PROGRAMER'S NOTEBOOK

GUI Application Programming using GTK+/GNOME

GTK+/GNOME による GUI アプリケーションプログラミング

菅谷 保之 著

THE TEO PROJECT

PROGRAMMER'S NOTEBOOK

PROGRAMMER'S NOTEBOOK シリーズ

GTK+/GNOME による GUI アプリケーションプログラミング

菅谷 保之 著

THE TEO PROJECT

まえがき

GTK+ は GIMP (*GNU Image Manipulation Program*) と呼ばれる画像編集ソフトを作成するために開発された GUI (*Graphical User Interface*) アプリケーションを作成するためのツールキット (toolkit) です.

現在では GIMP に留まらず様々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 にひけをとならいデスクトップ環境を実現している GNOME(*GNU Network Object Model Environment*) も, そのベースは GTK+ によって作られています. そのため, GTK+ は現在最も注目を浴びているツールキットだと言えるでしょう (GNOME と並んで 2 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様々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, 思ったような外観を作成するためにはプログラミングにかなりの手間がかかる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, ユーザが自由にその見た目を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+ や GNOME 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+/GNOME による GUI アプリケーション作成の学習に役立つと幸いです.

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.

開発環境の準備

1 章では, Linux で本書の内容を学習するための GTK+/GNOME の開発環境を設定する方法を説明します. 本書では Linux のディストリビューションとして Vine Linux 4.1 を採用し, 使用する開発環境やライブラリ等のインストール方法を示します.

GTK+ プログラミング入門

2 章では, GTK+ を使った簡単なプログラムを例に GTK+ のプログラムの流れやコンパイルの方法について解説します. また, GUI 作成にかかせないコンテナウィジェットやパッキングウィジェットをはじめとする基本的なウィジェットを紹介します.

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, GLIB を利用するとリストやハッシュなどの拡張データ構造とそれらのデータ構造に対する操作を簡単に行うことができます. 3 章では, GLIB が提供する便利な関数やデータ構造, それらに対する操作関数を具体例を挙げて紹介します.

GDK による図形の描画

4 章では, GTK+ のグラフィック部分を担当する GDK ライブラリによる図形描画について解説し, 直線や矩形, 円弧などの具体的な図形の描画方法について説明します.

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 を使用すると 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. 5 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.

ウィジェットリファレンス

GTK+ では操作性がよく, 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. 6 章では, これらのウィジェットについて, それぞれの機能や具体的な使用方法を紹介します.

拡張ウィジェットの作成

GTK+ では既存のウィジェットの他に独自のウィジェットを 1 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. 7 章では, 具体例を挙げて拡張ウィジェットの作成方法について解説します.

GNOME プログラミング入門

8 章では, GNOME アプリケーションに作成方法について解説します. GNOME のウィジェットは GTK+ のウィジェットよりさらに複雑になりますので, 本書では GNOME アプリケーション作成の導入のみ紹介します.

統合開発環境によるソフトウェア開発

9 章では, 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について解説します. 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+/GNOME アプリケーションを簡単に作成できるようになります.

必要なソフトウェア

本書ではソフトウェアインストールの容易さや日本語環境が充実していることから, Vine Linux 4.1 がインストールされた開発環境を想定して解説しています. また, 本書に載せている例題のサンプルソースは Vine Linux 4.1 上でコンパイルして, 動作を確認したものです. 1 章では Vine Linux 4.1 上に GTK+/GNOME の開発環境をインストールする手順について説明していますが, Vine Linux 4.1 以外のディストリビューションを使用する読者は, 必要な開発環境をインストールする必要があります.

ご意見とご質問

本書に関するご意見, ご質問は次のメールアドレスまでお願いいたします.

sugaya@iim.ics.tut.ac.jp

本書で掲載したサンプルプログラムのソースコードや正誤表などの情報は次のウェブページをご覧ください.

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

目次

第 1 章

プログラミングを始める前に

1.1 プログラミング環境を整えよう

まず, 本書はバージョン 2 以降の GTK+ を対象としていますので, バージョン 2 以降の GTK+ がインストールされたプログラミング環境が必要です. 最近のディストリビューションであれば間違いなくインストールされているでしょう.

本書では, ディストリビューションとして Vine Linux 4.1 を使用します. Vine Linux 4.1 では, 標準で最小限の開発環境しかインストールされません. 試しに Vine Linux 4.1 をフルインストールした直後の環境下で, 次のコマンドを実行してインストールされているパッケージを確認してみます. 本書では ↵ は改行記号の入力を表します.

```
% rpm -qa | grep gtk2 ↵
pygtk2-libglade-2.8.6-0vl1
gtk2-engines-2.7.4-0vl1
gtk2-2.8.20-0vl3
pygtk2-2.8.6-0vl1
```

開発用のヘッダファイルが含まれているのは xxx-devel というパッケージですから, gtk2 の開発環境がインストールされていないことがわかります. まずは, これから本書を読み進めるにあたって必要なパッケージをインストールする手順について説明します.

1.2 必要な開発パッケージのみをインストールする

ここでは, 個々のパッケージを選択してインストールする方法を説明します. 後になって必要なパッケージが出てくる度にいちいちインストールをするのが面倒な読者は次節の「全ての開発パッケージをインストール

する」をご覧下さい.

先程のコマンドで gtk2 パッケージの開発用パッケージがインストールされていないことがわかりましたので, まずは次のコマンドを実行し, 開発用パッケージ gtk2-devel をインストールしてみましょう.

```
% apt-get install gtk2-devel ↵
```

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ます. 指定したパッケージ以外にも, それに依存したパッケージが自動的にインストールされることがわかります. これらのパッケージをインストールしてもよい場合は y と入力し ↵ を押して下さい.

```
パッケージリストを読みこんでいます... 完了
依存情報ツリーを作成しています... 完了
以下の追加パッケージがインストールされます:
  XOrg-devel atk-devel cairo-devel freetype2-devel glib2-devel
  libpng-devel pango-devel zlib-devel
以下のパッケージが新たにインストールされます:
  XOrg-devel atk-devel cairo-devel freetype2-devel glib2-devel gtk2-devel
  libpng-devel pango-devel zlib-devel
アップグレード: 0 個, 新規インストール: 9 個, 削除: 0 個, 保留: 0 個
9416kB のアーカイブを取得する必要があります。
展開後に 38.3MB のディスク容量が追加消費されます。
続行しますか? [Y/n]y ↵
取得:1 http://updates.vinelinux.org 4.1/i386/main XOrg-devel 6.9.0-0vl23 [4371kB]
取得:2 http://updates.vinelinux.org 4.1/i386/main glib2-devel 2.12.7-0vl1 [1256kB]
取得:3 http://updates.vinelinux.org 4.1/i386/main atk-devel 1.12.1-0vl1 [117kB]
取得:4 http://updates.vinelinux.org 4.1/i386/main zlib-devel 1.2.3-0vl5 [95.3kB]
取得:5 http://updates.vinelinux.org 4.1/i386/main libpng-devel 2:1.2.12-0vl1.1 [177kB]
取得:6 http://updates.vinelinux.org 4.1/i386/main freetype2-devel 2.1.10-0vl8 [451kB]
取得:7 http://updates.vinelinux.org 4.1/i386/main cairo-devel 1.2.4-0vl2 [346kB]
取得:8 http://updates.vinelinux.org 4.1/i386/main pango-devel 1.14.7-0vl1 [265kB]
取得:9 http://updates.vinelinux.org 4.1/i386/main gtk2-devel 2.8.20-0vl3 [2337kB]
9416kB を 31s 秒で取得しました (302kB/s)
変更を適用しています...
準備中...                ########################################## [100%]
   1:XOrg-devel           ########################################## [ 11%]
   2:glib2-devel          ########################################## [ 22%]
   3:atk-devel            ########################################## [ 33%]
   4:zlib-devel           ########################################## [ 44%]
   5:libpng-devel         ########################################## [ 56%]
```

```
   6:freetype2-devel        ########################################### [ 67%]
   7:cairo-devel            ########################################### [ 78%]
   8:pango-devel            ########################################### [ 89%]
   9:gtk2-devel             ########################################### [100%]
完了
```

apt-get コマンドを使うとインストールするパッケージと依存関係にあるパッケージも同時にインストールすることができます. このように必要なファイルをダウンロードし, インストール完了のメッセージが表示されればパッケージのインストールは成功です. 一応確認のために gtk2-devel がインストールされているか確かめてみましょう. 初めに実行したコマンドで確認すると, 開発用パッケージがインストールされていることがわかります.

このテキストでは GNOME を用いたプログラムに関しても解説しますので, 同じ手順で libgnomeui-devel をインストールしておいて下さい.

```
% rpm -qa | grep gtk2 ↵
pygtk2-libglade-2.8.6-0vl1
gtk2-engines-2.7.4-0vl1
gtk2-devel-2.8.20-0vl3
gtk2-2.8.20-0vl3
pygtk2-2.8.6-0vl1
```

1.3 全ての開発パッケージをインストールする

必要なパッケージのみを選択してインストールするのが面倒で, ハードディスクの残り容量に余裕のある読者は次のコマンドを実行して, 開発用パッケージをインストールして下さい. ただし, このコマンドでは既にインストールされているパッケージに対する開発用パッケージしかインストールすることができません. すなわち, もともとインストールされていないパッケージに対する開発用パッケージはインストールされませんので注意して下さい.

```
% apt-get script install-devel.lua ↵
```

コマンドの実行結果を以下に示します. これをみてもわかるように 167 個ものパッケージ (242MB) がインストールされます.

```
パッケージリストを読みこんでいます... 完了
依存情報ツリーを作成しています... 完了
以下のパッケージが新たにインストールされます:
```

```
  GConf GConf-devel GConf2-devel ImageMagick-devel ORBit ORBit-devel
  ORBit2-devel VFlib-devel XOrg-gl-devel Xaw3d-devel aalib-devel alsa-lib-devel
  anthy-devel apache-devel apache2-devel apr-devel apr-util-devel apt-devel
  arts-devel aspell-devel audiofile-devel avahi-devel avahi-glib-devel
  beecrypt-devel beepmp-devel bind-devel bonobo bonobo-devel bzip2-devel
    ... 途中省略 ...
  pump-devel pygtk2-devel python-devel readline-devel rpm-devel ruby-devel
  samba-libsmbclient-devel scim-devel slang-devel sox-devel sqlite sqlite-devel
  sqlite3-devel startup-notification-devel taglib-devel vte-devel
  wireless-tools-devel
アップグレード: 0 個，新規インストール: 167 個，削除: 0 個，保留: 0 個
62.0MB のアーカイブを取得する必要があります。
展開後に 242MB のディスク容量が追加消費されます。
続行しますか? [Y/n] ↵
取得:1 http://updates.vinelinux.org 4.1/i386/main libxml2-devel 2.6.26-0vl1 [1929kB]
取得:2 http://updates.vinelinux.org 4.1/i386/main libIDL-devel 0.8.7-0vl1 [84.2kB]
取得:3 http://updates.vinelinux.org 4.1/i386/main ORBit2-devel 2.14.3-0vl1 [360kB]
取得:4 http://updates.vinelinux.org 4.1/i386/main GConf2-devel 2.14.0-0vl5 [183kB]
取得:5 http://updates.vinelinux.org 4.1/i386/main ImageMagick-devel 6.3.0.1-0vl1 [1657kB]
    ... 途中省略 ...
取得:163 http://updates.vinelinux.org 4.1/i386/main gnutls-devel 1.4.1-2vl1 [912kB]
取得:164 http://updates.vinelinux.org 4.1/i386/main gpgme-devel 1.0.3-0vl1 [76.2kB]
取得:165 http://updates.vinelinux.org 4.1/i386/main lcms-devel 1.15-0vl2 [130kB]
取得:166 http://updates.vinelinux.org 4.1/i386/main libtermcap-devel 2.0.8-44vl1 [55.1kB]
取得:167 http://updates.vinelinux.org 4.1/i386/main rpm-devel 4.4.2-0vl16 [1303kB]
62.0MB を 5m30s 秒で取得しました (188kB/s)
変更を適用しています...
準備中...                ########################################### [100%]
   1:libxml2-devel       ########################################### [  1%]
   2:libIDL-devel        ########################################### [  1%]
   3:ORBit2-devel        ########################################### [  2%]
   4:GConf2-devel        ########################################### [  2%]
   5:ImageMagick-devel   ########################################### [  3%]
    ... 途中省略 ...
 163:gnutls-devel        ########################################### [ 98%]
 164:gpgme-devel         ########################################### [ 98%]
 165:lcms-devel          ########################################### [ 99%]
 166:libtermcap-devel    ########################################### [ 99%]
 167:rpm-devel           ########################################### [100%]
完了
```

