OBJECT (button), "clicked",
45                     G_CALLBACK (cb_button), (gpointer) entry);
46   gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
```

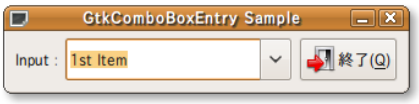


図 7.14　コンボボックスエントリウィジェット

```
47
48    gtk_widget_show_all (window);
49    gtk_main ();
50
51    return 0;
52  }
```

7.3.2　コンボボックスエントリ

コンボボックスエントリ（GtkComboBoxEntry）は，コンボボックスとエントリを合成した複合ウィジェットです．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，コンボボックスエントリはエントリに入力した文字列を新たな項目として追加可能なウィジェットです．

コンボボックス（エントリ）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. 179）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
          +----GtkObject
                 +----GtkWidget
                        +----GtkContainer
                               +----GtkBin
                                      +----GtkComboBox
                                             +----GtkComboBoxEntry
```

ウィジェットの作成

コンボボックスエントリウィジェットを作成する関数を以下に示します．

- gtk_combo_box_entry_new_text
  コンボボックスエントリを作成する一番簡単な関数です．

  ```
  GtkWidget* gtk_combo_box_entry_new_text (void);
  ```

- gtk_combo_box_entry_new
  コンボボックスエントリを作成する関数です．この関数を使ってウィジェットを作成した場合は，その後で関数 gtk_combo_box_set_model と関数 gtk_combo_box_entry_set_text_column を呼び出してコンボアイテムの設定をする必要があります．この手間を省いてくれるのが上記の関数 gtk_combo_box_entry_new_text です．

  ```
  GtkWidget* gtk_combo_box_entry_new (void);
  ```

- gtk_combo_box_entry_new_with_model
  コンボアイテムのモデルを指定してコンボボックスエントリを作成する関数です．この関数を使用するには GtkTreeModel の知識が必要です．

  ```
  GtkWidget*
  gtk_combo_box_entry_new_with_model (GtkTreeModel *model,
                                      gint          text_column);
  ```

シグナルとコールバック関数

表 7.10 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 *combobox,
                    gpointer     user_data);
```

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，ソース 7–3–2 のようにします．

ソース 7-3-2 コンボアイテムの文字列の取得

```
1  GtkComboBox  *combobox;
2  GtkTreeModel *model;
3  GtkTreeIter  iter;
4  gchar        *label;
5  model = gtk_combo_box_get_model (combobox);
6  if (gtk_combo_box_get_active_iter (combobox, &iter))
7    {
8      gtk_tree_model_get (model, &iter, 0, &label, -1);
9    }
```

関数 gtk_combo_box_get_active_iter は現在選択されているアイテムの GtkTreeIter を取得する関数です．選択されていない場合は FALSE が返ります．

```
gboolean gtk_combo_box_get_active_iter (GtkComboBox *combo_box,
                                        GtkTreeIter *iter);
```

関数 gtk_tree_model_get は設定されたモデルから指定した位置のデータを取得する関数です．第 3 番目の引数から，データのインデックス，データを格納する変数を並べて与えます．これは複数並べて記述することが可能です．関数の最後の引数は必ず −1 で終わらなければいけません．

```
void gtk_tree_model_get (GtkTreeModel *tree_model,
                         GtkTreeIter *iter,
                         ...);
```

サンプルプログラム

ソース 7–3–3 にコンボボックス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．コンボボックスにはあらかじめ 2 つのアイテムが登録されていて，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．

表 7.10 コンボボックスエントリ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．

ソース 7-3-3 コンボボックスエントリウィジェットのサンプルプログラム：gtkcomboboxentry-sample.c

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

#define COMBO_LIST_LIMIT 5

static GList *combolist = NULL;

static GList*
append_item (GtkComboBox *combobox,
             GList       *combolist,
             const gchar *item_label)
{
  gtk_combo_box_append_text (combobox, item_label);
  combolist = g_list_append (combolist, g_strdup (item_label));

  return combolist;
}

static GList*
remove_item (GtkComboBox *combobox,
             GList       *combolist,
             gint        index)
{
  gtk_combo_box_remove_text (combobox, index);
  g_free (g_list_nth_data (combolist, index));
  combolist
    = g_list_remove_link (combolist, g_list_nth (combolist, index));

  return combolist;
}

static void
cb_combo_changed (GtkComboBox *combobox,
                  gpointer    user_data)
{
  GtkTreeModel *model;
  GtkTreeIter  iter;
  gchar        *text;

  model = gtk_combo_box_get_model (combobox);
  if (gtk_combo_box_get_active_iter (combobox, &iter))
    {
      gtk_tree_model_get (model, &iter, 0, &text, -1);
      g_print ("Item␣number␣is␣%d␣(%s)\n",
               gtk_combo_box_get_active (combobox), text);
    }
}

static void
cb_combo_entry_activate (GtkEntry *entry,
                         gpointer user_data)
{
  GtkComboBox *combobox;
  GList       *list;
  gboolean    exist_flag = FALSE;
  const gchar *new_text;

  new_text = gtk_entry_get_text (entry);
  if (!new_text || strcmp (new_text, "") == 0) return;

  for (list = combolist; list; list = g_list_next (list))
    {
      if (strcmp (new_text, (gchar *) list->data) == 0)
        {
          exist_flag = TRUE;
          break;
        }
    }
  if (!exist_flag)
    {
      combobox = GTK_COMBO_BOX (user_data);
      combolist = append_item (combobox, combolist, new_text);
```

```
72        if (g_list_length (combolist) > COMBO_LIST_LIMIT)
73          {
74            combolist = remove_item (combobox, combolist, 0);
75          }
76        g_print ("Append␣a␣new␣item␣label␣'%s'.\n", new_text);
77      }
78  }
79
80  int
81  main (int argc, char **argv)
82  {
83    GtkWidget *window;
84    GtkWidget *hbox;
85    GtkWidget *label;
86    GtkWidget *combo;
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), "GtkComboBoxEntry␣Sample");
93    gtk_window_set_resizable (GTK_WINDOW (window), FALSE);
94    gtk_container_set_border_width (GTK_CONTAINER (window), 5);
95    g_signal_connect (G_OBJECT (window), "destroy",
96                      G_CALLBACK (gtk_main_quit), NULL);
97    hbox = gtk_hbox_new (FALSE, 5);
98    gtk_container_add (GTK_CONTAINER (window), hbox);
99
100   label = gtk_label_new ("Input␣:");
101   gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);
102
103   combo = gtk_combo_box_entry_new_text ();
104   combolist = append_item (GTK_COMBO_BOX (combo), combolist, "1st␣Item");
105   combolist = append_item (GTK_COMBO_BOX (combo), combolist, "2nd␣Item");
106   gtk_combo_box_set_active (GTK_COMBO_BOX (combo), 0);
107
108   g_signal_connect (G_OBJECT (combo), "changed",
109                     G_CALLBACK (cb_combo_changed), NULL);
110   g_signal_connect (G_OBJECT (GTK_BIN (combo)->child), "activate",
111                     G_CALLBACK (cb_combo_entry_activate), combo);
112
113   gtk_box_pack_start (GTK_BOX (hbox), combo, TRUE, TRUE, 0);
114
115   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
116   g_signal_connect (G_OBJECT (button), "clicked",
117                     G_CALLBACK (gtk_main_quit), NULL);
118   gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
119
120   gtk_widget_show_all (window);
121   gtk_main ();
122
123   return 0;
124 }
```

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
          +----GtkObject
                +----GtkWidget
                      +----GtkEntry
                            +----GtkSpinButton
```

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．

表 7.11　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.

- 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);
```

  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．このように GtkAdjustment で指定するのはオーバースペックで，かつ面倒なので，次に紹介する関数 gtk_spin_button_new_with_range を使うと便利です．

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  GtkObject *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,
7                                   step, page, page_size);
8  spinbutton = gtk_spin_button_new (GTK_ADJUSTMENT(adjustment),
9                                    step, digits);
```

- gtk_spin_button_new_with_range
  数値の範囲とステップ値を指定してスピンボタンを作成する関数です．

```
GtkWidget* gtk_spin_button_new_with_range (gdouble min,
                                           gdouble max,
                                           gdouble step);
```

シグナルとコールバック関数

表 7.11 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);
```

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．

- 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，取得したりできる．

```
void gtk_spin_button_set_increments (GtkSpinButton *spin_button,
                                     gdouble       step,
                                     gdouble       page);
```

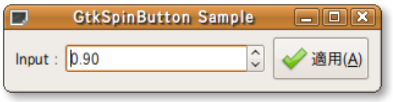


図 7.15　スピンボタンウィジェット

```
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 data)
{
  g_print ("value = %f\n", gtk_spin_button_get_value (spinbutton));
}

static void
cb_button (GtkButton *button, gpointer data)
{
  g_print ("value = %f\n",
           gtk_spin_button_get_value (GTK_SPIN_BUTTON (data)));
}

int
main (int argc, char **argv)
{
  GtkWidget *window;
  GtkWidget *hbox;
  GtkWidget *label;
  GtkWidget *spinbutton;
  GtkWidget *button;
  GtkObject *adjustment;
  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_hbox_new (FALSE, 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_stock (GTK_STOCK_APPLY);
  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．

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----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．

図 7.16　テキストビューウィジェット

- 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 data)
{
  GtkTextBuffer *buffer;
  GtkTextIter   start, end;
  gchar         *utf8_text;

  buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (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");
  gtk_widget_set_size_request (window, 300, 100);
  gtk_container_set_border_width (GTK_CONTAINER (window), 5);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);
```

```
44
45    vbox = gtk_vbox_new (FALSE, 5);
46    gtk_container_add (GTK_CONTAINER (window), vbox);
47
48    scrolledwindow = gtk_scrolled_window_new (NULL, NULL);
49    gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW
50                                         (scrolledwindow),
51                                         GTK_SHADOW_ETCHED_OUT);
52    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolledwindow),
53                                    GTK_POLICY_AUTOMATIC,
54                                    GTK_POLICY_AUTOMATIC);
55    gtk_box_pack_start (GTK_BOX (vbox), scrolledwindow, TRUE, TRUE, 0);
56
57    textview = gtk_text_view_new ();
58    gtk_container_add (GTK_CONTAINER (scrolledwindow), textview);
59    set_text (GTK_TEXT_VIEW (textview),
60              "This␣is␣a␣sample␣program␣of␣GtkTextView.\n"
61              "GtkTextView␣is␣a␣...\n"
62              "このプログラムはGtkTextViewウィジェットのサンプル\n"
63              "プログラムです。");
64    hbox = gtk_hbox_new (FALSE, 5);
65    gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
66
67    button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
68    g_signal_connect (G_OBJECT (button), "clicked",
69                      G_CALLBACK (gtk_main_quit), NULL);
70    gtk_box_pack_end (GTK_BOX (hbox), button, FALSE, FALSE, 0);
71
72    button = gtk_button_new_from_stock (GTK_STOCK_APPLY);
73    g_signal_connect (G_OBJECT (button), "clicked",
74                      G_CALLBACK (print_text), (gpointer) textview);
75    gtk_box_pack_end (GTK_BOX (hbox), button, FALSE, FALSE, 0);
76
77    gtk_widget_show_all (window);
78    gtk_main ();
79
80    return 0;
81 }
```

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 7.17）.

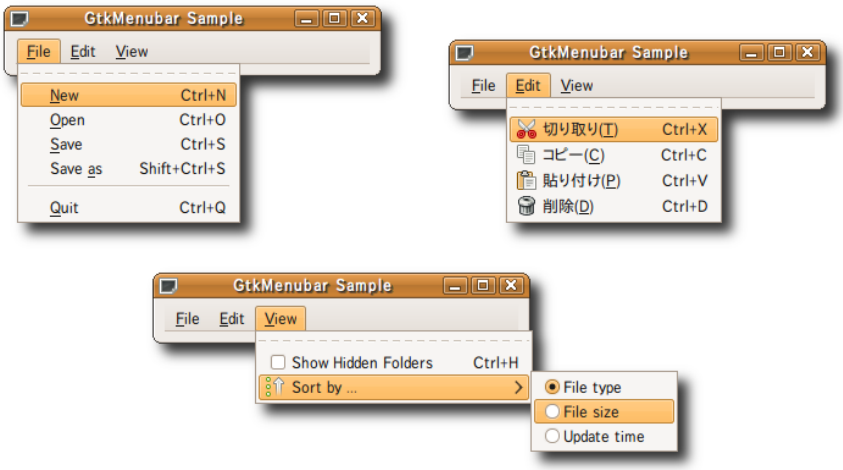


図 7.17　メニューバー

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----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. アイコン付きのメニューアイテム
   - gtk_image_menu_item_new
     アイコン付きのメニューアイテム（ラベルなし）を作成します．この関数でメニューアイテムを作成した時点ではアイコンは登録されていません.

     ```
     GtkWidget* gtk_image_menu_item_new (void);
     ```

     このメニューアイテムにアイコンを登録するには，関数 gtk_image_new などでアイコンデータを作成して，関数 gtk_image_menu_item_set_image を呼び出します.

     ```
     void gtk_image_menu_item_set_image
          (GtkImageMenuItem *image_menu_item, GtkWidget *image);
     ```

     アイコンの登録の例を以下に示します.

     ```
     GtkWidget *item;
     GtkWidget *image;

     image =
       gtk_image_new_from_stock (GTK_STOCK_OK, GTK_ICON_SIZE_MENU);
     item  = gtk_image_menu_item_new ();
     gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (item),
                                    image);
     ```

- gtk_image_menu_item_new_with_label
  ラベル付きのアイコンメニューアイテムを作成します．アイコンの登録方法は上記の方法と同様です．

```
GtkWidget*
gtk_image_menu_item_new_with_label (const gchar *label);
```

- gtk_image_menu_item_new_with_mnemonic
  アクセラレータ機能のあるラベル付きのアイコンメニューアイテムを作成します．アイコンの登録方法は上記の方法と同様です．

```
GtkWidget*
gtk_image_menu_item_new_with_mnemonic (const gchar *label);
```

- gtk_image_menu_item_new_from_stock
  GtkStockItem で定義された文字列を指定してアイコンメニューアイテムを作成します．自動的にショートカットも設定されるので，関数の引数には GtkAccelGroup 型の変数も与えます．

```
GtkWidget*
gtk_image_menu_item_new_from_stock (const gchar    *stock_id,
                                    GtkAccelGroup *accel_group);
```

3. 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);
```

4. 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);
```

5. その他のメニューアイテム
   セパレータやティアオフアイテム（切り離し可能アイテム）です．
   - gtk_separator_menu_item_new

```
GtkWidget* gtk_separator_menu_item_new (void);
```

   - gtk_tearoff_menu_item_new

```
GtkWidget* gtk_tearoff_menu_item_new (void);
```

ショートカットキーの設定

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，ショートカットキーが設定されていることがよくあります．関数 gtk_image_menu_item_new_from_stock を使ってメニューアイテムを作成すると，そのストックアイテムに対応したショートカットキーが自動的に設定されます．その他の関数でメニューアイテムを作成した場合には，関数 gtk_widget_add_accelerator を使ってメニューアイテムにショートカットを設定します．

```
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　ショートカットを設定するウィジェット
第 2 引数　ショートカット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　ショートカットキー
第 5 引数　装飾子
第 6 引数　アクセラレータフラグ

ショートカットキーの設定例をソース 7–4–1 に示します．ショートカットキー GDK_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）がショートカットとして設定されます．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 ショートカットキー設定のサンプル

```
1  GtkWidget     *item;
2  GtkAccelGroup *accel_group;
3
4  accel_group = gtk_accel_group_new ();
5  item = gtk_menu_item_new_with_mnemonic ("_Open");
6  gtk_widget_add_accelerator (item, "activate", accel_group,
7                              GDK_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_RELEASE_MASK  = 1 << 30,
  GDK_MODIFIER_MASK = GDK_RELEASE_MASK | 0x1fff
} 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_container_add (GTK_CONTAINER(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_container_add (GTK_CONTAINER(submenu), subitem);
```

サンプルプログラム

ソース 7–4–2 に，図 7.17 のメニューを作成するソースを示します．左端の 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．真ん中の Edit メニューは，アイコン付きのメニューアイテムです．右端の 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

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

static void
cb_quit (GtkWidget *widget, gpointer data)
{
  gtk_main_quit ();
}

static GtkWidget*
create_menu (void)
{
  GtkWidget     *menubar;
  GtkWidget     *menu;
  GtkWidget     *item;
  GtkWidget     *image;
  GtkAccelGroup *accel_group;
  GSList        *group = NULL;

  menubar = gtk_menu_bar_new ();
  accel_group = gtk_accel_group_new ();

  item = gtk_menu_item_new_with_mnemonic ("_File");
  gtk_container_add (GTK_CONTAINER (menubar), item);
  menu = gtk_menu_new ();
  gtk_menu_item_set_submenu (GTK_MENU_ITEM (item), menu);
  {
    item = gtk_tearoff_menu_item_new ();
    gtk_container_add (GTK_CONTAINER (menu), item);

    item = gtk_menu_item_new_with_mnemonic ("_New");
    gtk_container_add (GTK_CONTAINER (menu), item);
    gtk_widget_add_accelerator (item, "activate", accel_group,
                                GDK_N,
                                GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);

    item = gtk_menu_item_new_with_mnemonic ("_Open");
    gtk_container_add (GTK_CONTAINER (menu), item);
    gtk_widget_add_accelerator (item, "activate", accel_group,
                                GDK_O,
                                GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);

    item = gtk_menu_item_new_with_mnemonic ("_Save");
```

```
44      gtk_container_add (GTK_CONTAINER (menu), item);
45      gtk_widget_add_accelerator (item, "activate", accel_group,
46                                  GDK_S,
47                                  GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
48
49      item = gtk_menu_item_new_with_mnemonic ("Save␣_as");
50      gtk_container_add (GTK_CONTAINER (menu), item);
51      gtk_widget_add_accelerator (item, "activate", accel_group,
52                                  GDK_S,
53                                  GDK_CONTROL_MASK | GDK_SHIFT_MASK,
54                                  GTK_ACCEL_VISIBLE);
55
56      item = gtk_separator_menu_item_new ();
57      gtk_container_add (GTK_CONTAINER (menu), item);
58
59      item = gtk_menu_item_new_with_mnemonic ("_Quit");
60      gtk_container_add (GTK_CONTAINER (menu), item);
61      gtk_widget_add_accelerator (item, "activate", accel_group,
62                                  GDK_Q,
63                                  GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
64      g_signal_connect (G_OBJECT (item), "activate",
65                        G_CALLBACK (cb_quit), NULL);
66    }
67    item = gtk_menu_item_new_with_mnemonic ("_Edit");
68    gtk_container_add (GTK_CONTAINER (menubar), item);
69    menu = gtk_menu_new ();
70    gtk_menu_item_set_submenu (GTK_MENU_ITEM (item), menu);
71    {
72      item = gtk_tearoff_menu_item_new ();
73      gtk_container_add (GTK_CONTAINER (menu), item);
74
75      item = gtk_image_menu_item_new_from_stock (GTK_STOCK_CUT,
76                                                 accel_group);
77      gtk_container_add (GTK_CONTAINER (menu), item);
78
79      item = gtk_image_menu_item_new_from_stock (GTK_STOCK_COPY,
80                                                 accel_group);
81      gtk_container_add (GTK_CONTAINER (menu), item);
82
83      item = gtk_image_menu_item_new_from_stock (GTK_STOCK_PASTE,
84                                                 accel_group);
85      gtk_container_add (GTK_CONTAINER (menu), item);
86
87      item = gtk_image_menu_item_new_from_stock (GTK_STOCK_DELETE,
88                                                 accel_group);
89      gtk_container_add (GTK_CONTAINER (menu), item);
90      gtk_widget_add_accelerator (item, "activate", accel_group,
91                                  GDK_D,
92                                  GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
93    }
94    item = gtk_menu_item_new_with_mnemonic ("_View");
95    gtk_container_add (GTK_CONTAINER (menubar), item);
96    menu = gtk_menu_new ();
97    gtk_menu_item_set_submenu (GTK_MENU_ITEM (item), menu);
98    {
99      item = gtk_tearoff_menu_item_new ();
100     gtk_container_add (GTK_CONTAINER (menu), item);
101
102     item = gtk_check_menu_item_new_with_label ("Show␣Hidden␣Folders");
103     gtk_container_add (GTK_CONTAINER (menu), item);
104     gtk_widget_add_accelerator (item, "activate", accel_group,
105                                 GDK_H,
106                                 GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
107
108     item = gtk_image_menu_item_new_with_label ("Sort␣by␣...");
109     image = gtk_image_new_from_stock (GTK_STOCK_SORT_ASCENDING,
110                                       GTK_ICON_SIZE_MENU);
111     gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (item), image);
112     gtk_container_add (GTK_CONTAINER (menu), item);
113     menu = gtk_menu_new ();
114     gtk_menu_item_set_submenu (GTK_MENU_ITEM (item), menu);
115     {
116       item = gtk_radio_menu_item_new_with_label (group, "File␣type");
117       gtk_container_add (GTK_CONTAINER (menu), item);
118       gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM (item), TRUE);
119
```

```
120       group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM (item));
121       item = gtk_radio_menu_item_new_with_label (group, "File␣size");
122       gtk_container_add (GTK_CONTAINER (menu), item);
123
124       group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM (item));
125       item = gtk_radio_menu_item_new_with_label (group, "Update␣time");
126       gtk_container_add (GTK_CONTAINER (menu), item);
127     }
128   }
129   return menubar;
130 }
131
132 int
133 main (int argc, char **argv)
134 {
135   GtkWidget *window;
136   GtkWidget *menubar;
137
138   gtk_init (&argc, &argv);
139
140   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
141   gtk_window_set_title (GTK_WINDOW (window), "GtkMenubar␣Sample");
142   gtk_widget_set_size_request (window, 300, -1);
143   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
144   g_signal_connect (G_OBJECT (window), "destroy",
145                     G_CALLBACK (gtk_main_quit), NULL);
146
147   menubar = create_menu ();
148   gtk_container_add (GTK_CONTAINER (window), menubar);
149
150   gtk_widget_show_all (window);
151   gtk_main ();
152
153   return 0;
154 }
```

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 7.18）.

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----GtkWidget
                      +----GtkContainer
                            +----GtkMenuShell
                                  +----GtkMenu
```

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.

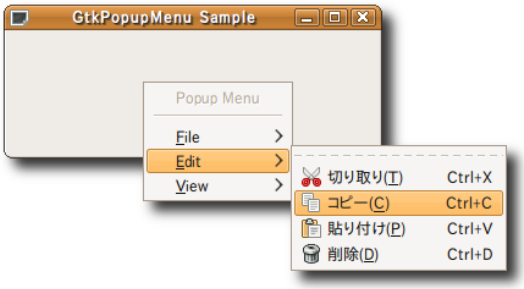


図 7.18 ポップアップメニュー

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 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     *item;
  GtkWidget     *image;
  GtkAccelGroup *accel_group;
  GSList        *group = NULL;

  popupmenu = gtk_menu_new ();
  accel_group = gtk_accel_group_new ();
  gtk_window_add_accel_group (parent, accel_group);

  item = gtk_menu_item_new_with_label ("Popup␣Menu");
  gtk_widget_set_sensitive (item, FALSE);
  gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), item);

```

```
42   item = gtk_separator_menu_item_new ();
43   gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), item);
44
45   item = gtk_menu_item_new_with_mnemonic ("_File");
46   gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), item);
47   menu = gtk_menu_new ();
48   gtk_menu_item_set_submenu (GTK_MENU_ITEM (item), menu);
49   {
50     item = gtk_tearoff_menu_item_new ();
51     gtk_container_add (GTK_CONTAINER (menu), item);
52
53     item = gtk_menu_item_new_with_mnemonic ("_New");
54     gtk_container_add (GTK_CONTAINER (menu), item);
55     gtk_widget_add_accelerator (item, "activate", accel_group,
56                                 GDK_N,
57                                 GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
58
59     item = gtk_menu_item_new_with_mnemonic ("_Open");
60     gtk_container_add (GTK_CONTAINER (menu), item);
61     gtk_widget_add_accelerator (item, "activate", accel_group,
62                                 GDK_O,
63                                 GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
64
65     item = gtk_menu_item_new_with_mnemonic ("_Save");
66     gtk_container_add (GTK_CONTAINER (menu), item);
67     gtk_widget_add_accelerator (item, "activate", accel_group,
68                                 GDK_S,
69                                 GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
70
71     item = gtk_menu_item_new_with_mnemonic ("Save␣_as");
72     gtk_container_add (GTK_CONTAINER (menu), item);
73     gtk_widget_add_accelerator (item, "activate", accel_group,
74                                 GDK_S,
75                                 GDK_CONTROL_MASK | GDK_SHIFT_MASK,
76                                 GTK_ACCEL_VISIBLE);
77
78     item = gtk_separator_menu_item_new ();
79     gtk_container_add (GTK_CONTAINER (menu), item);
80
81     item = gtk_menu_item_new_with_mnemonic ("_Quit");
82     gtk_container_add (GTK_CONTAINER (menu), item);
83     gtk_widget_add_accelerator (item, "activate", accel_group,
84                                 GDK_Q,
85                                 GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
86     g_signal_connect (G_OBJECT (item), "activate",
87                       G_CALLBACK (cb_quit), NULL);
88   }
89   item = gtk_menu_item_new_with_mnemonic ("_Edit");
90   gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), item);
91   menu = gtk_menu_new ();
92   gtk_menu_item_set_submenu (GTK_MENU_ITEM (item), menu);
93   {
94     item = gtk_tearoff_menu_item_new ();
95     gtk_container_add (GTK_CONTAINER (menu), item);
96
97     item = gtk_image_menu_item_new_from_stock (GTK_STOCK_CUT,
98                                                accel_group);
99     gtk_container_add (GTK_CONTAINER (menu), item);
100
101    item = gtk_image_menu_item_new_from_stock (GTK_STOCK_COPY,
102                                               accel_group);
103    gtk_container_add (GTK_CONTAINER (menu), item);
104
105    item = gtk_image_menu_item_new_from_stock (GTK_STOCK_PASTE,
106                                               accel_group);
107    gtk_container_add (GTK_CONTAINER (menu), item);
108
109    item = gtk_image_menu_item_new_from_stock (GTK_STOCK_DELETE,
110                                               accel_group);
111    gtk_container_add (GTK_CONTAINER (menu), item);
112    gtk_widget_add_accelerator (item, "activate", accel_group,
113                                GDK_D,
114                                GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
115  }
116  item = gtk_menu_item_new_with_mnemonic ("_View");
117  gtk_menu_shell_append (GTK_MENU_SHELL (popupmenu), item);
```

```
118   menu = gtk_menu_new ();
119   gtk_menu_item_set_submenu (GTK_MENU_ITEM (item), menu);
120   {
121     item = gtk_tearoff_menu_item_new ();
122     gtk_container_add (GTK_CONTAINER (menu), item);
123
124     item = gtk_check_menu_item_new_with_label ("Show␣Hidden␣Folders");
125     gtk_container_add (GTK_CONTAINER (menu), item);
126     gtk_widget_add_accelerator (item, "activate", accel_group,
127                                 GDK_H,
128                                 GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
129
130     item = gtk_image_menu_item_new_with_label ("Sort␣by␣...");
131     image = gtk_image_new_from_stock (GTK_STOCK_SORT_ASCENDING,
132                                       GTK_ICON_SIZE_MENU);
133     gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (item), image);
134     gtk_container_add (GTK_CONTAINER (menu), item);
135     menu = gtk_menu_new ();
136     gtk_menu_item_set_submenu (GTK_MENU_ITEM (item), menu);
137     {
138       item = gtk_radio_menu_item_new_with_label (group, "File␣type");
139       gtk_container_add (GTK_CONTAINER (menu), item);
140       gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM (item), TRUE);
141
142       group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM (item));
143       item = gtk_radio_menu_item_new_with_label (group, "File␣size");
144       gtk_container_add (GTK_CONTAINER (menu), item);
145
146       group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM (item));
147       item = gtk_radio_menu_item_new_with_label (group, "Update␣time");
148       gtk_container_add (GTK_CONTAINER (menu), item);
149     }
150   }
151   return popupmenu;
152 }
153
154 int
155 main (int argc, char **argv)
156 {
157   GtkWidget *window;
158   GtkWidget *eventbox;
159   GtkWidget *popupmenu;
160
161   gtk_init (&argc, &argv);
162
163   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
164   gtk_window_set_title  (GTK_WINDOW (window), "GtkPopupMenu␣Sample");
165   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
166   gtk_widget_set_size_request (window, 300, 100);
167   g_signal_connect (G_OBJECT (window), "destroy",
168                     G_CALLBACK (gtk_main_quit), NULL);
169
170   popupmenu = create_popupmenu (GTK_WINDOW (window));
171   gtk_widget_show_all (popupmenu);
172
173   eventbox = gtk_event_box_new ();
174   gtk_container_add (GTK_CONTAINER (window), eventbox);
175   g_signal_connect (G_OBJECT (eventbox), "button_press_event",
176                     G_CALLBACK (cb_popup_menu), popupmenu);
177
178   gtk_widget_show_all (window);
179   gtk_main ();
180
181   return 0;
182 }
```

7.4.3　UI マネージャ

UI マネージャ（GtkUIManager）は，これまで説明してきたメニューの作成を支援するウィジェットです．図 7.17 は UI マネージャを利用して作成したメニューの例です．最終的に出来上がるメニューはこれまで説明した方法で作成したメニューとほぼ同じものですが，メニューの作成は UI マネージャを利用するほうがはるかに簡単です．

オブジェクトの階層構造

```
GObject
    +----GtkUIManager
```

UI マネージャの作成

ウィジェットの作成には関数 gtk_ui_manager_new を使用します．

```
GtkUIManager* gtk_ui_manager_new (void);
```

メニューアイテムの定義

メニューアイテムの定義は GtkActionEntry という構造体を用いて行います．GtkActionEntry 構造体は次のように定義されています．

```
typedef struct {
  const gchar     *name;
  const gchar     *stock_id;
  const gchar     *label;
  const gchar     *accelerator;
  const gchar     *tooltip;
  GCallback       callback;
} GtkActionEntry;
```

構造体のメンバの説明を以下にまとめます．

- name
  メニューアイテムを特定するための文字列です．
- stock_id
  メニューアイテムの先頭に表示するアイコン用の GtkStockItem で定義された文字列です．
- label
  メニューアイテムに表示するラベルです．文字の前にアンダースコアを挿入することで，その文字の下にアンダースコアが表示されアクセラレータキーとして動作します．
- accelerator
  メニューアイテムのショートカットを設定するための文字列です．”<control>O” や ”<control><shift>S”, ”<alt><shift>X” のように指定します．
- tooltips
  メニューアイテムの説明のための文字列です．
- callback
  メニューアイテムがクリックされたときに呼び出すコールバック関数です．

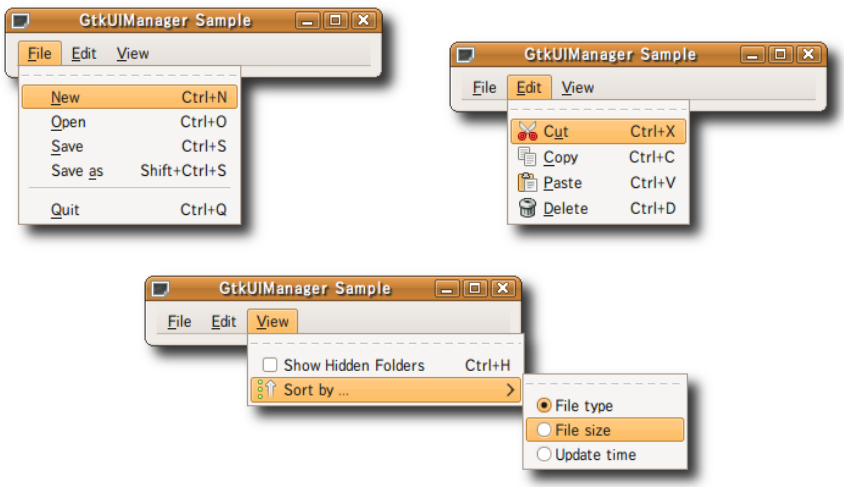


図 7.19 UI マネージャ

今説明したメニューアイテムは普通のメニューアイテムでチェックボタンやラジオボタンの機能を持つメニューアイテムの定義は，それぞれ GtkToggleActionEntry と GtkRadioActionEntry という構造体を用いて行います．GtkToggleActionEntry 構造体は次のように定義されています．

```
typedef struct {
  const gchar     *name;
  const gchar     *stock_id;
  const gchar     *label;
  const gchar     *accelerator;
  const gchar     *tooltip;
  GCallback       callback;
  gboolean        is_active;
} GtkToggleActionEntry;
```

GtkToggleActionEntry 構造体は，GtkActionEntry 構造体のメンバに is_active というアイテムがアクティブ状態かどうかを表す gboolean 型のメンバが追加されたものです．

GtkRadioActionEntry 構造体は次のように定義されています．

```
typedef struct {
  const gchar     *name;
  const gchar     *stock_id;
  const gchar     *label;
  const gchar     *accelerator;
  const gchar     *tooltip;
  gint            value;
} GtkRadioActionEntry;
```

GtkRadioActionEntry 構造体は，callback メンバがなく，同じラジオメニューアイテムの中の ID を表す value という gint 型のメンバが追加されています．ラジオメニューアイテムのコールバック関数はこの後説明する関数 gtk_action_group_add_radio_actions で設定します．

アクショングループの作成とメニューアイテムの登録

UI マネージャを利用したメニュー作成に密接に関連するウィジェットが，アクショングループ（GtkActionGroup）です．このウィジェットは，メニューアイテムをまとめて扱うためのウィジェットです．

アクショングループの作成は関数 gtk_action_group_new を使用します．引数の文字列には，作成するアクショングループを特定するための名前を指定します．

```
GtkActionGroup* gtk_action_group_new (const gchar *name);
```

そして次の 3 つの関数で，作成したアクショングループに対してメニューアイテムを登録します．

- gtk_action_group_add_actions
  GtkActionEntry 構造体で用意したメニューアイテムをアクショングループに登録するための関数です．第 3 引数にはメニューアイテムの数，第 4 引数にはコールバック関数に渡すデータを指定します．

```
void
gtk_action_group_add_actions (GtkActionGroup       *action_group,
                              const GtkActionEntry *entries,
                              guint                 n_entries,
                              gpointer              user_data);
```

- gtk_action_group_add_toggle_actions
  GtkActionToggleEntry 構造体で用意したメニューアイテムをアクショングループに登録するための関数です．

```
void gtk_action_group_add_toggle_actions
  (GtkActionGroup            *action_group,
   const GtkToggleActionEntry *entries,
   guint                      n_entries,
   gpointer                   user_data);
```

- gtk_action_group_add_radio_actions
  GtkActionRadioEntry 構造体で用意したメニューアイテムをアクショングループに登録するための関数です．第 4 引数には初期状態でアクティブにするアイテム番号（GtkActionRadioEntry 構造体の value の値）を指定します．また第 5 引数でコールバック関数を指定します．

```
void gtk_action_group_add_radio_actions
  (GtkActionGroup            *action_group,
   const GtkRadioActionEntry *entries,
   guint                     n_entries,
   gint                      value,
   GCallback                 on_change,
   gpointer                  user_data);
```

メニューの階層構造の定義

今まではメニューアイテム 1 つずつの定義について説明してきました．ではメニューの構成はどうやって決定したらいいのでしょう．前節までに説明した方法では，サブメニューを追加してどのメニューアイテムを配置するのかなどを，すべて GTK+ の関数を呼び出して設定する必要がありました．

GtkUIManager を使用してメニューを構成する場合には，メニューの階層構造をテキストで記述するだけで済んでしまいます．簡単なメニューの階層構造の例を次に示します．

```
static const gchar *ui_info =
"<ui>"
"␣␣<menubar␣name='MenuBar'>"
"␣␣␣␣<menu␣action='FileMenu'>"
"␣␣␣␣␣␣<menuitem␣action='New'/>"
"␣␣␣␣␣␣<menuitem␣action='Open'/>"
"␣␣␣␣␣␣<menuitem␣action='Save'/>"
"␣␣␣␣␣␣<menuitem␣action='SaveAs'/>"
"␣␣␣␣␣␣<separator/>"
"␣␣␣␣␣␣<menuitem␣action='Quit'/>"
"␣␣␣␣</menu>"
"␣␣</menubar>"
"</ui>";
```

文字列は “<ui>” で始まり，“</ui>” で終わる必要があります．基本的な記述は「識別子」と「属性」という形式で行います．メニューアイテムの記述は “<menuitem action='New'/>” のように記述して，action の属性には GtkActionEntry 構造体の name メンバの値を記述します．

サンプルプログラム

ソース 7–4–4 に GtkUIManager を用いたサンプルプログラムのソースコードを示します．

ソース 7-4-4 UI マネージャのサンプルプログラム：gtkuimanager-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_test1 (void)
5  {
6    g_print ("...\n");
7  }
8
9  static void
10 cb_test2 (GtkWidget *widget, gpointer user_data)
11 {
12   g_print ("%s\n", (gchar *) user_data);
13 }
14
15 static GtkActionEntry entries[] =
16 {
17   {"FileMenu", NULL, "_File"},
18   {"EditMenu", NULL, "_Edit"},
```

```
  {"ViewMenu", NULL, "_View"},
  {"SortMenu", GTK_STOCK_SORT_ASCENDING, "Sort␣by␣..."},
  {"New", NULL, "_New", "<control>N", NULL, cb_test1},
  {"Open", NULL, "_Open", "<control>O", NULL, G_CALLBACK (cb_test2)},
  {"Save", NULL, "_Save", "<control>S", NULL, NULL},
  {"SaveAs", NULL, "Save␣_as", "<shift><control>S", NULL, NULL},
  {"Quit", NULL, "_Quit", "<control>Q", NULL, gtk_main_quit},
  {"Cut", GTK_STOCK_CUT, "C_ut", "<control>X", NULL, NULL},
  {"Copy", GTK_STOCK_COPY, "_Copy", "<control>C", NULL, NULL},
  {"Paste", GTK_STOCK_PASTE, "_Paste", "<control>V", NULL, NULL},
  {"Delete", GTK_STOCK_DELETE, "_Delete", "<control>D", NULL, NULL}
};

static GtkToggleActionEntry toggle_entries[] =
{
  {"ShowHidden", NULL, "Show␣Hidden␣Folders", "<control>H",
   NULL, NULL, FALSE}
};

enum
{
  SORT_FILE_TYPE,
  SORT_FILE_SIZE,
  SORT_UPDATE_TIME
};

static GtkRadioActionEntry radio_entries[] =
{
  {"FileType", NULL, "File␣type", NULL, NULL, SORT_FILE_TYPE},
  {"FileSize", NULL, "File␣size", NULL, NULL, SORT_FILE_SIZE},
  {"UpdateTime", NULL, "Update␣time", NULL, NULL, SORT_UPDATE_TIME}
};

static guint n_entries = G_N_ELEMENTS (entries);
static guint n_toggle_entries = G_N_ELEMENTS (toggle_entries);
static guint n_radio_entries = G_N_ELEMENTS (radio_entries);

static const gchar *ui_info =
"<ui>"
"␣␣<menubar␣name='MenuBar'>"
"␣␣␣␣<menu␣action='FileMenu'>"
"␣␣␣␣␣␣<menuitem␣action='New'/>"
"␣␣␣␣␣␣<menuitem␣action='Open'/>"
"␣␣␣␣␣␣<menuitem␣action='Save'/>"
"␣␣␣␣␣␣<menuitem␣action='SaveAs'/>"
"␣␣␣␣␣␣<separator/>"
"␣␣␣␣␣␣<menuitem␣action='Quit'/>"
"␣␣␣␣</menu>"
"␣␣␣␣<menu␣action='EditMenu'>"
"␣␣␣␣␣␣<menuitem␣action='Cut'/>"
"␣␣␣␣␣␣<menuitem␣action='Copy'/>"
"␣␣␣␣␣␣<menuitem␣action='Paste'/>"
"␣␣␣␣␣␣<menuitem␣action='Delete'/>"
"␣␣␣␣</menu>"
"␣␣␣␣<menu␣action='ViewMenu'>"
"␣␣␣␣␣␣<menuitem␣action='ShowHidden'/>"
"␣␣␣␣␣␣<menu␣action='SortMenu'>"
"␣␣␣␣␣␣␣␣<menuitem␣action='FileType'/>"
"␣␣␣␣␣␣␣␣<menuitem␣action='FileSize'/>"
"␣␣␣␣␣␣␣␣<menuitem␣action='UpdateTime'/>"
"␣␣␣␣␣␣</menu>"
"␣␣␣␣</menu>"
"␣␣</menubar>"
"</ui>";

static void
activate_radio_action (GtkAction      *action,
                       GtkRadioAction *current)
{
  g_print ("Radio␣action␣\"%s\"␣selected\n",
           gtk_action_get_name (GTK_ACTION (current)));
}

static GtkWidget*
create_menu (GtkWidget *parent)
{
```

```
95   GtkUIManager   *ui;
96   GtkActionGroup *actions;
97   static gchar   *text = "Hello";
98
99   actions = gtk_action_group_new ("Actions");
100  gtk_action_group_add_actions (actions, entries, n_entries,
101                                (gpointer) text);
102  gtk_action_group_add_toggle_actions (actions,
103                                       toggle_entries, n_toggle_entries,
104                                       NULL);
105  gtk_action_group_add_radio_actions (actions,
106                                      radio_entries, n_radio_entries,
107                                      SORT_FILE_TYPE,
108                                      G_CALLBACK (activate_radio_action),
109                                      NULL);
110
111  ui = gtk_ui_manager_new ();
112  gtk_ui_manager_insert_action_group (ui, actions, 0);
113  gtk_ui_manager_set_add_tearoffs (ui, TRUE);
114  gtk_window_add_accel_group (GTK_WINDOW (parent),
115                              gtk_ui_manager_get_accel_group (ui));
116  gtk_ui_manager_add_ui_from_string (ui, ui_info, -1, NULL);
117
118  return gtk_ui_manager_get_widget (ui, "/MenuBar");
119 }
120
121 int
122 main (int argc, char **argv)
123 {
124   GtkWidget *window;
125   GtkWidget *menubar;
126
127   gtk_init (&argc, &argv);
128   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
129   gtk_window_set_title  (GTK_WINDOW (window), "GtkUIManager␣Sample");
130   gtk_widget_set_size_request (window, 300, -1);
131   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
132   g_signal_connect (G_OBJECT (window), "destroy",
133                     G_CALLBACK (gtk_main_quit), NULL);
134
135   menubar = create_menu (window);
136   gtk_container_add (GTK_CONTAINER (window), menubar);
137
138   gtk_widget_show_all (window);
139   gtk_main ();
140
141   return 0;
142 }
```

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．

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----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．
  GtkDialog は次のような構造をしています．ラベルや画像などのメッセージを配置する場合には，vbox メンバへ配置します．ボタンは action_area へ配置します．

```
struct GtkDialog {
  GtkWidget *vbox;
  GtkWidget *action_area;
};

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,
                             ...);
```

GtkDialogFlags は次のように定義されています．

```
typedef enum
{
  GTK_DIALOG_MODAL               = 1 << 0,
  GTK_DIALOG_DESTROY_WITH_PARENT = 1 << 1,
  GTK_DIALOG_NO_SEPARATOR        = 1 << 2
 } GtkDialogFlags;
```

また，ボタンの指定は第 4 引数からはストック ID 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．

```
GtkWidget *dialog
 = gtk_dialog_new_with_buttons ("My␣dialog",
                                main_app_window,
                                GTK_DIALOG_MODAL |
                                GTK_DIALOG_DESTROY_WITH_PARENT,
                                GTK_STOCK_OK,
                                GTK_RESPONSE_ACCEPT,
                                GTK_STOCK_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
```

図 7.20　ダイアログ

```
{
  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);
```

ウィジェットのプロパティ設定

ダイアログの構成は先ほど説明しましたが，vbox と action_area の間にセパレータを表示するか表示しないかを関数 gtk_dialog_set_has_sesparator で設定できます．また現在のセパレータの設定を関数 gtk_dialog_get_has_sesparator で取得できます．

```
void gtk_dialog_set_has_separator (GtkDialog *dialog,
                                   gboolean  setting);

gboolean gtk_dialog_get_has_separator (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 7.21 左のウィンドウが表示されます．保存ボタンを押すと，図 7.21 右のダイアログが表示されます．「はい」か「いいえ」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．

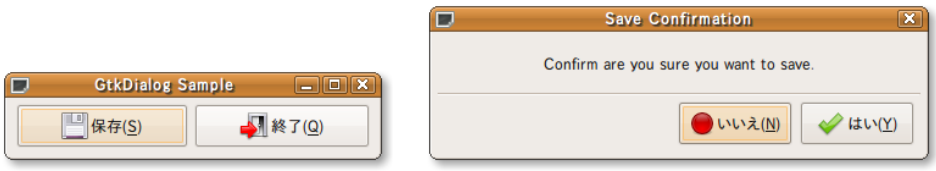


図 7.21 ダイアログウィジェットのサンプルプログラム

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 *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,
                                   GTK_STOCK_NO, GTK_RESPONSE_NO,
                                   GTK_STOCK_YES, GTK_RESPONSE_YES,
                                   NULL);
  label = gtk_label_new ("Confirm␣are␣you␣sure␣you␣want␣to␣save.");
  gtk_container_add (GTK_CONTAINER (GTK_DIALOG (dialog)->vbox), label);
  gtk_widget_set_size_request (dialog, 400, 100);
  gtk_widget_show_all (dialog);

  response = gtk_dialog_run (GTK_DIALOG (dialog));
  if (response == GTK_RESPONSE_YES)
    {
      g_print ("Yes␣button␣was␣pressed.\n");
    }
  else if (response == GTK_RESPONSE_NO)
    {
      g_print ("No␣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_hbox_new (FALSE, 5);
  gtk_container_add (GTK_CONTAINER (window), hbox);

  button = gtk_button_new_from_stock (GTK_STOCK_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_from_stock (GTK_STOCK_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．

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
           +----GtkObject
                  +----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 7.22 を参照）．

```
GtkWidget* gtk_message_dialog_new (GtkWindow      *parent,
                                   GtkDialogFlags flags,
                                   GtkMessageType type,
                                   GtkButtonsType buttons,
                                   const gchar    *message_format,
                                   ...);
```

GtkMessageType は表 7.12 のように定義されています．
GtkButtonType は次のように定義されています．

```
typedef enum
```

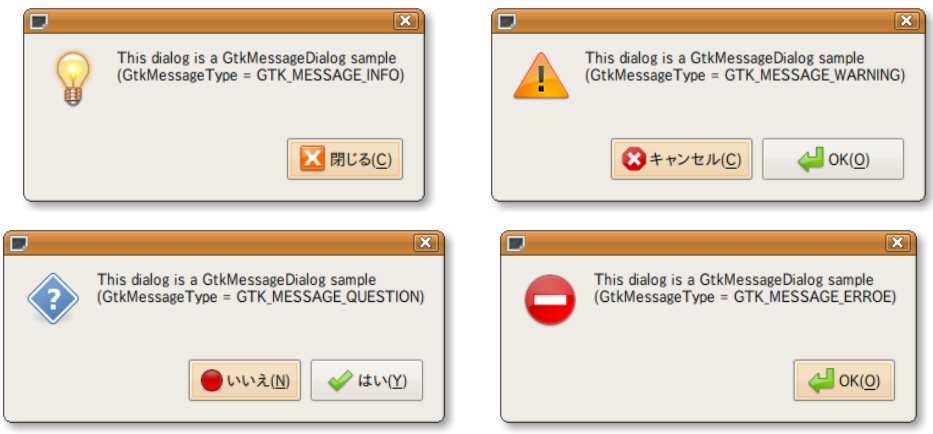


図 7.22 メッセージダイアログ

表 7.12　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 7.22 左上）.
GTK_MESSAGE_WARNING	警告ダイアログを作成する（図 7.22 右上）.
GTK_MESSAGE_QUESTION	質問ダイアログを作成する（図 7.22 左下）.
GTK_MESSAGE_ERROR	エラーダイアログを作成する（図 7.22 右下）.

```
{
  GTK_BUTTONS_NONE,
  GTK_BUTTONS_OK,
  GTK_BUTTONS_CLOSE,
  GTK_BUTTONS_CANCEL,
  GTK_BUTTONS_YES_NO,
  GTK_BUTTONS_OK_CANCEL
} GtkButtonsType;
```

- 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,
                                    ...);
```

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);
```

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 7.23）.

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  GtkWidget *window;
4
5  static void
6  show_dialog (GtkButton *button, gpointer user_data)
7  {
8    GtkWidget      *dialog;
9    GtkButtonsType btype[] = {GTK_BUTTONS_CLOSE,
10                             GTK_BUTTONS_OK_CANCEL,
11                             GTK_BUTTONS_YES_NO,
```

図 7.23　メッセージダイアログのサンプルプログラム

```
12                              GTK_BUTTONS_OK};
13    GtkMessageType mtype = (GtkMessageType) user_data;
14    gchar          *string[] = {"GTK_MESSAGE_INFO",
15                                "GTK_MESSAGE_WARNING",
16                                "GTK_MESSAGE_QUESTION",
17                                "GTK_MESSAGE_ERROE"};
18    gint           result;
19
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 *hbox;
59    GtkWidget *button;
60    gchar     *stock[] = {GTK_STOCK_DIALOG_INFO,
61                          GTK_STOCK_DIALOG_WARNING,
62                          GTK_STOCK_DIALOG_QUESTION,
63                          GTK_STOCK_DIALOG_ERROR};
64    int       n;
65
66    gtk_init (&argc, &argv);
67
68    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
69    gtk_window_set_title (GTK_WINDOW (window), "GtkMessageDialog␣Sample");
70    gtk_container_set_border_width (GTK_CONTAINER (window), 5);
71    g_signal_connect (G_OBJECT (window), "destroy",
72                      G_CALLBACK (gtk_main_quit), NULL);
73
74    hbox = gtk_hbox_new (TRUE, 5);
75    gtk_container_add (GTK_CONTAINER (window), hbox);
76
77    for (n = 0; n < 4; n++)
78      {
79        button = gtk_button_new_from_stock (stock[n]);
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_from_stock (GTK_STOCK_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 7.24）.

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 7.13 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.

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.

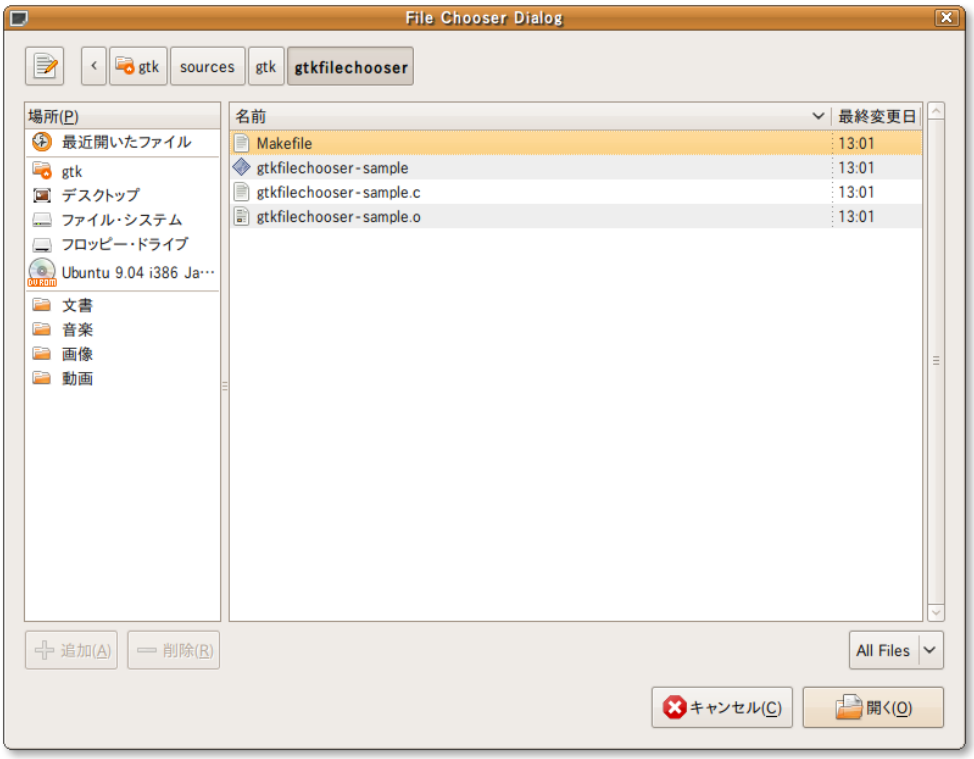


図 7.24　ファイル選択ダイアログ

表 7.13 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.

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，「開く」ボタンと「キャンセル」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．この例のように，ボタンのラベルとして GtkStockItem を使用できます.

```
dialog =
  gtk_file_chooser_dialog_new ("File␣Open␣Dialog",
                               GTK_WIDGET(parent),
                               GTK_FILE_CHOOSER_ACTION_OPEN,
                               GTK_STOCK_CANCEL,
                               GTK_RESPONSE_CANCE,
                               GTK_STOCK_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，「開く」ボタンを押すと図 7.24 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 79 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 data)
5  {
6    g_print ("File␣filter␣changed.\n");
7  }
8
9  static void
10 cb_button (GtkButton *button, gpointer 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   };
21   gint  response;
22   gchar *filename;
23   gchar *folder;
24
25   parent = GTK_WIDGET(g_object_get_data (G_OBJECT (data), "parent"));
26   entry  = GTK_ENTRY (data);
27
28   dialog = gtk_file_chooser_dialog_new ("File␣Chooser␣Dialog",
29                                         GTK_WINDOW (parent),
30                                         action[0],
31                                         GTK_STOCK_CANCEL,
32                                         GTK_RESPONSE_CANCEL,
33                                         GTK_STOCK_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
63                                                 (dialog));
64        g_print ("%s\n", folder);
65        g_free (folder);
66
67        gtk_entry_set_text (entry, filename);
68        g_free (filename);
69      }
70    else if (response == GTK_RESPONSE_CANCEL)
71      {
72        g_print ("Cancel␣button␣was␣pressed.\n");
73      }
74    else
75      {
76        g_print ("Another␣response␣was␣recieved.\n");
77      }
78    gtk_widget_destroy (dialog);
79  }
80
81  int
82  main (int argc, char **argv)
83  {
84    GtkWidget *window;
85    GtkWidget *hbox;
86    GtkWidget *label;
87    GtkWidget *entry;
88    GtkWidget *button;
89
90    gtk_init (&argc, &argv);
91
92    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
93    gtk_window_set_title  (GTK_WINDOW (window), "GtkFileChooser␣Sample");
94    gtk_container_set_border_width (GTK_CONTAINER (window), 5);
95    g_signal_connect (G_OBJECT (window), "destroy",
96                      G_CALLBACK (gtk_main_quit), NULL);
97
98    hbox = gtk_hbox_new (FALSE, 5);
99    gtk_container_add (GTK_CONTAINER (window), hbox);
100
101   label = gtk_label_new ("File␣:");
102   gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);
103
104   entry = gtk_entry_new ();
105   gtk_box_pack_start (GTK_BOX (hbox), entry, TRUE, TRUE, 0);
106   g_object_set_data (G_OBJECT (entry), "parent", (gpointer) window);
107
108   button = gtk_button_new_from_stock (GTK_STOCK_OPEN);
109   g_signal_connect (G_OBJECT (button), "clicked",
110                     G_CALLBACK (cb_button), (gpointer) entry);
111   gtk_box_pack_start (GTK_BOX (hbox), button, FALSE, FALSE, 0);
112
113   gtk_widget_show_all (window);
114   gtk_main ();
115
116   return 0;
117 }
```

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 7.25）．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 7.25 アバウトダイアログ

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----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_name (GtkAboutDialog *about,
                                const gchar    *name);

const gchar* gtk_about_dialog_get_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);
```

- 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 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_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)␣2009");
21   gtk_about_dialog_set_comments (about,
22                                  "This␣is␣a␣GtkAboutDialog␣sample␣"
23                                  "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
29   gtk_container_set_border_width (GTK_CONTAINER (dialog), 5);
30
31   gtk_widget_show_all (dialog);
32 }
33
34 int
35 main (int argc, char **argv)
36 {
37   GtkWidget *window;
38   GtkWidget *button;
39
40   gtk_init (&argc, &argv);
41
42   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
43   gtk_window_set_title (GTK_WINDOW (window), "GtkAboutDialog-Sample");
44   gtk_widget_set_size_request (window, 300, -1);
45   g_signal_connect (G_OBJECT (window), "destroy",
46                     G_CALLBACK (gtk_main_quit), NULL);
47
48   button = gtk_button_new_with_label ("Show␣About␣Dialog");
49   gtk_container_add (GTK_CONTAINER (window), button);
50   g_signal_connect (G_OBJECT (button), "clicked",
51                     G_CALLBACK (cb_show_dialog), NULL);
52
53   gtk_widget_show_all (window);
54   gtk_main ();
55
56   return 0;
57 }
```

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
           +----GtkObject
                  +----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 7.26）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

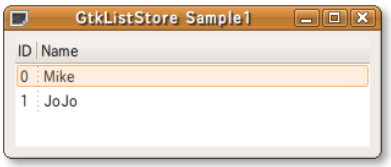


図 7.26　リストの作成

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．

表 7.14 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.

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 の登録（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,
                                    ...);

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 7.14 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．

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,
                                          ...);
```

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 7.27）．

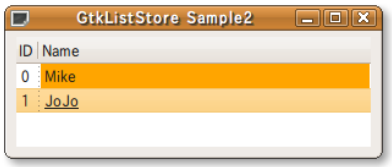


図 7.27 さまざまな列属性の設定

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 7.15 のようになっています．

表 7.15 文字列データに対する属性

属性	データ型	アクセス
background	gchararray	Write
underline	PangoUnderline	Read / Write

PangoUnderline は次のように定義されています．

```
typedef enum {
  PANGO_UNDERLINE_NONE,
  PANGO_UNDERLINE_SINGLE,
  PANGO_UNDERLINE_DOUBLE,
  PANGO_UNDERLINE_LOW,
  PANGO_UNDERLINE_ERROR
} PangoUnderline;
```

ソース 7-6-2 列属性の設定：gtkliststore-sample2.c

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

enum
{
  COLUMN_ID,
  COLUMN_NAME,
  COLUMN_NAME_COLOR,
  COLUMN_NAME_LINE,
  N_COLUMNS
};

typedef struct _ListData
{
  guint          id;
  gchar          *name;
  gchar          *color;
  PangoUnderline linestyle;
} ListData;

static ListData data[] =
{
  {0, "Mike", "Orange", PANGO_UNDERLINE_NONE},
  {1, "JoJo", "Red",    PANGO_UNDERLINE_SINGLE}
};

static void
add_data (GtkTreeView *treeview)
{
  GtkListStore *store;
  GtkTreeIter  iter;
  int          n;

  store = GTK_LIST_STORE (gtk_tree_view_get_model (treeview));

  for (n = 0; n < sizeof (data) / sizeof (data[0]); n++)
    {
      gtk_list_store_append (store, &iter);
      gtk_list_store_set (store, &iter,
                          COLUMN_ID,         data[n].id,
                          COLUMN_NAME,       data[n].name,
                          COLUMN_NAME_COLOR, data[n].color,
                          COLUMN_NAME_LINE,  data[n].linestyle, -1);
    }
}

static GtkWidget*
create_list_model (void)
{
  GtkWidget         *treeview;
  GtkListStore      *liststore;
  GtkCellRenderer   *renderer;
  GtkTreeViewColumn *column;

  liststore = gtk_list_store_new (N_COLUMNS,
                                  G_TYPE_UINT,
                                  G_TYPE_STRING,
                                  G_TYPE_STRING,
                                  G_TYPE_UINT);
  treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL (liststore));
  g_object_unref (liststore);

  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

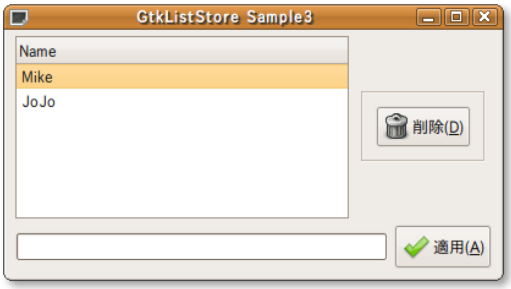


図 7.28 リストデータの削除

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．

```
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 7.29 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）．

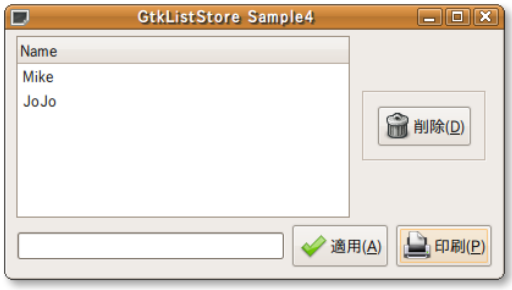


図 7.29　リストデータへの一括アクセス

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
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（4 行目），その次の行を関数 gtk_tree_model_iter_next で取得します（41, 47 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，21–22 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（24, 49 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) &&
21           gtk_tree_model_get_iter (GTK_TREE_MODEL (store),
22                                    &pre_iter, path))
23         {
24           gtk_list_store_swap (store, &pre_iter, &iter);
25         }
26     }
27 }
28
29 static void
30 down_list (GtkTreeView *treeview)
```

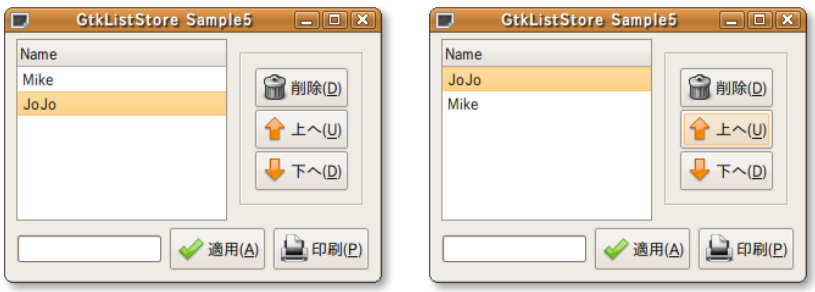


図 7.30　行の入れ換え

```
31 {
32   GtkListStore      *store;
33   GtkTreeSelection *selection;
34   GtkTreeIter       iter;
35   gboolean          success;
36
37   selection = gtk_tree_view_get_selection (treeview);
38   if (!selection) return;
39
40   store = GTK_LIST_STORE (gtk_tree_view_get_model (treeview));
41   success = gtk_tree_selection_get_selected (selection, NULL, &iter);
42   if (success)
43     {
44       GtkTreeIter next_iter;
45
46       next_iter = iter;
47       if (gtk_tree_model_iter_next (GTK_TREE_MODEL (store), &next_iter))
48         {
49           gtk_list_store_swap (store, &iter, &next_iter);
50         }
51     }
52 }
```

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 7.16 と表 7.17 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．

表 7.16 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．

図 7.31 GtkCellRenderer に対するコールバック関数の例

表 7.17　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.

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．

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，16–17 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,
17                       COLUMN_STATUS, !status, -1);
18   gtk_tree_path_free (path);
19 }
20
21 static void
22 cb_name_edited (GtkCellRendererText *renderer,
23                 const gchar         *path_string,
24                 const gchar         *new_text,
25                 gpointer            user_data)
26 {
27   GtkTreeModel *model;
28   GtkTreeIter  iter;
29   GtkTreePath  *path;
30   gboolean     status;
31
32   model = gtk_tree_view_get_model (GTK_TREE_VIEW (user_data));
33   path  = gtk_tree_path_new_from_string (path_string);
34
35   gtk_tree_model_get_iter (model, &iter, path);
36   gtk_list_store_set (GTK_LIST_STORE (model), &iter,
37                       COLUMN_NAME, new_text, -1);
38   gtk_tree_path_free (path);
39 }
40
41 static GtkWidget*
42 list_new (void)
43 {
44   GtkWidget         *treeview;
45   GtkListStore      *liststore;
46   GtkCellRenderer   *renderer;
47   GtkTreeViewColumn *column;
48
49   liststore = gtk_list_store_new (N_COLUMNS,
50                                   G_TYPE_BOOLEAN,
51                                   G_TYPE_STRING,
```

```
52                                      G_TYPE_BOOLEAN);
53   treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL (liststore));
54
55   renderer = gtk_cell_renderer_toggle_new ();
56   g_signal_connect (renderer, "toggled",
57                     G_CALLBACK (cb_status_toggled), treeview);
58   column =
59     gtk_tree_view_column_new_with_attributes ("Status", renderer,
60                                               "active", COLUMN_STATUS,
61                                               NULL);
62   gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
63
64   renderer = gtk_cell_renderer_text_new ();
65   g_signal_connect (renderer, "edited",
66                     G_CALLBACK (cb_name_edited), treeview);
67   column =
68     gtk_tree_view_column_new_with_attributes ("Name", renderer,
69                                               "text", COLUMN_NAME,
70                                               "editable",
71                                               COLUMN_EDITABLE,
72                                               NULL);
73   gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
74
75   return treeview;
76 }
```

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_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",
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

図 7.32 データのソート

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 7.33　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．

```
GtkCellRenderer* gtk_cell_renderer_pixbuf_new (void);
```

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

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

enum
{
  COLUMN_ICON,
  COLUMN_LABEL,
  N_COLUMNS
};

typedef struct _TreeData TreeData;
struct _TreeData
{
  gchar    *iconname;
  gchar    *label;
  TreeData *child;
};

static TreeData sublevel1[] =
{
  {"file.png", "file␣1-1", NULL},
  {"file.png", "file␣1-2", NULL},
  NULL
};

static TreeData sublevel2[] =
{
  {"file.png", "file␣2-1", NULL},
  {"file.png", "file␣2-2", NULL},
  NULL
};

static TreeData toplevel[] =
{
  {"folder.png", "Folder␣1", sublevel1},
  {"folder.png", "Folder␣2", sublevel2},
  NULL
};

static void
add_data (GtkTreeView *treeview)
{
  GtkTreeStore *store;
  TreeData     *top;

  store = GTK_TREE_STORE(gtk_tree_view_get_model (treeview));

  top = toplevel;
  while (top->iconname)
    {
      TreeData    *child;
      GdkPixbuf   *pixbuf;
      GtkTreeIter iter;

      child = top->child;
      pixbuf = gdk_pixbuf_new_from_file (top->iconname, NULL);
      gtk_tree_store_append (store, &iter, NULL);
      gtk_tree_store_set (store, &iter,
                          COLUMN_ICON,  pixbuf,
                          COLUMN_LABEL, top->label, -1);
      while (child->iconname)
        {
          GtkTreeIter child_iter;

          pixbuf = gdk_pixbuf_new_from_file (child->iconname, NULL);
          gtk_tree_store_append (store, &child_iter, &iter);
          gtk_tree_store_set (store, &child_iter,
                              COLUMN_ICON,  pixbuf,
                              COLUMN_LABEL, child->label, -1);
          child++;
        }
      top++;
    }
```