1.4 その他の必要なパッケージ

その他に本書では, GTK+/GNOME の統合開発環境 anjuta を使用しますので, anjuta とユーザインターフェイス作成ツール glade, gtranslator, devhelp など関連するアプリケーションをインストールしておきます. これらのパッケージのインストールも apt-get コマンドを使って行うことができます. 以下のようにインストールしたいパッケージを列挙すると依存関係にあるパッケージ含めて一度にインストールすることができます.

```
% apt-get install anjuta glade2 devhelp gtranslator ↵
パッケージリストを読みこんでいます... 完了
依存情報ツリーを作成しています... 完了
以下の追加パッケージがインストールされます:
  gdb gtkspell
以下のパッケージが新たにインストールされます:
  anjuta devhelp gdb glade2 gtkspell gtranslator
アップグレード: 0 個, 新規インストール: 6 個, 削除: 0 個, 保留: 0 個
8421kB のアーカイブを取得する必要があります。
展開後に 21.7MB のディスク容量が追加消費されます。
続行しますか? [Y/n]y
取得:1 http://updates.vinelinux.org 4.1/i386/plus gdb 6.4.90-0vl1 [3195kB]
取得:2 http://updates.vinelinux.org 4.1/i386/plus anjuta 1.2.4a-0vl1 [2564kB]
取得:3 http://updates.vinelinux.org 4.1/i386/plus devhelp 0.12-0vl1 [175kB]
取得:4 http://updates.vinelinux.org 4.1/i386/plus glade2 2.12.1-0vl1 [1866kB]
取得:5 http://updates.vinelinux.org 4.1/i386/plus gtkspell 2.0.11-0vl1 [27.5kB]
取得:6 http://updates.vinelinux.org 4.1/i386/plus gtranslator 1.1.6-0vl1 [593kB]
8421kB を 29s 秒で取得しました (284kB/s)
変更を適用しています...
準備中...                  ########################################## [100%]
   1:gdb                    ########################################## [ 17%]
   2:anjuta                 ########################################## [ 33%]
   3:devhelp                ########################################## [ 50%]
   4:glade2                 ########################################## [ 67%]
   5:gtkspell               ########################################## [ 83%]
   6:gtranslator            ########################################## [100%]
完了
```

Vine Linux 4.1 ではこれらのパッケージをインストールするとデスクトップのメニューにこれらのアプリケーションが追加されます．[アプリケーション]-[プログラミング] のメニューが図 1.1 のようになっていることを確認してみてください．

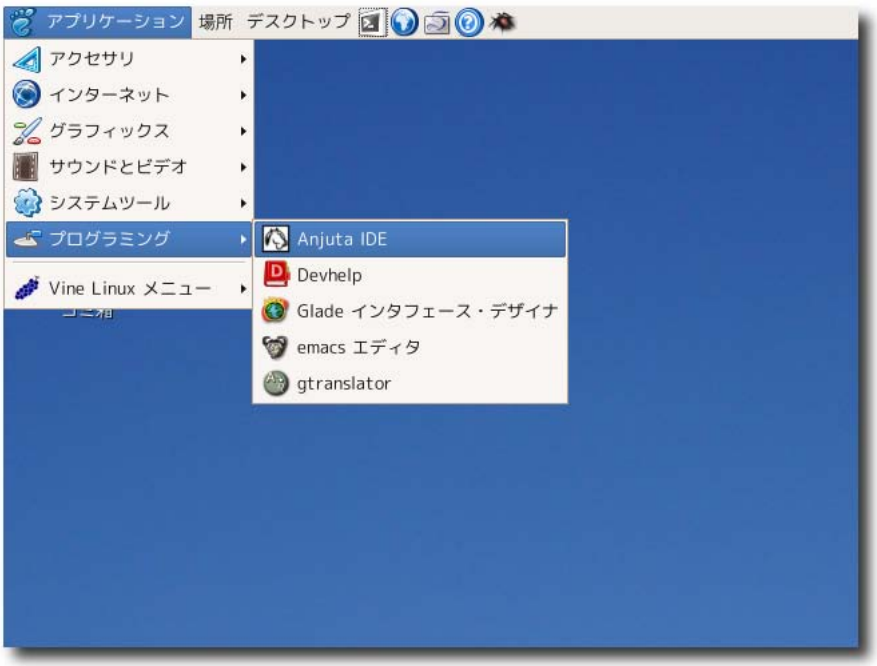


図 1.1　インストールしたアプリケーションのメニュー

gtranslator はプログラム内のメッセージの翻訳を支援するアプリケーションです．gtranslator の具体的な使い方は第 10 章 統合開発環境を使ったソフトウェア開発で紹介します．また devhelp は GTK+ などで使用する関数等を検索して使い方を調べるのに便利なアプリケーションです．GTK+ の関数は本書で紹介するもの以外にも膨大な数存在します．どんな関数が存在し，それらをどう使用するか調べる場合に大変役に立ちます．図 1.2 は devhelp の画面です．左上の検索欄に調べたい関数を入力すると関数の候補が左下にリストで表示され、詳細は右ウィンドウに表示されます．

✣ コラム〜GTK+ のテーマ機能

GTK+ にはテーマ機能というものがあり，ウィジェットの概観をテーマに応じて変えることができます．また同様にデスクトップのアイコンやウィンドウマネージャなどもテーマ機能を持つものが主流となってきています．GNOME-LOOK.ORG (http://gnome-look.org/) というサイトにさまざまなテーマが登録されています．

本書で使用しているテーマは Vine Linux 4.1 標準のテーマではないため，標準テーマをそのまま使用しているユーザは本書に掲載した実行例を見て戸惑うかもしれません．

テーマの変更についての詳細や本書で使用しているテーマについては付録 A GTK+ テーマの変更を参照してください．

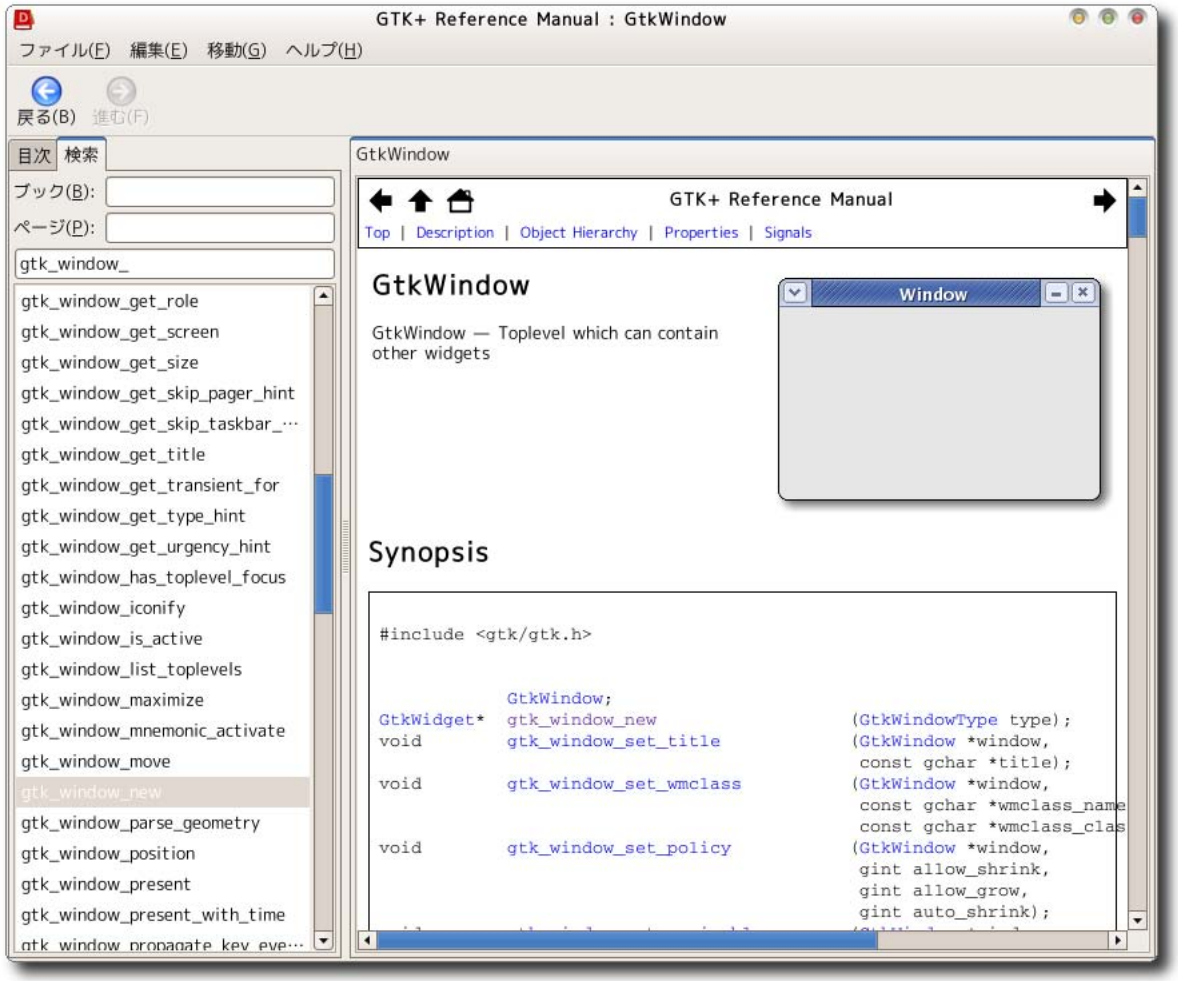


図 1.2　devhelp の起動画面

1.5　本書で使用する開発環境

表 1.1 に本書で使用する開発環境をまとめておきます．Vine Linux 4.1 以外のディストリビューションを使用する読者は参考にしてください．

表 1.1　本書で使用する開発環境

	名称	バージョン
ディストリビューション	Vine Linux	4.1
カーネル	kernel	2.6.16
コンパイラ	gcc	3.3.6
パッケージ	autoconf	2.59
	automake	1.9.6
	gtk	2.8.20
	libgnome	2.14.1
	libgnomeui	2.14.1
	libgnomeprint	2.12.1
	libgnomeprintui	2.12.1
	gnome-vfs	2.14.2
	GConf	2.14.0
	anjuta	1.2.4a
	glade2	2.12.1
	devhelp	0.12
	gtranslator	1.1.6

第 2 章

はじめての GTK+

2.1 簡単なプログラム

まずは, 簡単な例を解説しながら GTK+ の GUI アプリケーションがどのような仕組で動作するのか, どうやってプログラムをコンパイルするのかを説明していきます. ここでは, 図 2.1 のようなボタン付きのウィンドウを作成します. このプログラムのソースコードをソース 2–1 に示します.

図 2.1 Hello World

ソース 2–1 Hello World : hello_world.c

```
1  #include <gtk/gtk.h>
2
3  static void cb_button (GtkWidget *widget, gpointer data) {
4    g_print ("Hello World\n");
5  }
6
7  int main (int argc, char *argv[]) {
8    GtkWidget *window;
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), "Hello␣World");
15   gtk_widget_set_size_request (window, 250, 80);
16   gtk_container_set_border_width (GTK_CONTAINER(window), 10);
17
18   button = gtk_button_new_with_label ("Hello␣World");
19   gtk_container_add (GTK_CONTAINER(window), button);
20
21   g_signal_connect (G_OBJECT(window),
22                     "destroy", G_CALLBACK(gtk_main_quit), NULL);
23   g_signal_connect (G_OBJECT(button),
24                     "clicked", G_CALLBACK(cb_button), NULL);
25
26   gtk_widget_show_all (window);
27   gtk_main ();
28
29   return 0;
30 }
```

ソース 2–1 のソースコードをコンパイルするには次のようにします．pkg-config に–cflags –libs オプションをつけることでコンパイルに必要なヘッダファイルのあるディレクトリやライブラリを gcc のオプションとして与えることができます．

```
% gcc hello_world.c -o hello_world ‘pkg-config gtk+-2.0 --cflags --libs‘
```

それでは，ソース 2–1 の内容を解説しましょう．

ヘッダファイルのインクルード (1 **行目**)

GTK+ 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 gtk.h をインクルードしています．Vine Linux 4.1 では/usr/include/gtk-2.0/gtk というディレクトリ内に gtk.h がインストールされています．

GTK+ **の初期化** (11 **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ための関数です．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);
```

GUI **の作成** (13–19 **行目**)

まず，アプリケーションのベースとなるウィンドウを作成して，ウィンドウのタイトルバーの文字を ”Hello World” としています．そして，ボタンを作成し，ウィンドウウィジェットの中に作成したボタンウィジェットを配置しています．ウィンドウウィジェットのように他のウィジェットを一つ格納できるウィジェットをコンテナと呼びます．また，複数のウィジェットを格納できるウィジェットをパッキングボックスと呼びます．

16 行目に出てくる GTK_CONTAINER は GtkWidget 型の変数を GtkContainer 型にキャストするマクロです．GTK+ のウィジェットは C++ のクラスのような構造をしています．各ウィジェットの階層構造は図 2.2 を参照してください．

コールバック関数の設定 (21–24 **行目**)

この行では作成したウィンドウウィジェットとボタンウィジェットの持つイベントに対して, そのイベントが発生したときに呼び出す関数を関連づけています. この関数をコールバック関数と呼びます.

メインループ (27 **行目**)

関数 gtk_main を呼び出すことで, アプリケーションはイベント待ち状態となります. なんらかのイベントが発生し, それに対するコールバック関数が定義されていれば, コールバック関数が呼び出されます.

コールバック関数 (3–5 **行目**)

イベントに対するコールバック関数を定義しています. ここではボタンが押されるとボタンウィジェットの ”clicked” シグナルに関連づけられたコールバック関数 cb_button が呼び出され, 関数 g_print によってターミナルに Hello World の文字が出力されます.

2.2　ウィジェットの階層構造

前節の例ではウィンドウの中にボタンが配置されている簡単な GUI を作成しました. これらのウィンドウやボタンをまとめてウィジェットと呼びます. ウィジェットとは GUI を構成する部品のことです. 前節でも簡単に説明しましたが, ウィジェットは C++ のクラスのような構造を持ちます. 全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. 階層関係にあるウィジェット間で型変換を行うために, マクロが定義されています. 例えば, GtkWindow 型の変数をその上位の GtkContainer 型にキャストするためには, マクロ GTK_CONTAINER を使用します.

ウィジェットの階層構造の一部を図 2.2 に示します. 全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://developer.gnome.org/doc/API/2.0/gtk/ch01.html) を参照してください.

2.3　イベントとコールバック関数

ここではウィジェットに対する操作 (イベント) と, そのイベントが発生したときに実行する関数 (コールバック関数) について解説します.

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. 例えば, ボタンウィジェットはボタンをクリックしたときに”clicked” という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設定しておけば, ボタンをクリックしたときにある動作を実行するということができるのです. 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知るには, GTK+ 2.0 Tutorial の Appendix A. GTK Signals (http://www.gtk.org/tutorial/a2700.html) を参照するといいでしょう. また, 9 章で説明する glade を使うとシグナルを確認することができます.