```
 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",
 94                                       COLUMN_ICON);
 95
 96   renderer = gtk_cell_renderer_text_new ();
 97   gtk_tree_view_column_pack_start (column, renderer, TRUE);
 98   gtk_tree_view_column_add_attribute (column, renderer, "text",
 99                                       COLUMN_LABEL);
100
101   gtk_tree_view_append_column (GTK_TREE_VIEW (treeview), column);
102
103   return treeview;
104 }
105
106 int
107 main (int argc, char **argv)
108 {
109   GtkWidget *window;
110   GtkWidget *treeview;
111
112   gtk_init (&argc, &argv);
113
114   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
115   gtk_window_set_title (GTK_WINDOW (window), "GtkTreeStore␣Sample1");
116   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
117   g_signal_connect (G_OBJECT (window), "destroy",
118                     G_CALLBACK (gtk_main_quit), NULL);
119   gtk_widget_set_size_request (window, 280, 240);
120
121   treeview = create_tree_model ();
122   gtk_tree_view_set_headers_visible (GTK_TREE_VIEW (treeview), FALSE);
123   gtk_container_add (GTK_CONTAINER (window), treeview);
124
125   add_data (GTK_TREE_VIEW (treeview));
126
127   gtk_widget_show_all (window);
128   gtk_main ();
129
130   return 0;
131 }
```

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 7.34）．

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 7.34 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_hbox_new (TRUE, 0);
18   gtk_container_add (GTK_CONTAINER (window), hbox);
19
20   tooltips = gtk_tooltips_new ();
21
22   button = gtk_button_new_from_stock (GTK_STOCK_NEW);
23   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
24   gtk_widget_set_tooltip_text (button, "Run␣new␣program");
25
26   button = gtk_button_new_from_stock (GTK_STOCK_OPEN);
27   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
28   gtk_widget_set_tooltip_text (button, "Open␣a␣file");
29
30   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
31   gtk_box_pack_start (GTK_BOX (hbox), button, TRUE, TRUE, 0);
32   gtk_widget_set_tooltip_text (button, "Quit␣this␣program");
33
34   g_signal_connect (G_OBJECT (button), "clicked",
35                     G_CALLBACK (gtk_main_quit), NULL);
36
37   gtk_widget_show_all (window);
38   gtk_main ();
39
40   return 0;
41 }
```

図 7.34　ツールチップ

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 7.35 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
          +----GtkObject
                +----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．

  ```
  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_fracktion によって，値を更新できます．

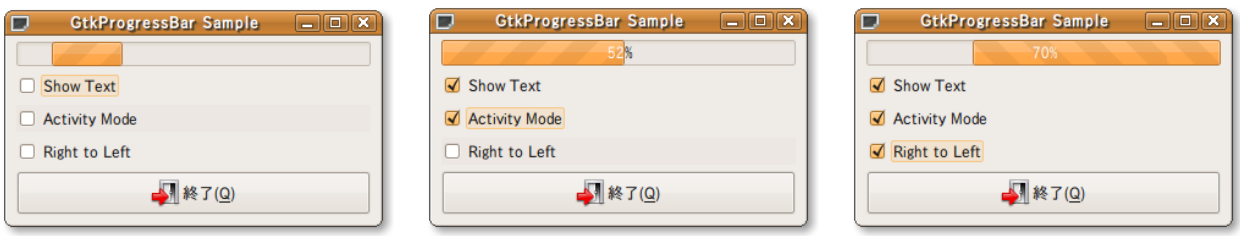


図 7.35　プログレスバー

```
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モードがプログレッシブモードの場合，関数 gtk_progress_bar_set_orientation により，プログレスバーの進行方向を指定できます．また，関数 gtk_progress_bar_get_orientation によって，現在の設定を取得できます．

```
void
gtk_progress_bar_set_orientation (GtkProgressBar *pbar,
                                  GtkProgressBarOrientation
                                                    orientation);
```

  GtkProgressBarOrientation は次のように定義されています．

```
typedef enum
{
  GTK_PROGRESS_LEFT_TO_RIGHT,
  GTK_PROGRESS_RIGHT_TO_LEFT,
  GTK_PROGRESS_BOTTOM_TO_TOP,
  GTK_PROGRESS_TOP_TO_BOTTOM
} GtkProgressBarOrientation;
```

- プログレスバーに表示するテキスト
  プログレスバーがプログレッシブモードの場合は，プログレスバー上にテキストを表示できます．進行状況をプログレスバーの長さと同時に，テキストで何 % と表示すると，よりわかりやすくなります．プログレスバー上にテキストを表示するには，関数 gtk_progress_bar_set_text を使用します．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);
G_CONST_RETURN 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 7.35 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. 63）．

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    if (show_text)
37      {
38        gtk_progress_bar_set_text (GTK_PROGRESS_BAR (user_data), "");
39        show_text = FALSE;
40      }
41    else
42      {
43        show_text = TRUE;
44      }
45  }
46
47  static void
48  cb_activity_mode (GtkToggleButton *widget, gpointer user_data)
49  {
50    activity_mode = gtk_toggle_button_get_active (widget);
51    if (activity_mode)
52      {
53        gtk_progress_bar_set_fraction (GTK_PROGRESS_BAR (user_data), 0.0);
54      }
55    else
56      {
57        gtk_progress_bar_pulse (GTK_PROGRESS_BAR (user_data));
58      }
59  }
60
61  static void
62  cb_orientation (GtkToggleButton *widget, gpointer user_data)
63  {
64    GtkProgressBarOrientation orientation;
65
66    orientation =
67      gtk_progress_bar_get_orientation (GTK_PROGRESS_BAR (user_data));
68    switch (orientation)
69      {
70      case GTK_PROGRESS_LEFT_TO_RIGHT:
71        gtk_progress_bar_set_orientation (GTK_PROGRESS_BAR (user_data),
72                                          GTK_PROGRESS_RIGHT_TO_LEFT);
73        break;
74      case GTK_PROGRESS_RIGHT_TO_LEFT:
75        gtk_progress_bar_set_orientation (GTK_PROGRESS_BAR (user_data),
76                                          GTK_PROGRESS_LEFT_TO_RIGHT);
77        break;
78      default:
79        break;
80      }
81  }
82
83  static void
84  cb_quit (GtkButton *widget, gpointer data)
85  {
86    g_source_remove (timer);
87    gtk_main_quit ();
88  }
89
90  int
91  main (int argc, char **argv)
92  {
93    GtkWidget *window;
94    GtkWidget *vbox;
95    GtkWidget *progressbar;
96    GtkWidget *button;
97
98    gtk_init (&argc, &argv);
99
```

```
100   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
101   gtk_window_set_title  (GTK_WINDOW (window), "GtkProgressBar␣Sample");
102   gtk_widget_set_size_request (window, 300, -1);
103   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
104   g_signal_connect (G_OBJECT (window), "destroy",
105                     G_CALLBACK (cb_quit), NULL);
106
107   vbox = gtk_vbox_new (FALSE, 5);
108   gtk_container_add (GTK_CONTAINER (window), vbox);
109
110   progressbar = gtk_progress_bar_new ();
111   gtk_box_pack_start (GTK_BOX (vbox), progressbar, FALSE, FALSE, 0);
112
113   button = gtk_check_button_new_with_label ("Show␣Text");
114   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
115   g_signal_connect (G_OBJECT (button), "clicked",
116                     G_CALLBACK (cb_show_text), progressbar);
117
118   button = gtk_check_button_new_with_label ("Activity␣Mode");
119   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
120   g_signal_connect (G_OBJECT (button), "clicked",
121                     G_CALLBACK (cb_activity_mode), progressbar);
122
123   button = gtk_check_button_new_with_label ("Right␣to␣Left");
124   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
125   g_signal_connect (G_OBJECT (button), "clicked",
126                     G_CALLBACK (cb_orientation), progressbar);
127
128   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
129   gtk_box_pack_start (GTK_BOX (vbox), button, FALSE, FALSE, 0);
130   g_signal_connect (G_OBJECT (button), "clicked",
131                     G_CALLBACK (gtk_main_quit), NULL);
132
133   timer = g_timeout_add (100, progressbar_update, progressbar);
134
135   gtk_widget_show_all (window);
136   gtk_main ();
137
138   return 0;
139 }
```

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
          +----GtkObject
                +----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次の関数を使用します．

- gtk_hseparator_new
  横方向のセパレータを作成します．つまり垂直ボックスに配置して上下のウィジェットを仕切る場合に用います．

```
GtkWidget* gtk_hseparator_new (void);
```

- gtk_vseparator_new
  縦方向のセパレータを作成します．

```
GtkWidget* gtk_vseparator_new (void);
```

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 7.36）．

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
           +----GtkObject
                  +----GtkWidget
                         +----GtkRange
                                +----GtkScale
                                       +----GtkHScale
                                       +----GtkVScale
```

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_hscale_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_hscale_new (GtkAdjustment *adjustment);
```

- gtk_hscale_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_hscale_new_with_range (gdouble min,
                                      gdouble max,
                                      gdouble step);
```

- gtk_vscale_new
  GtkAdjustment で数値の範囲等を設定し，関数の引数に指定して垂直スケールを作成します．

```
GtkWidget* gtk_vscale_new (GtkAdjustment *adjustment);
```

- gtk_vscale_new_with_range
  数値の範囲とステップ幅を引数に指定して，垂直スケールを作成します．

```
GtkWidget* gtk_vscale_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．

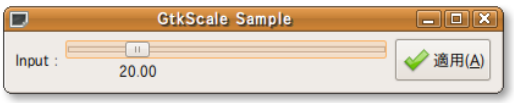


図 7.36　スケール

- 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

```
1  #include <gtk/gtk.h>
2
3  static void
4  cb_value_changed (GtkScale *scale, gpointer user_data)
5  {
6    g_print ("value␣=␣%f\n", gtk_range_get_value (GTK_RANGE (scale)));
7  }
8
9  static void
10 cb_button (GtkButton *button, gpointer user_data)
11 {
12   g_print ("value␣=␣%f\n", gtk_range_get_value (GTK_RANGE (user_data)));
13 }
14
15 int
16 main (int argc, char **argv)
17 {
18   GtkWidget *window;
19   GtkWidget *hbox;
20   GtkWidget *label;
21   GtkWidget *scale;
22   GtkWidget *button;
23   gdouble   min = 0.0, max = 100.0, step = 0.1;
24
25   gtk_init (&argc, &argv);
26
27   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
28   gtk_window_set_title (GTK_WINDOW (window), "GtkScale␣Sample");
```

```
29   gtk_widget_set_size_request (window, 400, -1);
30   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
31   g_signal_connect (G_OBJECT (window), "destroy",
32                     G_CALLBACK (gtk_main_quit), NULL);
33
34   hbox = gtk_hbox_new (FALSE, 5);
35   gtk_container_add (GTK_CONTAINER (window), hbox);
36
37   label = gtk_label_new ("Input␣:");
38   gtk_box_pack_start (GTK_BOX (hbox), label, FALSE, FALSE, 0);
39
40   scale = gtk_hscale_new_with_range (min, max, step);
41   gtk_scale_set_digits (GTK_SCALE (scale), 2);
42   gtk_scale_set_draw_value (GTK_SCALE (scale), TRUE);
43   gtk_scale_set_value_pos (GTK_SCALE (scale), GTK_POS_BOTTOM);
44
45   g_signal_connect (G_OBJECT (scale), "value-changed",
46                     G_CALLBACK (cb_value_changed), NULL);
47   gtk_box_pack_start (GTK_BOX (hbox), scale, TRUE, TRUE, 0);
48
49   button = gtk_button_new_from_stock (GTK_STOCK_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 7.37）．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．

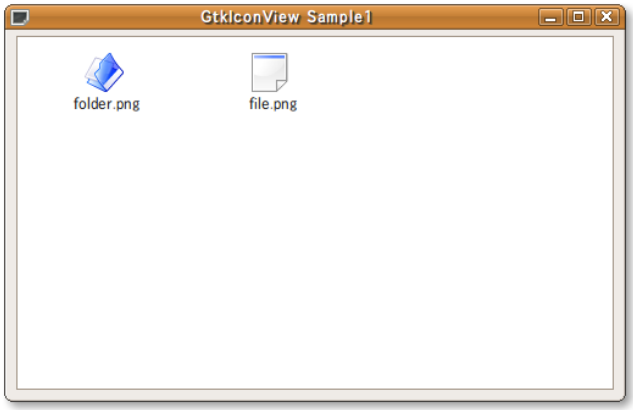


図 7.37　アイコンビューウィジェットの例

オブジェクトの階層構造

```
GObject
    +----GInitiallyUnowned
          +----GtkObject
                +----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);
```

表 7.18　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.

```
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;
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.

```
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. 180）を参照してください.

シグナルとコールバック関数

表 7.18 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 7.38 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);
```

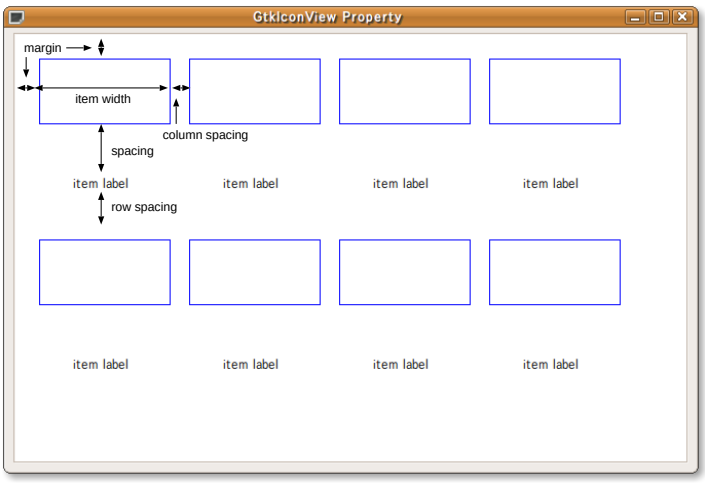


図 7.38　アイコンビューウィジェットのプロパティ

```
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 の値で指定します．

```
void gtk_icon_view_set_selection_mode (GtkIconView *icon_view,
                                       GtkSelectionMode mode);

GtkSelectionMode
gtk_icon_view_get_selection_mode (GtkIconView *icon_view);

typedef enum
{
  GTK_SELECTION_NONE,
  GTK_SELECTION_SINGLE,
  GTK_SELECTION_BROWSE,
  GTK_SELECTION_MULTIPLE,
  GTK_SELECTION_EXTENDED = GTK_SELECTION_MULTIPLE
} GtkSelectionMode;
```

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

```
1  #include <gtk/gtk.h>
2
3  enum
4  {
5    COLUMN_NAME,
6    COLUMN_PIXBUF,
7    N_COLUMNS
8  };
9
10 static void
11 add_data (GtkIconView *iconview)
12 {
13   GdkPixbuf    *folder_pixbuf;
14   GdkPixbuf    *file_pixbuf;
15   GtkListStore *store;
16   GtkTreeIter  iter;
17
18   file_pixbuf   = gdk_pixbuf_new_from_file ("./file.png", NULL);
19   folder_pixbuf = gdk_pixbuf_new_from_file ("./folder.png", NULL);
20
21   store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));
```

```
22
23    gtk_list_store_clear (store);
24
25    gtk_list_store_append (store, &iter);
26    gtk_list_store_set (store, &iter,
27                        COLUMN_NAME, "folder.png",
28                        COLUMN_PIXBUF, folder_pixbuf, -1);
29    g_object_unref (folder_pixbuf);
30
31    gtk_list_store_append (store, &iter);
32    gtk_list_store_set (store, &iter,
33                        COLUMN_NAME, "file.png",
34                        COLUMN_PIXBUF, file_pixbuf, -1);
35    g_object_unref (file_pixbuf);
36  }
37
38  static GtkWidget*
39  create_icon_view_widget (void)
40  {
41    GtkWidget    *iconview;
42    GtkListStore *store;
43
44    store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING, GDK_TYPE_PIXBUF);
45    iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL (store));
46    g_object_unref (store);
47
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,
63                        COLUMN_NAME, &name, -1);
64    g_print ("item␣'%s'␣is␣clicked.\n", name);
65    g_free (name);
66  }
67
68  int
69  main (int argc, char **argv)
70  {
71    GtkWidget *window;
72    GtkWidget *scroll_window;
73    GtkWidget *iconview;
74
75    gtk_init (&argc, &argv);
76
77    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
78    gtk_window_set_title (GTK_WINDOW (window), "GtkIconView␣Sample1");
79    gtk_widget_set_size_request (window, 500, 300);
80    gtk_container_set_border_width (GTK_CONTAINER (window), 5);
81    g_signal_connect (G_OBJECT (window), "destroy",
82                      G_CALLBACK (gtk_main_quit), NULL);
83
84    scroll_window = gtk_scrolled_window_new (NULL, NULL);
85    gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW
86                                         (scroll_window),
87                                         GTK_SHADOW_ETCHED_IN);
88    gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scroll_window),
89                                    GTK_POLICY_AUTOMATIC,
90                                    GTK_POLICY_AUTOMATIC);
91    gtk_container_add (GTK_CONTAINER (window), scroll_window);
92
93    iconview = create_icon_view_widget ();
94    gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview),
95                                   COLUMN_NAME);
96    gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW(iconview),
97                                     COLUMN_PIXBUF);
```

```
98    gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 128);
99    g_signal_connect (G_OBJECT (iconview), "item-activated",
100                      G_CALLBACK (cb_item_activated), NULL);
101
102   gtk_container_add (GTK_CONTAINER (scroll_window), iconview);
103
104   add_data (GTK_ICON_VIEW(iconview));
105
106   gtk_widget_show_all (window);
107   gtk_main ();
108
109   return 0;
110 }
```

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．

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);

typedef enum
{
  GTK_SORT_ASCENDING,
  GTK_SORT_DESCENDING
} GtkSortType;
```

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);
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)
```

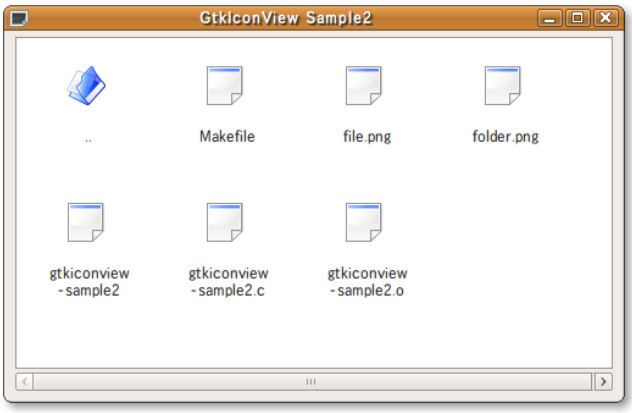


図 7.39　簡易ファイルブラウザ

```
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␣Sample2");
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 7.40 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

```
1  #include <gtk/gtk.h>
2
3  enum
4  {
5    COLUMN_COMPLETION_TEXT,
6    N_COLUMNS
7  };
8
9  static GtkEntryCompletion*
10 create_completion_widget (void)
11 {
12   GtkEntryCompletion *completion;
13   GtkListStore       *store;
14   GtkTreeIter        iter;
15
16   completion = gtk_entry_completion_new ();
17   store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING);
18   gtk_entry_completion_set_model (completion, GTK_TREE_MODEL (store));
```

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 1

```
19   g_object_unref (store);
20   gtk_entry_completion_set_text_column (completion, 0);
21
22   gtk_list_store_append (store, &iter);
23   gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "a", -1);
24
25   gtk_list_store_append (store, &iter);
26   gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "ai", -1);
27
28   gtk_list_store_append (store, &iter);
29   gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "aiu", -1);
30
31   return completion;
32 }
33
34 int
35 main (int argc, char **argv)
36 {
37   GtkWidget          *window;
38   GtkWidget          *entry;
39   GtkEntryCompletion *completion;
40
41   gtk_init (&argc, &argv);
42
43   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
44   gtk_window_set_title (GTK_WINDOW (window),
45                         "GtkEntryCompletion␣Sample1");
46   gtk_widget_set_size_request (window, 320, -1);
47   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
48   g_signal_connect (G_OBJECT (window), "destroy",
49                     G_CALLBACK (gtk_main_quit), NULL);
50
51   entry = gtk_entry_new ();
52   gtk_container_add (GTK_CONTAINER (window), entry);
53
54   completion = create_completion_widget ();
55   gtk_entry_set_completion (GTK_ENTRY (entry), completion);
56   g_object_unref (completion);
57
58   gtk_widget_show_all (window);
59   gtk_main ();
60
61   return 0;
62 }
```

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 7.41）．

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
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] != '.')
```

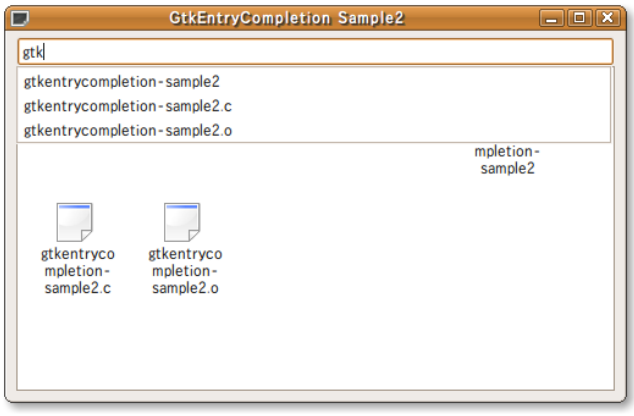


図 7.41　エントリ補完ウィジェットのサンプルプログラム 2

```
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,
119                       -1);
120   gtk_tree_model_get (model, b,
121                       COLUMN_IS_DIRECTORY, &is_dir_b,
122                       COLUMN_DISPLAY_NAME, &name_b,
123                       -1);
124   if (!is_dir_a && is_dir_b)
125     {
126       result = 1;
127     }
128   else if (is_dir_a && !is_dir_b)
129     {
130       result = -1;
131     }
132   else
133     {
134       result = g_utf8_collate (name_a, name_b);
135     }
136   return result;
137 }
138
139 static GtkWidget*
140 create_icon_view_widget (void)
141 {
142   GtkWidget    *iconview;
143   GtkListStore *store;
144
145   currentdir = g_get_current_dir ();
146   store = gtk_list_store_new (N_ICONVIEW_COLUMNS,
147                               G_TYPE_STRING,
```

```
148                               G_TYPE_STRING,
149                               GDK_TYPE_PIXBUF,
150                               G_TYPE_BOOLEAN);
151   gtk_tree_sortable_set_default_sort_func (GTK_TREE_SORTABLE (store),
152                                            sort_func, NULL, NULL);
153   gtk_tree_sortable_set_sort_column_id
154     (GTK_TREE_SORTABLE (store),
155      GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID, GTK_SORT_ASCENDING);
156
157   iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL (store));
158   g_object_unref (store);
159
160   return iconview;
161 }
162
163 static void
164 cb_item_activated (GtkIconView *iconview,
165                    GtkTreePath *treepath,
166                    gpointer    data)
167 {
168   GtkListStore          *store;
169   GtkTreeIter           iter;
170   gchar                 *path;
171   gboolean              is_dir;
172   GtkEntryCompletion    *completion;
173
174   store = GTK_LIST_STORE (gtk_icon_view_get_model (iconview));
175   gtk_tree_model_get_iter (GTK_TREE_MODEL (store), &iter, treepath);
176   gtk_tree_model_get (GTK_TREE_MODEL (store), &iter,
177                       COLUMN_PATH, &path,
178                       COLUMN_IS_DIRECTORY, &is_dir,
179                       -1);
180   if (is_dir)
181     {
182       g_free (currentdir);
183       currentdir = g_strdup (path);
184       add_data (iconview, GTK_ENTRY_COMPLETION (data));
185     }
186   g_free (path);
187 }
188
189 int
190 main (int argc, char **argv)
191 {
192   GtkWidget          *window;
193   GtkWidget          *vbox;
194   GtkWidget          *entry;
195   GtkWidget          *scrolled_window;
196   GtkWidget          *iconview;
197   GtkEntryCompletion *completion;
198
199   gtk_init (&argc, &argv);
200
201   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
202   gtk_window_set_title (GTK_WINDOW (window),
203                         "GtkEntryCompletion␣Sample2");
204   gtk_widget_set_size_request (window, 500, 300);
205   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
206   g_signal_connect (G_OBJECT (window), "destroy",
207                     G_CALLBACK (gtk_main_quit), NULL);
208
209   vbox = gtk_vbox_new (FALSE, 0);
210   gtk_container_add (GTK_CONTAINER (window), vbox);
211
212   entry = gtk_entry_new ();
213   gtk_box_pack_start (GTK_BOX (vbox), entry, FALSE, FALSE, 0);
214
215   completion = create_completion_widget ();
216   gtk_entry_set_completion (GTK_ENTRY (entry), completion);
217   g_object_unref (completion);
218
219   scrolled_window = gtk_scrolled_window_new (NULL, NULL);
220   gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW
221                                        (scrolled_window),
222                                        GTK_SHADOW_ETCHED_IN);
223   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolled_window),
```

```
224                                     GTK_POLICY_AUTOMATIC,
225                                     GTK_POLICY_AUTOMATIC);
226   gtk_box_pack_start (GTK_BOX (vbox), scrolled_window, TRUE, TRUE, 0);
227
228   load_pixbuf ();
229
230   iconview = create_icon_view_widget ();
231   gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview),
232                                  COLUMN_DISPLAY_NAME);
233   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW (iconview),
234                                    COLUMN_PIXBUF);
235   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 80);
236   g_signal_connect (G_OBJECT (iconview), "item_activated",
237                     G_CALLBACK (cb_item_activated), completion);
238
239   gtk_container_add (GTK_CONTAINER (scrolled_window), iconview);
240
241   add_data (GTK_ICON_VIEW (iconview), completion);
242
243   gtk_widget_show_all (window);
244   gtk_main ();
245
246   return 0;
247 }
```

第 8 章

拡張ウィジェットの作成

本章では独自ウィジェットの作成方法について解説します．全く新しいウィジェットを作成することもできますが，ここでは実用上要求が高いと思われる既存ウィジェットをベースとした拡張ウィジェットの作成方法を，具体例を挙げて説明します．

8.1 アイコン付きボタンウィジェットの作成

ここでは図 8.1 に示すようなアイコンとラベルが 1 つになったアイコン付きボタンウィジェット（GtkIconButton）を作成しながら，拡張ウィジェットの作成方法を説明します．

作成するアイコン付きボタンウィジェットには，以下のような機能を実装することにします．

- ボタン中にアイコンを挿入できる．アイコンデータは次の 4 種類から指定できる．
  - PNG フォーマット，JPEG フォーマットの画像ファイル
  - GtkImage ウィジェット
  - GdkPixbuf データ
  - インラインデータ
- ラベルの配置位置をアイコンの上，下，右，左のいずれかから指定できる．

そして，このウィジェットの関数として次のような関数を実装します．

- ウィジェット作成関数
  - gtk_icon_button_new
  - gtk_icon_button_new_from_image
  - gtk_icon_button_new_from_pixbuf
  - gtk_icon_button_new_from_inline
  - gtk_icon_button_new_with_label
  - gtk_icon_button_new_from_image_label
  - gtk_icon_button_new_from_pixbuf_label
  - gtk_icon_button_new_from_inline_label
  - gtk_icon_button_new_with_mnemonic

図 8.1　アイコン付きボタンウィジェット

  - gtk_icon_button_new_from_image_mnemonic
  - gtk_icon_button_new_from_pixbuf_mnemonic
  - gtk_icon_button_new_from_inline_mnemonic
- プロパティ取得関数
  - gtk_icon_button_get_label
  - gtk_icon_button_get_icon
  - gtk_icon_button_get_text_position
- プロパティ変更関数
  - gtk_icon_button_set_label
  - gtk_icon_button_set_icon
  - gtk_icon_button_set_text_position

8.2　ヘッダファイルの作成

アイコン付きボタンウィジェットは，ボタンウィジェットの拡張ウィジェットなので，ボタンウィジェットのヘッダファイル（gtkbutton.h）を参考にして，アイコン付きボタンウィジェット（gtkiconbutton.h）を作成することにします．gtkbutton.h の一部をソース 8–1 に示します．

ヘッダファイルのインクルード制御（1–2 行目）

この記述は，このヘッダファイルを複数のファイルにインクルードするのを防ぐための記述です．このファイルをインクルードしたファイル内でマクロ__GTK_BUTTON__が定義されていない場合は，マクロ__GTK_BUTTON__を定義して，それ以降の記述を有効にします．

このマクロを GtkIconButton 用に__GTK_ICON_BUTTON__と修正します．

C++ への対応（7，22 行目）

この記述は，このヘッダファイルを C++ のソースコードにインクルードする場合に必要な記述です．

ソース 8-1 GtkButton ウィジェットのヘッダファイル：gtkbutton.h から一部抜粋

```
 1 #ifndef __GTK_BUTTON_H__
 2 #define __GTK_BUTTON_H__
 3
 4 #include <gtk/gtkbin.h>
 5 #include <gtk/gtkimage.h>
 6
 7 G_BEGIN_DECLS
 8
 9 #define GTK_TYPE_BUTTON             (gtk_button_get_type ())
10 #define GTK_BUTTON(obj)             (G_TYPE_CHECK_INSTANCE_CAST ((obj), \
11                                      GTK_TYPE_BUTTON, GtkButton))
12 #define GTK_BUTTON_CLASS(klass)     (G_TYPE_CHECK_CLASS_CAST ((klass), \
13                                      GTK_TYPE_BUTTON, GtkButtonClass))
14 #define GTK_IS_BUTTON(obj)          (G_TYPE_CHECK_INSTANCE_TYPE ((obj), \
15                                      GTK_TYPE_BUTTON))
16 #define GTK_IS_BUTTON_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE ((klass), \
17                                      GTK_TYPE_BUTTON))
18 #define GTK_BUTTON_GET_CLASS(obj)  (G_TYPE_INSTANCE_GET_CLASS ((obj), \
19                                      GTK_TYPE_BUTTON, GtkButtonClass))
20  ...
21
22 G_END_DECLS
23
24 #endif /* __GTK_BUTTON_H__ */
```

ウィジェット専用マクロ定義（9–19 行目）

ここで定義されているのは，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．これらのマクロも名前を修正して，GtkIconButton 用に書き換えます．

これに加えて，GtkIconButton で使用するアイコンデータのソースタイプを表す列挙体を定義します．定義した列挙体には，typedef を用いて GtkIconButtonSource という名前をつけておきます．

```
typedef enum
{
  GTK_ICON_BUTTON_SOURCE_FILE,
  GTK_ICON_BUTTON_SOURCE_INLINE,
  GTK_ICON_BUTTON_SOURCE_IMAGE,
  GTK_ICON_BUTTON_SOURCE_PIXBUF
} GtkIconButtonSource;
```

また，ラベルテキストの配置位置を表す列挙体には，既に定義されている GtkPositionType を使用することにします．

```
enum
{
  GTK_POS_LEFT,
  GTK_POS_RIGHT,
  GTK_POS_TOP,
  GTK_POS_BOTTOM
} GtkPositionType;
```

修正を行ったヘッダファイル gtkiconbutton.h の一部をソース 8–2 に示します．

ソース 8-2 GtkIconButton のヘッダファイル：gtkiconbutton.h から一部抜粋

```
#ifndef __GTK_ICON_BUTTON_H__
#define __GTK_ICON_BUTTON_H__

#include <gtk/gtk.h>

G_BEGIN_DECLS

#define GTK_TYPE_ICON_BUTTON          (gtk_icon_button_get_type ())
#define GTK_ICON_BUTTON(obj)          (G_TYPE_CHECK_INSTANCE_CAST((obj),\
                                       GTK_TYPE_ICON_BUTTON,\
                                       GtkIconButton))
#define GTK_ICON_BUTTON_CLASS(klass) (G_TYPE_CHECK_CLASS_CAST((klass),\
                                       GTK_TYPE_ICON_BUTTON,\
                                       GtkIconButtonClass))
#define GTK_IS_ICON_BUTTON(obj)       (G_TYPE_CHECK_INSTANCE_TYPE((obj),\
                                       GTK_TYPE_ICON_BUTTON))
#define GTK_IS_ICON_BUTTON_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE\
                                          ((klass), \
                                          GTK_TYPE_ICON_BUTTON))
#define GTK_ICON_BUTTON_GET_CLASS(obj) (G_TYPE_INSTANCE_GET_CLASS((obj),\
                                         GTK_TYPE_ICON_BUTTON,\
                                         GtkIconButtonClass))

typedef enum
{
  GTK_ICON_BUTTON_SOURCE_FILE,
  GTK_ICON_BUTTON_SOURCE_INLINE,
  GTK_ICON_BUTTON_SOURCE_IMAGE,
  GTK_ICON_BUTTON_SOURCE_PIXBUF
} GtkIconButtonSource;

 ...

G_END_DECLS

#endif  /* __GTK_ICON_BUTTON_H__ */
```

8.3　クラス構造体の定義

次にクラス構造体を定義します．まずウィジェットの構造体ですが，このウィジェットの親クラスは GtkButton とするので，構造体の最初のメンバは GtkButton 型の変数とします．そして拡張部分のメンバとして，ラベルテキストの配置位置を表す変数 text_position，アイコンデータのソースタイプを表す変数 icon_source，アイコンデータを表す変数 pixbuf を定義します．クラス構造体のメンバには親クラスの変数のみ与えます．