```
GObject
    +---- GtkAccelGroup
    +---- GtkAccelMap
    +---- GtkObject
              +---- GtkWidget
              |          +---- GtkMisc
              |          |          +---- GtkLabel
              |          |          |          +---- GtkAccelLabel
              |          |          |          +---- GtkTipsQuery
              |          |          +---- GtkArrow
              |          |          +---- GtkImage
              |          |          +---- GtkPixmap
              |          +---- GtkContainer
              |          |          +---- GtkBin
              |          |          |          +---- GtkButton
              |          |          |          +---- GtkWindow
              |          |          |          ...
              |          |          +---- GtkBox
              |          |                     +---- GtkButtonBox
              |          |                     +---- GtkHBox
              |          |                     +---- GtkVBox
              |          +---- GtkDrawingArea
              |          +---- GtkEntry
              |          |          +---- GtkSpinButton
              |          +---- GtkRange
              |          |          +---- GtkScale
              |          |          |          +---- GtkHScale
              |          |          |          +---- GtkVScale
              |          |          +---- GtkScrollbar
              |          |                     +---- GtkHScrollbar
              |          |                     +---- GtkVScrollbar
              |          +---- GtkSeparator
              |          ...       +---- GtkHSeparator
              |                    +---- GtkVSeparator
              +---- GtkAdjustment
              +---- GtkCellRenderer
              |          +---- GtkCellRendererPixbuf
              |          +---- GtkCellRendererText
              |          +---- GtkCellRendererToggle
              +---- GtkTooltips
              +---- GtkTreeViewColumn
              ...
```

図 2.2 ウィジェットの階層構造

では具体的にシグナルとコールバック関数を関連付ける方法ですが, それには関数 g_signal_connect を使用します. ソース 2–1 の 23–24 行目を見てみましょう.

```
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は一般に次のような形をしています.

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

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. この際にデータを gpointer 型にキャストするのを忘れないように注意してください. 3 章で説明しますが, gpointer 型は gtypes.h で次のように定義されています.

```
typedef void* gpointer;
```

コールバック関数の書式は上記のものだけではなく, 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を確認するには, GTK+ 2.0 Tutorial の Appendix A. GTK Signals (http://www.gtk.org/tutorial/a2700.html) を参照してください.

2.4 ウィジェットのパッキング

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 (パッキング) することで, 簡単に, そして, 見た目のよい GUI を作成することができます.

本節では, ウィジェットのパッキングについて説明します. ここではウィジェットのパッキング方法として代表的な 3 つの方法を挙げて, 具体例とともに解説します.

2.4.1 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. ソース 2–1 に出てきた 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次の関数を使います. 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.

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

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 (ボーダ幅) を空けるかを指定することができます. 図 2.3 の矢印で示した空白がボーダ幅になります. ボーダ幅の設定には関数 gtk_container_set_border_width を使用します.

図 2.3　コンテナウィジェットのボーダ幅

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

2.4.2　パッキングボックス

コンテナはウィジェットを 1 つしか配置できませんので, コンテナだけでは複雑な GUI を作成することができません. 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することの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);
```

また, 垂直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);
```

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. それぞれの引数は次のようになっています. 同じような関数に gtk_box_pack_end があります. 関数 gtk_box_pack_start がウィジェットを左 (GtkVbox の場合は上) から右 (GtkVbox の場合は下) へ配置するのに対して, 関数 gtk_box_pack_end はウィジェットを右 (下) から左 (上) へ配置していきます.

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 or FALSE)
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 or FALSE)
第 5 引数　ボックスの両端にどれだけすき間を空けるか

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

表 2.1　gtk_box_pack_start に与えたパラメータ

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

この関数の第 3, 第 4 引数の与え方によって, GUI の見た目が変化します. 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. ソース 2–2 のソースコードをコンパイルして実行した結果が図 2.4 です. これは垂直ボックスに水平ボックスを 3 つ配置して, それぞれの水平ボックスに expand と fill の値を変えてボタンを配置したものです. 各行でボタンの大きさや配置が異なるのがわかるでしょう.

表 2.1 に関数 gtk_box_pack_start に与えたパラメータをまとめました. 図 2.4 の 1 行目を見てもわかるとおり, expand, fill ともに TRUE とした場合には, 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ています. 2 行目では, 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, 全てのウィジェットが最小限の大きさに留まっているのがわかります. fill パラメータは expand パラメータが TRUE のときに有効であることに注意してください.

ソース 2–2 ウィジェットの配置： packing-sample.c

```
1  #include <gtk/gtk.h>
2
3  int main (int argc, char *argv[]) {
4    GtkWidget *window;
5    GtkWidget *vbox, *hbox;
6    GtkWidget *button;
7
8    gtk_init (&argc, &argv);
9
10   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
11   gtk_window_set_title (GTK_WINDOW(window), "Packing␣Sample");
12   gtk_container_set_border_width(GTK_CONTAINER(window), 10);
13   g_signal_connect (G_OBJECT(window),
14                     "destroy", G_CALLBACK(gtk_main_quit), NULL);
15
```

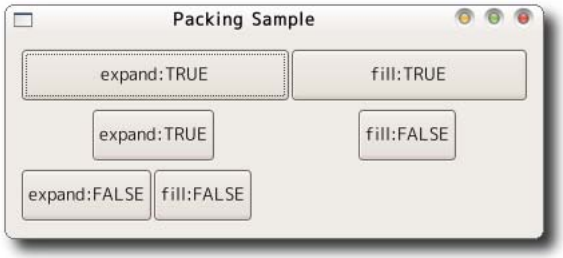


図 2.4　ウィジェットの配置方法

```
16    vbox = gtk_vbox_new (TRUE, 5);
17    gtk_container_add (GTK_CONTAINER(window), vbox);
18
19    hbox = gtk_hbox_new (FALSE, 0);
20    gtk_box_pack_start (GTK_BOX(vbox), hbox, TRUE, TRUE, 0);
21    button = gtk_button_new_with_label ("expand:TRUE");
22    gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
23    button = gtk_button_new_with_label ("fill:TRUE");
24    gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
25
26    hbox = gtk_hbox_new (FALSE, 0);
27    gtk_box_pack_start (GTK_BOX(vbox), hbox, TRUE, TRUE, 0);
28    button = gtk_button_new_with_label ("expand:TRUE");
29    gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, FALSE, 0);
30    button = gtk_button_new_with_label ("fill:FALSE");
31    gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, FALSE, 0);
32
33    hbox = gtk_hbox_new (FALSE, 0);
34    gtk_box_pack_start (GTK_BOX(vbox), hbox, TRUE, TRUE, 0);
35    button = gtk_button_new_with_label ("expand:FALSE");
36    gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
37    button = gtk_button_new_with_label ("fill:FALSE");
38    gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
39
40    gtk_widget_show_all (window);
41    gtk_main ();
42
43    return 0;
44 }
```

2.4.3 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, 第 2 引数で縦に配置するウィジェット数, 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, 全てのセルの大きさが一番大きなウィジェットの大きさに統一されます (図 2.5 上段). FALSE を指定した場合は, 行と列で独立にセルの大きさを調整します (図 2.5 下段).

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

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. 関数の第 1 引数と第 2 引数には, テーブルウィジェットと配置するウィジェットを指定します.

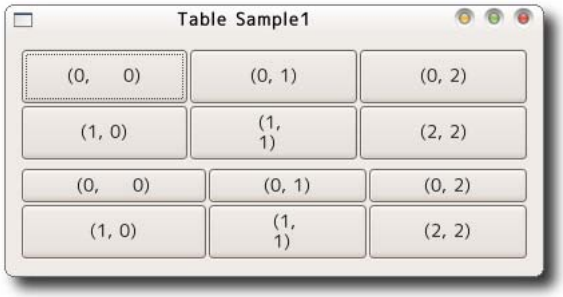


図 2.5 テーブルウィジェットへの配置 その 1

```
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 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 2.6 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 2.6 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.

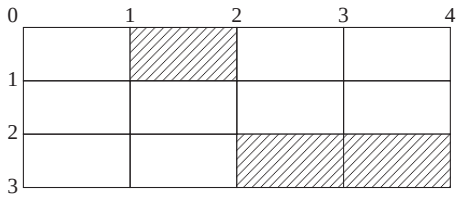


図 2.6 テーブルウィジェットへの配置指定

xoptoins, 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 は次のように定義されていて, 論理和演算によって複数指定することができます.

```
typedef enum
{
  GTK_EXPAND = 1 << 0,
  GTK_SHRINK = 1 << 1,
  GTK_FILL   = 1 << 2
} GtkAttachOptions;
```

GTK_EXPAND ... もしテーブルの格子があるウィジットよりも大きい場合は, ウィジットが広がります.

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ります. そのような場合に GTK_SHRINK が指定されていると,

表 2.2　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

ウィジットはテーブルの中で縮みます.

GTK_FILL　... 領域を埋めるようにテーブルを広げます.

xoptions, yoptions にそれぞれ, 表 2.2 に示した値を与えた場合, 図 2.7 のようになります. 図 2.7 のソースコードをソース 2–3 に示します.

ソース 2–3 テーブルへの配置：table-sample2.c

```
1  #include <gtk/gtk.h>
2
3  int main (int argc, char *argv[]) {
4    GtkWidget *window, *vbox, *table, *button;
5
6    gtk_init (&argc, &argv);
7
8    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
9    gtk_window_set_title (GTK_WINDOW(window), "Table␣Sample2");
10   gtk_container_set_border_width(GTK_CONTAINER(window), 10);
11   g_signal_connect (G_OBJECT(window),
12                     "destroy", G_CALLBACK(gtk_main_quit), NULL);
13   vbox = gtk_vbox_new (FALSE, 5);
14   gtk_container_add (GTK_CONTAINER(window), vbox);
15
16   table = gtk_table_new (3, 4, FALSE);
17   gtk_box_pack_start (GTK_BOX(vbox), table, TRUE, TRUE, 0);
18   {
19     button = gtk_button_new_with_label ("January");
20     gtk_table_attach (GTK_TABLE(table), button, 0, 1, 0, 1,
21                       GTK_FILL | GTK_SHRINK | GTK_EXPAND,
```

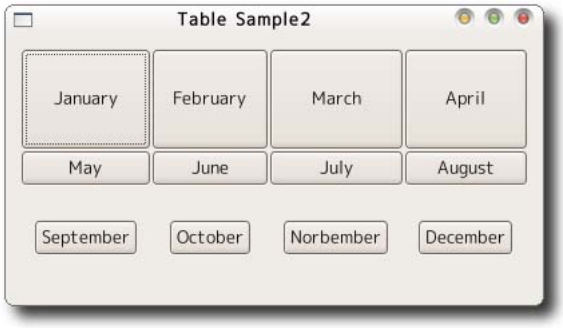


図 2.7　テーブルウィジェットへの配置 その 2

```
22                      GTK_FILL | GTK_SHRINK | GTK_EXPAND, 0, 0);
23    button = gtk_button_new_with_label ("February");
24    gtk_table_attach (GTK_TABLE(table), button, 1, 2, 0, 1,
25                      GTK_FILL | GTK_SHRINK | GTK_EXPAND,
26                      GTK_FILL | GTK_SHRINK | GTK_EXPAND, 0, 0);
27    button = gtk_button_new_with_label ("March");
28    gtk_table_attach (GTK_TABLE(table), button, 2, 3, 0, 1,
29                      GTK_FILL | GTK_SHRINK | GTK_EXPAND,
30                      GTK_FILL | GTK_SHRINK | GTK_EXPAND, 0, 0);
31    button = gtk_button_new_with_label ("April");
32    gtk_table_attach (GTK_TABLE(table), button, 3, 4, 0, 1,
33                      GTK_FILL | GTK_SHRINK | GTK_EXPAND,
34                      GTK_FILL | GTK_SHRINK | GTK_EXPAND, 0, 0);
35    button = gtk_button_new_with_label ("May");
36    gtk_table_attach (GTK_TABLE(table), button, 0, 1, 1, 2,
37                      GTK_FILL | GTK_SHRINK,
38                      GTK_FILL | GTK_SHRINK, 0, 0);
39    button = gtk_button_new_with_label ("June");
40    gtk_table_attach (GTK_TABLE(table), button, 1, 2, 1, 2,
41                      GTK_FILL | GTK_SHRINK,
42                      GTK_FILL | GTK_SHRINK, 0, 0);
43    button = gtk_button_new_with_label ("July");
44    gtk_table_attach (GTK_TABLE(table), button, 2, 3, 1, 2,
45                      GTK_FILL | GTK_SHRINK,
46                      GTK_FILL | GTK_SHRINK, 0, 0);
47    button = gtk_button_new_with_label ("August");
48    gtk_table_attach (GTK_TABLE(table), button, 3, 4, 1, 2,
49                      GTK_FILL | GTK_SHRINK,
50                      GTK_FILL | GTK_SHRINK, 0, 0);
51    button = gtk_button_new_with_label ("September");
52    gtk_table_attach (GTK_TABLE(table), button, 0, 1, 2, 3,
53                      GTK_SHRINK | GTK_EXPAND,
54                      GTK_SHRINK | GTK_EXPAND, 0, 0);
55    button = gtk_button_new_with_label ("October");
56    gtk_table_attach (GTK_TABLE(table), button, 1, 2, 2, 3,
57                      GTK_SHRINK | GTK_EXPAND,
58                      GTK_SHRINK | GTK_EXPAND, 0, 0);
59    button = gtk_button_new_with_label ("Norbember");
60    gtk_table_attach (GTK_TABLE(table), button, 2, 3, 2, 3,
61                      GTK_SHRINK | GTK_EXPAND,
62                      GTK_SHRINK | GTK_EXPAND, 0, 0);
63    button = gtk_button_new_with_label ("December");
64    gtk_table_attach (GTK_TABLE(table), button, 3, 4, 2, 3,
65                      GTK_SHRINK | GTK_EXPAND,
66                      GTK_SHRINK | GTK_EXPAND, 0, 0);
67  }
68  gtk_widget_show_all (window);
69  gtk_main ();
70  return 0;
71 }
```

2.5　シグナルとコールバック関数の詳細

ここでは先ほど簡単に説明した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.

2.5.1　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);
```

引数の意味は以下の通りです. 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 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

第 5 引数に与える関数は次のように定義されています. この関数の第 1 引数に第 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.