ソース 8-3 GtkIconButton の構造体定義：gtkiconbutton.h から一部抜粋

```
1  typedef struct  _GtkIconButton      GtkIconButton;
2  typedef struct  _GtkIconButtonClass GtkIconButtonClass;
3
4  struct _GtkIconButton
5  {
6    GtkButton button;
7  };
8
9  struct _GtkIconButtonClass
10 {
11   GtkButtonClass parent_class;
12 };
```

8.4　初期化関数

まず，型を定義するためのマクロを次のように設定します．3 番目の値で，このウィジェットの親ウィジェットが GtkButton ウィジェットであることを表しています．

```
G_DEFINE_TYPE (GtkIconButton, gtk_icon_button, GTK_TYPE_BUTTON)
```

次に初期化関数を定義します．初期化関数にはクラスの初期化関数 gtk_icon_button_class_init と，ウィジェット初期化関数 gtk_icon_button_init があります．クラス初期化関数では，このウィジェット内部で扱う変数をまとめた構造体を追加しています．ウィジェット初期化関数では変数の初期化のみを行います．

ソース 8-4 初期化関数

```
1  static void
2  gtk_icon_button_class_init (GtkIconButtonClass *klass)
3  {
4    GObjectClass *gobject_class;
5
6    gobject_class  = G_OBJECT_CLASS (klass);
7
8    g_type_class_add_private (gobject_class,
9                              sizeof (GtkIconButtonPrivate));
10 }
11
12 static void
13 gtk_icon_button_init (GtkIconButton *button)
14 {
15   GtkIconButtonPrivate *priv = GTK_ICON_BUTTON_GET_PRIVATE (button);
16
17   priv->pixbuf = NULL;
18   priv->text_pos = GTK_POS_RIGHT;
19   priv->icon_source = GTK_ICON_BUTTON_SOURCE_FILE;
20 }
```

8.5　ウィジェット作成関数

実際にウィジェットを作成する関数は，はじめに示したように引数にさまざまなバリエーションがあります．そこですべての引数を包括するローカルなウィジェット作成関数を実装して，それぞれのウィジェット関数からはこのローカル関数を呼び出すようにします（ソース 8–5）．この関数内での処理は以下のようになります．

- ラベルテキストの設定
- プロパティの設定
- アイコンデータの作成（関数_set_icon の呼び出し）
- コンポーネントの作成（関数 gtk_icon_button_construct_child の呼び出し）

ソース 8-5 ウィジェット作成関数

```
1  static GtkWidget*
2  _gtk_icon_button_new (const gchar      *label_text,
3                        guint            use_underline,
4                        GtkPositionType  text_pos,
5                        guint            icon_source,
6                        gpointer         user_data)
7  {
8    GtkIconButtonPrivate *priv;
9    GtkIconButton        *ibutton;
10   GtkWidget            *widget;
11   GtkButton            *button;
12
13   widget  = g_object_new (GTK_TYPE_ICON_BUTTON, NULL);
14   ibutton = GTK_ICON_BUTTON (widget);
15   button  = GTK_BUTTON(ibutton);
16   priv    = GTK_ICON_BUTTON_GET_PRIVATE (ibutton);
17
18   if (label_text)
19     {
20       button->label_text = g_strdup (label_text);
21     }
22   else
23     {
24       button->label_text = NULL;
25     }
26   button->use_underline = use_underline;
27   priv->text_pos    = text_pos;
28   priv->icon_source     = icon_source;
29   _set_icon (ibutton, user_data);
30
31   gtk_icon_button_construct_child (ibutton);
32
33   return widget;
34 }
```

アイコンデータ作成関数

アイコンデータを作成する部分は，別の関数_set_icon（ソース 8–6）を呼び出します．ここではアイコンデータの種類によって 4 通りの方法でアイコンデータ（GdkPixbuf データ）を作成します．

ソース 8-6 アイコンデータ作成関数

```
1  static void
2  _set_icon (GtkIconButton *ibutton, gpointer user_data)
3  {
4    GtkIconButtonPrivate *priv;
5
6    priv = GTK_ICON_BUTTON_GET_PRIVATE (ibutton);
7
8    if (priv->pixbuf) g_object_unref (priv->pixbuf);
9
10   switch (priv->icon_source)
11     {
12     case GTK_ICON_BUTTON_SOURCE_FILE:
13       priv->pixbuf
14         = gdk_pixbuf_new_from_file ((gchar *) user_data, NULL);
15       break;
16     case GTK_ICON_BUTTON_SOURCE_INLINE:
17       priv->pixbuf
18         = gdk_pixbuf_new_from_inline (-1, (guint8 *) user_data,
19                                       TRUE, NULL);
20       break;
21     case GTK_ICON_BUTTON_SOURCE_IMAGE:
22       priv->pixbuf = gtk_image_get_pixbuf (GTK_IMAGE (user_data));
23       break;
24     case GTK_ICON_BUTTON_SOURCE_PIXBUF:
25       priv->pixbuf = g_object_ref (user_data);
26       break;
27     }
28 }
```

コンポーネント作成関数

そしてアイコン用のウィジェットやラベルを作成するために，次の関数 gtk_icon_button_construct_child（ソース 8–7）を呼び出します．

このようにコンポーネントウィジェットを作成する関数を別に作成する方法は，GtkButton のウィジェット作成方法を引き継いでいます．これは，ウィジェットのプロパティが変更されてウィジェットを作成し直さなければならなくなったときに，この関数を呼び出すことでコンポーネントを作成し直すことができるからです．この関数も，gtkbutton.c 内の gtk_button_construct_child に若干の修正を加えて作成しました．

実は，GtkButton ウィジェットでもストックアイコンを指定してアイコン付きボタンを実現できますが，GtkButton ウィジェットではアイコンをラベルテキストの左側にしか配置できません．GtkIconButton ウィジェットでは，21–22 行目でラベルテキストの配置位置を示すプロパティ text_position によって，水平ボックスを生成するか垂直ボックスを生成するかを切り替えています．そして，58–59 行目でプロパティ text_position によって，アイコンとテキストラベルを配置する順番をコントロールしています．

ソース 8-7 コンポーネント作成関数 gtk_icon_button_construct_child

```
1  static void
2  gtk_icon_button_construct_child (GtkIconButton *ibutton)
3  {
4    GtkButton            *button;
5    GtkIconButtonPrivate *priv;
6    GtkWidget            *child;
7    GtkWidget            *label = NULL;
8    GtkWidget            *box;
9    GtkWidget            *align;
10   const gchar          *label_text;
11
12   button = GTK_BUTTON (ibutton);
13   priv   = GTK_ICON_BUTTON_GET_PRIVATE (ibutton);
14   child  = gtk_bin_get_child (GTK_BIN (button));
15
16   if (child)
17     {
18       gtk_container_remove (GTK_CONTAINER (button), child);
19     }
20   if (priv->text_pos == GTK_POS_TOP ||
21       priv->text_pos == GTK_POS_BOTTOM)
22     {
23       box = gtk_vbox_new (FALSE, 2);
24     }
25   else
26     {
27       box = gtk_hbox_new (FALSE, 2);
28     }
29   label_text = gtk_button_get_label (button);
30   if (label_text)
31     {
32       if (gtk_button_get_use_underline (button))
33         {
34           label = gtk_label_new_with_mnemonic (label_text);
35           gtk_label_set_mnemonic_widget (GTK_LABEL (label),
36                                          GTK_WIDGET (button));
37         }
38       else
39         {
40           label = gtk_label_new (label_text);
41         }
42     }
43   if (priv->pixbuf)
44     {
45       priv->image = gtk_image_new_from_pixbuf (priv->pixbuf);
46       g_object_set (priv->image,
47                     "visible", show_image (button), "no_show_all", TRUE,
48                     NULL);
49     }
50   if (label_text && priv->align_set)
51     {
52       align = gtk_alignment_new (priv->xalign, priv->yalign, 0.0, 0.0);
53       gtk_misc_set_alignment (GTK_MISC (label),
54                               priv->xalign, priv->yalign);
55     }
```

```
56    else
57      {
58        align = gtk_alignment_new (0.5, 0.5, 0.0, 0.0);
59      }
60    if (priv->text_pos == GTK_POS_TOP ||
61        priv->text_pos == GTK_POS_LEFT)
62      {
63        if (label)
64          {
65            gtk_box_pack_start (GTK_BOX (box), label, FALSE, FALSE, 0);
66          }
67        if (priv->image)
68          {
69            gtk_box_pack_end (GTK_BOX (box), priv->image, FALSE, FALSE, 0);
70          }
71      }
72    else
73      {
74        if (priv->image)
75          {
76            gtk_box_pack_start (GTK_BOX (box),
77                                priv->image, FALSE, FALSE, 0);
78          }
79        if (label)
80          {
81            gtk_box_pack_end (GTK_BOX (box), label, FALSE, FALSE, 0);
82          }
83      }
84    gtk_container_add (GTK_CONTAINER (button), align);
85    gtk_container_add (GTK_CONTAINER (align), box);
86    gtk_widget_show_all (align);
87
88    return;
89  }
```

8.6　プロパティ取得関数

ウィジェットを作成する関数を実装したので，残りはウィジェットのプロパティへのアクセス関数の実装です．ここではウィジェットのプロパティを取得する関数を実装します．

具体的にはラベルテキスト，アイコンデータ，ラベルテキストの配置位置を取得する関数を考えます．これらはウィジェット構造体のメンバの値を返せばよいので，実装結果はソース 8–8 のようになります．

ソース 8-8 プロパティ取得関数

```
1  G_CONST_RETURN gchar*
2  gtk_icon_button_get_label (GtkIconButton *ibutton)
3  {
4    g_return_val_if_fail (GTK_IS_ICON_BUTTON (ibutton), NULL);
5
6    return gtk_button_get_label (GTK_BUTTON (ibutton));
7  }
8
9  GdkPixbuf*
10 gtk_icon_button_get_icon (GtkIconButton *ibutton)
11 {
12   GtkIconButtonPrivate *priv;
13
14   g_return_val_if_fail (GTK_IS_ICON_BUTTON (ibutton), NULL);
15
16   priv = GTK_ICON_BUTTON_GET_PRIVATE (ibutton);
17
18   return g_object_ref (priv->pixbuf);
19 }
20
21 GtkPositionType
22 gtk_icon_button_get_text_position (GtkIconButton *ibutton)
23 {
24   GtkIconButtonPrivate *priv;
25
26   g_return_val_if_fail (GTK_IS_ICON_BUTTON (ibutton), 0);
27
28   priv = GTK_ICON_BUTTON_GET_PRIVATE (ibutton);
```

```
29
30   return priv->text_pos;
31 }
```

8.7　プロパティ変更関数

プロパティを変更する関数を実装します．それぞれの引数に与えられた新しいプロパティで現在のプロパティを更新して，関数 gtk_icon_button_construct_child を呼び出してコンポーネントを再構成するようにします．

ソース 8-9 プロパティ変更関数

```
1  void
2  gtk_icon_button_set_label (GtkIconButton *ibutton,
3                             const gchar   *label)
4  {
5    g_return_if_fail (GTK_IS_ICON_BUTTON (ibutton));
6
7    gtk_button_set_label (GTK_BUTTON (ibutton), label);
8
9    gtk_icon_button_construct_child (ibutton);
10   g_object_notify (G_OBJECT (ibutton), "label");
11 }
12
13 void
14 gtk_icon_button_set_icon (GtkIconButton *ibutton,
15                           GdkPixbuf     *pixbuf)
16  {
17   GtkIconButtonPrivate *priv;
18
19   g_return_if_fail (GTK_IS_ICON_BUTTON (ibutton));
20
21   priv = GTK_ICON_BUTTON_GET_PRIVATE (ibutton);
22   if (priv->pixbuf) g_object_unref (priv->pixbuf);
23   priv->pixbuf = g_object_ref (pixbuf);
24
25   gtk_icon_button_construct_child (ibutton);
26 }
27
28 void
29 gtk_icon_button_set_text_position (GtkIconButton   *ibutton,
30                                    GtkPositionType text_pos)
31 {
32   GtkIconButtonPrivate *priv;
33
34   g_return_if_fail (GTK_IS_ICON_BUTTON (ibutton));
35
36   priv = GTK_ICON_BUTTON_GET_PRIVATE (ibutton);
37   priv->text_pos = text_pos;
38
39   gtk_icon_button_construct_child (ibutton);
40 }
```

8.8　ウィジェットのテスト

これでアイコン付きボタンウィジェットは完成しました（一部ソースコードは省略しています）．最後に，作成したアイコン付きボタンウィジェットの動作を確認するテストプログラムを紹介します．このプログラムは，クリックするたびにラベルテキストの配置位置を変更していきます．実行してみると，図 8.1 のように，ボタンをクリックするたびにラベルテキストの配置が変わっていくのがわかります．

ソース 8-10 GtkIconButton ウィジェットのテストプログラム

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

static void
cb_click (GtkWidget *widget, gpointer user_data)
{
  static guint position = 0;

  position++;
  if (position == 4) position = 0;
  gtk_icon_button_set_text_position (GTK_ICON_BUTTON (widget),
                                     (GtkPositionType) position);
}

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

  gtk_init (&argc, &argv);

  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "GtkIconButton␣Test");
  gtk_widget_set_size_request (window, 300, 100);
  g_signal_connect (G_OBJECT (window), "destroy",
                    G_CALLBACK (gtk_main_quit), NULL);

  ibutton = gtk_icon_button_new_with_label ("gnome-tigert.png",
                                            "Push␣me!!",
                                            GTK_POS_TOP);
  g_signal_connect (G_OBJECT (ibutton), "clicked",
                    G_CALLBACK(cb_click), NULL);
  gtk_container_add (GTK_CONTAINER (window), ibutton);

  gtk_widget_show_all (window);
  gtk_main ();

  return 0;
}
```

第 9 章

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．

なお，本書で解説する Anjuta のバージョンは 2 以降のもので，バージョン 1 のものとは使い方が異なるところがあるので，ご注意ください．

9.1 プロジェクトの作成

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，メニューの「アプリケーション」-「プログラミング」-「Anjuta IDE」を選択してください．

Anjuta を起動すると，図 9.1 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，Anjuta のメニューから「ファイル (N)」-「新規」-「4. プロジェクト」を選択してください．

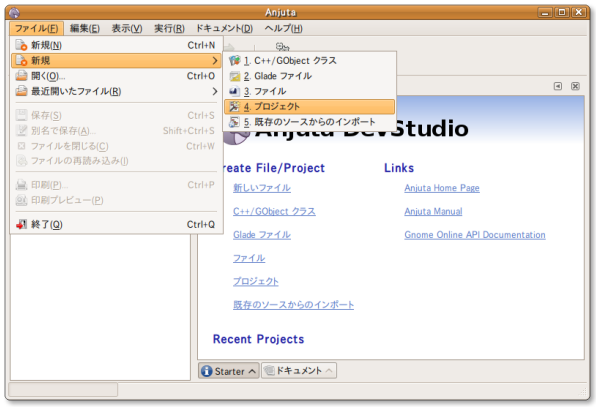


図 9.1　新規プロジェクトの作成

新規プロジェクトの作成を選択するとまず，図 9.2(a) のような画面が表示されるので，「GTK+」を選択し，「進む (F)」ボタンを押して次に進みます．次にプロジェクト名や開発者といった，プロジェクトに関する基本情報を入力するダイアログになります（図 9.2(b)）．

本プロジェクトでは，簡単な画像処理を行うアプリケーション ImageOperator を作成します．図 9.2(b) のように基本的な情報を入力したら，「進む (F)」ボタンを押して，次の画面に移ります．次のプロジェクトのオプション設定では，標準で図 9.2(c) のようになっているので，何も変更せずに「進む (F)」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)」ボタンを押して終了です．

アプリケーション作成ウィザードが終了したら，図 9.3 のようなメインウィンドウが表示されます．この時点でウィンドウだ

(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 9.2　プロジェクト作成ウィザード

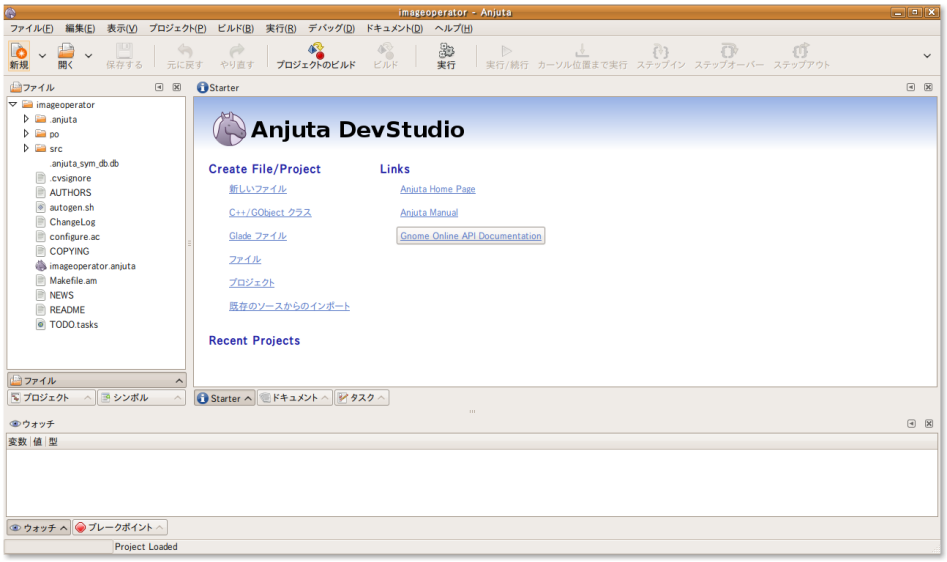


図 9.3　Anjuta のメインウィンドウ

けを表示するようなソースコードが自動的に生成されているので，メニューから「ビルド (B)」-「プロジェクトのビルド (B)」を選択すると，プロジェクトをビルドしてアプリケーションを作成できます．

しかし，自動的に生成されたソースコードには一部問題があり，そのままビルドすると GUI を構成するファイルを読み込むことができません．そこで，ビルドする前に，src ディレクトリにある main.c ファイルを修正します．

Anjuta の画面左側にプロジェクト内のファイルがツリー上に表示されているので，src ディレクトリ内にある main.c をダブルクリックしてみてください．エディタが設定されていないときは，図 9.4 のような画面が表示されるので，好みに応じて好きなエディタを選択してください．無事エディタが起動すると，中央の画面に main.c の内容が表示されるので，61 行目の

```
#define GLADE_FILE "src/imageoperator.glade"
```

の部分を次のように修正してください．

図 9.4 エディタの選択画面

```
#define GLADE_FILE "imageoperator.glade"
```

修正したら，メニューから「ビルド (B)」-「プロジェクトのビルド (B)」を選択するか，ツールバーの「プロジェクトのビルド」をクリックしてプロジェクトをビルドします．なお，プロジェクトの 1 回目のビルドのときに図 9.5 のような画面が表示されるので，そのまま「実行 (E)」ボタンを押してください．

図 9.5 プロジェクトの構成画面

無事ビルドが終了したら，メニューから「実行 (R)」-「実行」を選択するか，ツールバーの「実行」をクリックしてビルドしたアプリケーションを起動してみてください．図 9.6 のような画面が表示されれば成功です．

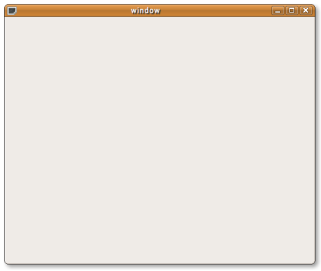


図 9.6 初期状態のアプリケーション

9.2 GUI の作成

9.2.1 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，次に本格的にアプリケーション用の 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 は単独でも動作しますが，Anjuta から Glade を起動するには，画面左側のファイルツリーから src ディレクトリ内にある，imageoperator.glade というファイルをダブルクリックします．Anjuta の画面内で Glade が起動すると図 9.7 のような画面が表示されます．

Glade によって GUI を作成するには Anjuta の画面左側に表示される「ウィジェット」，「パレット」，「プロパティ」の 3 つのタブ（図 9.8）を使用します．それぞれのタブは次のような役割をします．

- ウィジェット
  現在の GUI の構成をツリー形式で表示します．
- パレット
  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，中央の画面のウィジェットを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．

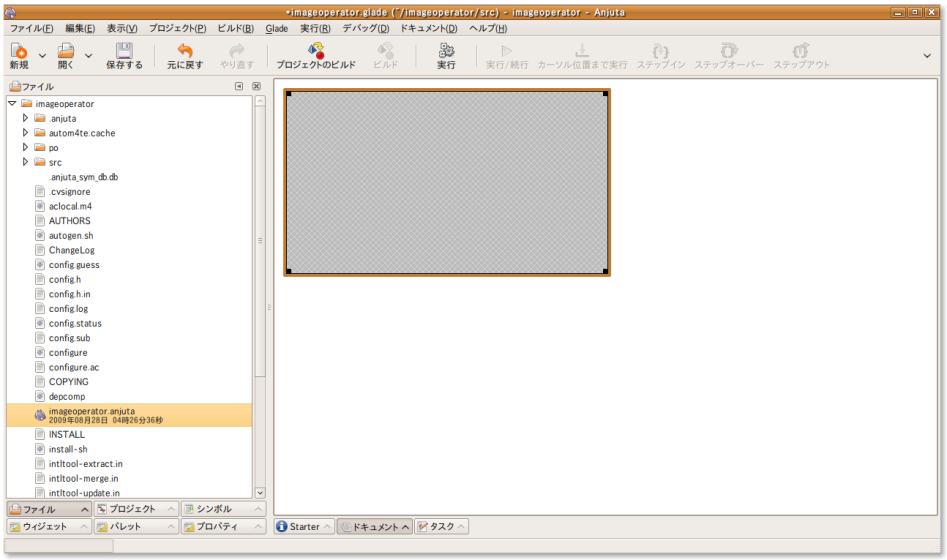


図 9.7　Glade による GUI 編集画面

- プロパティ
  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．

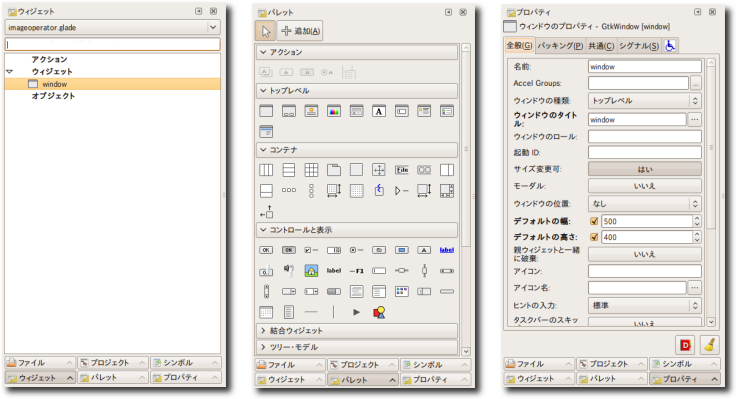


図 9.8　Glade で使用する 3 つのタブ

9.2.2　GUI の構成

まず，ウィンドウ内に必要なウィジェットを配置していくので，タブを「パレット」に切り替えて，配置するウィジェットを選択できる状態にします．図 9.9 に，今回作成するアプリケーションのウィジェット配置を示しておきます．これを参考にしながら，ウィジェット配置について説明します．

はじめにパレット内のコンテナの垂直ボックスをクリックしてから，画面中央のウィンドウウィジェット内でマウスの左ボタンをクリックしてください．すると，垂直ボックス何に個のウィジェットを配置するのかを聞いてくるので，アイテム数を 2 に修正して「OK(O)」ボタンを押してください．

垂直ボックスの上部にはメニューバーを配置しますが，今回は UI マネージャを使ってメニューを作成することにします．そのために，垂直ボックスの上部にメニューバーを配置するための垂直ボックスを 1 つ追加しておきます．この垂直ボックスに追加するアイテム数は 1 にしておきます．次の順番で，はじめに配置した垂直ボックスの下部にウィジェットを階層的に配置します．

1. スクロール・ウィンドウ
2. ビューポート
3. アラインメント
4. 描画領域

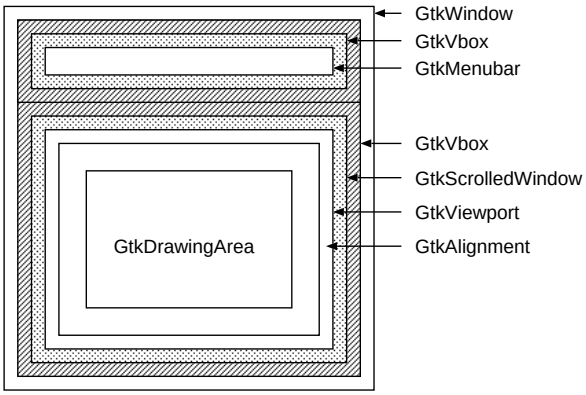


図 9.9 アプリケーションのウィジェット配置

ウィジェットの配置が終了したら，タブを「ウィジェット」に切り替えて，配置したウィジェットの階層構造が図 9.10 のようになっているか確認してください．

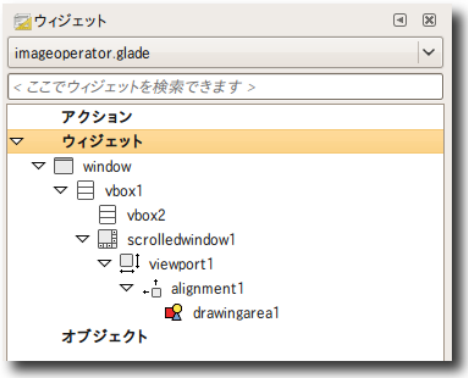


図 9.10 ウィジェットの階層構造

9.2.3 プロパティの設定

ウィジェット配置の確認ができたら，いくつかのウィジェットのプロパティを設定します．「ウィジェット」タブでウィジェットを選択してから「プロパティ」タブに切り替えると，そのウィジェットのプロパティが表示されます．

ウィンドウのプロパティ設定

ウィンドウのプロパティでは，ウィンドウのタイトルを “window” から “ImageOperator” に修正します．

2 つ目の垂直ボックス（vbox2）のプロパティ設定

メニューバーを配置するための垂直ボックスのプロパティで，「パッキング」タブの「拡張」プロパティを「いいえ」に変更してください．

アラインメントのプロパティ設定

ここでは「水平スケール」プロパティと「垂直スケール」プロパティを，1.00 から 0.00 に修正してください．このパラメータを 0 に設定しておくと，アラインメントウィジェット内に配置したウィジェット（この場合は描画領域）を，ウィジェットの中央に配置するようになります．

9.2.4 メニューの生成

メニューも Glade で作成できるのですが，操作が少々面倒なのでチュートリアルで紹介した UI マネージャ（7.4.3，p. 160）を利用することにします．ここではメニュー情報の定義やメニュー作成関数用に新しい menu.c というファイルを追加します．

新しいファイルを追加するには，まず，メニューの「ファイル (N)」-「新規」-「3. ファイル」を選択するか，ツールバーの

「新規」-「3. ファイル」をクリックします．図 9.11(a) のようなウィンドウが表示されるので，「名前」の欄に”menu.c”と入力し，オプションの項目の「プロジェクトに追加する」をチェックして，「OK(O)」を押してください．次に図 9.11(b) のようなウィンドウが表示されるので，「other files」を選択して，「追加 (A)」ボタンを押してください．

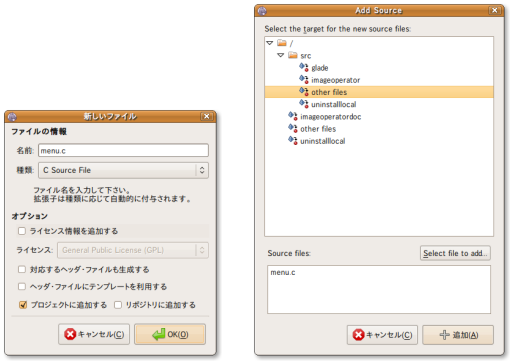


図 9.11　ファイルの追加

中央の画面に menu.c の編集画面が表示されるので，ソース 9–1 の内容を入力します．UI マネージャを利用したメニュー作成については，第 2 章のチュートリアルや第 7 章のウィジェットリファレンスを参照してください．

ソース 9-1 メニュー作成：menu.c

```
1  #include <gtk/gtk.h>
2  #include <glade/glade.h>
3  #include "callbacks.h"
4  #include "image_operator.h"
5
6  static const gchar* menu_info =
7      "<ui>"
8      "␣␣<menubar␣name='Menubar'>"
9      "␣␣␣␣<menu␣action='File'>"
10     "␣␣␣␣␣␣<menuitem␣action='Open'/>"
11     "␣␣␣␣␣␣<menuitem␣action='Save'/>"
12     "␣␣␣␣␣␣<menuitem␣action='SaveAs'/>"
13     "␣␣␣␣␣␣<separator/>"
14     "␣␣␣␣␣␣<menuitem␣action='Quit'/>"
15     "␣␣␣␣</menu>"
16     "␣␣␣␣<menu␣action='Edit'>"
17     "␣␣␣␣␣␣<menuitem␣action='Undo'/>"
18     "␣␣␣␣␣␣<menuitem␣action='Grayscale'/>"
19     "␣␣␣␣</menu>"
20     "␣␣␣␣<menu␣action='Help'>"
21     "␣␣␣␣␣␣<menuitem␣action='About'/>"
22     "␣␣␣␣</menu>"
23     "␣␣</menubar>"
24     "</ui>";
25
26 static GtkActionEntry entries[] = {
27   {"File", NULL, "_File"},
28   {"Open", GTK_STOCK_OPEN, "_Open", "<control>O",
29    "Open␣an␣Image", G_CALLBACK(cb_open)},
30   {"Save", GTK_STOCK_SAVE, "_Save", "<control>S",
31    "Save␣the␣Image", G_CALLBACK(cb_save)},
32   {"SaveAs", GTK_STOCK_SAVE_AS, "Save_as", "<control><shift>S",
33    "Save␣the␣Image␣with␣other␣name", G_CALLBACK(cb_saveas)},
34   {"Quit", GTK_STOCK_QUIT, "_Quit", "<control>Q",
35    "Quit␣this␣program", G_CALLBACK(cb_quit)},
36   {"Edit", NULL, "_Edit"},
37   {"Undo", GTK_STOCK_UNDO, "_Undo", "<control>Z",
38    "Undo␣the␣operation", G_CALLBACK(cb_undo)},
39   {"Grayscale", NULL, "_Grayscale", NULL,
40    "Convert␣to␣grayscale", G_CALLBACK(cb_grayscale)},
41   {"Help", NULL, "_Help"},
42   {"About", GTK_STOCK_ABOUT, "_About", NULL,
43    "About␣this␣program", G_CALLBACK(cb_about)}
44 };
45
```

```
46 void
47 create_menu (GtkWidget* parent)
48 {
49   GtkUIManager   *ui;
50   GtkActionGroup *actions;
51   GtkWidget      *vbox;
52   GtkWidget      *menubar;
53
54   ui = gtk_ui_manager_new ();
55   actions = gtk_action_group_new ("menu");
56   gtk_action_group_add_actions (actions, entries,
57                                 sizeof(entries)/sizeof(entries[0]),
58                                 parent);
59   gtk_ui_manager_insert_action_group (ui, actions, 0);
60   gtk_ui_manager_add_ui_from_string (ui, menu_info, -1, NULL);
61   gtk_window_add_accel_group (GTK_WINDOW(parent),
62                               gtk_ui_manager_get_accel_group (ui));
63
64   vbox = glade_xml_get_widget (gxml, "vbox2");
65   menubar = gtk_ui_manager_get_widget (ui, "/Menubar");
66   gtk_box_pack_start (GTK_BOX(vbox), menubar, FALSE, FALSE, 0);
67   gtk_widget_show_all (menubar);
68 }
```

63 行目では，libglade の関数 glade_xml_get_widget によって，GUI 作成時にそれぞれのウィジェットにつけた名前でウィジェットを取得しています．

```
GtkWidget* glade_xml_get_widget (GladeXML *self, const char *name);
```

第 1 引数　GladeXML 型の変数
第 2 引数　ウィジェットに付けた名前

今後，他の関数内で drawingarea ウィジェットにアクセスする必要があるので，menu.c を追加したのと同様に image_operator.h というファイルを追加して，ソース 9–2 のように GladeXML 型の変数をグローバル変数として宣言します．

ソース 9-2 image_operator.h から一部抜粋

```
1  #include <gtk/gtk.h>
2  #include <glade/glade.h>
3
4  GladeXML *gxml;
```

これに合わせて，main.c の関数 create_window 内の GladeXML 型の変数の宣言を削除します．そのほかに main.c 内に以下の 2 つの行を追加してください．

callback.h をインクルードしている行の後:

```
void create_menu (GtkWidget *parent);
```

関数 main 内の関数 gtk_main の前:

```
create_menu (window);
```

また，それぞれのメニューアイテムに設定したコールバック関数を callback.c に追加します．それぞれの関数の中身は後で実装することにします．同様に callback.h にコールバック関数のプロトタイプ宣言を追加します．

callback.c 内:

```
void cb_open (GtkAction* action, gpointer user_data) { }
void cb_save (GtkAction* action, gpointer user_data) { }
void cb_saveas (GtkAction* action, gpointer user_data) { }
void cb_quit (GtkAction* action, gpointer user_data) { }
void cb_undo (GtkAction* action, gpointer user_data) { }
void cb_grayscale (GtkAction* action, gpointer user_data) { }
void cb_about (GtkAction* action, gpointer user_data) { }
```

callback.h 内:

```
void cb_open (GtkAction* action, gpointer user_data);
void cb_save (GtkAction* action, gpointer user_data);
void cb_saveas (GtkAction* action, gpointer user_data);
void cb_quit (GtkAction* action, gpointer user_data);
void cb_undo (GtkAction* action, gpointer user_data);
void cb_grayscale (GtkAction* action, gpointer user_data);
void cb_about (GtkAction* action, gpointer user_data);
```

最後に Makefile.am の imageoperator_SOURCES に menu.c，image_operator.h を追加します．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 9.12 に示します．

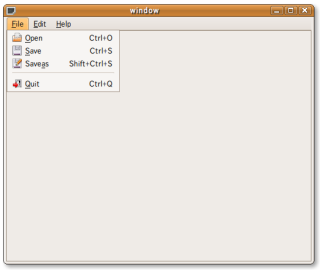


図 9.12　アプリケーションの GUI 画面

9.3　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 9.1 にまとめました．

9.3.1　グローバル変数の設定

関数を実装する前にそれぞれの関数で共通に使用する変数を定義しておきます．先ほど作成した image_operator.h に新しく 2 つのグローバル変数を以下のように定義します．pixbuf は表示画像データ，backup はアンドゥ用の画像データです．

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

GladeXML  *gxml;
GdkPixbuf *pixbuf;
GdkPixbuf *backup;
```

表 9.1　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_undo	画像処理関数（アンドゥ）
cb_grayscale	画像処理関数（濃淡画像の作成）
cb_about	アプリケーション情報を表示する関数
cb_expose	画面描画関数

9.3.2　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_expose 内に実装します．現時点でドローイングエリアのコールバック関数の設定を行っていないので，はじめにドローイングエリア（描画領域）の **expose-event シグナル** に対するコールバック関数を設定します．

コールバック関数を設定するために，Anjuta の画面左側のタブを「ウィジェット」に切り替えて drawingarea1 を選択してから，タブを「プロパティ」に切り替えます．drawingarea1 のプロパティが表示されたら，その中の「シグナル」タブをクリックしてウィジェットに対するシグナル一覧を表示させます．この一覧から expose-event を見つけて，ハンドラ名の欄に cb_expose と入力してください（図 9.13）．

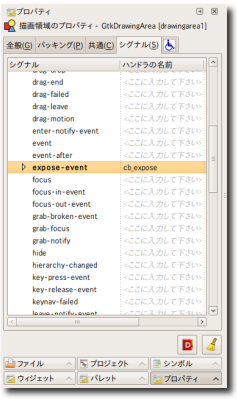


図 9.13　コールバック関数の設定

次に callbacks.c 内にコールバック関数を実装します．画像データをドローイングエリアに描画する方法は，6.3.2 GtkDrawingArea ウィジェットによる画像の表示を参考にすることにします．実装したコールバック関数をソース 9–3 に示します．画像データが読み込まれている場合にだけ描画処理を行うことに注意してください．

ソース 9-3　関数 cb_canvas_expose

```
1  gboolean
2  cb_canvas_expose (GtkWidget       *widget,
3                    GdkEventExpose  *event,
4                    gpointer         user_data)
5  {
6      GdkPixmap*pixmap;
7
8      if (pixbuf)
9        {
10         gtk_widget_set_size_request (widget,
11                                      gdk_pixbuf_get_width (pixbuf),
12                                      gdk_pixbuf_get_height (pixbuf));
13         gdk_pixbuf_render_pixmap_and_mask (pixbuf, &pixmap, NULL, 255);
14         gdk_window_set_back_pixmap (widget->window, pixmap, FALSE);
15         gdk_window_clear(widget->window);
16         g_object_unref (pixmap);
17       }
```

```
18        return FALSE;
19    }
```

9.3.3　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．

コールバック関数 cb_open

ファイルメニューで「開く」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 9–4 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．ここでは，画像の保存のことも考えて，現在開いているファイル名を保持するためのグローバル変数 opened_filename を新たに image_operator.h に追加して，ファイル選択ダイアログで選択したファイル名を保持させています．

ソース 9-4 画像の読み込み関連の関数：callbacks.c から一部抜粋

```
1  void
2  cb_open (GtkAction* action, gpointer user_data)
3  {
4    GtkWidget *window;
5    GtkWidget *dialog;
6    GtkWidget *drawingarea;
7    gint      response;
8    gchar     *filename;
9
10   window = GTK_WIDGET (user_data);
11   dialog = gtk_file_chooser_dialog_new ("Open␣an␣image",
12                                         GTK_WINDOW (window),
13                                         GTK_FILE_CHOOSER_ACTION_OPEN,
14                                         GTK_STOCK_CANCEL,
15                                         GTK_RESPONSE_CANCEL,
16                                         GTK_STOCK_OPEN,
17                                         GTK_RESPONSE_ACCEPT,
18                                         NULL);
19   gtk_widget_show_all (dialog);
20
21   response = gtk_dialog_run (GTK_DIALOG(dialog));
22   if (response == GTK_RESPONSE_ACCEPT)
23     {
24       filename
25         = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER(dialog));
26       if (pixbuf)
27         {
28           g_object_unref (pixbuf);
29         }
30       pixbuf = gdk_pixbuf_new_from_file (filename, NULL);
31       drawingarea = glade_xml_get_widget (gxml, "drawingarea1");
32       gtk_widget_queue_draw (drawingarea);
33       if (opened_filename)
34         {
35           g_free (opened_filename);
36           opened_filename = g_strdup (filename);
37         }
38       g_free (filename);
39     }
40   gtk_widget_destroy (dialog);
41 }
```

9.3.4　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「保存」と「別名で保存」の 2 つがあります．「保存」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．「別名で保存」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，「保存」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．画像の保存に関するソースコードを，ソース 9–5 に示します．関数_save では，指定されたファイル名の拡張子を調べて，関数 gdk_pixbuf_save に対応している JPEG フォーマットか PNG フォーマットであれば，指定されたファイル名で画像を保存します．

ソース 9-5 画像の保存関連の関数：callbacks.c から一部抜粋

```
1  static void
2  _save (gchar *filename)
3  {
4    gchar *ext;
5
6    ext = g_strrstr (filename, ".");
7    if (ext)
8      {
9        ext++;
10       if (g_strcmp0 (ext, "png") == 0)
11         {
12           gdk_pixbuf_save (pixbuf, filename, "png", NULL, NULL);
13         }
14       else if (g_strcmp0 (ext, "jpg") == 0)
15         {
16           gdk_pixbuf_save (pixbuf, filename, "jpeg", NULL, NULL);
17         }
18       else
19         {
20           g_printerr ("Image␣format␣'%s'␣is␣not␣suported\n", ext);
21         }
22       g_free (opened_filename);
23       opened_filename = g_strdup (filename);
24     }
25 }
26
27 void
28 cb_save (GtkAction* action, gpointer user_data)
29 {
30   _save (opened_filename);
31 }
32
33 void
34 cb_saveas (GtkAction* action, gpointer user_data)
35 {
36   GtkWidget *window;
37   GtkWidget *dialog;
38   gint      response;
39   gchar     *filename;
40
41   if (!pixbuf) return;
42
43   window = GTK_WIDGET (user_data);
44   dialog = gtk_file_chooser_dialog_new ("Save␣the␣image",
45                                         GTK_WINDOW (window),
46                                         GTK_FILE_CHOOSER_ACTION_SAVE,
47                                         GTK_STOCK_CANCEL,
48                                         GTK_RESPONSE_CANCEL,
49                                         GTK_STOCK_SAVE,
50                                         GTK_RESPONSE_ACCEPT,
51                                         NULL);
52   gtk_widget_show_all (dialog);
53
54   response = gtk_dialog_run (GTK_DIALOG(dialog));
55   if (response == GTK_RESPONSE_ACCEPT)
56     {
57       filename
58         = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER(dialog));
59       _save (filename);
60       g_free (filename)
61     }
62   gtk_widget_destroy (dialog);
63 }
```

9.3.5　画像処理関数の実装

画像処理関数を作成する前に，指定した座標の画素値を GdkPixbuf データから取得したり，書き込むマクロを，ソース 9–6 のように定義します．これは，5.4 節の簡単な画像処理アプリケーションの作成（p. 73）で使用したマクロと同様です．

ソース 9-6 画素値の読み書きを行うマクロ：image_operator.h から一部抜粋

```
#define gdk_pixbuf_get_pixel(pixbuf,x,y,p) \
(*(gdk_pixbuf_get_pixels((pixbuf)) + \
 gdk_pixbuf_get_rowstride((pixbuf)) * (y) + \
 gdk_pixbuf_get_n_channels((pixbuf)) * (x) + (p)))

#define gdk_pixbuf_put_pixel(pixbuf,x,y,p,val) \
(*(gdk_pixbuf_get_pixels((pixbuf)) + \
 gdk_pixbuf_get_rowstride((pixbuf)) * (y) + \
 gdk_pixbuf_get_n_channels((pixbuf)) * (x) + (p)) = (val))
```

アンドゥ処理

次に，画像処理結果を処理前に戻すアンドゥ処理を実装します．メニューアイテムから呼び出されるコールバック関数 cb_undo と，現在の画像データをメモリ上に保存する関数 do_backup を，ソース 9–7 に示します．

はじめにアンドゥ用の変数 backup を main 関数内で NULL に初期化しておきます．そして，変数 backup の実メモリ領域が確保されている場合には，関数 gdk_pixbuf_copy_area を呼び出し，データをコピーします．メモリ領域が確保されていない場合には，関数 gdk_pixbuf_copy を呼び出します．

コールバック関数 cb_undo では関数 gdk_pixbuf_copy_area を使って，変数 backup から変数 pixbuf へデータをコピーします．ここでは，アンドゥ処理の取り消しにも対応するために，はじめに変数 pixbuf のデータをいったん別変数_backup にコピーしておき，最後にそのデータを変数 backup にコピーします．

ソース 9-7 アンドゥ処理関連の関数：callbacks.c から一部抜粋

```
static void
do_backup (void)
{
  if (backup)
    {
      gdk_pixbuf_copy_area (pixbuf, 0, 0,
                            gdk_pixbuf_get_width (pixbuf),
                            gdk_pixbuf_get_height(pixbuf),
                            backup, 0, 0);
    }
  else
    {
      backup = gdk_pixbuf_copy (pixbuf);
    }
}

void
cb_undo (GtkAction *action,
         gpointer  user_data)
{
  GtkWidget *drawingarea;
  GdkPixbuf *_backup;

  if (backup)
    {
      _backup = gdk_pixbuf_copy (pixbuf);
      gdk_pixbuf_copy_area (backup, 0, 0,
                            gdk_pixbuf_get_width(backup),
                            gdk_pixbuf_get_height(backup),
                            pixbuf, 0, 0);
      gdk_pixbuf_copy_area (_backup, 0, 0,
                            gdk_pixbuf_get_width(_backup),
                            gdk_pixbuf_get_height(_backup),
                            backup, 0, 0);
      g_object_unref (_backup);
      drawingarea = glade_xml_get_widget (gxml, "drawingarea1");
      gtk_widget_queue_draw (drawingarea);
    }
}
```

最後に画像処理メニューの中の 1 つのグレースケール化の実装例をソース 9–8 に示します．まず 11 行目でアンドゥ処理用に関数 do_backup を呼び出し，データをバックアップします．次に 12–24 行目で画像を走査してカラー画像をグレースケールに変換します．画像処理結果を画面に反映させるために 25 行目で関数 gtk_widget_queue_draw を呼び出して expose-event シグナルを発生させます．

ソース 9-8 関数 cb_grayscale：callbacks.c から一部抜粋

```
1  void
2  cb_grayscale (GtkAction *action,
3                gpointer  user_data)
4  {
5    GtkWidget *drawingarea;
6    int       x, y;
7    guchar    val, r, g, b;
8
9    if (pixbuf)
10     {
11       do_backup ();
12       for (y = 0; y < gdk_pixbuf_get_height(pixbuf); y++)
13         {
14           for (x = 0; x < gdk_pixbuf_get_width(pixbuf); x++)
15             {
16               r = gdk_pixbuf_get_pixel(pixbuf, x, y, 0);
17               g = gdk_pixbuf_get_pixel(pixbuf, x, y, 1);
18               b = gdk_pixbuf_get_pixel(pixbuf, x, y, 2);
19               val = (guchar) (0.299 * r + 0.587 * g + 0.114 * b);
20               gdk_pixbuf_put_pixel(pixbuf, x, y, 0, val);
21               gdk_pixbuf_put_pixel(pixbuf, x, y, 1, val);
22               gdk_pixbuf_put_pixel(pixbuf, x, y, 2, val);
23             }
24         }
25       drawingarea = glade_xml_get_widget (gxml, "drawingarea1");
26       gtk_widget_queue_draw (canvas);
27   }
28 }
```

9.3.6　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メッセージダイアログを使用します．メッセージダイアログの詳細については 7.5.2 （p. 169）を参照してください．

コールバック関数 cb_about をソース 9–9 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 9.14 に示します．

ソース 9-9 関数 cb_about

```
1  void
2  cb_about (GtkAction* action, gpointer user_data)
3  {
4    GtkWidget *dialog;
5    gint      response;
6    gchar     *copyright = "Copyright(C)␣...";
7    gchar     *message = "is␣a␣simple␣image␣proccessing␣application.";
8
```

図 9.14　アプリケーション情報ダイアログ

```
 9   dialog = gtk_message_dialog_new (GTK_WINDOW (user_data),
10                                    GTK_DIALOG_MODAL |
11                                    GTK_DIALOG_DESTROY_WITH_PARENT,
12                                    GTK_MESSAGE_INFO,
13                                    GTK_BUTTONS_CLOSE,
14                                    "%s␣%s\n%s\n%s␣%s\n",
15                                    PACKAGE, VERSION,
16                                    copyright, PACKAGE, message);
17   gtk_widget_show_all (dialog);
18   response = gtk_dialog_run (GTK_DIALOG (dialog));
19   gtk_widget_destroy (dialog);
20 }
```

9.3.7　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 9–10 に示します．

ソース 9-10 関数 cb_quit

```
1 void
2 cb_quit (GtkAction *action,
3          gpointer  user_data)
4 {
5   if (pixbuf) g_object_unref (pixbuf);
6   if (backup) g_object_unref (backup);
7   if (opened_filename) g_free (opened_filename);
8   gtk_main_quit ();
9 }
```

9.4　アプリケーションの実行

ここまででアプリケーションの実装は終了しました．そこで，メニューから「ビルド (B)」-「プロジェクトのビルド (B)」を選択するか，ツールバーの「プロジェクトのビルド」をクリックして，プロジェクトをビルドします．

無事ビルドが終了したら，メニューから「実行 (R)」-「実行」を選択するか，ツールバーの「実行」をクリックして，ビルドしたアプリケーションを起動してみてください．

図 9.15 はアプリケーションを実行して，画像を読み込んだものです．

図 9.15　完成したアプリケーション

9.5 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．

9.5.1 main.c の修正

配布パッケージを作成する前に，main.c 内の GUI 構成ファイル（imageoperator.glade）の場所を修正する必要があります．パッケージをインストールすると，このファイルは，PACKAGE_DATA_DATA/imageoperator というディレクトリ内に置かれるので，以下のように修正します．

```
#define GLADE_FILE PACKAGE_DATA_DIR"/imageoperator/imageoperator.glade"
```

9.5.2 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．

9.5.3 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/imageoperator/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 ~/imageoperator/imageoperator-0.1.tar.gz ↵
...
$ cd imageoperator-0.1 ↵
$ ./configure --prefix=/tmp/imageoperator/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/imageoperator ↵
$ ls -R
.:
usr/

./usr:
bin/  doc/  share/
```

```
./usr/bin:
imageoperator

./usr/doc:
imageoperator

./usr/doc/imageoperator:
AUTHORS  COPYING  ChangeLog  INSTALL  NEWS  README

./usr/share:
imageoperator

./usr/share/imageoperator/glade:
imageoperator.glade
```

9.6　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．

9.6.1　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，callbacks.c と menu.c と imageoperator.glade 内に翻訳対象となる文字列が含まれます（imageoperator.glade 内のウィンドウタイトルの文字列 ImageOperator が翻訳対象文字列になります）．

ただし，メニューの文字列については，_() ではなく，N_() で囲んでください．これは文字列用の領域を確保する際に，文字列を翻訳メッセージに置き換えてしまうことを防ぐことが目的です．

また，現在は_() 等のマクロが main.c 内にしか定義されていないので，main.c の以下の部分を，image_operator.h にカット&ペーストします．そして，menu.c で image_operator.h をインクルードしてください．

```
/*
 * Standard gettext macros.
 */
#ifdef ENABLE_NLS
#  include <libintl.h>
#  undef _
#  define _(String) dgettext (PACKAGE, String)
#  ifdef gettext_noop
#    define N_(String) gettext_noop (String)
#  else
#    define N_(String) (String)
#  endif
#else
#  define textdomain(String) (String)
#  define gettext(String) (String)
#  define dgettext(Domain,Message) (Message)
#  define dcgettext(Domain,Message,Type) (Message)
#  define bindtextdomain(Domain,Directory) (Domain)
#  define _(String) (String)
#  define N_(String) (String)
```

```
#endif
```

次に，menu.c 内の関数 create_menu に，以下のように関数 gtk_action_group_set_translation_domain を追加してください．この関数を追加しないと，メニューの文字列が日本語に翻訳されて表示されません．

```
actions = gtk_action_group_new ("menu");
gtk_action_group_set_translation_domain (actions, GETTEXT_PACKAGE);
```

次に，プロジェクトディレクトリ（imageoperator）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 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，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/callbacks.c
src/menu.c
src/imageoperator.glade
```

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 imageoperator/po ↵
$ intltool-update -p ↵
```

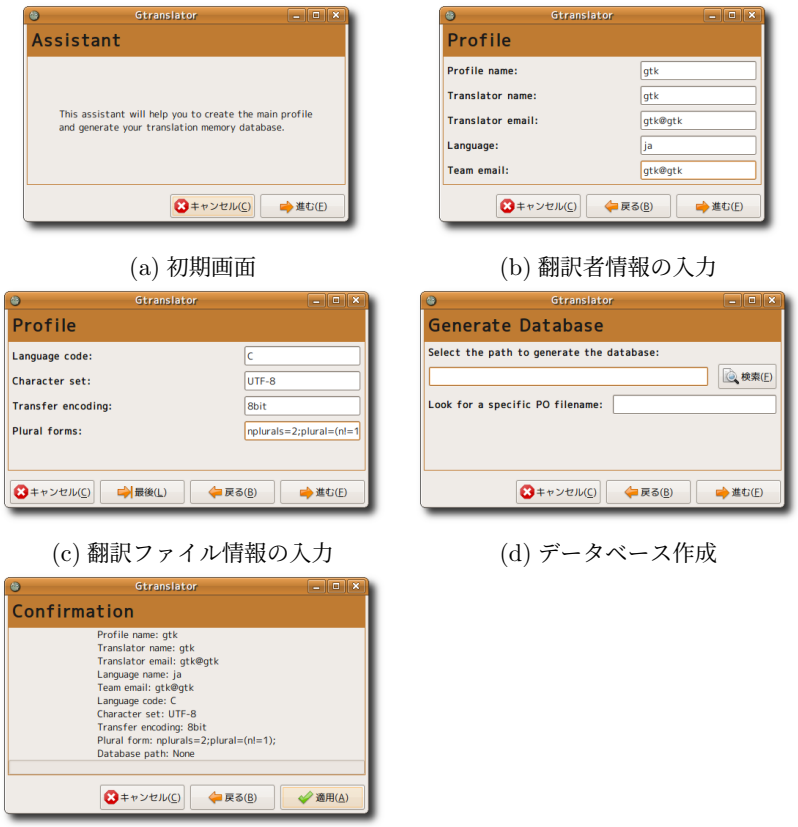
正常にコマンドが終了したら，imageoperator.pot というファイルが生成されます．これを ja.po という名前でコピーして，このファイルに翻訳メッセージを入力していきます．翻訳に進む前に，ja.po 内の以下の部分を修正してください．

```
"Project-Id-Version: imageoperator 0.1\n"
...
"Last-Translator: GTK <gtk@>\n"
"Language-Team:GTK <gtk@>\n"
...
"Content-Type: text/plain; charset=UTF-8\n"
```

9.6.2　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，メニューの「アプリケーション」-「プログラミング」-「Gtranslator PO Editor」を選択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 9.16 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) 翻訳ファイル情報の入力　(d) データベース作成

(e) 確認画面

図 9.16　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 9.17 に示します．画面左下の「Original Text」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．

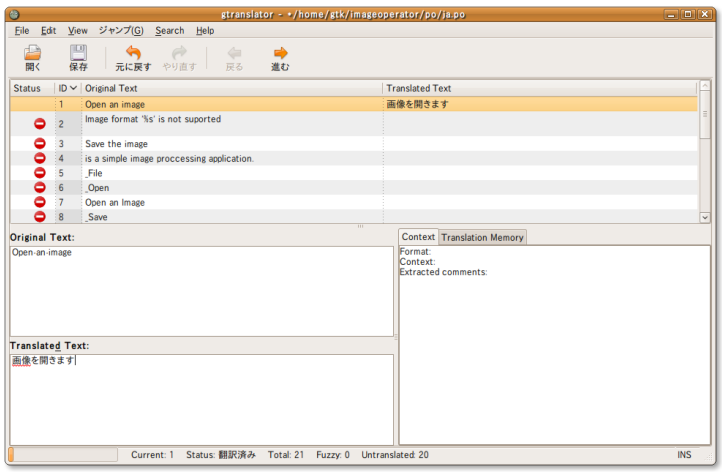


図 9.17　メッセージの翻訳画面

9.6.3　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 9.5 節（p. 243）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 9.18 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 9.18　メッセージを日本語化したアプリケーション

9.7　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．古いバージョンに対するものですが，日本語訳されたドキュメントが以下の URL で公開されているので，興味のある方は目を通してみてください．

- Anjuta IDE マニュアル：http://www.gnome.gr.jp/docs/anjuta-manual/
- Anjuta 統合開発環境 FAQ：http://www.gnome.gr.jp/docs/anjuta-faqs/

第 10 章

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
- オリジナルストックアイテムの作成
- プログラムオプションの解析

10.1 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．

10.1.1 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 GtkEventMask 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_ALL_EVENTS_MASK            = 0x3FFFFE
} GdkEventMask;
```

10.1.2　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．

10.1.3　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 }
```

10.1.4 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 }
```

10.1.5　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 }
```

10.1.6　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．

10.1.7　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. ビットマップデータ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_CURSOR_IS_PIXMAP    = -1
} GdkCursorType;
```

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 (window->window, cursor);
21
22   gtk_main ();
23
24   return 0;
25 }
```

ビットマップデータからカーソルを生成する方法

作成したいカーソル画像とそのマスクデータが char 型の配列で与えられているとき，まず関数 gdk_bitmap_new_from_data によってカーソル画像とそのマスクデータのビットマップを生成します．

```
GdkBitmap* gdk_bitmap_create_from_data (GdkDrawable *drawable,
                                        const gchar *data,
                                        gint        width,
                                        gint        height);
```

ビットマップデータが用意できたら，関数 gdk_cursor_new_from_pixmap でカーソルを生成します．

```
GdkCursor* gdk_cursor_new_from_pixmap (GdkPixmap      *source,
                                       GdkPixmap      *mask,
```

```
                                        const GdkColor *fg,
                                        const GdkColor *bg,
                                        gint           x,
                                        gint           y);
```

関数の第 5 引数と第 6 引数の (x, y) には，カーソル座標のカーソル画像中の位置を指定します．また関数の第 3 引数と第 4 引数には，カーソルの前景色と背景色を GdkColor で指定する必要があります．

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 ビットマップから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 char hand_open_data[] =
 7 {
 8   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00,
 9   0x60, 0x36, 0x00, 0x60, 0x36, 0x00, 0xc0, 0x36, 0x01, 0xc0, 0xb6, 0x01,
10   0x80, 0xbf, 0x01, 0x98, 0xff, 0x01, 0xb8, 0xff, 0x00, 0xf0, 0xff, 0x00,
11   0xe0, 0xff, 0x00, 0xe0, 0x7f, 0x00, 0xc0, 0x7f, 0x00, 0x80, 0x3f, 0x00,
12   0x00, 0x3f, 0x00, 0x00, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
13 };
14
15 static char hand_open_mask[] =
16 {
17   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x60, 0x3f, 0x00,
18   0xf0, 0x7f, 0x00, 0xf0, 0x7f, 0x01, 0xe0, 0xff, 0x03, 0xe0, 0xff, 0x03,
19   0xd8, 0xff, 0x03, 0xfc, 0xff, 0x03, 0xfc, 0xff, 0x01, 0xf8, 0xff, 0x01,
20   0xf0, 0xff, 0x01, 0xf0, 0xff, 0x00, 0xe0, 0xff, 0x00, 0xc0, 0x7f, 0x00,
21   0x80, 0x7f, 0x00, 0x80, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
22 };
23
24 static char hand_close_data[] =
25 {
26   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
27   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x80, 0x3f, 0x00,
28   0x80, 0xff, 0x00, 0x80, 0xff, 0x00, 0xb0, 0xff, 0x00, 0xf0, 0xff, 0x00,
29   0xe0, 0xff, 0x00, 0xe0, 0x7f, 0x00, 0xc0, 0x7f, 0x00, 0x80, 0x3f, 0x00,
30   0x00, 0x3f, 0x00, 0x00, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
31 };
32
33 static char hand_close_mask[] =
34 {
35   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
36   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x80, 0x3f,
37   0x00, 0xc0, 0xff, 0x00, 0xc0, 0xff, 0x01, 0xf0, 0xff, 0x01,
38   0xf8, 0xff, 0x01, 0xf8, 0xff, 0x01, 0xf0, 0xff, 0x01, 0xf0,
39   0xff, 0x00, 0xe0, 0xff, 0x00, 0xc0, 0x7f, 0x00, 0x80, 0x7f,
40   0x00, 0x80, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
41 };
42
43 static GdkCursor*
44 make_cursor (char *cursor_data,
45              char *cursor_mask,
46              int  width,
47              int  height)
48 {
49   GdkBitmap *data;
50   GdkBitmap *mask;
51   GdkColor black = {0, 0, 0, 0};
52   GdkColor white = {0, 0xffff, 0xffff, 0xffff};
53   GdkCursor *cursor;
54
55   data = gdk_bitmap_create_from_data (NULL, cursor_data, width, height);
56   mask = gdk_bitmap_create_from_data (NULL, cursor_mask, width, height);
57   cursor = gdk_cursor_new_from_pixmap (data, mask, &white, &black,
58                                        width / 2, height / 2);
59   g_object_unref (data);
```

```
60   g_object_unref (mask);
61
62   return cursor;
63 }
64
65 static gboolean
66 cb_mouse_press (GtkWidget      *widget,
67                 GdkEventButton *event,
68                 gpointer       user_data)
69 {
70   gdk_window_set_cursor (widget->window, hand_close);
71
72   return FALSE;
73 }
74
75 static gboolean
76 cb_mouse_release (GtkWidget      *widget,
77                   GdkEventButton *event,
78                   gpointer       user_data)
79 {
80   gdk_window_set_cursor (widget->window, hand_open);
81
82   return FALSE;
83 }
84
85 int
86 main (int argc, char *argv[])
87 {
88   GtkWidget *window;
89
90   gtk_init (&argc, &argv);
91
92   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
93   gtk_window_set_title (GTK_WINDOW (window), "Mouse␣Tips5-2");
94   gtk_widget_set_size_request (window, 300, 200);
95   g_signal_connect (G_OBJECT (window), "destroy",
96                     G_CALLBACK (gtk_main_quit), NULL);
97
98   gtk_widget_add_events (window,
99                          GDK_BUTTON_PRESS_MASK |
100                         GDK_BUTTON_RELEASE_MASK);
101  g_signal_connect (G_OBJECT (window), "button-press-event",
102                    G_CALLBACK (cb_mouse_press), NULL);
103  g_signal_connect (G_OBJECT (window), "button-release-event",
104                    G_CALLBACK (cb_mouse_release), NULL);
105
106  hand_open  = make_cursor (hand_open_data, hand_open_mask, 20, 20);
107  hand_close = make_cursor (hand_close_data, hand_close_mask, 20, 20);
108
109  gtk_widget_show_all (window);
110  gdk_window_set_cursor (window->window, hand_open);
111  gtk_main ();
112
113  return 0;
114 }
```

10.2　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．

10.2.1　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．

10.2.2　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);
```

10.2.3　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 10–2 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 10-2 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

```
1  #include <gtk/gtk.h>
2
3  static gboolean
4  cb_key_press (GtkWidget   *widget,
5                GdkEventKey *event,
6                gpointer    user_data)
7  {
8    g_print ("keyval␣=␣%d,␣state␣=␣%d,␣string␣=␣%s\n",
9             event->keyval, event->state, event->string);
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), "Key␣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_KEY_PRESS_MASK);
27   g_signal_connect (G_OBJECT (window), "key-press-event",
28                     G_CALLBACK (cb_key_press), NULL);
29
30   gtk_widget_show_all (window);
31   gtk_main ();
```

```
32
33   return 0;
34 }
```

10.3 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．

10.3.1 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);
```

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 10.1 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 10.1 GtkDestDefault 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 10.2 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 10.2 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 10.3 を参照）の論理和で指定します．

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．

表 10.3　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.