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

GConnectFlags は次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.

```
typedef enum
{
  G_CONNECT_AFTER   = 1 << 0,
  G_CONNECT_SWAPPED = 1 << 1
} GConnectFlags;
```

関数 g_signal_connect_data を使ったプログラムの例をソース 2–4 に示します. プログラムを実行すると図 2.8 のウィンドウが表示されます. 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. このデータは関数 g_strdup で領域確保した文字列ですので, プログラムの終了時にはこの領域を解放する必要があります. 関数の第 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ですので, 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.” と表示されてプログラムが終了します.

図 2.8　コールバック関数の例 1

```
% ./signal-sample1 ↵
Hello World
Hello World
Destroy the callback function data.
%
```

ソース 2–4　コールバック関数の例 1：signal-sample1.c

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

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 GClosureFunc を必要としないことが多いです．そのため，関数 g_signal_connect_data のマクロとして以下に示す 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関数がこの関数です．この関数はマクロとして以下のように定義されています．関数 g_signal_connect_data の最後の引数は 0 となっており，このとき，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．

```
#define g_signal_connect(instance, detailed_signal, c_handler, data)\
        g_signal_connect_data ((instance), (detailed_signal), \
                               (c_handler), (data), NULL, \
                               (GConnectFlags) 0)
```

- 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．

```
#define g_signal_connect_after(instance, detailed_signal, \
                               c_handler, data) \
        g_signal_connect_data ((instance), (detailed_signal), \
                               (c_handler), (data), NULL, \
                               G_CONNECT_AFTER)
```

- g_signal_connect_swapped
  この関数は関連づ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づ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．

```
#define g_signal_connect_swapped(instance, detailed_signal, \
                                 c_handler, data) \
        g_signal_connect_data ((instance), (detailed_signal), \
                               (c_handler), (data), NULL, \
                               G_CONNECT_SWAPPED)
```

これらの関数について具体的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 2–5 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 (33–40 行目)．このように一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．そして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てみます．プログラムを実行してボタンをクリックすると以下のようにターミナルにメッセージが表示されます．

```
% ./signal-sample1 ↵
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 2–5 コールバック関数の動作：signal-sample2.c

```
1  #include <gtk/gtk.h>
2
3  static void cb_button1 (GtkWidget *widget, gpointer data) {
4    g_print ("function␣1.\n");
5  }
6
7  static void cb_button2 (GtkWidget *widget, gpointer data) {
8    g_print ("function␣2.\n");
9  }
10
11 static void cb_button3 (GtkWidget *widget, gpointer data) {
12   g_print ("function␣3.\n");
13 }
14
15 static void cb_button4 (GtkWidget *widget, gpointer data) {
16   g_print ("function␣4.\n");
17 }
18
19 int main (int argc, char *argv[]) {
20   GtkWidget *window, *button;
21
22   gtk_init (&argc, &argv);
23
24   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
25   gtk_window_set_title (GTK_WINDOW(window), "Signal␣Sample2");
26   g_signal_connect (G_OBJECT(window),
27                     "destroy", G_CALLBACK(gtk_main_quit), NULL);
28
29   button = gtk_button_new_with_label ("Click␣me!");
30   gtk_container_add (GTK_CONTAINER(window), button);
31
32   g_signal_connect_after (G_OBJECT(button), "clicked",
33                           G_CALLBACK(cb_button3), NULL);
34   g_signal_connect_after (G_OBJECT(button), "clicked",
```

```
35                              G_CALLBACK(cb_button4), NULL);
36   g_signal_connect (G_OBJECT(button),
37                     "clicked", G_CALLBACK(cb_button1), NULL);
38   g_signal_connect (G_OBJECT(button),
39                     "clicked", G_CALLBACK(cb_button2), NULL);
40
41   gtk_widget_show_all (window);
42   gtk_main ();
43
44   return 0;
45 }
```

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 2–6 です．

28–29 行目では一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 33–34 行目で二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一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ますので二つ目のボタンのコールバック関数を関数 g_signal_connect 等で設定すると二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 引数に一つ目のウィジェットを指定していますので，二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一つ目のボタン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一つ目のボタンになります．

プログラムを実行すると，図 2.9 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．

図 2.9 関数 g_signal_connect_swapped の例

ソース 2–6 関数 g_signal_connect_swapped の例：signal-sample3.c

```
1  #include <gtk/gtk.h>
2
3  void cb_button (GtkWidget *widget, gpointer data) {
4    static gint count = 0;
5    gchar       buf[64];
6
7    sprintf (buf, "%d␣time(s)␣clicked.", ++count);
8    gtk_button_set_label (GTK_BUTTON(widget), buf);
9  }
10
11 int main (int argc, char *argv[]) {
12   GtkWidget *window;
13   GtkWidget *hbox;
14   GtkWidget *button1, *button2;
15
16   gtk_init (&argc, &argv);
17
18   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
19   gtk_window_set_title (GTK_WINDOW(window), "Signal␣Sample3");
20   g_signal_connect (G_OBJECT(window),
21                     "destroy", G_CALLBACK(gtk_main_quit), NULL);
22
23   hbox = gtk_hbox_new (TRUE, 0);
24   gtk_container_add (GTK_CONTAINER(window), hbox);
25
26   button1 = gtk_button_new_with_label ("0␣time(s)␣clicked.");
27   gtk_box_pack_start (GTK_BOX(hbox), button1, TRUE, TRUE, 0);
28   g_signal_connect (G_OBJECT(button1), "clicked",
29                     G_CALLBACK(cb_button), NULL);
30
31   button2 = gtk_button_new_with_label ("Click␣me!");
32   gtk_box_pack_start (GTK_BOX(hbox), button2, TRUE, TRUE, 0);
33   g_signal_connect_swapped (G_OBJECT(button2), "clicked",
34                             G_CALLBACK(cb_button), (gpointer) button1);
35
36   gtk_widget_show_all (window);
37   gtk_main ();
38
39   return 0;
40 }
```

2.5.2　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);
```

ソース 2–7 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 2.10 のように二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 42–43 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．

右のボタンをクリックすると 49–50 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．しかし，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．もう一つの方法は，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 46–47 行目の関数 g_object_set_data です．この関数は第 1 引数のオブジェクトに第 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関連付けるものです．

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

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);
```

また，関数 disconnect_handler 内では何度も関数 g_signal_handler_disconnect が実行されるのを回避するために，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);
```

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

ソース 2–7 関数 g_signal_connect_swapped の例：signal-sample4.c

```
1  #include <gtk/gtk.h>
2
3  void cb_button (GtkWidget *widget, gpointer data) {
4    g_printf ("Callback function is called.\n");
5  }
6
7  void disconnect_handler (GtkWidget *widget, gpointer data) {
8    GtkWidget *button;
9    gulong    handle;
10
11   button = GTK_WIDGET(g_object_get_data (G_OBJECT(widget), "button1"));
12   handle = (gulong) g_object_get_data (G_OBJECT(widget), "handle");
13
14   if (g_signal_handler_is_connected (button, handle)) {
15     g_signal_handler_disconnect (button, handle);
16     g_print ("Callback function is diconnected.\n");
17     return;
18   } else {
19     g_print ("Callback function is already diconnected.\n");
20   }
21 }
22
23 int main (int argc, char *argv[]) {
24   GtkWidget *window;
25   GtkWidget *hbox;
26   GtkWidget *button1, *button2;
27   gulong    handle;
28
29   gtk_init (&argc, &argv);
30
31   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
32   gtk_window_set_title (GTK_WINDOW(window), "Signal Sample4");
33   gtk_widget_set_size_request (window, 500, 50);
34   g_signal_connect (G_OBJECT(window),
35                     "destroy", G_CALLBACK(gtk_main_quit), NULL);
36
37   hbox = gtk_hbox_new (TRUE, 0);
38   gtk_container_add (GTK_CONTAINER(window), hbox);
39
```

図 2.10　コールバック関数の解除

```
40    button1 = gtk_button_new_with_label ("Click␣me!");
41    gtk_box_pack_start (GTK_BOX(hbox), button1, TRUE, TRUE, 0);
42    handle = g_signal_connect (G_OBJECT(button1),
43                               "clicked", G_CALLBACK(cb_button), NULL);
44
45    button2 = gtk_button_new_with_label ("Disconnect␣button1␣callbacks.");
46    g_object_set_data (G_OBJECT(button2), "button1", (gpointer) button1);
47    g_object_set_data (G_OBJECT(button2), "handle",  (gpointer) handle);
48    gtk_box_pack_start (GTK_BOX(hbox), button2, TRUE, TRUE, 0);
49    g_signal_connect (G_OBJECT(button2),
50                      "clicked", G_CALLBACK(disconnect_handler), NULL);
51
52    gtk_widget_show_all (window);
53    gtk_main ();
54
55    return 0;
56 }
```

2.5.3 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．そのしくみや動作を解説するのは大変ですので，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について紹介します．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” というイベントが発生します．ソース 2–8 では，35–36 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 3–22 行目です．ここでは図 2.11 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 6.5 節ダイアログ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” イベント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” イベントが発生します．37–38 行目で対応するコールバック関数 cb_destroy を設定していますので，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” イベントに対するコールバック関数の戻り値が TRUE の場合，”destroy” イベントは発生せずプログラムは続行します．

図 2.11 シグナルの伝搬

ソース 2–8 シグナルの伝搬：signal-sample5.c

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

第 3 章

GLib

GLib は C 言語で定義された関数を拡張した関数や, プラットホ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ための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について紹介します.

3.1 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ホ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 3.1 のように定義しています.

3.2 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.

3.2.1 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);
```

- 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. sprintf と同じ働きをしますが, 予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, va_list args);
```

表 3.1 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	gint
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
gint8	signed char
guint8	unsigned char
gint16	signed short
guint16	unsigned short
gint32	signed int
guint32	unsigned int
gfloat	float
gdouble	double

- 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);
```

- 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);
```

- 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);
```

- g_strfreev

  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);
```

3.2.2 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);
```

  GFileTest は次のように定義されています．

```
typedef enum
{
  G_FILE_TEST_IS_REGULAR    = 1 << 0,
  G_FILE_TEST_IS_SYMLINK    = 1 << 1,
  G_FILE_TEST_IS_DIR        = 1 << 2,
  G_FILE_TEST_IS_EXECUTABLE = 1 << 3,
  G_FILE_TEST_EXISTS        = 1 << 4
} GFileTest;
```

- 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次の関数 g_dir_read_name を使用します．

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

- 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 が帰ります．

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

- 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);
```

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 3–1 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 (15 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 (17–22 **行目**)

17 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一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ますので, 21 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 (23 **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.