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 10.1）．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，71–73 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，74–75 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 23–55 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．

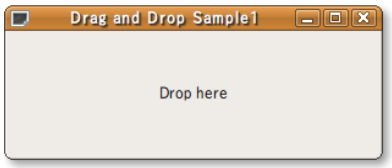


図 10.1　ドロップ操作のサンプル

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},
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   int   n;
35
36   g_print ("data->target␣=␣%s\n", gdk_atom_name (data->target));
37   g_print ("data->type␣␣␣=␣%s\n", gdk_atom_name (data->type));
38   g_print ("data->length␣=␣%d\n", data->length);
39   g_print ("data->format␣=␣%d\n", data->format);
40   g_print ("info␣␣␣␣␣␣␣␣␣=␣%d\n", info);
41
42   if (data->length > 0 && data->format == 8)
43     {
44       g_print ("Received␣string␣=␣");
45       received = g_strchomp (data->data);
46       for (n = 0; n < data->length; n++)
47         {
48           if (received[n] != '\0') g_print ("%c", received[n]);
49         }
50       g_print ("\n");
51       gtk_drag_finish (context, TRUE, FALSE, time);
52       return;
53     }
54   gtk_drag_finish (context, FALSE, FALSE, time);
55 }
56
57 int
58 main (int argc, char **argv)
59 {
```

```
60   GtkWidget *window;
61   GtkWidget *label;
62
63   gtk_init (&argc, &argv);
64
65   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
66   gtk_window_set_title (GTK_WINDOW (window), "Drag␣and␣Drop␣Sample1");
67   gtk_widget_set_size_request (window, 300, 100);
68   g_signal_connect (G_OBJECT (window), "destroy",
69                     G_CALLBACK (gtk_main_quit), NULL);
70
71   gtk_drag_dest_set (window,
72                      GTK_DEST_DEFAULT_ALL, target_table, ntargets - 1,
73                      GDK_ACTION_COPY | GDK_ACTION_MOVE);
74   g_signal_connect (window, "drag-data-received",
75                     G_CALLBACK (cb_drag_data_received), NULL);
76
77   label = gtk_label_new ("Drop␣here");
78   gtk_container_add (GTK_CONTAINER (window), label);
79
80   gtk_widget_show_all (window);
81   gtk_main ();
82
83   return 0;
84 }
```

10.3.2　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);
```

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);
```

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．

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 10.2 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．

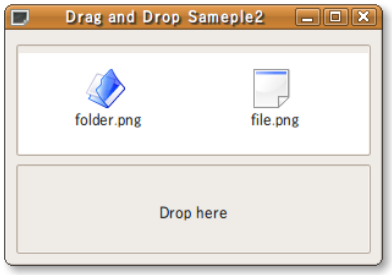


図 10.2　ドラッグ&ドロップアプリケーション

ソース 10-3-2 ドラッグ機能の実装：dnd-sample2.c

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

enum
{
  DROP_URI_LIST,
  DROP_X_MOZ_URL,
  DROP_HTML,
  DROP_TEXT_PLAIN,
  DROP_STRING
};

static GtkTargetEntry target_table[] =
{
  {"text/uri-list",  0, DROP_URI_LIST},
  {"text/x-moz-url", 0, DROP_X_MOZ_URL},
  {"text/html",      0, DROP_HTML},
  {"text/plain",     0, DROP_TEXT_PLAIN},
  {"STRING",         0, DROP_STRING}
};

static guint ntargets = sizeof (target_table) / sizeof (target_table[0]);

enum
{
  COLUMN_NAME,
  COLUMN_PIXBUF,
  N_COLUMNS
};

static void
add_data (GtkIconView *iconview)
{
  GdkPixbuf     *folder_pixbuf;
  GdkPixbuf     *file_pixbuf;
  GtkListStore  *store;
  GtkTreeIter   iter;

  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;
83
84   model
85     = GTK_TREE_MODEL (gtk_icon_view_get_model (GTK_ICON_VIEW (widget)));
86   root = gtk_icon_view_get_selected_items (GTK_ICON_VIEW (widget));
87
88   for (list = root; list; list = g_list_next (list))
89     {
90       GtkTreePath *path;
91       GtkTreeIter iter;
92       gchar       *name;
93
94       path = (GtkTreePath *) list->data;
95       gtk_tree_model_get_iter (model, &iter, path);
96       gtk_tree_model_get (model, &iter, COLUMN_NAME, &name, -1);
97       string_list = g_list_append (string_list, name);
98     }
99   string = g_string_new ((const gchar *) string_list->data);
100  for (list = g_list_next (string_list); list; list = g_list_next (list))
101    {
102      string = g_string_append (string, "\n");
103      string = g_string_append (string, (const gchar *) list->data);
104    }
105  gtk_selection_data_set (data, data->target, 8,
106                          string->str, strlen (string->str));
107  g_string_free (string, FALSE);
108 }
109
110 static void
111 cb_drag_data_received (GtkWidget         *widget,
112                        GdkDragContext    *context,
113                        gint              x,
114                        gint              y,
115                        GtkSelectionData  *data,
```

```
116                         guint            info,
117                         guint            time)
118 {
119   gchar *received;
120   int   n;
121
122   if ((data->length >= 0) && (data->format == 8))
123     {
124       g_print ("Received␣string␣=␣");
125       received = g_strchomp (data->data);
126       for (n = 0; n < data->length; n++)
127         {
128           if (data->data[n] != '\0') g_print ("%c", data->data[n]);
129         }
130       g_print ("\n");
131       return;
132     }
133   gtk_drag_finish (context, FALSE, FALSE, time);
134 }
135
136 int
137 main (int argc, char **argv)
138 {
139   GtkWidget *window;
140   GtkWidget *vbox;
141   GtkWidget *iconview;
142   GtkWidget *frame;
143   GtkWidget *label;
144
145   gtk_init (&argc, &argv);
146
147   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
148   gtk_window_set_title (GTK_WINDOW (window), "Drag␣and␣Drop␣Sameple2");
149   gtk_widget_set_size_request (window, 300, -1);
150   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
151   g_signal_connect (G_OBJECT (window), "destroy",
152                     G_CALLBACK (gtk_main_quit), NULL);
153
154   vbox = gtk_vbox_new (FALSE, 0);
155   gtk_container_add (GTK_CONTAINER (window), vbox);
156
157   frame = gtk_frame_new ("");
158   gtk_box_pack_start (GTK_BOX (vbox), frame, TRUE, TRUE, 0);
159   gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_IN);
160
161   iconview = create_icon_view_widget ();
162   gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview), COLUMN_NAME);
163   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW (iconview),
164                                    COLUMN_PIXBUF);
165   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 128);
166
167   gtk_icon_view_enable_model_drag_source (GTK_ICON_VIEW (iconview),
168                                           GDK_BUTTON1_MASK,
169                                           target_table,
170                                           ntargets,
171                                           GDK_ACTION_COPY |
172                                           GDK_ACTION_MOVE);
173
174   gtk_icon_view_set_selection_mode (GTK_ICON_VIEW (iconview),
175                                     GTK_SELECTION_MULTIPLE);
176   g_signal_connect (iconview, "drag-data-get",
177                     G_CALLBACK (source_drag_data_get), NULL);
178   gtk_container_add (GTK_CONTAINER (frame), iconview);
179   add_data (GTK_ICON_VIEW (iconview));
180
181   frame = gtk_frame_new ("");
182   gtk_widget_set_size_request (frame, -1, 80);
183   gtk_box_pack_start (GTK_BOX (vbox), frame, TRUE, TRUE, 0);
184   gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_IN);
185
186   gtk_drag_dest_set (frame,
187                      GTK_DEST_DEFAULT_ALL, target_table, ntargets,
188                      GDK_ACTION_COPY | GDK_ACTION_MOVE);
189   g_signal_connect (frame, "drag-data-received",
190                     G_CALLBACK(cb_drag_data_received), NULL);
191
```

```
192   label = gtk_label_new ("Drop here");
193   gtk_container_add (GTK_CONTAINER (frame), label);
194
195   gtk_widget_show_all (window);
196   gtk_main ();
197
198   return 0;
199 }
```

10.3.3　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 10.3 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，123–143 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 105–108 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．

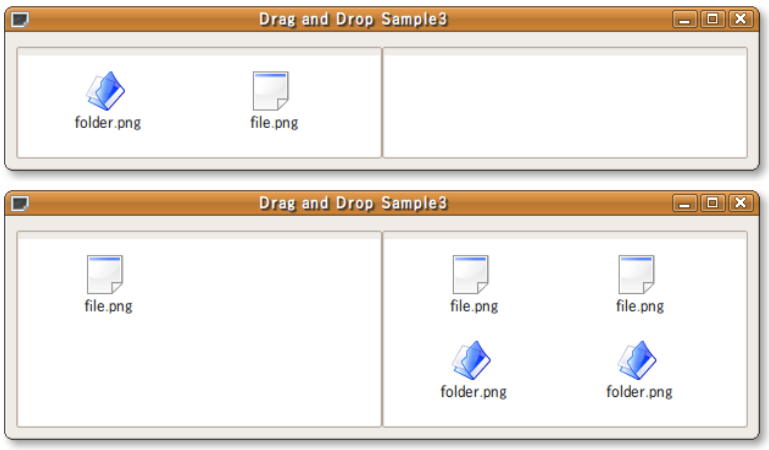


図 10.3　ドラッグ＆ドロップの実装

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, data->target, 8,
57                               string->str, strlen (string->str));
58       g_string_free (string, FALSE);
59     }
60 }
61
62 static void
63 cb_drag_data_received (GtkWidget         *widget,
64                        GdkDragContext    *context,
65                        gint              x,
66                        gint              y,
67                        GtkSelectionData  *data,
68                        guint             info,
69                        guint             time)
70 {
71   GtkWidget     *source;
72
73   source = gtk_drag_get_source_widget (context);
74   if (source == widget)
75     {
76       gtk_drag_finish (context, FALSE, FALSE, time);
77       return;
78     }
79   if ((data->length >= 0) && (data->format == 8))
80     {
81       GtkTreeModel *src_model;
82       GtkTreeModel *dst_model;
83       gchar        **strlist;
84       gchar        *received;
85       int          n;
86
87       src_model = gtk_icon_view_get_model (GTK_ICON_VIEW (source));
88       dst_model = gtk_icon_view_get_model (GTK_ICON_VIEW (widget));
89
90       received = g_strchomp ((gchar *) data->data);
91       strlist = g_strsplit (received, ",", 0);
92
93       for (n = 0; strlist[n]; n++)
94         {
95           GtkTreeIter iter;
96           GdkPixbuf   *pixbuf;
97           gchar       *name;
```

```
98
99            gtk_tree_model_get_iter_from_string (src_model,
100                                                 &iter, strlist[n]);
101           gtk_tree_model_get (src_model, &iter,
102                               COLUMN_NAME, &name,
103                               COLUMN_PIXBUF, &pixbuf, -1);
104
105           if (context->action == GDK_ACTION_MOVE)
106             {
107               gtk_list_store_remove (GTK_LIST_STORE (src_model), &iter);
108             }
109
110           gtk_list_store_append (GTK_LIST_STORE (dst_model), &iter);
111           gtk_list_store_set (GTK_LIST_STORE (dst_model), &iter,
112                               COLUMN_NAME, name,
113                               COLUMN_PIXBUF, pixbuf, -1);
114         }
115       g_strfreev (strlist);
116       gtk_drag_finish (context, TRUE, FALSE, time);
117
118       return;
119     }
120   gtk_drag_finish (context, FALSE, FALSE, time);
121 }
122
123 static void
124 cb_drag_data_delete (GtkWidget      *widget,
125                      GdkDragContext *drag_context,
126                      gpointer       user_data)
127 {
128   GtkTreeModel  *model;
129   GList         *root, *list;
130
131   model
132     = GTK_TREE_MODEL(gtk_icon_view_get_model (GTK_ICON_VIEW(widget)));
133   root = gtk_icon_view_get_selected_items (GTK_ICON_VIEW(widget));
134   for (list = root; list; list = g_list_next(list))
135     {
136       GtkTreePath *path;
137       GtkTreeIter iter;
138
139       path = (GtkTreePath *) list->data;
140       gtk_tree_model_get_iter (model, &iter, path);
141       gtk_list_store_remove (GTK_LIST_STORE(model), &iter);
142     }
143 }
144
145 static void
146 add_data (GtkIconView *iconview)
147 {
148   GdkPixbuf     *folder_pixbuf;
149   GdkPixbuf     *file_pixbuf;
150   GtkListStore  *store;
151   GtkTreeIter   iter;
152
153   folder_pixbuf = gdk_pixbuf_new_from_file ("./folder.png", NULL);
154   file_pixbuf = gdk_pixbuf_new_from_file ("./file.png", NULL);
155
156   store = GTK_LIST_STORE(gtk_icon_view_get_model (iconview));
157
158   gtk_list_store_clear (store);
159
160   gtk_list_store_append (store, &iter);
161   gtk_list_store_set (store, &iter,
162                       COLUMN_NAME, "folder.png",
163                       COLUMN_PIXBUF, folder_pixbuf, -1);
164   g_object_unref (folder_pixbuf);
165
166   gtk_list_store_append (store, &iter);
167   gtk_list_store_set (store, &iter,
168                       COLUMN_NAME, "file.png",
169                       COLUMN_PIXBUF, file_pixbuf, -1);
170   g_object_unref (file_pixbuf);
171 }
172
173 static GtkWidget*
```

```
174 create_icon_view_widget (void)
175 {
176   GtkWidget     *iconview;
177   GtkListStore *store;
178
179   store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING, GDK_TYPE_PIXBUF);
180   iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL (store));
181   g_object_unref (store);
182
183   gtk_icon_view_set_text_column (GTK_ICON_VIEW (iconview), COLUMN_NAME);
184   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW (iconview),
185                                    COLUMN_PIXBUF);
186   gtk_icon_view_set_item_width (GTK_ICON_VIEW (iconview), 128);
187
188   gtk_icon_view_enable_model_drag_source (GTK_ICON_VIEW (iconview),
189                                           GDK_BUTTON1_MASK |
190                                           GDK_SHIFT_MASK,
191                                           target_table,
192                                           ntargets,
193                                           GDK_ACTION_COPY |
194                                           GDK_ACTION_MOVE);
195   gtk_drag_dest_set (iconview,
196                      GTK_DEST_DEFAULT_ALL, target_table, ntargets,
197                      GDK_ACTION_COPY | GDK_ACTION_MOVE);
198   gtk_icon_view_set_selection_mode (GTK_ICON_VIEW (iconview),
199                                     GTK_SELECTION_MULTIPLE);
200   g_signal_connect (iconview, "drag-data-get",
201                     G_CALLBACK (source_drag_data_get), NULL);
202   g_signal_connect (iconview, "drag-data-received",
203                     G_CALLBACK (cb_drag_data_received), NULL);
204   g_signal_connect (iconview, "drag-data-delete",
205                     G_CALLBACK (cb_drag_data_delete), NULL);
206
207   return iconview;
208 }
209
210 int
211 main (int argc, char **argv)
212 {
213   GtkWidget *window;
214   GtkWidget *hbox;
215   GtkWidget *iconview;
216   GtkWidget *frame;
217
218   gtk_init (&argc, &argv);
219
220   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
221   gtk_window_set_title (GTK_WINDOW (window), "Drag␣and␣Drop␣Sample3");
222   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
223   g_signal_connect (G_OBJECT (window), "destroy",
224                     G_CALLBACK (gtk_main_quit), NULL);
225
226   hbox = gtk_hbox_new (FALSE, 0);
227   gtk_container_add (GTK_CONTAINER (window), hbox);
228
229   frame = gtk_frame_new ("");
230   gtk_widget_set_size_request (frame, 300, -1);
231   gtk_box_pack_start (GTK_BOX (hbox), frame, TRUE, TRUE, 0);
232   gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_IN);
233
234   iconview = create_icon_view_widget ();
235   gtk_container_add (GTK_CONTAINER (frame), iconview);
236   add_data (GTK_ICON_VIEW (iconview));
237
238   frame = gtk_frame_new ("");
239   gtk_widget_set_size_request (frame, 300, -1);
240   gtk_box_pack_start (GTK_BOX (hbox), frame, TRUE, TRUE, 0);
241   gtk_frame_set_shadow_type (GTK_FRAME (frame), GTK_SHADOW_IN);
242
243   iconview = create_icon_view_widget ();
244   gtk_container_add (GTK_CONTAINER (frame), iconview);
245
246   gtk_widget_show_all (window);
247   gtk_main ();
248
249   return 0;
```

```
250 }
```

10.4　オリジナルストックアイテムの作成

ここでは，独自のストックアイテムを作成する方法を解説します．独自のストックアイテムを作成できれば，既存の関数を使って，自分で作ったアイコンを使用したボタンやツールバーなどを簡単に作れます．しかしわざわざストックアイテムを作成しなくても，既存の関数をいくつか組み合わせることで，独自のアイコンを使用したボタンなどを作ることは可能です．では独自のストックアイテムを作成する利点は何でしょうか．

それはメニューの作成です．UI マネージャでは，メニューに表示するアイコンをストックアイテムで指定するので，独自のストックアイテムを作成することで，UI マネージャの枠組を保ったまま，オリジナルのメニューを作成できるようになるのです．

10.4.1　ストックアイテム作成

オリジナルストックアイテムの作成は次のような流れになります．

1. GtkIconFactory の生成
2. アイコン情報の設定
3. ストックアイテムの登録
4. ストックアイテムの有効化

GtkIconFactory の生成

GtkIconFactory は，新しく作成するストックアイテムを管理するオブジェクトです．新しいストックアイテムの作成は，ストックアイテムの情報をこのオブジェクトに登録することに相当します．このオブジェクトを作成するには次の関数を使用します．

```
GtkIconFactory* gtk_icon_factory_new (void);
```

アイコン情報の設定

ここでいうアイコン情報とは，アイコン画像のファイル名もしくはアイコンの GdkPixbuf データのことを指します．このアイコン情報は GtkIconSource を経由して GtkIconSet に渡します．それぞれのウィジェットの作成には，次の関数を使用します．

```
GtkIconSource* gtk_icon_source_new (void);

GtkIconSet* gtk_icon_set_new (void);
```

アイコン情報は，アイコン画像のファイル名もしくはアイコンの GdkPixbuf データのどちらかを指定します．どちらを使用するかによって以下の関数を使い分けます．なお，アイコン画像のファイル名は絶対パスで指定する必要があります．

```
void gtk_icon_source_set_filename (GtkIconSource *source,
                                   const gchar   *filename);

void gtk_icon_source_set_pixbuf (GtkIconSource *source,
                                 GdkPixbuf     *pixbuf);
```

そして，アイコン情報をセットした GtkIconSource 型の変数を，次の関数で GtkIconSet 型の変数に渡します．

```
void gtk_icon_set_add_source (GtkIconSet          *icon_set,
                              const GtkIconSource *source);
```

ストックアイテムの登録

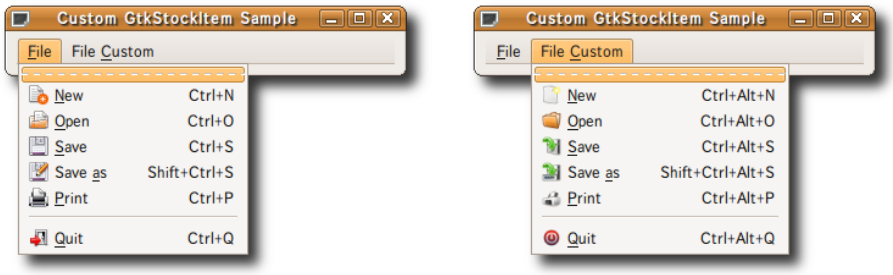
次にアイコン情報をセットした GtkIconSet 型の変数とストック ID（ストックアイテムを特定するための文字列）を組にして，GtkIconFactroy 型の変数に登録します．

```
void gtk_icon_factory_add (GtkIconFactory *factory,
                           const gchar    *stock_id,
                           GtkIconSet     *icon_set);
```

ストックアイテムの有効化

GtkIconFactory にストックアイテムを登録しただけでは，プログラムの中で使うことができません．登録した新しいストックアイテムを有効にするためには，次の関数を呼び出す必要があります．

```
void gtk_icon_factory_add_default (GtkIconFactory *factory);
```



(a) 標準のストックアイテム　　(b) 独自のストックアイテム

図 10.4　独自のストックアイテムを使用したメニュー

10.4.2　サンプルプログラム

オリジナルストックアイテムを使用したメニューのサンプルプログラムを，ソース 10–4 に示します．プログラムの実行結果は図 10.4 になります．

ソース 10-4 オリジナルストックアイテムのサンプルプログラム：customstockitem-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void cb_quit (GtkAction *action, gpointer user_data);
4
5  typedef struct
6  {
7    gchar *location;
8    gchar *stock_id;
9  } CustomStockIcon;
10
11 /*
12  * 実行環境に応じてMakefile内のICON_PATHを修正してください．
13  */
14 static CustomStockIcon custom_icon_list[] =
15 {
16   {ICON_PATH"images/new.png", "custom-stock-new"},
17   {ICON_PATH"images/open.png", "custom-stock-open"},
18   {ICON_PATH"images/save.png", "custom-stock-save"},
19   {ICON_PATH"images/saveas.png", "custom-stock-saveas"},
20   {ICON_PATH"images/print.png", "custom-stock-print"},
21   {ICON_PATH"images/quit.png", "custom-stock-quit"},
22   {NULL, NULL}
23 };
24
25 static GtkActionEntry entries[] =
26 {
27   {"FileMenu1", NULL, "_File"},
28   {"New", GTK_STOCK_NEW, "_New", "<control>N", NULL, NULL},
29   {"Open", GTK_STOCK_OPEN, "_Open", "<control>O", NULL, NULL},
30   {"Save", GTK_STOCK_SAVE, "_Save", "<control>S", NULL, NULL},
31   {"SaveAs", GTK_STOCK_SAVE_AS, "Save␣_as", "<shift><control>S",
32    NULL, NULL},
33   {"Print", GTK_STOCK_PRINT, "_Print", "<control>P", NULL, NULL},
34   {"Quit", GTK_STOCK_QUIT, "_Quit", "<control>Q", NULL,
35    G_CALLBACK (cb_quit)},
36
37   {"FileMenu2", NULL, "File␣_Custom"},
38   {"cNew", "custom-stock-new",
39    "_New", "<alt><control>N", NULL, NULL},
40   {"cOpen", "custom-stock-open",
```

```
41     "_Open", "<alt><control>O", NULL, NULL},
42    {"cSave", "custom-stock-save",
43     "_Save", "<alt><control>S", NULL, NULL},
44    {"cSaveAs", "custom-stock-saveas",
45     "Save␣_as", "<alt><shift><control>S", NULL, NULL},
46    {"cPrint", "custom-stock-print",
47     "_Print", "<alt><control>P", NULL, NULL},
48    {"cQuit", "custom-stock-quit",
49     "_Quit", "<alt><control>Q", NULL, G_CALLBACK (cb_quit)},
50  };
51
52  static guint n_entries = G_N_ELEMENTS (entries);
53
54  const gchar *ui_info =
55  "<ui>"
56  "␣␣<menubar␣name='MenuBar'>"
57  "␣␣␣␣<menu␣action='FileMenu1'>"
58  "␣␣␣␣␣␣<menuitem␣action='New'/>"
59  "␣␣␣␣␣␣<menuitem␣action='Open'/>"
60  "␣␣␣␣␣␣<menuitem␣action='Save'/>"
61  "␣␣␣␣␣␣<menuitem␣action='SaveAs'/>"
62  "␣␣␣␣␣␣<menuitem␣action='Print'/>"
63  "␣␣␣␣␣␣<separator/>"
64  "␣␣␣␣␣␣<menuitem␣action='Quit'/>"
65  "␣␣␣␣</menu>"
66  "␣␣␣␣<menu␣name='FileMenu2'␣action='FileMenu2'>"
67  "␣␣␣␣␣␣<menuitem␣action='cNew'/>"
68  "␣␣␣␣␣␣<menuitem␣action='cOpen'/>"
69  "␣␣␣␣␣␣<menuitem␣action='cSave'/>"
70  "␣␣␣␣␣␣<menuitem␣action='cSaveAs'/>"
71  "␣␣␣␣␣␣<menuitem␣action='cPrint'/>"
72  "␣␣␣␣␣␣<separator/>"
73  "␣␣␣␣␣␣<menuitem␣action='cQuit'/>"
74  "␣␣␣␣</menu>"
75  "␣␣</menubar>"
76  "</ui>";
77
78  static void
79  cb_quit (GtkAction *action, gpointer user_data)
80  {
81    GObject *window = G_OBJECT (user_data);
82
83    g_object_unref (g_object_get_data (window, "factory"));
84    g_object_unref (g_object_get_data (window, "ui"));
85    gtk_main_quit ();
86  }
87
88  static void
89  create_custom_stocks (GtkIconFactory  *factory,
90                        CustomStockIcon *list)
91  {
92    GtkIconSource *source;
93    GtkIconSet    *iconset;
94    int           n;
95
96    for (n = 0; list[n].location != NULL; n++)
97      {
98        source  = gtk_icon_source_new ();
99        iconset = gtk_icon_set_new ();
100       gtk_icon_source_set_filename (source, list[n].location);
101       gtk_icon_set_add_source (iconset, source);
102       gtk_icon_factory_add (factory, list[n].stock_id, iconset);
103       gtk_icon_source_free (source);
104       gtk_icon_set_unref (iconset);
105     }
106   gtk_icon_factory_add_default (factory);
107 }
108
109 static GtkUIManager*
110 create_menu (GtkWidget *parent)
111 {
112   GtkUIManager   *ui;
113   GtkActionGroup *actions;
114
115   actions = gtk_action_group_new ("Actions");
116   gtk_action_group_add_actions (actions, entries, n_entries,
```

```
117                                 (gpointer) parent);
118   ui = gtk_ui_manager_new ();
119   gtk_ui_manager_insert_action_group (ui, actions, 0);
120   gtk_ui_manager_set_add_tearoffs (ui, TRUE);
121   gtk_window_add_accel_group (GTK_WINDOW(parent),
122                               gtk_ui_manager_get_accel_group (ui));
123   gtk_ui_manager_add_ui_from_string (ui, ui_info, -1, NULL);
124
125   return ui;
126 }
127
128 int
129 main (int argc, char **argv)
130 {
131   GtkWidget      *window;
132   GtkIconFactory *factory;
133   GtkWidget      *menubar;
134   GtkUIManager   *ui;
135
136   gtk_init (&argc, &argv);
137   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
138   gtk_window_set_title  (GTK_WINDOW (window),
139                          "Custom␣GtkStockItem␣Sample");
140   gtk_widget_set_size_request (window, 320, -1);
141   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
142   g_signal_connect (G_OBJECT (window), "destroy",
143                     G_CALLBACK (gtk_main_quit), NULL);
144
145   factory = gtk_icon_factory_new ();
146   create_custom_stocks (factory, custom_icon_list);
147   g_object_set_data (G_OBJECT (window), "factory", factory);
148
149   ui = create_menu (window);
150   g_object_set_data (G_OBJECT (window), "ui", ui);
151   menubar = gtk_ui_manager_get_widget (ui, "/MenuBar");
152   gtk_container_add (GTK_CONTAINER (window), menubar);
153
154   gtk_widget_show_all (window);
155   gtk_main ();
156
157   return 0;
158 }
```

10.5　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，ソース 10–5 を例に解説します．

10.5.1　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 10.4 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．

表 10.4　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.

メンバ arg には，列挙体 GOptionArg で定義された値（表 10.5 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．

表 10.5　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.

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．

ソース 10–5 では，51–74 行目でオプションを定義しています．この例では，以下に示すタイプのオプションを使用しています．また，この例では，オプションの引数等をまとめて扱うために，構造体 Option を定義しています（4–11 行目）．

- 引数を取らないオプション（G_OPTION_ARG_NONE）
- 整数値の引数を取るオプション（G_OPTION_ARG_INT）
- 実数値の引数を取るオプション（G_OPTION_ARG_DOUBLE）
- ファイル名を引数に取るオプション（G_OPTION_ARG_FILENAME）
- 文字列を引数に取る複数使用可能なオプション（G_OPTION_ARG_STRNIG_ARRAY）
- 独自のオプション解析関数を使用するオプション（G_OPTION_ARG_CALLBACK）

最後のオプションは，GOptionArg に G_OPTION_ARG_CALLBACK を指定したオプションで，これはこのオプションで与えた引数をユーザが設定した独自の関数で解析したいときに使用します．

呼び出すコールバック関数を 71 行目で指定して，24–49 行目でコールバック関数を定義しています．このオプションでは，コロンで区切って 2 つの引数を指定することにし，コールバック関数で引数の文字列を分解して，オプションの引数を解析しています．

10.5.2　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．

ソース 10–5 の例は，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します．そのため，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);
```

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);
```

また，関数 g_option_group_new で GOptionGroup 型の変数を作成します．

```
GOptionGroup* g_option_group_new (const gchar *name,
                                  const gchar *description,
                                  const gchar *help_description,
                                  gpointer user_data,
                                  GDestroyNotify destroy);
```

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);
```

さらに，関数 g_option_group_set_parse_hooks によって，オプション解析の前後に呼び出す関数を設定できます．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);
```

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);
```

ソース 10–5 では 132–134 行目でこの設定を行い，76–93 行目でオプション解析前に呼ばれる関数を定義し，オプションメンバの初期化処理を行い，95–110 行目でオプション解析後に呼ばれる関数を定義しています．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 10–5 の 136–141 行目では，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います．

10.5.3　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);
```

もし，オプション解析に失敗した場合には FALSE を返します．また，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．

ソース 10–5 の 144 行目以降は，与えられた引数とオプションの値を用いてメインの処理を行っています．

ソース 10-5 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   if (nargs < 2)
36     {
37       g_print ("For␣%s␣option,␣"
38                "please␣specify␣two␣option␣arguments␣by␣separating␣"
39                "\":\".\n\n",
40                option_name);
41       return FALSE;
42     }
43   option.step = atof (option_strings[0]);
44   option.iterations = atoi (option_strings[1]);
45
46   g_strfreev (option_strings);
47
48   return TRUE;
49 }
50
51 static GOptionEntry entries[] =
52 {
53   {"verbose", 'v',
54    G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_NONE,
55    &option.verbose, "Verbose␣mode", NULL},
56   {"iterations", 'i',
57    G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_INT,
58    &option.iterations, "Number␣of␣iterations␣(Default␣10)", NULL},
59   {"step", 's',
60    G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_DOUBLE,
61    &option.step, "Step␣value␣(Default␣1)", NULL},
62   {"filename", 'f',
63    G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_FILENAME,
64    &option.filename, "Specify␣a␣filename", NULL},
65   {"string-array", 'a',
66    G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_STRING_ARRAY,
67    &option.string_array,
68    "Specify␣a␣string.␣This␣option␣can␣be␣used␣multiple␣times.", NULL},
69   {"callback", 'c',
```

```
70    G_OPTION_FLAG_NOALIAS, G_OPTION_ARG_CALLBACK,
71    special_option_parse_func, "The␣sample␣callback␣option",
72    "step:iterations"},
73   NULL
74 };
75
76 static gboolean
77 pre_parse_func (GOptionContext  *context,
78                 GOptionGroup    *group,
79                 gpointer        user_data,
80                 GError          *error)
81 {
82   Option *option = (Option *) user_data;
83
84   g_print ("\nInitializing␣option␣members␣...\n\n");
85
86   option->verbose      = FALSE;
87   option->iterations   = 10;
88   option->step         = 1;
89   option->filename     = NULL;
90   option->string_array = NULL;
91
92   return TRUE;
93 }
94
95 static gboolean
96 post_parse_func (GOptionContext *context,
97                  GOptionGroup   *group,
98                  gpointer       user_data,
99                  GError         *error)
100 {
101   Option *option = (Option *) user_data;
102
103   if (option->verbose)
104     {
105       g_print ("Parsing␣option␣is␣done.\n"
106                "(This␣message␣is␣shown␣if␣the␣verbose␣option␣is␣"
107                "specified.)\n\n");
108     }
109   return TRUE;
110 }
111
112 int
113 main (int argc, char *argv[])
114 {
115   GError         *error = NULL;
116   GOptionContext *context;
117   GOptionGroup   *option_group;
118   gint           n;
119   gchar          *help_message;
120   gdouble        val;
121   gboolean       result;
122
123   context = g_option_context_new ("#start");
124   g_option_context_add_main_entries (context, entries, NULL);
125   option_group
126     = g_option_group_new ("example",
127                           "Example:␣./parse-option␣-s␣0.1␣-i␣5␣1",
128                           "Show␣an␣example",
129                           (gpointer) &option,
130                           (GDestroyNotify) option_free);
131   g_option_group_add_entries (option_group, entries);
132   g_option_group_set_parse_hooks (option_group,
133                                   (GOptionParseFunc) pre_parse_func,
134                                   (GOptionParseFunc) post_parse_func);
135   g_option_context_add_group (context, option_group);
136   g_option_context_set_summary (context,
137                                 "This␣program␣is␣an␣option␣parser␣"
138                                 "example.");
139   g_option_context_set_description (context,
140                                 "In␣this␣line,␣please␣describe␣detailed"
141                                 "descriptions␣about␣this␣program.\n");
142   result = g_option_context_parse (context, &argc, &argv, &error);
143
144   if (!result || argc != 2)
145     {
```

```
146       help_message = g_option_context_get_help (context, TRUE, NULL);
147       g_print ("%s\n", help_message);
148       g_free (help_message);
149       g_option_context_free (context);
150       exit (1);
151     }
152   val = atof (argv[1]);
153
154   if (option.verbose)
155     {
156       g_print ("␣␣␣␣n␣|␣number\n------+----------\n");
157     }
158   for (n = 0; n < option.iterations; n++)
159     {
160       g_print ("%5d␣␣␣%f\n", n + 1, val);
161       val += option.step;
162     }
163   if (option.filename)
164     {
165       g_print ("option␣filename␣=␣%s\n", option.filename);
166     }
167   if (option.string_array)
168     {
169       n = 0;
170       while (option.string_array[n])
171         {
172           g_print ("option␣string-array␣=␣%s\n",
173                    option.string_array[n++]);
174         }
175     }
176   g_option_context_free (context);
177
178   return 0;
179 }
```

付録 A

GTK+ の開発環境の構築

A.1　GTK+ 開発環境のインストール – Fedora 11 編 –

Fedora 11 を使用する場合，OS のインストール途中で「ソフトウェア開発」をインストールするように選択しておけば，自動的に GTK+ の開発環境がインストールされます．ここでは「ソフトウェア開発」がインストールされていない場合について，Ubuntu と同様に GUI によるインストール方法とコマンドラインによるインストール方法の 2 つを紹介します．

GUI によるアプリケーションのインストール – Fedora 11 編 –

Fedora 11 で GUI によって GTK+ の開発環境をインストールするには，以下の手順に従って作業を行ってください．

1. メニューバーから「システム」→「管理」→「ソフトウェアの追加/削除」を選択します（図 A.1）．

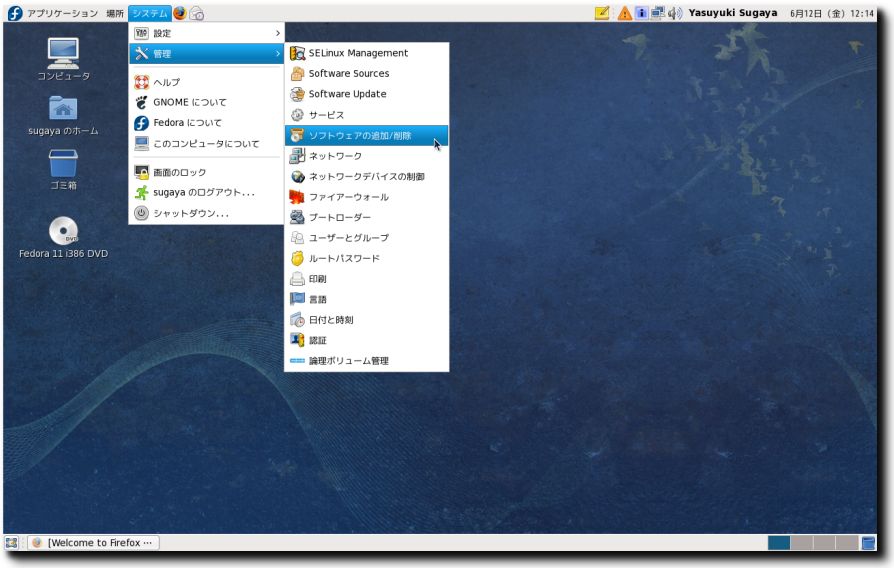
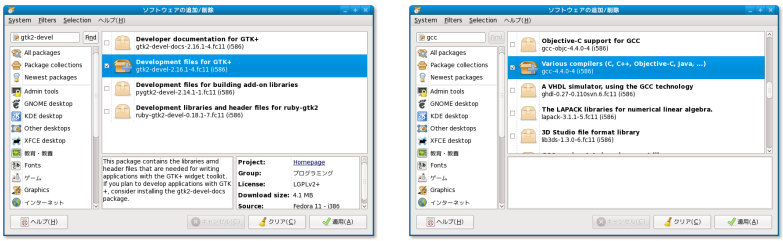


図 A.1　ソフトウェアの追加/削除の起動

2. 次に，表示されたウィンドウの左上の入力ボックスに “gtk2-devel” と入力し，検出された「gtk2-devel」をクリックしてチェックを入れます（図 A.2(a)）．
3. 同様に “gcc” を検索して，検出された「gcc」をクリックしてチェックを入れます（図 A.2(b)）．
4. 2 つのパッケージを選択したら，ウィンドウ右下の「適用 (A)」ボタンをクリックします．図 A.3 のウィンドウが表示さ

れるので，「install」ボタンを押してください．



(a)　(b)

図 A.2　パッケージの検索

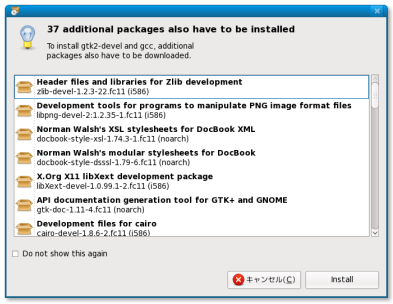


図 A.3　選択したパッケージのインストール

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 – Fedora 11 編 –

Fedora 11 では yum というコマンドを利用してパッケージの追加や削除を行います．gtk2-devel と gcc を yum コマンドを利用してインストールするには次のようにします．

```
$ su ↵ パスワードの入力
$ yum install gtk2-devel gcc ↵
```

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くるので，“y” を入力して処理を続けます（-y オプションを指定しておくと，この “y” を入力を省くことができます）．

```
Loaded plugins: refresh-packagekit
Setting up Install Process
Parsing package install arguments
Resolving Dependencies
...
Transaction Summary
================================================================================
Install      38 Package(s)
Update        0 Package(s)
Remove        0 Package(s)

Total download size: 30 M
Is this ok [y/N]: y
```

このほかに anjuta や devhelp，gtranslator も，以下のようにインストールすることができます．

```
$ su ↵ パスワードの入力
$ yum install anjuta devhelp gtranslator ↵
```

A.2　GTK+ 開発環境のインストール – OpenSUSE 11.1 編 –

OpenSUSE 11.1 でも Fedora 10 と同様に，OS をインストールしただけの状態では GTK+ の開発環境のみならず gcc もインストールされていないので，GUI もしくはコマンドラインから GTK+ の開発環境と gcc をインストールする必要があります．

GUI によるアプリケーションのインストール – OpenSUSE 11.1 編 –

OpenSUSE 11.1 で GTK+ の開発環境を GUI を利用してインストールするには以下のようにします．

1. デスクトップ画面の左下にあるコンピュータボタンをクリックして，メニューを表示し，メニュー右側にある「ソフトウェアのインストール」をクリックしてください（図 A.4）．メニューに「ソフトウェアのインストール」がない場合には，メニューからコントロール・センターを表示して，コントロール・センターから「ソフトウェアのインストール」をクリックして，ソフトウェアマネージャを起動してください（図 A.5）．

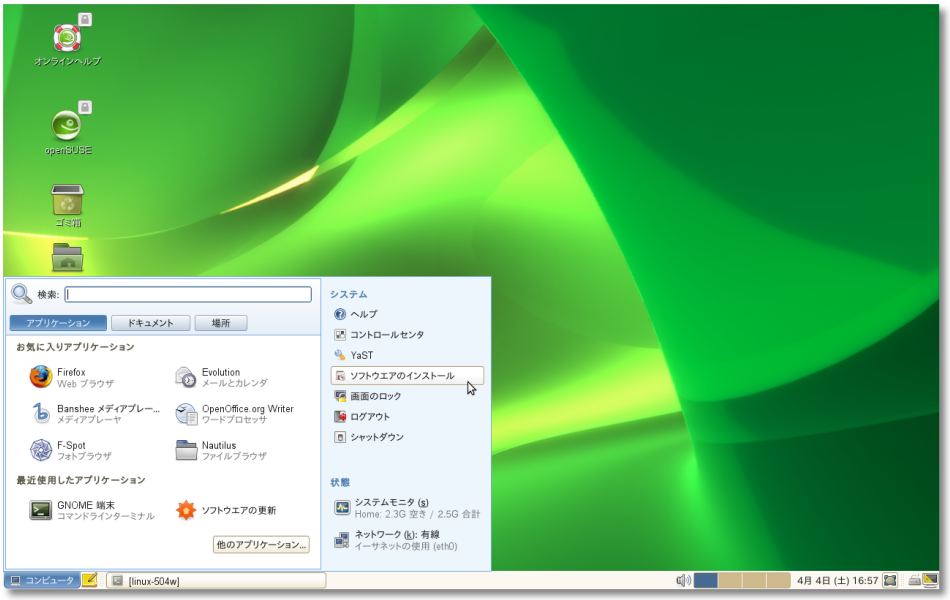


図 A.4　ソフトウェアマネージャの起動

2. 「ソフトウェアマネージャ」を起動するためにシステム管理者のパスワードの入力を求められるので，入力してください（図 A.6）．
3. ソフトウェアマネージャ画面のパッケージ一覧というラベルの右にある入力ボックスに “gtk2-devel” と入力してください．パッケージ一覧に「gtk2-devel」が表示されるので，右下にある「インストール (I)」ボタンをクリックしてください（図 A.7(a)）．
4. 同様に入力ボックスに “gcc” と入力して，パッケージ一覧に表示された「gcc」を選択し，「インストール (I)」ボタンをクリックしてください（図 A.7(b)）．
5. 「適用 (p)」ボタンをクリックください．インストールが開始し，実行経過が表示されます．

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 – OpenSUSE 11.1 編 –

OpenSUSE 11.1，では zypper というコマンドを利用してパッケージの追加や削除を行います．gtk2-devel と gcc を，zypper コマンドを利用してインストールするには次のようにします．

図 A.5　コントロール・センターからのソフトウェアマネージャの起動

図 A.6　パスワードの入力

(a)　(b)

図 A.7　パッケージの選択

```
$ su ↵ パスワードの入力
$ zypper install gtk2-devel gcc ↵
```

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くるので，“y” を入力して処理を続けます．

```
Loaded plugins: refresh-packagekit
リポジトリのデータを読み込んでいます...
インストール済みのパッケージを読み込んでいます...
パッケージの依存関係を解決しています...
...
全ダウンロードサイズ: 20.0 M この操作を行なうには、追加で 89.3 M の容量が必要です。
```

```
続行しますか？　[はい (Y)/いいえ (n)]: y
```

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

```
$ su ↵ パスワードの入力
$ zypper install anjuta devhelp gtranslator ↵
```

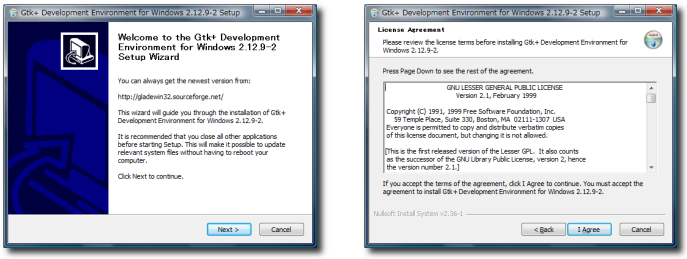
A.3　GTK+ 開発環境のインストール – Windows Vista 編 –

Windows 上で動作する GTK+ のパッケージも何種類か公開されています．本書では，Glade for Win32[*1] で公開されているパッケージのインストール方法を紹介します．

ただし，Windows 版のパッケージには GNOME ライブラリは含まれていないため，GNOME ライブラリを使用する第 9 章の内容を Windows 上で確認することはできません．あらかじめご了承ください．

以下の手順で GTK+ の開発環境をインストールします．

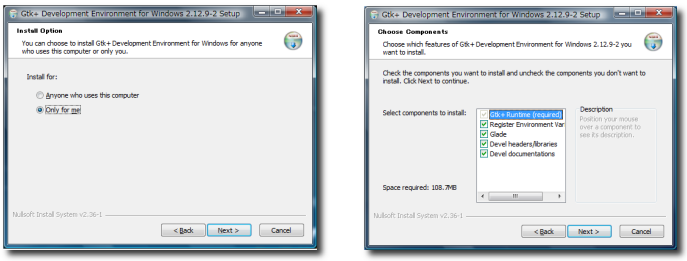
1. 上記の Web サイトから GTK+ の「Development Environment」(本書執筆時点では http://downloads.sourceforge.net/gladewin32/gtk-dev-2.12.9-win32-2.exe) をダウンロードします．
2. ダウンロードしたファイルをダブルクリックして，インストーラを起動します．図 A.8(a) のウィンドウが表示されるので，「Next>」ボタンをクリックして次に進みます．
3. 次に，ライセンスに同意するか聞かれるので，ライセンス文に同意する場合は「I Agree」ボタンをクリックしてください（図 A.8(b)）．



(a)　(b)

図 A.8　Window 版パッケージのインストール（1）

4. インストールする環境を現在のユーザだけで使用するのか，すべてのユーザで使用するのかご自分の使用する環境に合わせてどちらかを選択して，「Next>」ボタンをクリックしてください．ここでは現在のユーザでのみ使用するという項目を選択することにします（図 A.9(a)）．
5. 次にインストールするコンポーネントが表示されるので，得に問題がなければそのまま「Next>」ボタンをクリックしてください（図 A.9(b)）．

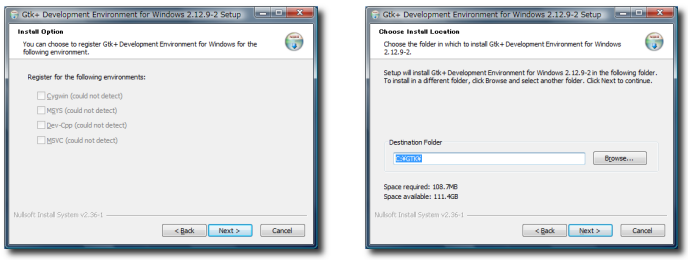


(a)　(b)

図 A.9　Window 版パッケージのインストール（2）

[*1] http://gladewin32.sourceforge.net/

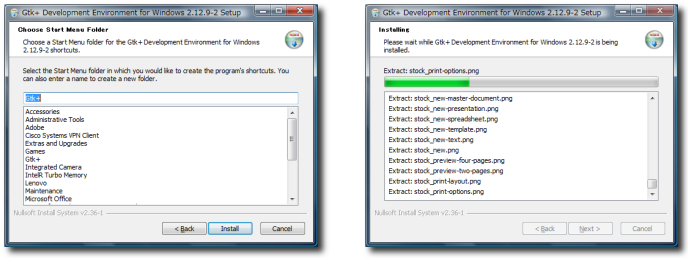
6. 使用している Windows にインストールされている開発環境を自動検出して，開発環境ごとに環境設定を行うかどうかたずねてきます．Visual Studio 2008 Express 版は検出されないようなので，図 A.10(a) のように表示されます．そのまま，「Next>」ボタンをクリックしてください．
7. 次にインストール先をたずねてくるので，ここでは標準で指定される C:¥GTK にインストールすることにして，「Next>」ボタンをクリックしてください（図 A.10(b)）．



(a)　(b)

図 A.10　Window 版パッケージのインストール（3）

8. メニューの追加先をたずねてくるので，そのまま「Install」ボタンをクリックしてください（図 A.11(a)）．インストールが開始され，図 A.11(b) のウィンドウが表示されます．あとはインストールが無事終了するのを待ちましょう．



(a)　(b)

図 A.11　Window 版パッケージのインストール（4）

付録 B

Visual Studio 2008 Express Edition でのビルド方法

Visual Studio 2008 Express Edition で GTK+ のプログラムをビルドする方法を紹介します．

付録 A で紹介した手順で GTK+ の開発環境をインストールすると，自動的に開発用の環境変数（INCLUDE，LIB）が設定されるようですが，Visual Studio の中では有効ではないようです．そのため，プロジェクトごとに設定する必要があります．ここでは，第 2 章（p. 7）で示したプログラムを例にして，プロジェクトの作成方法と設定の方法を説明します．

1. スタートメニューなどから Microsoft Visual C++ 2008 Express Edition を起動します．
2. メニューバーから「ファイル (F)」→「新規作成 (N)」→「プロジェクト (P)...」を選択します．図 B.1 のようなウィンドウが表示されるので，プロジェクタ名に “image-viewer” と入力して，「OK」ボタンをクリックしてください．

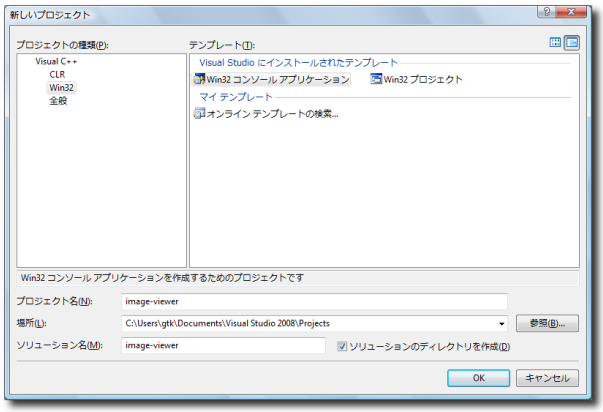
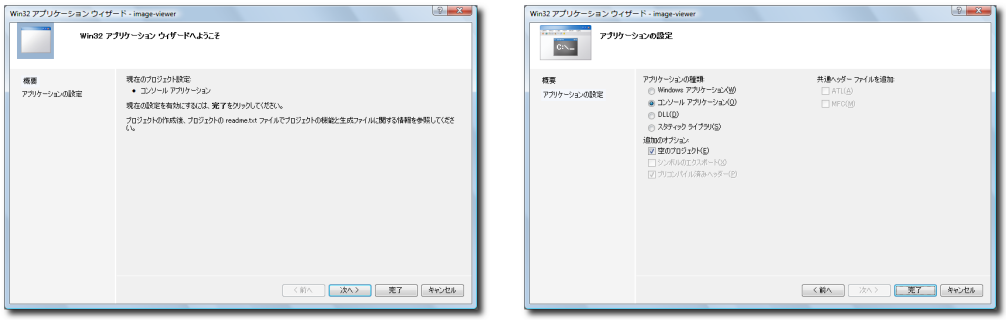


図 B.1　新規プロジェクトの作成

3. 次に表示されたウィンドウの「次へ >」ボタンをクリックしてください（図 B.2(a)）．
4. 「アプリケーションの設定」画面で，「空のプロジェクト (E)」にチェックを入れて，「完了」ボタンをクリックしてください（図 B.2(b)）．
5. 新規プロジェクトが作成できたら，次にソース 2–1 を入力するために，新規ファイルを追加します．図 B.3 の画面左側にあるソリューションエクスプローラの「ソースファイル」でマウスの右ボタンをクリックして， メニューを表示し，「追加 (D)」→「新しい項目 (W)...」を選択します．
6. 「新しい項目の追加」画面で「C++ ファイル (.cpp)」を選択して，ファイル名に “image-viewer.c” と入力し，「追加 (A)」ボタンをクリックしてください．
7. 新規ファイルを作成したら，ソース 2–1 を入力してください．入力が終了したら，メニューバーから「プロジェクト (P)」→「image-viewer のプロパティ (P)...」を選択してください（図 B.5）．



(a)　(b)

図 B.2　アプリケーションウィザード

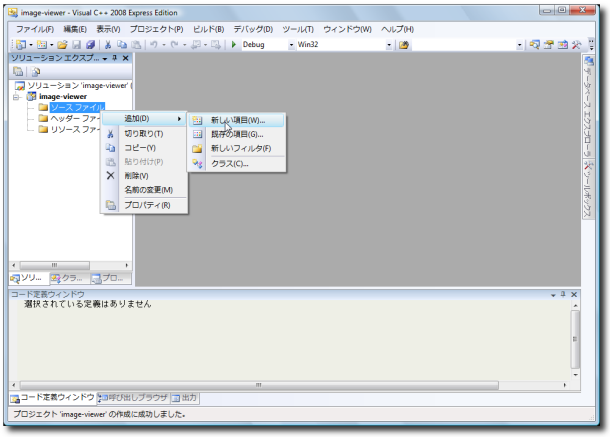


図 B.3　新規ファイルの追加 (1)

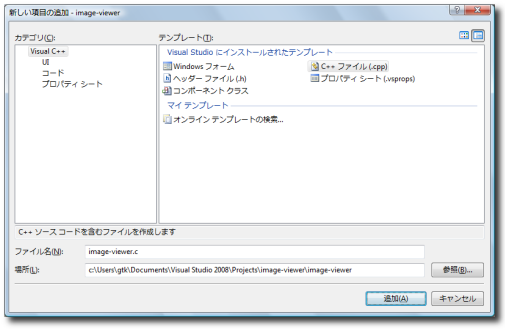


図 B.4　新規ファイルの追加 (2)

8.「image-viewer プロパティページ」というウィンドウが表示されるので，まず，「構成プロパティ」→「C/C++」→「全般」を選択してください．そして，中央の画面の「追加のインクルードディレクトリ」に GTK+ のヘッダファイルがインストールされているディレクトリを入力します（図 B.6(a)）．
最低限指定しなければいけないディレクトリは以下の通りです．

- C:¥GTK¥INCLUDE
- C:¥GTK¥INCLUDE¥GTK-2.0
- C:¥GTK¥INCLUDE¥GLIB-2.0
- C:¥GTK¥INCLUDE¥PANGO-1.0
- C:¥GTK¥INCLUDE¥CAIRO
- C:¥GTK¥INCLUDE¥ATK-1.0
- C:¥GTK¥LIB¥GTK-2.0¥INCLUDE
- C:¥GTK¥LIB¥GLIB-2.0¥INCLUDE

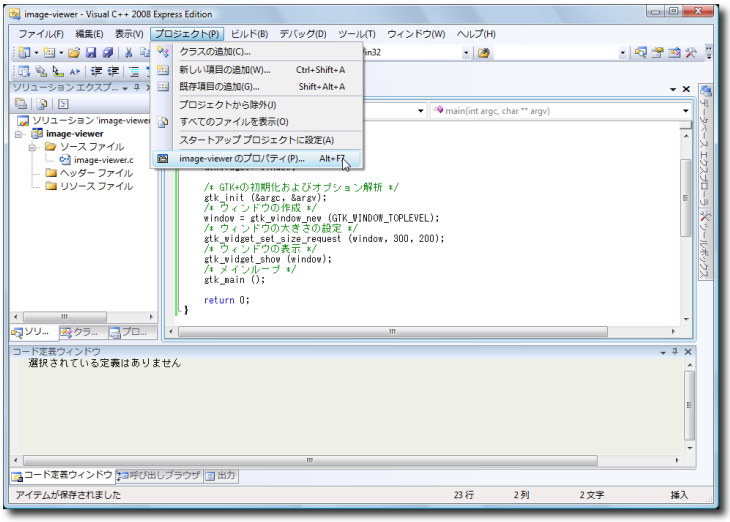
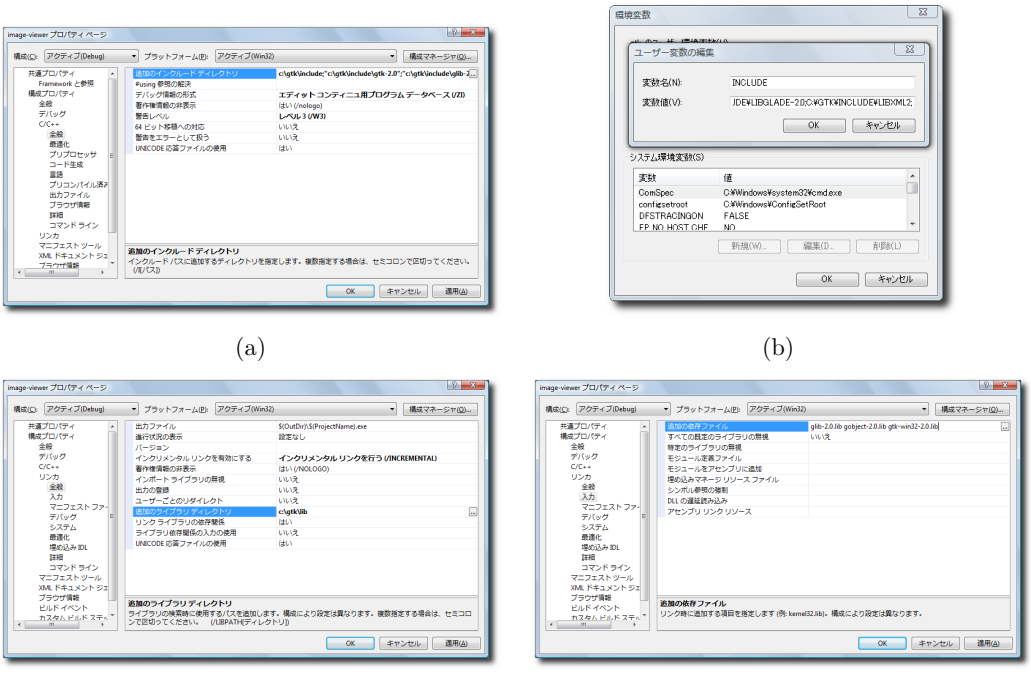


図 B.5　ビルドのための設定 (1)

これをいちいち入力するのが面倒な場合は，コントロールパネルのユーザーアカウントの欄から「環境変数の変更」をクリックして環境変数を表示し，環境変数 “INCLUDE” の内容をコピーして，貼り付けると簡単です（図 B.6(b)）．

9. 次に，「構成プロパティ」→「リンカ」→「全般」を選択してください．そして，中央の画面の「追加のライブラリディレクトリ」に “c:¥gtk¥lib” と入力してください（図 B.6(c)）．
10. 次に，「構成プロパティ」→「リンカ」→「入力」を選択してください．そして，中央の画面の「追加の依存ファイル」に以下に示すライブラリを入力してください（図 B.6(d)）．
    - glib-2.0.lib
    - gobject-2.0.lib
    - gtk-win32-2.0.lib



(a)　(b)

(c)　(d)

図 B.6　ビルドのための設定 (2)

11. 以上で設定が完了したので，メニューバーから「ビルド (B)」→「プロジェクトのビルド (B)」を選択してください．ビ

ルドが正常終了したら，メニューバーから「デバッグ (D)」→「デバッグ開始 (S)」もしくは「デバッグなしで開始 (H)」を選択してください．図 B.7 のようなウィンドウが表示されれば OK です．

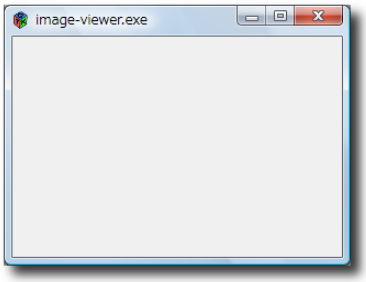


図 B.7　Windows 版 image-viewer

付録 C

GtkStockItem 一覧

GTK+ では，あらかじめ GtkStockItem と呼ばれるアイコンが定義されています．図 C.1 は gtk-demo というデモプログラムを使って表示させた GtkStockItem の例です（ただし，アイコンは GTK+ のテーマによって異なります）．

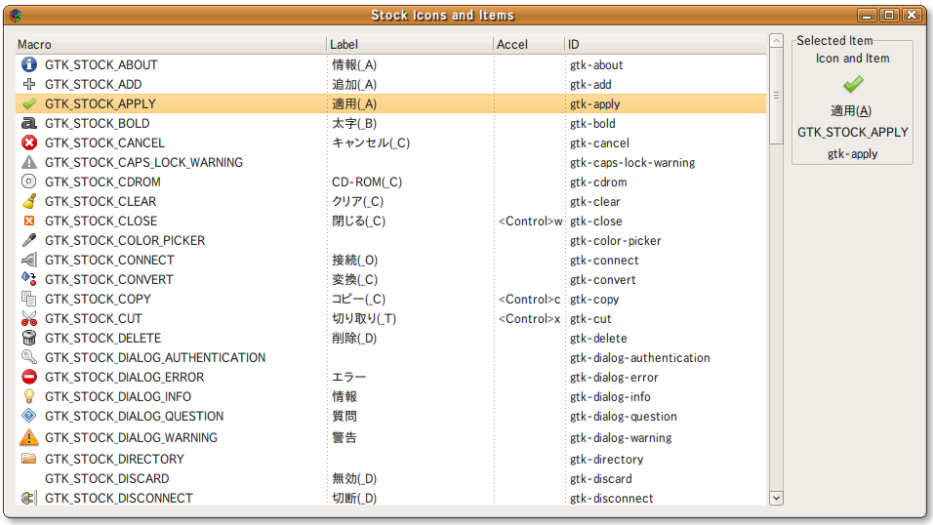


図 C.1　GtkStockItem の例

ソース A–1 に標準で定義されている GtkStockItem の一覧を示します．

ソース A-1 GtkStockItem の一覧

```
1  #define     GTK_STOCK_ABOUT
2  #define     GTK_STOCK_ADD
3  #define     GTK_STOCK_APPLY
4  #define     GTK_STOCK_BOLD
5  #define     GTK_STOCK_CANCEL
6  #define     GTK_STOCK_CDROM
7  #define     GTK_STOCK_CLEAR
8  #define     GTK_STOCK_CLOSE
9  #define     GTK_STOCK_COLOR_PICKER
10 #define     GTK_STOCK_CONVERT
11 #define     GTK_STOCK_CONNECT
12 #define     GTK_STOCK_COPY
13 #define     GTK_STOCK_CUT
14 #define     GTK_STOCK_DELETE
```

```
15  #define     GTK_STOCK_DIALOG_AUTHENTICATION
16  #define     GTK_STOCK_DIALOG_ERROR
17  #define     GTK_STOCK_DIALOG_INFO
18  #define     GTK_STOCK_DIALOG_QUESTION
19  #define     GTK_STOCK_DIALOG_WARNING
20  #define     GTK_STOCK_DIRECTORY
21  #define     GTK_STOCK_DISCONNECT
22  #define     GTK_STOCK_DND
23  #define     GTK_STOCK_DND_MULTIPLE
24  #define     GTK_STOCK_EDIT
25  #define     GTK_STOCK_EXECUTE
26  #define     GTK_STOCK_FILE
27  #define     GTK_STOCK_FIND
28  #define     GTK_STOCK_FIND_AND_REPLACE
29  #define     GTK_STOCK_FLOPPY
30  #define     GTK_STOCK_FULLSCREEN
31  #define     GTK_STOCK_GOTO_BOTTOM
32  #define     GTK_STOCK_GOTO_FIRST
33  #define     GTK_STOCK_GOTO_LAST
34  #define     GTK_STOCK_GOTO_TOP
35  #define     GTK_STOCK_GO_BACK
36  #define     GTK_STOCK_GO_DOWN
37  #define     GTK_STOCK_GO_FORWARD
38  #define     GTK_STOCK_GO_UP
39  #define     GTK_STOCK_HARDDISK
40  #define     GTK_STOCK_HELP
41  #define     GTK_STOCK_HOME
42  #define     GTK_STOCK_INDENT
43  #define     GTK_STOCK_INDEX
44  #define     GTK_STOCK_INFO
45  #define     GTK_STOCK_ITALIC
46  #define     GTK_STOCK_JUMP_TO
47  #define     GTK_STOCK_JUSTIFY_CENTER
48  #define     GTK_STOCK_JUSTIFY_FILL
49  #define     GTK_STOCK_JUSTIFY_LEFT
50  #define     GTK_STOCK_JUSTIFY_RIGHT
51  #define     GTK_STOCK_LEAVE_FULLSCREEN
52  #define     GTK_STOCK_MEDIA_FORWARD
53  #define     GTK_STOCK_MEDIA_NEXT
54  #define     GTK_STOCK_MEDIA_PAUSE
55  #define     GTK_STOCK_MEDIA_PLAY
56  #define     GTK_STOCK_MEDIA_PREVIOUS
57  #define     GTK_STOCK_MEDIA_RECORD
58  #define     GTK_STOCK_MEDIA_REWIND
59  #define     GTK_STOCK_MEDIA_STOP
60  #define     GTK_STOCK_MISSING_IMAGE
61  #define     GTK_STOCK_NETWORK
62  #define     GTK_STOCK_NEW
63  #define     GTK_STOCK_NO
64  #define     GTK_STOCK_OK
65  #define     GTK_STOCK_OPEN
66  #define     GTK_STOCK_PASTE
67  #define     GTK_STOCK_PREFERENCES
68  #define     GTK_STOCK_PRINT
69  #define     GTK_STOCK_PRINT_PREVIEW
70  #define     GTK_STOCK_PROPERTIES
71  #define     GTK_STOCK_QUIT
72  #define     GTK_STOCK_REDO
73  #define     GTK_STOCK_REFRESH
74  #define     GTK_STOCK_REMOVE
75  #define     GTK_STOCK_REVERT_TO_SAVED
76  #define     GTK_STOCK_SAVE
77  #define     GTK_STOCK_SAVE_AS
78  #define     GTK_STOCK_SELECT_COLOR
79  #define     GTK_STOCK_SELECT_FONT
80  #define     GTK_STOCK_SORT_ASCENDING
81  #define     GTK_STOCK_SORT_DESCENDING
82  #define     GTK_STOCK_SPELL_CHECK
83  #define     GTK_STOCK_STOP
84  #define     GTK_STOCK_STRIKETHROUGH
85  #define     GTK_STOCK_UNDELETE
86  #define     GTK_STOCK_UNDERLINE
87  #define     GTK_STOCK_UNDO
88  #define     GTK_STOCK_UNINDENT
89  #define     GTK_STOCK_YES
90  #define     GTK_STOCK_ZOOM_100
91  #define     GTK_STOCK_ZOOM_FIT
```

```
92  #define     GTK_STOCK_ZOOM_IN
93  #define     GTK_STOCK_ZOOM_OUT
```

付録 D

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://www.iim.ics.tut.ac.jp/~sugaya/books/GUI-ApplicationProgramming/
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
         +--- display1/
         +--- display2/
         +--- display3
         +--- image_application/
         +--- read/
  +--- glib
         +--- file/
         +--- ghashtable/
         +--- glist/
         +--- timer/
  +--- gtk
         +--- hello_world/
         +--- packing/
         +--- signal/
         +--- table/
  +--- 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/
```

付録 E

Gdk による図形の描画

E.1 Gdk の概要

この章では, Gdk ライブラリを使った図形の描画について解説します. Gdk は, GTK+ のグラフィックス関係を担当するライブラリです. Gdk ライブラリで重要な概念は, ドローアブル (GdkDrawable) とグラフィックコンテキスト (GdkGC) です. ドローアブルとは, 絵を描画するキャンバスみたいなものです. ドローアブルと対になるのがグラフィックコンテキストで, これは筆やペンのようなものを指します.

GTK+ で使用される代表的なドローアブルは GdkWindow や GdkPixmap, GdkBitmap です. ウィンドウ上に描画した図形を表示するためには, GdkWindow に描画を行う必要がありますが, 本書では主に GtkDrawingArea ウィジェットのメンバ window を使用します.

グラフィックコンテキストは描画する線の太さや色などの設定を行います. これらの設定方法については, 次の節で紹介します. ドローアブルとグラフィックコンテキストの概念さえおさえておけば, 線や矩形, 円弧を描画する関数はライブラリが提供してくれますので, 簡単に図形を描画することが可能です. これらの関数についても後の節で紹介します.

E.2 グラフィックコンテキスト

E.2.1 グラフィックコンテキストの生成

グラフィックコンテキスト (以下 GC) の生成には関数 gdk_gc_new を用います.

```
GdkGC* gdk_gc_new (GdkWindow *drawable);
```

関数の引数には GdkWindow や GdkPixmap 等のドローアブルを与えます. 反対に生成した GC を解放するには, 関数 g_object_unref を使います. GC を含めて GObject から派生したオブジェクトは, リファレンスカウンタという概念を持ちます. これらのオブジェクトは生成時にリファレンスカウンタの値が 1 に設定され, 関数 g_object_ref を呼び出すごとにリファレンスカウンタの値が 1 つ増えます. 反対に関数 g_object_unref を呼び出すごとにリファレンスカウンタの値が 1 つ減ります. リファレンスカウンタの値が 0 になったとき, そのオブジェクトは解放されます.

```
gpointer g_object_ref   (gpointer object);
void     g_object_unref (gpointer object);
```

E.2.2 色の設定

点や線などを描画する際の色の設定は, GdkColor 構造体で行い, GC に割り当てます. GdkColor 構造体は次のように定義されています.

```
struct GdkColor {
  guint32 pixel;
```

```
  guint16 red;
  guint16 green;
  guint16 blue;
};
```

GdkColor 構造体の red, green, blue メンバに 0 から 65535 までの 16 ビットの範囲で値を指定します．pixel メンバの値は，関数 gdk_color_alloc を呼び出すと自動的に設定されます．関数 gdk_colormap_get_system を使うとデフォルトのカラーマップを取得することができます．

```
gint gdk_color_alloc (GdkColormap *colormap, GdkColor *color);
GdkColormap* gdk_colormap_get_system (void);
```

GdkColor 構造体の変数に対して色の設定をした後，関数 gdk_gc_set_foreground と関数 gdk_gc_set_background によって GC に色を割り当てます．関数 gdk_gc_set_foreground は，点や線などを描画する前景色を設定します．関数 gdk_gc_set_background は背景色を設定します．

```
void gdk_gc_set_foreground (GdkGC *gc, const GdkColor *color);
void gdk_gc_set_background (GdkGC *gc, const GdkColor *color);
```

色の設定の例をソース E–1 に示します．

ソース E-1 色の設定

```
1  GdkGC     *gc;
2  GdkColor color;
3
4  gc = gdk_gc_new (window);
5
6  color.red   = 0x0000;
7  color.green = 0x0000;
8  color.blue  = 0xffff;
9
10 gdk_color_alloc (gdk_colormap_get_system (), &color);
11 gdk_gc_set_foreground (gc, &color);
```

E.3　図形の描画

E.3.1　点の描画

点の描画には関数 gdk_draw_point と関数 gdk_draw_points の 2 つの関数が用意されています．関数 gdk_draw_point は指定した座標に点を描画する関数です．複数の点を一度に描画したい場合は，関数 gdk_draw_points を用います．このとき，GdkPoint 構造体の配列で複数の点を指定します．

```
void gdk_draw_point  (GdkDrawable *drawable,
                      GdkGC       *gc,
                      gint        x,
                      gint        y);

void gdk_draw_points (GdkDrawable *drawable,
                      GdkGC       *gc,
                      GdkPoint    *points,
                      gint        npoints);
```

GdkPoint 構造体は，次のように定義されています．

```
struct GdkPoint {
  gint x;
  gint y;
};
```

E.3.2　線分の描画

線分の描画関数

線分を描画する関数には次の 3 種類があります.