以下にプログラムの実行結果を示します.

```
% gcc file-sample.c -o file-sample `pkg-config --cflags --libs glib-2.0` ↵
% ./file-sample . ↵
file-sample.c   (file)
file-sample.o   (file)
testfolder      (folder)
file-sample     (file)
Makefile        (file)
%
```

ソース 3–1 ファイル操作の例 : file-sample.c

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

```
22      }
23      g_dir_close (dir);
24    }
25    return 0;
26  }
```

3.2.3 Unicode に関する関数

- g_locale_to_utf8
  C 言語中で文字列用のエンコーディングで与えられた文字列を UTF8 でエンコーディングされた文字列に変換する関数です．この関数によって新しい文字列用のメモリ領域が確保されますので，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 など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);
```

- g_locale_from_utf8
  UTF8 でエンコーディングされた文字列を C 言語中で文字列用のエンコーディングに変換する関数です．この関数によって新しい文字列用のメモリ領域が確保されますので，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 など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);
```

サンプルプログラム

GTK+ のウィジェット内で扱われる文字コードは UTF8 です．しかし，関数 g_dir_open などで読み出したファイル名やプログラムの引数として与えた文字列などは，ロケール指定の文字コードとなります．ここでは，プログラムの引数として与えた文字列をそのままラベルとして表示した場合と，文字コードを UTF8 に変換してラベルとして表示した場合を比較してみます．図 3.1 はプログラムの実行例です．

```
% gcc utf8-sample.c -o utf8-sample `pkg-config --cflags --libs glib-2.0` ↵
% ./utf8-sample こんにちは世界 ↵
```

上のラベルはプログラムの引数に与えた文字列 ”こんにちは世界” をそのまま使用したラベルです．下のラベルはプログラムの引数に与えた文字列の文字コードを関数 g_locale_to_utf8 で UTF8 に変更して使用したラベルです．関数の第 2 引数には変換する文字列の長さを指定しますが，文字列を全て変換する場合には -1 を

与えても構いません. 第 3 引数から第 5 引数までは通常 NULL を指定しておいても構いません.

ソース 3–2 文字コード変換：utf8-sample.c

```
1  #include <gtk/gtk.h>
2  #include <stdlib.h>
3
4  int main (int argc, char *argv[]) {
5    GtkWidget      *window, *label, *vbox;
6    gchar          *name, *utf8name;
7
8    if (argc != 2) {
9      g_print ("Usage:␣./utf8-sample.c␣string\n");
10     exit (1);
11   }
12   gtk_init (&argc, &argv);
13
14   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
15   gtk_window_set_title (GTK_WINDOW(window), "UTF8␣Sample");
16   g_signal_connect (G_OBJECT(window),
17                     "destroy", G_CALLBACK(gtk_main_quit), NULL);
18
19   vbox = gtk_vbox_new (TRUE, 0);
20   gtk_container_add (GTK_CONTAINER(window), vbox);
21
22   name = argv[1];
23   label = gtk_label_new (name);
24   gtk_box_pack_start (GTK_BOX(vbox), label, TRUE, TRUE, 0);
25
26   utf8name = g_locale_to_utf8 (name, -1, NULL, NULL, NULL);
27   label = gtk_label_new (utf8name);
28   gtk_box_pack_start (GTK_BOX(vbox), label, TRUE, TRUE, 0);
29   g_free (utf8name);
30
31   gtk_widget_show_all (window);
32   gtk_main ();
33
34   return 0;
35 }
```

図 3.1 文字コード変換のサンプルプログラム

3.2.4 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);
```

- g_get_current_dir
  現在のディレクトリを取得する関数です.

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

- 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);
```

- 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);
```

- 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, ...);
```

3.3 連結リスト

GLib では基本的なデータ型の他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 型を用いていますので, 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;
};
```

3.3.1 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);
```

- 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);
```

- 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_);
```

- 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);
```

- g_list_foreach

  リストのおのおのの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);
```

  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);
```

- 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);
```

- 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);
```

- 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)
```

- 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)
```

- g_list_nth
  n 番目のノードを返します.

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

- 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);
```

- 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);
```

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

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

3.3.2 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 3–3 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. 以下に実行結果を示します. 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 (14–16 **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 (19 **行目**)
関数 g_list_append を用いて, リストを作成します.

初期リストの表示 (20–23 **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 (25 **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 (26 **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全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 (29 **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 3–3 リストのソート : list_sort.c

```
1  #include <glib.h>
2
3  gint compare_function (gconstpointer a, gconstpointer b) {
4    return strcmp ((gchar *) a, (gchar *) b);
5  }
6
7  void print_data (gpointer data, gpointer user_data) {
8    g_print ("%s␣", (gchar *) data);
9  }
10
11 int main (int  argc,
12            char *argv[]) {
13   GList *list = NULL;
14   gchar *data[] = {"January", "February", "March", "April", "May", "June",
15                    "July", "August", "September", "October", "November",
16                    "December"};
17   int   n;
18
19   for (n = 0; n < 12; n++) list = g_list_append (list, (gpointer) data[n]);
20   for (n = 0; n < 12; n++) {
21     g_print ("%s␣", (gchar *) g_list_nth_data (list, n));
22   }
23   g_print ("\n");
24
25   list = g_list_sort (list, compare_function);
26   g_list_foreach (list, (GFunc) print_data, NULL);
27   g_print ("\n");
28
29   g_list_free (list);
30
31   return 0;
32 }
```

3.3.3 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 3–4). 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 (8–10 **行目**)

始めに, 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 (16 **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 (22–23 **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ますので, 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 (29–30 **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ましたので, 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 3–4 リストの追加・削除 : list_operation.c

```
1  #include <glib.h>
2
3  int main (int argc, char *argv[]) {
4    GList *list = NULL, *work;
5    gchar *data[] = {"January", "February", "April"};
6    int   n;
7
8    for (n = 0; n < 3; n++) {
9      list = g_list_append (list, (gpointer) g_strdup (data[n]));
10   }
11   for (work = list; work; work = g_list_next (work)) {
12     g_print ("%s␣", (gchar *) work->data);
```

```
13    }
14    g_print ("\n");
15
16    list = g_list_insert (list, (gpointer) g_strdup ("March"), 2);
17    for (work = list; work; work = g_list_next (work)) {
18      g_print ("%s␣", (gchar *) work->data);
19    }
20    g_print ("\n");
21
22    g_free (g_list_nth_data (list, 3));
23    list = g_list_delete_link (list, g_list_nth (list, 3));
24    for (work = list; work; work = g_list_next (work)) {
25      g_print ("%s␣", (gchar *) work->data);
26    }
27    g_print ("\n");
28
29    g_list_foreach (list, (GFunc) g_free, NULL);
30    g_list_free (list);
31
32    return 0;
33 }
```

3.4 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_ からなる関数によってハッシュテーブルを操作します.

3.4.1 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, 二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);
```

- 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);
```

- 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);
```

- 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);
```

- 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);
```

- 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, まず最初にそれらを解放する必要があります.

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

3.4.2 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 3–3). 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 `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 (8–10 **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ますので, GDestroyNotify 関数として g_free を指定しています.

値の登録 (11–13 **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 (15–19 **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 (20 **行目**)

使用し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りますので，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 3–5 ハッシュテーブルの操作：hash-sample.c

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

3.5 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);
```

インターバルの値は 1/1000 秒単位で指定します．従って 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ループ 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ループ ID を指定します.

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

ソース 3–6 にイベントループを利用したタイマープログラムを紹介します. 関数 g_timeout_add によって 1 秒ごとに関数 count_down を呼び出し, 変数 count の値が 0 になったときにプログラムを終了します.

ソース 3–6 タイマープログラム：timer-sample.c

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

図 3.2　タイマーサンプル

```
34    timer_id = g_timeout_add (1000, (GSourceFunc) count_down, label);
35
36    gtk_widget_show_all (window);
37    gtk_main ();
38
39    return 0;
40 }
```

第 4 章

Gdk による図形の描画

4.1 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.

4.2 グラフィックコンテキスト

4.2.1 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);
```

4.2.2 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 4–1 に示します．

ソース 4–1 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);
```

4.3 図形の描画

4.3.1 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;
};
```

4.3.2 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 4.1 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.

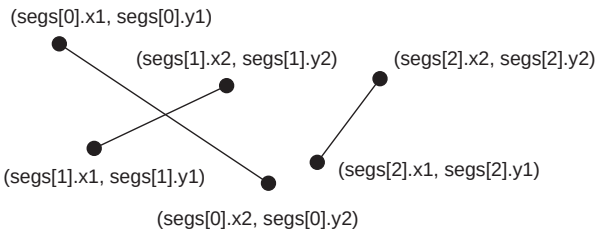


図 4.1　複数セグメントの描画

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 4.1 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

表 4.1　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)

(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 4.2 に示した 4 種類があります. 描画される線端は図 4.2 のようになります.

表 4.2　線端の種類

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.

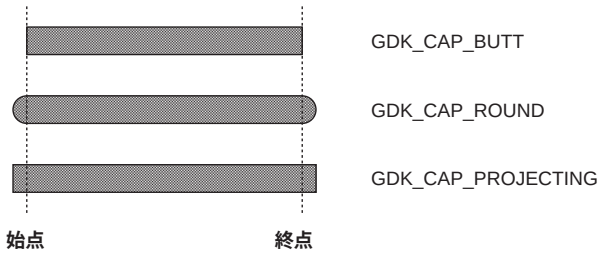


図 4.2　線端の描画

- 接続の種類

  線分の接続部分の形状には, 表 4.3 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 4.3 のようになります.

表 4.3　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.

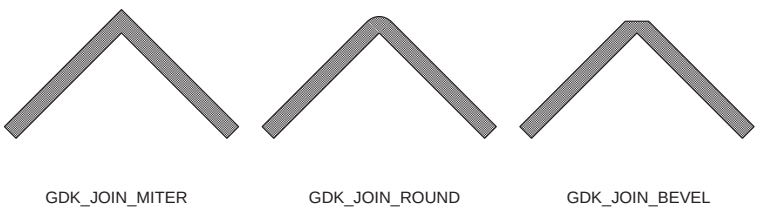


図 4.3　接続部分の描画

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 4–2 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.

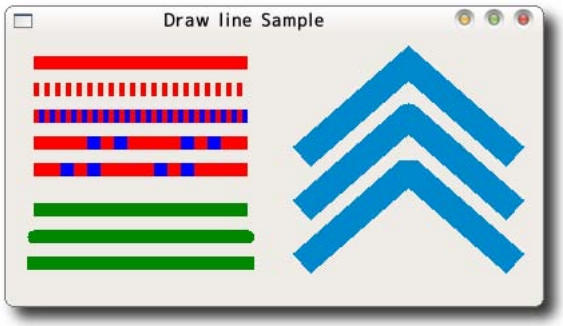


図 4.4　線分描画の例

ソース 4–2 線分の描画 : draw_lines.c

```
1  #include <gtk/gtk.h>
2
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 }
```

4.3.3 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 4.5 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.

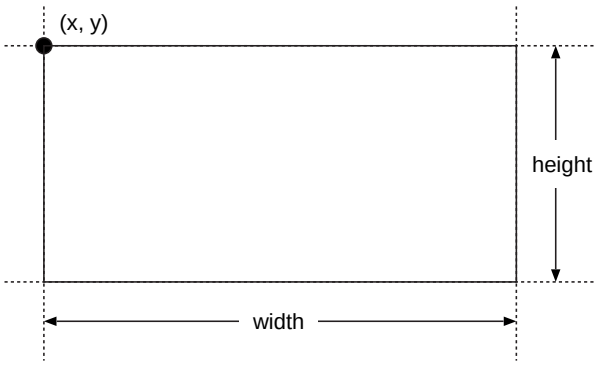


図 4.5 矩形の描画

4.3.4 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 4.6). 引数 filled を TRUE にすると, 円弧内を前景色で塗りつぶします.

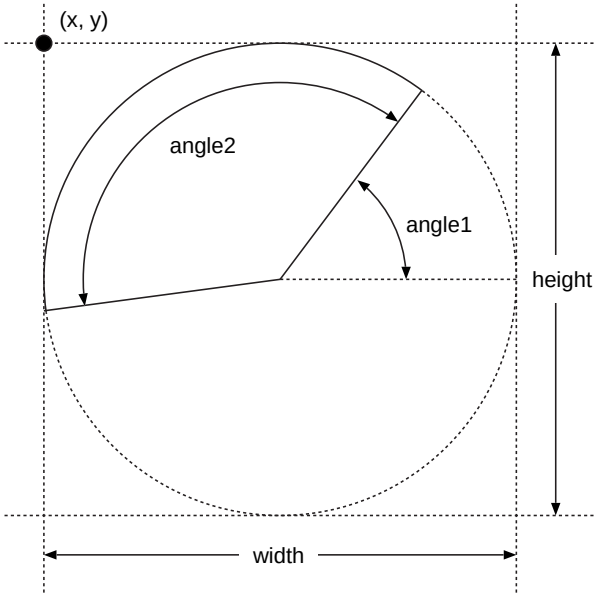


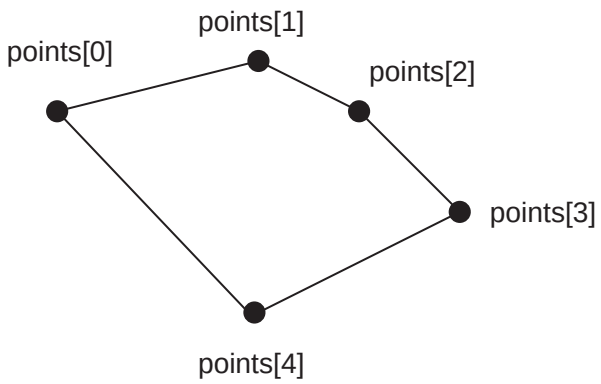
図 4.6　円弧の描画

4.3.5　多角形の描画

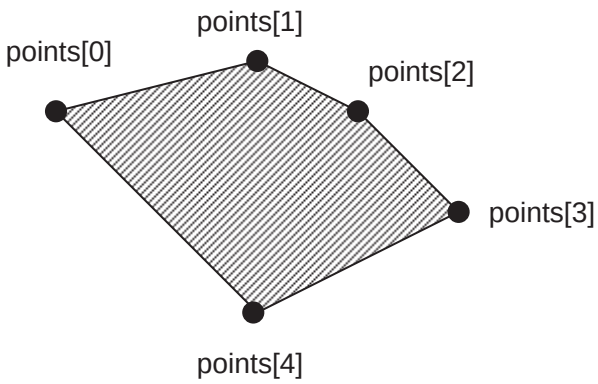
多角形の描画を行うには, 関数 gdk_draw_polygon を使用します.

```
void gdk_draw_polygon (GdkDrawable *drawable,
                       GdkGC       *gc,
                       gboolean    filled,
                       GdkPoint    *points,
                       gint        npoints);
```

図 4.7 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.