- gdk_draw_line
  指定した 2 点 $(x1, y1), (x2, y2)$ を結んだ線分を描画する関数です.

```
void gdk_draw_line (GdkDrawable *drawable,
                    GdkGC       *gc,
                    gint         x1,
                    gint         y1,
                    gint         x2,
                    gint         y2);
```

- gdk_draw_lines
  GdkPoint 構造体の配列で指定した点を順番に結んだ複数の線分を描画する関数です.

```
void gdk_draw_lines (GdkDrawable *drawable,
                     GdkGC       *gc,
                     GdkPoint    *points,
                     gint         npoints);
```

- gdk_draw_segments
  GdkSegment 構造体の配列で指定した複数の線分を描画する関数です.

```
void gdk_draw_segments (GdkDrawable *drawable,
                        GdkGC       *gc,
                        GdkSegment  *segs,
                        gint         nsegs);
```

GdkSegment 構造体は, 次のように定義されています.

```
struct GdkSegment {
  gint x1;
  gint y1;
  gint x2;
  gint y2;
};
```

関数 gdk_draw_lines が折れ線のような連続した線分を描画するのに対して, 関数 gdk_draw_segments は図 E.1 のような複数の個別の線分 (セグメント) を描画する関数です.

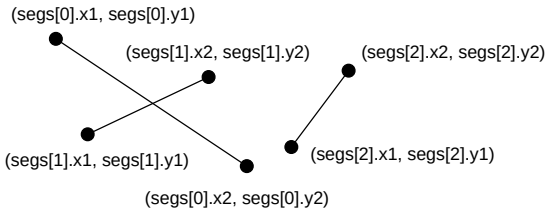


図 E.1　複数セグメントの描画

線分の属性

線分を描画する際には, 線分に対して次の属性を設定することができます.

- 線分の種類
  線分の種類には, 表 E.1 に示した 3 種類があります.　点線を描画する場合には, 関数 gdk_gc_set_dashes で点線の間隔を設定することができます.

```
void gdk_gc_set_dashes (GdkGC *gc,
```

```
gint   dash_offset,
gint8  dash_list[],
gint   n);
```

dash_offset は前景色を何ピクセル目から描画するかを指定します．また，dash_list には点線のパターンを配列で指定します．例えば，次のような配列を与えると，前景色 4 ピクセル・空白 (GDK_LINE_DOUBLE_DASH なら背景色)2 ピクセル・前景色 2 ピクセル・空白 2 ピクセルというパターンを繰り返し描画します．n には配列の要素数を与えます．

```
gint8 dash_list[] = {4, 2, 2, 2};
```

- 線端の種類
  線端の種類には，表 E.2 に示した 4 種類があります．描画される線端は図 E.2 のようになります．

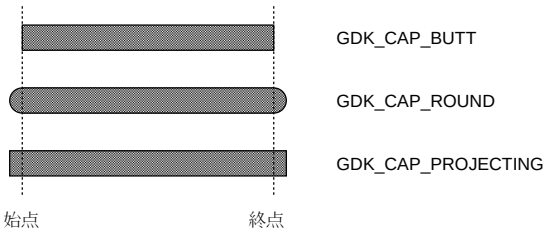


図 E.2　線端の描画

- 接続の種類
  線分の接続部分の形状には，表 E.3 に示した 3 種類があります．描画される接続部分の形状は図 E.3 のようになります．

線分の描画例

色の設定や線分の描画で説明した内容をまとめて，ソース E–2 に示します．

線分の描画に関する部分は先に詳しく解説してありますのでソースコードの説明は省略します．この例では，図形描画用のドローアブルとして，GtkDrawingArea ウィジェットの window メンバを使用しています．115 行目で設定しているコールバック関数は，ウィジェットが画面の前面に表示されたときなど画面の再描画が必要なときに発生するシグナル”expose_event” シグナルに対する関数です．

ユーザがウィンドウなどのドローアブル上に図形を描画する場合，図形を描画しているウィンドウが他のウィンドウに隠れた領域はユーザが再び図形の描画を行わない限り自動的に描画されることはありません．このように再描画が必要になったときに発生するシグナルが ”expose_event” シグナルです．従って，このシグナルに対するコールバック関数に図形を描画するコードを書くのが一般的です．

ソース E-2 線分の描画： draw_lines.c

```
1 #include <gtk/gtk.h>
2
```

表 E.1　線の種類

スタイル	説明
GDK_LINE_SOLID	実線
GDK_LINE_ON_OFF_DASH	点線
GDK_LINE_DOUBLE_DASH	点線 (前景色と背景色を交互に描画)

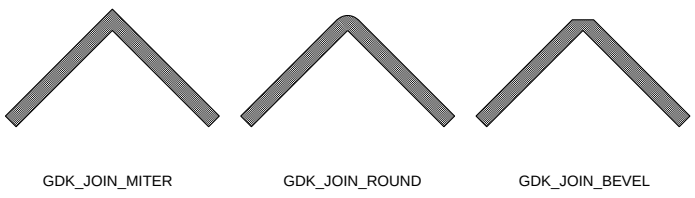


図 E.3　接続部分の描画

スタイル	説明
GDK_CAP_NOT_LAST	線端は GDK_CAP_BUTT と同様ですが, 線幅が 0 のとき, 最後の点を描画しません.
GDK_CAP_BUTT	始点と終点をそのまま描画します.
GDK_CAP_ROUND	線端を丸く描画します.
GDK_CAP_PROJECTING	線端を線幅の半分だけはみだして描画します.

表 E.3　接続の種類

スタイル	説明
GDK_JOIN_MITER	接続部分をとがらせます.
GDK_JOIN_ROUND	接続部分を丸くします.
GDK_JOIN_BEVEL	接続部分のとがった部分をカットしたような形にします.

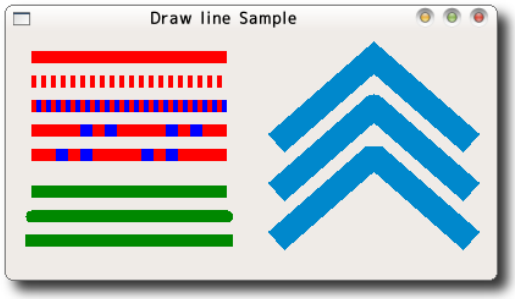


図 E.4　線分描画の例

```
3  static void set_color (GdkGC    *gc,
4                         GdkColor *color,
5                         guint16  red,
6                         guint16  green,
7                         guint16  blue,
8                         void (*set_function) (GdkGC           *gc,
9                                               const GdkColor *color)) {
10   color->red   = red;
11   color->green = green;
12   color->blue  = blue;
13   gdk_color_alloc (gdk_colormap_get_system (), color);
14   set_function (gc, color);
15 }
16
17 gboolean cb_expose_event (GtkWidget       *widget,
18                           GdkEventExpose  *event,
19                           gpointer        user_data) {
20   GdkWindow *drawable = widget->window;
21   GdkGC     *gc;
22   GdkColor  color;
23   GdkPoint  point[3];
24   gint      line_width = 10;
25   gint8     dash_style[] = {40, 10, 10, 10};
26
27   gc = gdk_gc_new (drawable);
28
29   /* 線種の違い */
30   set_color (gc, &color, 0xffff, 0x0000, 0x0000, gdk_gc_set_foreground);
31   gdk_gc_set_line_attributes (gc, line_width,
32                               GDK_LINE_SOLID, GDK_CAP_BUTT,GDK_JOIN_MITER);
33   gdk_draw_line (drawable, gc, 20, 20, 180, 20);
34
35   gdk_gc_set_line_attributes (gc, line_width,
36                               GDK_LINE_ON_OFF_DASH,
37                               GDK_CAP_BUTT, GDK_JOIN_MITER);
38   gdk_draw_line (drawable, gc, 20, 40, 180, 40);
39
40   set_color (gc, &color, 0x0000, 0x0000, 0xffff, gdk_gc_set_background);
```

```
41   gdk_gc_set_line_attributes (gc, line_width,
42                               GDK_LINE_DOUBLE_DASH,
43                               GDK_CAP_BUTT, GDK_JOIN_MITER);
44   gdk_draw_line (drawable, gc, 20, 60, 180, 60);
45
46   /* 点線の設定 */
47   gdk_gc_set_dashes (gc, 0, dash_style, 4);
48   gdk_draw_line (drawable, gc, 20, 80, 180, 80);
49
50   gdk_gc_set_dashes (gc, 20, dash_style, 4);
51   gdk_draw_line (drawable, gc, 20, 100, 180, 100);
52
53   /* 線端の種類 */
54   set_color (gc, &color, 0x0000, 0x8888, 0x0000, gdk_gc_set_foreground);
55
56   gdk_gc_set_line_attributes(gc, line_width,
57                              GDK_LINE_SOLID, GDK_CAP_BUTT, GDK_JOIN_MITER);
58   gdk_draw_line (drawable, gc, 20, 130, 180, 130);
59
60   gdk_gc_set_line_attributes(gc, line_width,
61                              GDK_LINE_SOLID,
62                              GDK_CAP_ROUND, GDK_JOIN_MITER);
63   gdk_draw_line (drawable, gc, 20, 150, 180, 150);
64
65   gdk_gc_set_line_attributes (gc, line_width,
66                               GDK_LINE_SOLID,
67                               GDK_CAP_PROJECTING, GDK_JOIN_MITER);
68   gdk_draw_line (drawable, gc, 20, 170, 180, 170);
69
70   /* 接続の種類 */
71   set_color (gc, &color, 0x0000, 0x8888, 0xcccc, gdk_gc_set_foreground);
72   line_width = 20;
73
74   gdk_gc_set_line_attributes (gc, line_width,
75                               GDK_LINE_SOLID,
76                               GDK_CAP_BUTT, GDK_JOIN_MITER);
77   point[0].x =  220; point[0].y = 90;
78   point[1].x =  300; point[1].y = 20;
79   point[2].x =  380; point[2].y = 90;
80   gdk_draw_lines (drawable, gc, point, 3);
81
82   gdk_gc_set_line_attributes (gc, line_width,
83                               GDK_LINE_SOLID,
84                               GDK_CAP_BUTT, GDK_JOIN_ROUND);
85   point[0].x =  220; point[0].y = 130;
86   point[1].x =  300; point[1].y = 60;
87   point[2].x =  380; point[2].y = 130;
88   gdk_draw_lines (drawable, gc, point, 3);
89
90   gdk_gc_set_line_attributes (gc, line_width,
91                               GDK_LINE_SOLID,
92                               GDK_CAP_BUTT, GDK_JOIN_BEVEL);
93   point[0].x =  220; point[0].y = 170;
94   point[1].x =  300; point[1].y = 100;
95   point[2].x =  380; point[2].y = 170;
96   gdk_draw_lines (drawable, gc, point, 3);
97
98   g_object_unref (gc);
99
100  return FALSE;
101 }
102
103 int main (int argc, char *argv[]) {
104   GtkWidget *window;
105   GtkWidget *canvas;
106
107   gtk_init (&argc, &argv);
108
109   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
110   gtk_window_set_title (GTK_WINDOW(window), "Line␣Examples");
111   gtk_widget_set_size_request (window, 400, 200);
112
113   canvas = gtk_drawing_area_new ();
114   gtk_container_add (GTK_CONTAINER(window), canvas);
115   g_signal_connect (G_OBJECT(canvas), "expose_event",
116                     G_CALLBACK (cb_expose_event), NULL);
```

```
117
118   gtk_widget_show_all (window);
119   gtk_main ();
120
121   return 0;
122 }
```

E.3.3 矩形の描画

矩形の描画を行うには, 関数 gdk_draw_rectangle を使用します.

```
void gdk_draw_rectangle (GdkDrawable *drawable,
                         GdkGC       *gc,
                         gboolean    filled,
                         gint        x,
                         gint        y,
                         gint        width,
                         gint        height);
```

関数 gdk_draw_rectangle では図 E.5 に示すように, 矩形の左上の座標 x, y と幅 width, 高さ height を指定します. 引数 filled を TRUE にすると矩形内を前景色で塗りつぶします.

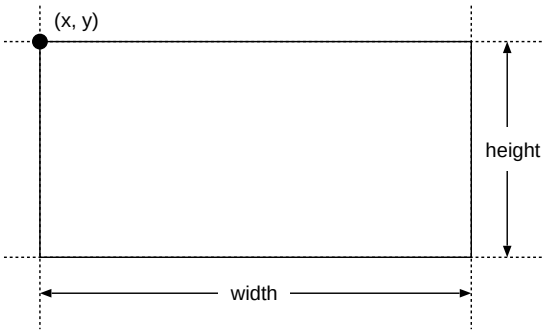


図 E.5 矩形の描画

E.3.4 円弧の描画

円弧の描画を行うには, 関数 gdk_draw_arc を使用します.

```
void gdk_draw_arc (GdkDrawable *drawable,
                   GdkGC       *gc,
                   gboolean    filled,
                   gint        x,
                   gint        y,
                   gint        width,
                   gint        height,
                   gint        angle1,
                   gint        angle2);
```

引数 x, y は円弧を含む矩形の左上の座標を表します. また, width, height はその矩形の幅と高さを表します. angle1 は, 円弧の開始角を表します. 水平方向右方向 (3 時の方向) を 0° として, 1/64° 単位で与えます. angle2 は angle1 からの相対的な角度を与えます (図 E.6). 引数 filled を TRUE にすると, 円弧内を前景色で塗りつぶします.

E.3.5 多角形の描画

多角形の描画を行うには, 関数 gdk_draw_polygon を使用します.

```
void gdk_draw_polygon (GdkDrawable *drawable,
                       GdkGC       *gc,
```

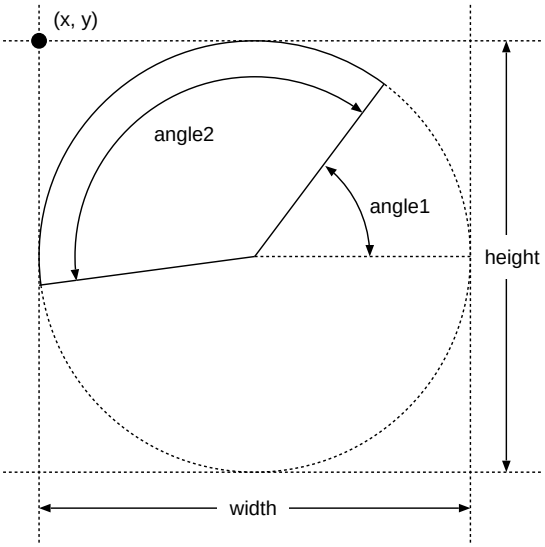


図 E.6　円弧の描画

```
gboolean     filled,
GdkPoint     *points,
gint         npoints);
```

図 E.7 に示すように多角形の頂点を GdkPoint 構造体の配列で指定します．引数 filled を TRUE にすると，多角形内を前景色で塗りつぶします．

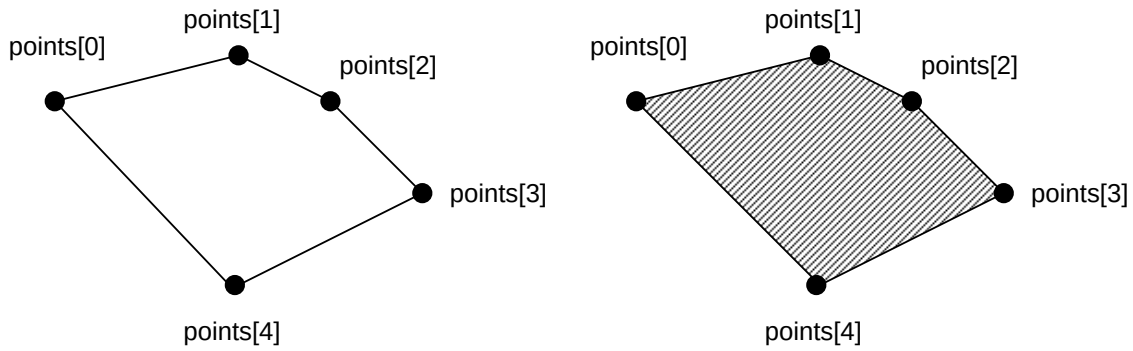
E.4　ピックスマップへの描画

GdkPixmap もドローアブルの一つです．図形などをウィンドウに描画するには GdkWindow を使えばいいことは先ほどまでの例で理解できたと思います．では GdkPixmap はどういうときに使用するのでしょうか．ウィンドウにたくさんの図形を描画する場合，直接 GdkWindow に描画すると画面がちらつくことがあります．このときに図形を先に GdkPixmap に描画しておき，それを GdkWindow に張り付けることで画面のちらつきを回避することができます．

ソース E–3 に図形を描画したピックスマップをウィンドウに張り付ける例を示します．

ピックスマップの生成 (5–14 **行目**)

この関数はウィンドウの ”configure_event” シグナルに対するコールバック関数です．”configure_event” はウィジェットが生成されたときに一度だけ発生するシグナルです．この関数が呼び出されたときにピックスマップを作成して，白で全体を塗りつぶしています．



(a) パラメータ filled が FALSE の場合　　(b) パラメータ filled が TRUE の場合

図 E.7　多角形の描画

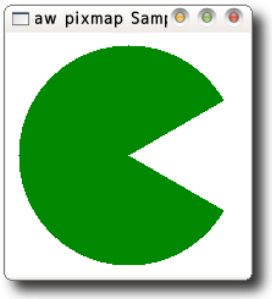


図 E.8　ピックスマップの描画

図形の描画 (16–37 **行目**)

この関数内で, ピックスマップに図形を描画して, そのピックスマップをウィンドウに張り付けています. ピックスマップの張り付けには, 関数 gdk_window_set_back_pixmap を使用しています. この関数はピックスマップをウィンドウの背景として設定する関数です. 関数 gdk_window_clear を呼び出すことで, 設定した背景でウィンドウを描画することができます.

```
void gdk_window_set_back_pixmap (GdkWindow *window,
                                 GdkPixmap *pixmap,
                                 gboolean  parent_relative);
void gdk_window_clear (GdkWindow *window);
```

ここでは使用しませんでしたが, ピックスマップの描画には関数 gdk_draw_drawable を使うこともできます.

```
void gdk_draw_drawable (GdkDrawable *drawable,
                        GdkGC       *gc,
                        GdkDrawable *src,
                        gint        xsrc,
                        gint        ysrc,
                        gint        xdest,
                        gint        ydest,
                        gint        width,
                        gint        height);
```

ソース E-4 ピックスマップの描画：draw_pixmap.c

```
1  #include <gtk/gtk.h>
2
3  static GdkPixmap *pixmap = NULL;
4
5  gboolean cb_configure_event (GtkWidget *widget, GdkEventConfigure *event) {
6    pixmap = gdk_pixmap_new (widget->window,
7                             widget->allocation.width,
8                             widget->allocation.height,
9                             -1);
10   gdk_draw_rectangle (pixmap, widget->style->white_gc, TRUE, 0, 0,
11                       widget->allocation.width,
12                       widget->allocation.height);
13   return TRUE;
14 }
15
16 gboolean cb_expose_event (GtkWidget       *widget,
17                           GdkEventExpose  *event,
18                           gpointer        user_data) {
19   GdkWindow *drawable = widget->window;
20   GdkGC     *gc;
21   GdkColor  color;
22
23   gc = gdk_gc_new (pixmap);
24   color.red   = 0x0000;
25   color.green = 0x8888;
26   color.blue  = 0x0000;
27   gdk_color_alloc (gdk_colormap_get_system (), &color);
28   gdk_gc_set_foreground (gc, &color);
```

```
29
30    gdk_draw_arc (pixmap, gc, TRUE, 10, 10, 180, 180, 30 * 64, 300 * 64);
31    gdk_window_set_back_pixmap (drawable, pixmap, FALSE);
32
33    gdk_window_clear(drawable);
34    g_object_unref (gc);
35
36    return FALSE;
37  }
38
39  int main (int argc, char *argv[]) {
40    GtkWidget *window;
41    GtkWidget *canvas;
42
43    gtk_init (&argc, &argv);
44
45    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
46    gtk_window_set_title (GTK_WINDOW(window), "Draw␣Pixmap");
47    gtk_widget_set_size_request (window, 200, 200);
48
49    canvas = gtk_drawing_area_new ();
50    gtk_container_add (GTK_CONTAINER(window), canvas);
51    g_signal_connect (G_OBJECT(canvas), "configure_event",
52                      G_CALLBACK (cb_configure_event), NULL);
53    g_signal_connect (G_OBJECT(canvas), "expose_event",
54                      G_CALLBACK (cb_expose_event), NULL);
55
56    gtk_widget_show_all (window);
57    gtk_main ();
58
59    return 0;
60  }
```

付録 F

GNOME プログラミング入門

GNOME というとデスクトップ環境を指すことが多いのですが, ここで扱うのはその GNOME デスクトップ環境を構築するために使用されている GUI ライブラリです. GNOME の GUI ライブラリで使用できるウィジェットは, GTK+ のウィジェットをベースとして開発された発展的なウィジェットですので, それらのウィジェットを使用することで便利なアプリケーションを簡単に作成することが可能になります. 本章では, GNOME ウィジェットを使用した簡単なアプリケーションの作成方法について解説します.

F.1 簡単な GNOME プログラムの作成

ここでは図 F.1 に示すようなアイコン一覧を表示するプログラムの作成を通して, GNOME アプリケーションのプログラミング方法を説明します. ソースコードはソース F–1 に示します.

GNOME アプリケーションといってもこれまでに学んだ GTK+ アプリケーションの作成方法とそれほど変わりませんので気楽に読み進めてください.

ソース F–1 簡単な GNOME アプリケーション：gnome-sample.c

```
1  #include <gnome.h>
2
3  int
4  main (int argc, char **argv)
5  {
6    GnomeProgram *app;
7    GtkWidget    *window;
8    GtkWidget    *iconlist;
9    int           n;
10
11   app = gnome_program_init ("gnome-sample", "1.0.0", LIBGNOMEUI_MODULE,
12                             argc, argv, NULL);
```

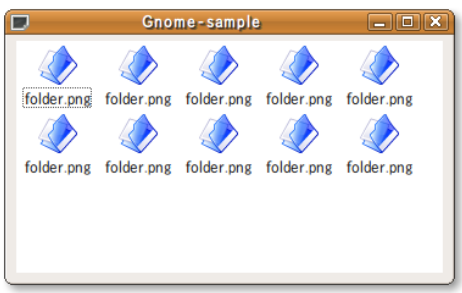


図 F.1 簡単な GNOME アプリケーション

```
13
14   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
15   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
16   gtk_window_set_title (GTK_WINDOW (window), "Gnome-sample");
17   gtk_widget_set_size_request (window, 360, 200);
18
19   iconlist = gnome_icon_list_new (64, NULL, 0);
20   gtk_container_add (GTK_CONTAINER (window), iconlist);
21
22   for (n = 0; n < 10; n++)
23     {
24       gnome_icon_list_append (GNOME_ICON_LIST(iconlist),
25                               "folder.png", "folder.png");
26     }
27   gtk_widget_show_all (window);
28   gtk_main ();
29   return 0;
30 }
```

ヘッダファイルのインクルード (1 **行目**)

GNOME ライブラリ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 gnome.h をインクルードしています. Ubuntu 9.04 では/usr/include/libgnomeui-2.0 というディレクトリ内に gnome.h がインストールされています.

GNOME **アプリケーションの初期化** (11-12 **行目**)

関数 gnome_program_init は作成したアプリケーションを GNOME アプリケーションとして動作させる場合に, 始めに呼び出す必要のある初期化関数です (この関数を使用せずに関数 gtk_init を呼び出しても問題ありません).

```
GnomeProgram*
gnome_program_init (const char            *app_id,
                    const char            *app_version,
                    const GnomeModuleInfo *module_info,
                    int                   argc,
                    char                  **argv,
                    const char            *first_property_name,
                    ...);
```

この関数の戻り値である GnomeProgram 型の変数は, 関数 gnome_program_get_app_version の引数に指定して, アプリケーションのバージョンを調べたりするときに使用します.

```
const char* gnome_program_get_app_version (GnomeProgram *program);
```

関数の第 3 引数に指定する GnomeModuleInfo 構造体は次のように定義されており, 各自で設定するのは大変そうですが, 通常はライブラリ側で定義されている LIBGNOMEUI_MODULE を指定すればいいでしょう.

```
struct GnomeModuleInfo {
    const char               *name;
    const char               *version;
    const char               *description;
    GnomeModuleRequirement   *requirements;
    GnomeModuleHook          instance_init;
    GnomeModuleHook          pre_args_parse, post_args_parse;
    struct poptOption        *options;
    GnomeModuleInitHook      init_pass;
    GnomeModuleClassInitHook class_init;
    const char               *opt_prefix;
    gpointer                 expansion1;
};
```

メインウィンドウの作成 (14 **行目**)

ここでアプリケーションのベースとなるメインウィンドウを作成しています. GNOME のウィジェットを使用してアプリケーションを作成する場合も GTK+ と同様にベースとなるのは GtkWindow です.

アイコンリストの作成 (19–26 行目)

アイコンリストウィジェット (GnomeIconList) はアイコンを並べて表示するウィジェットです[*1]. このウィジェットを使用するとファイルや GdkPixbuf データから簡単にアイコン一覧を作成することができます.

アイコンリストウィジェットを作成するには, 関数 gnome_icon_list_new を使用します.

```
GtkWidget* gnome_icon_list_new (guint         icon_width,
                                GtkAdjustment *adj,
                                int           flags);
```

作成したアイコンリストにアイコンを追加するには, 関数 gnome_icon_list_append を使用します. この関数を使用すると指定したファイルからアイコンデータを作成しリストに追加します. GdkPixbuf データからアイコンを追加するには, 関数 gnome_icon_list_append_pixbuf を使用します. 関数の戻り値には追加したアイコンの番号が返ります.

```
int gnome_icon_list_append (GnomeIconList *gil,
                            const char    *icon_filename,
                            const char    *text);
int gnome_icon_list_append_pixbuf (GnomeIconList *gil,
                                   GdkPixbuf     *im,
                                   const char    *icon_filename,
                                   const char    *text);
```

これらの関数の他に, 指定した位置にアイコンを挿入する関数として, ファイルからアイコンを挿入する関数 gnome_icon_list_insert と Pixbuf データからアイコンを挿入する関数 gnome_icon_list_insert_pixbuf があります.

```
void gnome_icon_list_insert (GnomeIconList *gil,
                             int           pos,
                             const char    *icon_filename,
                             const char    *text);
void gnome_icon_list_insert_pixbuf (GnomeIconList *gil,
                                    int           pos,
                                    GdkPixbuf     *im,
                                    const char    *icon_filename,
                                    const char    *text);
```

GNOME アプリケーションのコンパイル

GNOME ライブラリは多くのライブラリを使用して作成されているため, GNOME アプリケーションを作成するためには依存関係にある多くのライブラリをリンクする必要があります. pkg-config を使用すればこれらのオプションを簡単に与えることができます. GNOME ウィジェットを使用した場合には pkg-config の引数に libgnomeui-2.0 を指定します.

```
$ gcc gnome-sample.c -o gnome-sample `pkg-config libgnomeui-2.0 --cflags --libs` ↵
```

F.2 GNOME のウィジェット

ここでは GNOME ウィジェットの中から便利なウィジェットを 2 つ紹介します.

F.2.1 ファイルエントリ

ファイルエントリウィジェット (GnomeFileEntry)[*2]はエントリにファイル名を指定することを目的とした複合ウィジェットです. 参照ボタンをクリックするとファイル選択ダイアログが表示され, ダイアログから選択したファイル名がエントリに反映されます.

ウィジェットの作成

ファイルエントリを作成するには, 関数 gnome_file_entry_new を使用します.

[*1] このウィジェットは既に GTK+ の GtkIconView ウィジェットとして実装されていて, リファレンスマニュアルでは GnomeIconList を使用しないことを奨めています.

[*2] ドキュメントではこのウィジェットはもう使用しない方がいいとのことです.

```
GtkWidget*
gnome_file_entry_new (const char *history_id,
                      const char *browse_dialog_title);
```

ファイル名の取得

エントリ内のファイル名を取得するには, 関数 gnome_file_entry_get_full_path を使用します. 反対にファイル名をエントリにセットする関数は gnome_file_entry_set_filename です.

```
char* gnome_file_entry_get_full_path (GnomeFileEntry *fentry,
                                      gboolean       file_must_exist);
void gnome_file_entry_set_filename (GnomeFileEntry *fentry,
                                    const char     *filename);
```

サンプルプログラム

ファイルエントリのサンプルプログラムをソース F–2 に示します.

ソース F–2 ファイルエントリ：gnomefileentry-sample.c

```
1  #include <gnome.h>
2
3  int
4  main (int argc, char **argv)
5  {
6    GtkWidget *window;
7    GtkWidget *fileentry;
8
9    gnome_program_init ("gnome_file_entry-sample", "1.0.0", LIBGNOMEUI_MODULE,
10                       argc, argv, NULL);
11
12   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
13   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
14   gtk_window_set_title (GTK_WINDOW (window), "GnomeFileEntry-Sample");
15   gtk_widget_set_size_request (window, 360, -1);
16   g_signal_connect (G_OBJECT (window), "destroy",
17                     G_CALLBACK (gtk_main_quit), NULL);
18
19   fileentry = gnome_file_entry_new (NULL, NULL);
20   gtk_container_add (GTK_CONTAINER (window), fileentry);
21
22   gtk_widget_show_all (window);
23   gtk_main ();
24
25   return 0;
26 }
```

F.2.2　アバウトダイアログ

アバウトダイアログ (GnomeAbout)[*3]はアプリケーションの製作者やバージョン等の情報を表示するためのダイアログです. アイコンデータや製作者, バージョン等の必要な情報を与えるだけで簡単にダイアログを作成することができます.

ダイアログの作成

アバウトダイアログを作成するには, 関数 gnome_about_new を使用します. 関数のプロトタイプ宣言は次のようになります.

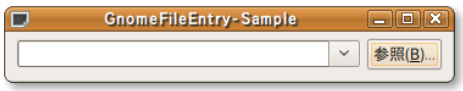


図 F.2　ファイルエントリウィジェットのサンプル

[*3] このウィジェットは既に GTK+ の GtkAboutDialog ウィジェットとして実装されていて, リファレンスマニュアルでは GnomeAbout を使用しないことを奨めています.

```
GtkWidget* gnome_about_new (const gchar  *name,
                            const gchar  *version,
                            const gchar  *copyright,
                            const gchar  *comments,
                            const gchar  **authors,
                            const gchar  **documenters,
                            const gchar  *translator_credits,
                            GdkPixbuf    *logo_pixbuf);
```

関数の引数に以下に示す情報を与えることで簡単にアプリケーション用のアバウトダイアログを作成することができます.

第 1 引数　アプリケーション名
第 2 引数　バージョン
第 3 引数　コピーライト
第 4 引数　アプリケーションの説明
第 5 引数　アプリケーション製作者の一覧
第 6 引数　ドキュメント製作者の一覧
第 7 引数　翻訳者
第 8 引数　ロゴ用の GdkPixbuf データ

サンプルプログラム

アバウトダイアログのサンプルプログラムをソース F–3 に示します.

ソース F–3 アバウトダイアログ：gnomeaboutdialog-sample.c

```
1  #include <gnome.h>
2
3  static void
4  cb_show_dialog (GtkWidget *widget, gpointer data)
5  {
6    GtkWidget   *dialog;
7    const gchar *authors[] = {"Yasuyuki␣Sugaya", NULL};
8    const gchar *documenters[] = {"Yasuyuki␣Sugaya", NULL};
9    gchar       *translators = "Yasuyuki␣Sugaya";
10
11   dialog = gnome_about_new ("GnomeAbout-Sample", "1.0.0",
12                             "Copyright␣(C)␣2009␣...",
13                             "This␣is␣a␣GnomeAbout␣dialog␣sample␣program.",
14                             authors, documenters, translators, NULL);
15   gtk_container_set_border_width (GTK_CONTAINER (dialog), 5);
16
17   gtk_widget_show_all (dialog);
18 }
19
20 int
21 main (int argc, char **argv)
22 {
```

図 F.3　アバウトダイアログのサンプル

```
23   GtkWidget *window;
24   GtkWidget *button;
25
26   gnome_program_init ("gnome_about_dialog-sample", "1.0.0",
27                       LIBGNOMEUI_MODULE, argc, argv, NULL);
28
29   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
30   gtk_container_set_border_width (GTK_CONTAINER (window), 5);
31   gtk_window_set_title (GTK_WINDOW (window), "GnomeAbout-Sample");
32   gtk_widget_set_size_request (window, 250, -1);
33   g_signal_connect (G_OBJECT (window), "destroy",
34                     G_CALLBACK (gtk_main_quit), NULL);
35
36   button = gtk_button_new_with_label ("Show␣About␣Dialog");
37   gtk_container_add (GTK_CONTAINER (window), button);
38   g_signal_connect (G_OBJECT (button), "clicked",
39                     G_CALLBACK (cb_show_dialog), NULL);
40
41   gtk_widget_show_all (window);
42   gtk_main ();
43
44   return 0;
45 }
```

索引

著者紹介

菅谷保之（すがや やすゆき）

筑波大学大学院博士課程修了．博士（工学，筑波大学）．
2001 年から 2006 年まで岡山大学工学部助手．
2006 年 4 月から豊橋技術科学大学情報工学系講師．
画像処理，コンピュータビジョンの研究に従事．

入門 GTK+

2006 年　3 月　1 日　初　版
2007 年　4 月　30 日　第 2 版
2009 年　10 月　25 日　第 3 版

著者　菅谷 保之
E-mail　sugaya@iim.ics.tut.ac.jp