(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 4.7　多角形の描画

4.4　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 4–3 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．

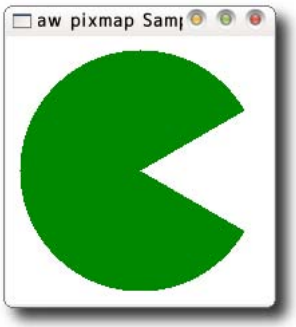


図 4.8　ピックスマップの描画

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．

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 4–3 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 }
```

第 5 章

GdkPixbuf

GdkPixbuf は画像ファイルの読み書きなど, 画像データの扱いを担当するライブラリです. 以前は独立したパッケージとしてメンテナンスされていましたが, 現在では GTK+ のパッケージに取り込まれ, GTK+ との親和性が高くなっています. この章では GdkPixbuf が提供する機能のうち, 以下の項目について解説します.

- ファイルの読み書き
- 画像情報の取得
- 画像の描画

そして, 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[*1].

```
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;
```

[*1] これはバージョン 2.8.20 現在の情報です.

表 5.1 GdkPixbuf で扱える画像フォーマット

画像フォマット	読み込み	書き込み
jpeg	○	○
png	○	○
ico	○	○
ani	○	×
bmp	○	×
gif	○	×
pnm	○	×
ras	○	×
tga	○	×
tiff	○	×
xbm	○	×
xpm	○	×

ファイルの読み込み等でエラーが発生した場合は, 関数の戻り値は NULL となり, GError 構造体の変数 error にはエラー情報が格納されます. 以下は読み込み許可のないファイルと存在しないファイルを引数に与えた場合のエラーコードとエラーメッセージの例です.

```
% ./read_image cannotread.png
error->code : 2
error->message :
 ファイル 'cannotread.png' のオープンに失敗しました: 許可がありません

% ./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他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 は次のように定義されていますので, 関数の引数には GDK_COLORSPACE_RGB を

与えます.

```
typedef enum {
  GDK_COLORSPACE_RGB
} GdkColorspace;
```

また, has_alpha は アルファプレーンが必要な場合は TRUE を, そうでない場合は FALSE を与えます. bits_per_sample は現在では 8 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);
```

- 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 を指定すると全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することができます.

```
% 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 オプションで指定することができます．

```
/* 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..."
  ...
```

✣ コラム〜rowstride の値

rowstride の値は画像の 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 (アルファチャンネルがない場合です)．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することができます．

$$rowstride = 3W + (4 - 3W\%4)$$

ここで $a\%b$ は a を b で割った余りを表すものとします．

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キー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することができます. これから情報は, 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.

表 5.2 GdkPixbuf で扱える画像フォーマット

画像フォーマット	キー	キーの値
jpeg	quality	[0:100]
png	compression	[0:9]
ico	depth	16, 24, 32

```
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);
```

✣ コラム〜TEO 画像からの GdkPixbuf データの生成

GdkPixbuf は TEO ファイルの読み書きに対応していません. TEO 画像を GTK+ のアプリケーションで利用したい場合はどうしたらいいでしょう. ここでは一番簡単な方法を紹介します.
TEO 画像を C 言語のプログラム中で扱う場合, 画像データに対して TEOIMAGE 構造体の変数を通じてアクセスします. libteo のマクロ TeoData を使うと直接画像データ (unsigned char 型の一次元配列) を取得することができます. これを利用して, 関数 gdk_pixbuf_new_from_data により TEO 画像から GdkPixbuf データを作成します.

```
TEOIMAGE  *img;
GdkPixbuf *pixbuf;
pixbuf =
  gdk_pixbuf_new_from_data ((guchar *) TeoData(img),
                            GDK_COLORSPACE_RGB,
                            FALSE, 8,
                            TeoWidth(img), TeoHeight(img),
                            TeoWidth(img) * 3,
                            NULL,
                            NULL);
```

rowstride には画像 1 行分のバイト数, すなわち画像の幅 × チャネル数を与えます. GdkPixbuf では現在, 1 画素当たりのデータ量が 8 ビットでチャネル数 3, もしくは 4(4 番目のチャネルはアルファチャネルを表す) のデータしかサポートしていませんので, この方法で使用できる TEO 画像は TEO_UINT8 型の RGB 画像のみになります.

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

一番簡単な方法は, 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
3  int main (int argc, char *argv[]) {
4    GtkWidget *window;
5    GtkWidget *image;
6
7    gtk_init (&argc, &argv);
8
9    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
10   gtk_window_set_title (GTK_WINDOW(window), "Display␣Image␣1");
11   image = gtk_image_new_from_file (argv[1]);
12   gtk_container_add (GTK_CONTAINER(window), image);
13   gtk_widget_show_all (window);
14   gtk_main ();
15
16   return 0;
17 }
```

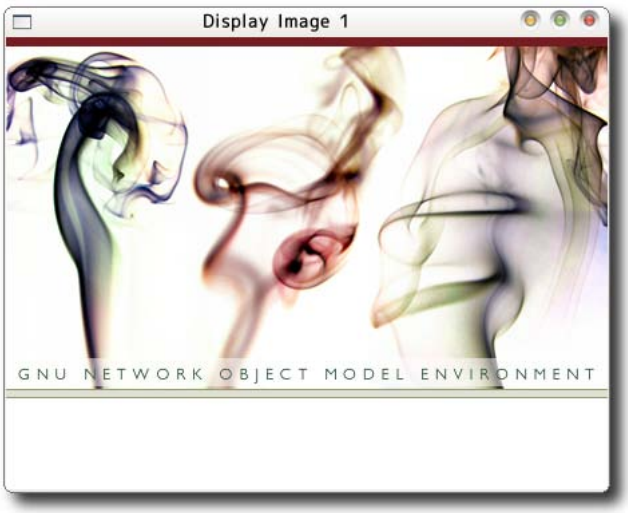


図 5.1 GdkImage ウィジェットによる画像の表示

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ることが可能です. 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することができます.

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

GdkPixbuf や GdkPixmap から GtkImage ウィジェットを作成することも可能です.

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

逆に 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, 図形の表示で扱った GtkDrawingArea ウィジェットを利用することもできます. GtkImage ウィジェットに比べて扱いが面倒ですが, 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 に示します.

画像の読み込み (18 **行目**)

ウィンドウに表示するための画像を関数 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** (25–28 **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する関数です.

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

コールバック関数の設定 (29–30 **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.

画像の描画 (6–7 **行目**)

関数 gdk_draw_pixbuf を使って, 画像データを描画します. 第 1 引数 drawable には画像データを描画するドローアブルを指定します. ソース 5–2 では, GtkDrawingArea の window メンバをドローアブルとして指定しています. 第 2 引数の gc は特に指定せずに, NULL を与えておいても構いません.

第 3, 4 引数はドローアブルのどの座標から描画を開始するかを指定します. そして, 第 5 引数から第 8 引数で, 描画する画像の範囲 (開始座標と幅, 高さ) を指定します. width, height に −1 を指定すると, 自動的に画像全体の幅と高さとなります.

```
void gdk_draw_pixbuf (GdkDrawable  *drawable,
                      GdkGC        *gc,
```

```
                                GdkPixbuf     *pixbuf,
                                gint          src_x,
                                gint          src_y,
                                gint          dest_x,
                                gint          dest_y,
                                gint          width,
                                gint          height,
                                GdkRgbDither  dither,
                                gint          x_dither,
                                gint          y_dither);
```

GdkRgbDither は次のように定義されています.

```
typedef enum
{
  GDK_RGB_DITHER_NONE,
  GDK_RGB_DITHER_NORMAL,
  GDK_RGB_DITHER_MAX
} GdkRgbDither;
```

ソース 5–2 GtkDrawingArea ウィジェットによる画像の表示 1：display2.c

```
1  #include <gtk/gtk.h>
2
3  static gint cb_expose (GtkWidget        *widget,
4                         GdkEventExpose   *event,
5                         gpointer         data) {
6    gdk_draw_pixbuf (widget->window, NULL, (GdkPixbuf *) data,
7                     0, 0, 0, 0, -1, -1, GDK_RGB_DITHER_NONE, 0, 0);
8    return TRUE;
9  }
10
11 int main (int argc, char *argv[]) {
12   GtkWidget     *window;
13   GtkWidget     *canvas;
14   GdkPixbuf     *pixbuf;
```

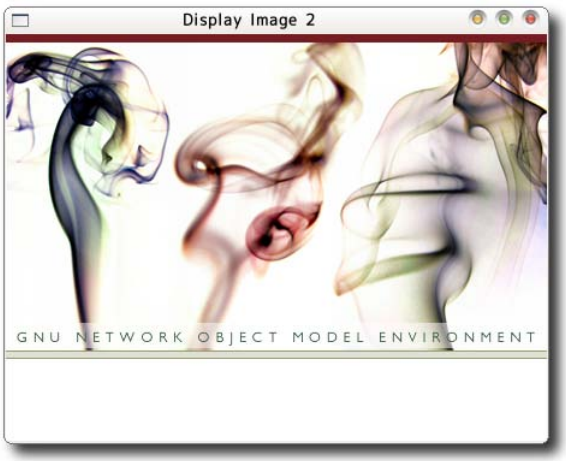


図 5.2 GtkDrawingArea ウィジェットによる画像の表示 (その 1)

```
15
16   gtk_init (&argc, &argv);
17
18   pixbuf = gdk_pixbuf_new_from_file (argv[1], NULL);
19
20   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
21   gtk_window_set_title (GTK_WINDOW(window), "Display␣Image␣2");
22
23   canvas = gtk_drawing_area_new ();
24   gtk_widget_set_size_request (canvas,
25                                gdk_pixbuf_get_width (pixbuf),
26                                gdk_pixbuf_get_height(pixbuf));
27   g_signal_connect (G_OBJECT(canvas), "expose_event",
28                     G_CALLBACK(cb_expose), pixbuf);
29   gtk_container_add (GTK_CONTAINER(window), canvas);
30
31   gtk_widget_show_all (window);
32   gtk_main ();
33
34   return 0;
35 }
```

GtkDrawingArea ウィジェットを使う方法として, GdkPixbuf データからピックスマップ (GdkPixmap) とビットマップ (GdkBitmap) を作成して, 関数 gdk_window_set_back_pixmap で画像データを GdkWindow の背景として設定する方法があります. この方法を使って一旦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. GdkPixbuf データからピックスマップとビットマップを生成するには, 関数 gdk_pixbuf_render_pixmap_and_mask を使います.

ソース 5–3 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 (0–255)
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 の巾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

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 は次のように定義されています.

```
typedef enum {
  GDK_INTERP_NEAREST,
  GDK_INTERP_TILES,
  GDK_INTERP_BILINEAR,
  GDK_INTERP_HYPER
} GdkInterpType;
```

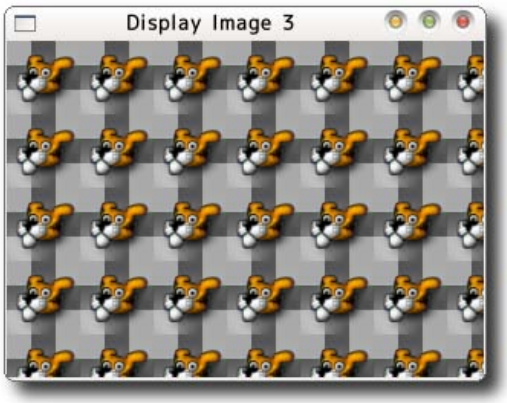


図 5.3 GtkDrawingArea ウィジェットによる画像の表示 (その 2)

ソース 5–3 GtkDrawingArea ウィジェットによる画像の表示 2 : display3.c

```
1  #include <gtk/gtk.h>
2
3  static gint cb_expose (GtkWidget        *widget,
4                         GdkEventExpose   *event,
5                         gpointer         data) {
6    GdkPixbuf     *pixbuf = (GdkPixbuf *) data;
7    GdkPixbuf     *background;
8    GdkPixmap     *pixmap;
9    int           w, h;
10
11   w = gdk_pixbuf_get_width (pixbuf);
12   h = gdk_pixbuf_get_height(pixbuf);
13   background = gdk_pixbuf_new (GDK_COLORSPACE_RGB, FALSE, 8, w, h);
14   gdk_pixbuf_composite_color (pixbuf, background,
15                               0, 0, w, h, 0, 0, 1.0, 1.0,
16                               GDK_INTERP_BILINEAR, 255, 0, 0, 16,
17                               0xaaaaaa, 0x555555);
18   gdk_pixbuf_render_pixmap_and_mask (background, &pixmap, NULL, 255);
19   gdk_window_set_back_pixmap (widget->window, pixmap, FALSE);
20   gdk_window_clear(widget->window);
21
22   g_object_unref (background);
23   g_object_unref (pixmap);
24
25   return TRUE;
26 }
27
28 int main (int argc, char *argv[]) {
29   GtkWidget     *window;
30   GtkWidget     *canvas;
31   GdkPixbuf     *pixbuf;
32
33   gtk_init (&argc, &argv);
34
35   pixbuf = gdk_pixbuf_new_from_file (argv[1], NULL);
36
37   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
38   gtk_window_set_title (GTK_WINDOW(window), "Display␣Image␣3");
39
40   canvas = gtk_drawing_area_new ();
41   gtk_widget_set_size_request (canvas,
42                                gdk_pixbuf_get_width (pixbuf),
43                                gdk_pixbuf_get_height(pixbuf));
44   g_signal_connect (G_OBJECT(canvas), "expose_event",
45                     G_CALLBACK(cb_expose), pixbuf);
46   gtk_container_add (GTK_CONTAINER(window), canvas);
47
```

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

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 5.4 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.

このアプリケーションの中で次のウィジェットが新しく登場します. これらのウィジェットの使い方についても紹介します.

- スクロール機能付きウィンドウ (GtkScrolledWindow)
- アラインメントウィジェット (GtkAlignment)
- スピンボタンウィジェット (GtkSpinButton)

5.4.1 ウィジェットの配置

図 5.4 のアプリケーションのウィジェット配置を図 5.5 に示します. ここでのポイントは 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, ウィンドウの大きさが小さすぎたり, 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. ウィンドウの大きさよりも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

図 5.4 画像をしきい値で 2 値化するアプリケーション

ウ内に納まらない部分も表示することができます．では，画像の大きさがウィンドウの大きさよりも小さい場合はどうでしょう．図 5.3 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方が大きいと，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，GtkAlignment ウィジェットを使用します．

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．

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

xalign, 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, 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

図 5.5 に示すようにそれぞれのウィジェットを配置して GUI を作成します．

ベースウィジェットの作成 (17–23 **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 (25–31 **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

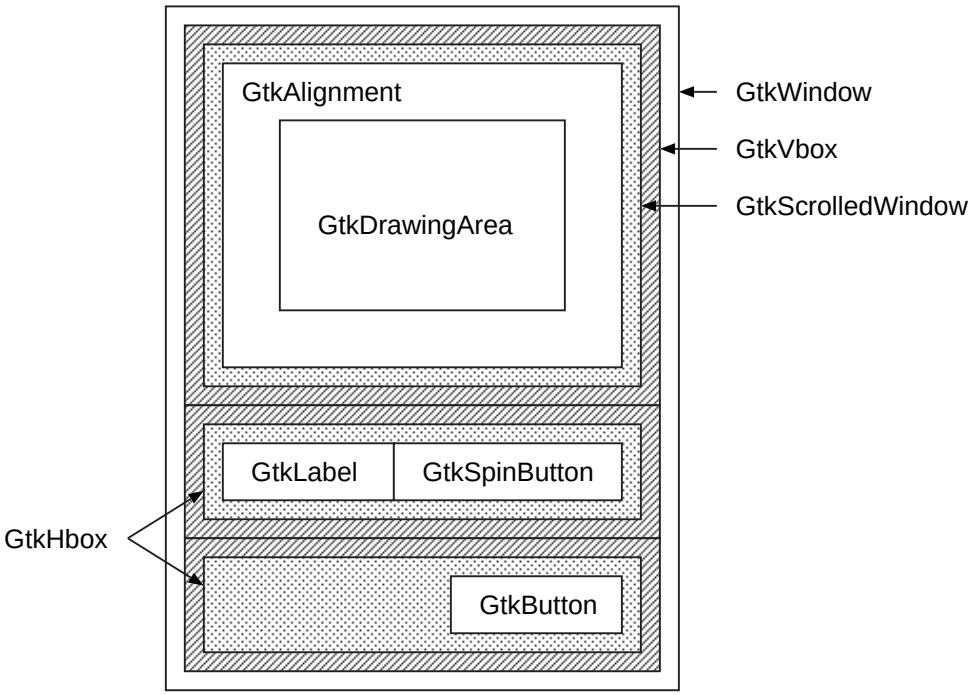


図 5.5 画像をしきい値で 2 値化するアプリケーション

表 5.3 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.

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 5.3 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);
```

GtkScrolledWindow の設定として, ウィンドウ内の装飾の設定があります.
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 は次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 5.6 を参考にしてください.

```
typedef enum
{
  GTK_SHADOW_NONE,
  GTK_SHADOW_IN,
  GTK_SHADOW_OUT,
  GTK_SHADOW_ETCHED_IN,
  GTK_SHADOW_ETCHED_OUT
} GtkShadowType;
```

アラインメントウィジェットの作成 (33–34 **行目**)

GtkAlignment ウィジェットは関数 gtk_alignment_new で作成します. 今回は, この中に GtkDrawingArea ウィジェットを中央に配置したいので, それぞれの引数を次のように指定しています.

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

スピンボタンの作成 (53–57 **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. 56–57 行目では, この後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. スピンボタンの詳細については 6.6.3 節スピンボタンを参照してください.

```
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);
```

ソース 5–4–1 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* make_interface (const gchar    *title,
5                                    GdkPixbuf     *pixbuf) {
6    GtkWidget *window;
7    GtkWidget *vbox;
8    GtkWidget *scrolledwindow;
9    GtkWidget *alignment;
10   GtkWidget *drawingarea;
11   GtkWidget *hbox;
12   GtkWidget *label;
13   GtkObject *adjustment;
14   GtkWidget *spinbutton;
15   GtkWidget *button;
16
17   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
18   gtk_window_set_title (GTK_WINDOW(window), title);
```

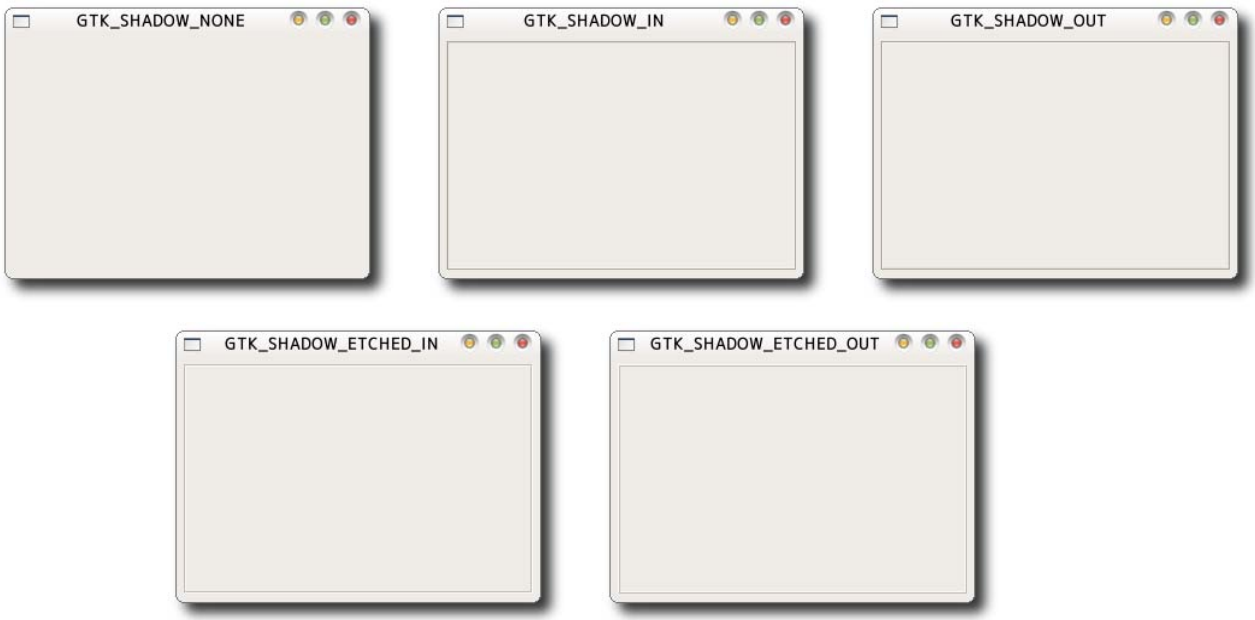


図 5.6 ウィンドウ装飾の種類

```
19   gtk_widget_set_size_request (window, WINDOW_WIDTH, WINDOW_HEIGHT);
20
21   vbox = gtk_vbox_new (FALSE, 3);
22   gtk_container_add (GTK_CONTAINER(window), vbox);
23   gtk_container_set_border_width (GTK_CONTAINER(vbox), 5);
24
25   scrolledwindow = gtk_scrolled_window_new (NULL, NULL);
26   gtk_box_pack_start (GTK_BOX(vbox), scrolledwindow, TRUE, TRUE, 0);
27   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW(scrolledwindow),
28                                   GTK_POLICY_AUTOMATIC,
29                                   GTK_POLICY_AUTOMATIC);
30   gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW(scrolledwindow),
31                                        GTK_SHADOW_ETCHED_IN);
32   {
33     alignment = gtk_alignment_new (0.5, 0.5, 0, 0);
34     gtk_scrolled_window_add_with_viewport(GTK_SCROLLED_WINDOW
35                                           (scrolledwindow), alignment);
36     {
37       drawingarea = gtk_drawing_area_new ();
38       gtk_container_add (GTK_CONTAINER(alignment), drawingarea);
39       gtk_widget_set_size_request (drawingarea,
40                                    gdk_pixbuf_get_width (pixbuf),
41                                    gdk_pixbuf_get_height (pixbuf));
42       g_signal_connect (G_OBJECT(drawingarea),
43                         "expose_event", G_CALLBACK(cb_expose),
44                         (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     label = gtk_label_new ("Threshold␣:");
51     gtk_box_pack_start (GTK_BOX(hbox), label, FALSE, FALSE, 0);
52
53     adjustment = gtk_adjustment_new (128, 0, 255, 1, 5, 0);
54     spinbutton = gtk_spin_button_new (GTK_ADJUSTMENT(adjustment), 1, 0);
55     gtk_box_pack_start (GTK_BOX(hbox), spinbutton, TRUE, TRUE, 0);
56     g_signal_connect (G_OBJECT(spinbutton), "value_changed",
57                       G_CALLBACK (cb_value_changed), drawingarea);
58   }
59   hbox = gtk_hbox_new (FALSE, 0);
60   gtk_box_pack_start (GTK_BOX(vbox), hbox, FALSE, FALSE, 0);
61   {
62     button = gtk_button_new_from_stock ("gtk-quit");
63     gtk_box_pack_end (GTK_BOX(hbox), button, FALSE, FALSE, 0);
64     g_signal_connect (G_OBJECT(button),
65                       "clicked", G_CALLBACK(cb_quit), (gpointer) pixbuf);
66   }
67   return window;
68 }
```

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するシグナルです. 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 (15 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 (17–29 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, 5.3.2 節で紹介した通りです.

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, 関数 gtk_spin_button_get_value がありますが, 今回は 0 から 255 までの整数しか扱いませんので, 前者の関数を使います.

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 5–4–2 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 cb_expose (GtkWidget        *widget,
8                         GdkEventExpose   *event,
9                         gpointer         data) {
10   GdkPixbuf     *pixbuf = (GdkPixbuf *) data;
```

```
11   GdkPixbuf      *background;
12   GdkPixmap      *pixmap;
13   int            w, h;
14
15   create_binarized_image (pixbuf, bin, threshold);
16
17   w = gdk_pixbuf_get_width (pixbuf);
18   h = gdk_pixbuf_get_height(pixbuf);
19   background = gdk_pixbuf_new (GDK_COLORSPACE_RGB, FALSE, 8, w, h);
20   gdk_pixbuf_composite_color (bin, background,
21                               0, 0, w, h, 0, 0, 1.0, 1.0,
22                               GDK_INTERP_BILINEAR, 255, 0, 0, 16,
23                               0xaaaaaa, 0x555555);
24   gdk_pixbuf_render_pixmap_and_mask (background, &pixmap, NULL, 255);
25   gdk_window_set_back_pixmap (widget->window, pixmap, FALSE);
26   gdk_window_clear(widget->window);
27
28   g_object_unref (background);
29   g_object_unref (pixmap);
30
31   return TRUE;
32 }
33
34 static void cb_value_changed (GtkSpinButton      *spinbutton,
35                               gpointer           data) {
36   threshold=gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(spinbutton));
37   gtk_widget_queue_draw (GTK_WIDGET(data));
38 }
```

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 5–4–3 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．

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

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関数として，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 で計算することができます．

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

ソース 5–4–3 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 5–4–4 の operation.c の create_binarized_image 中で行っています. 12–14 行目で点 (x, y) の RGB 値を取得して, 15 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 5–4–4 2 値化関数：operation.c

```
1 #include <gtk/gtk.h>
2 #include "operation.h"
3
4 void create_binarized_image (const GdkPixbuf     *src,
5                              GdkPixbuf           *dst,
6                              gint                threshold) {
7   int           row, col;
8   guchar        r, g, b, y, val;
9
10  for (row = 0; row < gdk_pixbuf_get_height (src); row++) {
11    for (col = 0; col < gdk_pixbuf_get_width (src); col++) {
12      r = gdk_pixbuf_get_pixel(src, col, row, 0);
13      g = gdk_pixbuf_get_pixel(src, col, row, 1);
14      b = gdk_pixbuf_get_pixel(src, col, row, 2);
15      y = (guchar) (0.299 * r + 0.587 * g + 0.114 * b);
16      val = (y >= threshold) ? 255 : 0;
17      gdk_pixbuf_put_pixel(dst, col, row, 0, val);
18      gdk_pixbuf_put_pixel(dst, col, row, 1, val);
19      gdk_pixbuf_put_pixel(dst, col, row, 2, val);
20    }
21  }
22 }
```

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 5–4–5 のメイン関数を合わせて, 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. これまでのように一つ一つコマンドラインで gcc コマンドを実行していると煩わしいので, ここでは Makefile を使ってソースコード

を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 5–4–6 に示します．

作成したアプリケーションの実行例を図 5.7 に示します．図 5.7 の左側が入力画像です．これを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 5.7 の右側になります．

```
% make ↵
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 ↵
```

ソース 5–4–5 メイン関数：binarize.c から一部抜粋

```
1  int main (int argc, char *argv[]) {
2    GtkWidget      *window;
3    GdkPixbuf      *pixbuf;
4
5    gtk_init (&argc, &argv);
6
7    pixbuf = gdk_pixbuf_new_from_file (argv[1], NULL);
8    bin    = gdk_pixbuf_copy (pixbuf);
9    window = make_interface ("Image␣Binarization", pixbuf);
10   gtk_widget_show_all (window);
11   gtk_main ();
12
13   return 0;
14 }
```

図 5.7　入力画像とアプリケーションの実行結果

ソース 5–4–6 Makefile

```
1  CC              = gcc -O2
2
3  INSTALL         = /usr/bin/install
4  PREFIX          = $(HOME)
5  PKG_CONFIG      = /usr/bin/pkg-config
6
7  CFLAGS          = -Wall `$(PKG_CONFIG) --cflags gtk+-2.0`
8  LDFLAGS         = `$(PKG_CONFIG) --libs gtk+-2.0`
9  LIBS            = `$(PKG_CONFIG) --libs gtk+-2.0`
10
11 SRCS            = binarize.c operation.c
12 HDRS            = operation.h
13 OBJS            = $(SRCS:.c=.o)
14
15 DEST            = $(PREFIX)/bin
16
17 PROGRAM         = binarize_gui
18
19 all:            $(PROGRAM)
20
21 $(PROGRAM):     $(OBJS) $(HDRS)
22                 $(CC) $(OBJS) $(LDFLAGS) $(LIBS) -o $(PROGRAM)
23
24 clean:;         rm -f *.o *~ $(PROGRAM)
25
26 install:        $(PROGRAM)
27                 $(INSTALL) -s $(PROGRAM) $(DEST)
28                 strip $(DEST)/$(PROGRAM)
```

第 6 章

ウィジェットリファレンス

この章では様々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.

6.1 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, ある動作のトリガー役をする普通のボタン, ある項目のオン, オフを設定するためのチェックボタン, 複数の項目の中から選択を行うラジオボタンの 3 つのボタンが存在します.

6.1.1 普通のボタン

オブジェクトの階層構造

```
GObject
   +----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 6.1 普通のボタン

- 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 6.1 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 6.1 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 6.1 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 とか 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 6.1 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用さ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.

表 6.1　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です.
”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です.
”pressed”	ボタンが押されたときに発生するシグナルです.
”release”	ボタンが離されたときに発生するシグナルです.
”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です.

```
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 6–1–1 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 6.2 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 6.2 右のように更新されます.

図 6.2 ボタンウィジェットのサンプルプログラム

ソース 6–1–1 ボタンウィジェットのサンプルプログラム：gtkbutton-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void cb_button_clicked (GtkButton *widget, gpointer data) {
4    static int     count = 0;
5    char           buf[1024];
6
7    sprintf (buf, "%d␣time(s)␣clicked.", ++count);
8    gtk_button_set_label (widget, buf);
9  }
10
11 int main (int argc, char **argv) {
12   GtkWidget      *window;
13   GtkWidget      *box;
14   GtkWidget      *button;
15
16   gtk_init (&argc, &argv);
17
18   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
19   gtk_window_set_title  (GTK_WINDOW(window), "GtkButton␣Sample");
20   gtk_widget_set_size_request (window, 240, -1);
21   g_signal_connect (G_OBJECT(window), "destroy",
22                     G_CALLBACK (gtk_main_quit), NULL);
23
24   box = gtk_vbox_new (TRUE, 0);
25   gtk_container_add (GTK_CONTAINER(window), box);
26
27   button = gtk_button_new_with_label ("Please␣click␣me.");
28   gtk_box_pack_start (GTK_BOX(box), button, TRUE, TRUE, 0);
```

```
29   g_signal_connect (G_OBJECT(button), "clicked",
30                     G_CALLBACK (cb_button_clicked), NULL);
31
32   gtk_widget_show_all (window);
33   gtk_main ();
34
35   return 0;
36 }
```

6.1.2 チェックボタン

オブジェクトの階層構造

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

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 6.3 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 6.3 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 6.3 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);
  ```

図 6.3　チェックボタン

表 6.2 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です.

シグナルとコールバック関数

表 6.2 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で調べます. 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) することができます.

```
void gtk_toggle_button_toggled (GtkToggleButton *toggle_button);
```

サンプルプログラム

チェックボタンウィジェットのサンプルプログラムをソース 6–1–2 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.

図 6.4 チェックボタンウィジェットのサンプルプログラム

ソース 6–1–2 チェックボタンウィジェットのサンプルプログラム：gtkcheckbutton-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void cb_button_toggled (GtkToggleButton *widget, gpointer data) {
4    gboolean       active;
5    gchar          *str[] = {"FALSE", "TRUE"};
6
7    active = gtk_toggle_button_get_active (widget);
8    g_print ("Check␣button␣state␣:␣%s\n", str[active]);
9  }
10
11 int main (int argc, char **argv) {
12   GtkWidget      *window;
13   GtkWidget      *box;
14   GtkWidget      *button;
15
16   gtk_init (&argc, &argv);
17
18   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
19   gtk_window_set_title  (GTK_WINDOW(window), "GtkCheckButton␣Sample");
20   gtk_widget_set_size_request (window, 300, -1);
21   g_signal_connect (G_OBJECT(window), "destroy",
22                     G_CALLBACK(gtk_main_quit), NULL);
23
24   box = gtk_vbox_new (TRUE, 0);
25   gtk_container_add (GTK_CONTAINER(window), box);
26
27   button = gtk_check_button_new_with_label ("Please␣clicke␣me.");
28   gtk_box_pack_start (GTK_BOX(box), button, TRUE, TRUE, 0);
29   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON(button), TRUE);
30   g_signal_connect (G_OBJECT(button), "toggled",
31                     G_CALLBACK (cb_button_toggled), NULL);
32
33   gtk_widget_show_all (window);
34   gtk_main ();
35
36   return 0;
37 }
```

6.1.3 ラジオボタン

オブジェクトの階層構造

```
GObject
   +----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 6.5 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 6.5 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 6.5 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,
```

図 6.5 ラジオボタン

```
                                                const gchar     *label);
```

- gtk_radio_button_new_with_mnemonic_from_widget

```
GtkWidget*
gtk_radio_button_new_with_mnemonic_from_widget(GtkRadioButton *group,
                                               const gchar    *label);
```

ラジオボタンを作成する例をソース 6–1–3 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て関数 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 6–1–3 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 6.3 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．

表 6.3　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です．
”toggled”	ボタンの状態が変化したときに発生するシグナルです．

```
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 6–1–4 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．

ソース 6–1–4 ラジオボタンウィジェットのサンプルプログラム：gtkradiobutton-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void cb_button_toggled (GtkRadioButton *widget, gpointer data) {
4    g_print ("Catch␣the␣toggled␣signal␣from␣the␣radio␣button%d.\n",
5             (gint) data);
6    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON(widget))) {
7      g_print ("The␣radio␣button%d␣is␣selected.\n", (gint) data);
8    }
9  }
10
11 int main (int argc, char **argv) {
12   GtkWidget      *window;
13   GtkWidget      *box;
14   GtkWidget      *button[3];
15   GSList         *group = NULL;
16
17   gtk_init (&argc, &argv);
18
19   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
20   gtk_window_set_title  (GTK_WINDOW(window), "GtkRadioButton␣Sample");
21   gtk_widget_set_size_request (window, 300, -1);
22   g_signal_connect (G_OBJECT(window), "destroy",
23                     G_CALLBACK (gtk_main_quit), NULL);
24
25   box = gtk_vbox_new (TRUE, 0);
26   gtk_container_add (GTK_CONTAINER(window), box);
27
28   button[0] = gtk_radio_button_new_with_label (group, "Red");
29   gtk_box_pack_start (GTK_BOX(box), button[0], TRUE, TRUE, 0);
30
```

図 6.6 ラジオボタンウィジェットのサンプルプログラム

```
31   button[1] = gtk_radio_button_new_with_label_from_widget
32     (GTK_RADIO_BUTTON(button[0]), "Green");
33   gtk_box_pack_start (GTK_BOX(box), button[1], TRUE, TRUE, 0);
34
35   button[2] = gtk_radio_button_new_with_label_from_widget
36     (GTK_RADIO_BUTTON(button[0]), "Blue");
37   gtk_box_pack_start (GTK_BOX(box), button[2], TRUE, TRUE, 0);
38
39   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON(button[1]), TRUE);
40
41   g_signal_connect (G_OBJECT(button[0]), "toggled",
42                     G_CALLBACK(cb_button_toggled), (gpointer) 0);
43   g_signal_connect (G_OBJECT(button[1]), "toggled",
44                     G_CALLBACK(cb_button_toggled), (gpointer) 1);
45   g_signal_connect (G_OBJECT(button[2]), "toggled",
46                     G_CALLBACK(cb_button_toggled), (gpointer) 2);
47
48   gtk_widget_show_all (window);
49   gtk_main ();
50
51   return 0;
52 }
```

6.2　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.

6.2.1　ウィンドウ

ウィンドウウィジェット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
   +----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. 引数にはウィンドウタイプを GtkWindowType で指定します. 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);

typedef enum
{
  GTK_WINDOW_TOPLEVEL,
  GTK_WINDOW_POPUP
} GtkWindowType;
```

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. 関数 gtk_window_get_title と関数 gtk_window_set_title でタイトルの取得や設定を行います.

```
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 を使うとウィンドウサイズをユーザが変更できるかどうかを設定することができます. また, 関数 gtk_window_get_resizable を使うと現在の設定を取得することができます.

```
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したり, ウィンドウをアイコン化した際のアイコンです.

```
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することができます.

```
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
  ウィンドウサイズの指定は関数 gtk_widget_set_size_request 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_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, 関数 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一つの方法は関数 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);
```

6.2.2 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よくしてくれます. また, 枠に対して見出しもつけることができます. 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することができ, 見た目がよくなるだけではなく, 操作性も向上させることができます.

オブジェクトの階層構造

```
GObject
   +----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.

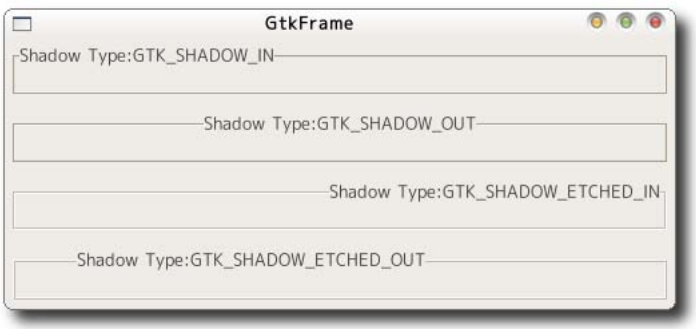


図 6.7 フレーム表示の例

ウィジェットのプロパティ設定

フレームウィジェットのプロパティには次の 3 つの項目が存在します.

- 見出しテキスト

  見出しテキストは次の関数を使って取得したり, 設定したりできます.

```
G_CONST_RETURN gchar* gtk_frame_get_label (GtkFrame *frame);

void gtk_frame_set_label (GtkFrame *frame, const gchar *label);
```

- 見出し位置

  見出しをフレームのどの位置に表示するかを設定します. xalign が 0 なら左, 1 なら右に表示します.

```
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 5.4.2 節で紹介した GtkShadowType で指定します. フレームの装飾の種類は図 6.7 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);
```

6.2.3　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ることができることです (図 6.8).

図 6.8　ペインによる領域の分割

オブジェクトの階層構造

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

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で作成することができます.

```
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_hpaned_new と関数 gtk_vpaned_new 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することができます. 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);
```

6.2.4 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用されるウィジェットです.

オブジェクトの階層構造

```
GObject
    +----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);
```

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方法は 3 種類あります.

- gtk_toolbar_append_item

  ボタンをツールバーに配置する関数です.

```
GtkWidget*
gtk_toolbar_append_item (GtkToolbar     *toolbar,
                         const char     *text,
                         const char     *tooltip_text,
                         const char     *tooltip_private_text,
                         GtkWidget      *icon,
                         GtkSignalFunc  callback,
                         gpointer       user_data);
```

  関数の引数は以下の通りです.

  第 1 引数：　ツールバーウィジェット
  第 2 引数：　ボタンラベル
  第 3 引数：　ツールチップテキスト
  第 4 引数：　ツールチップテキスト
  第 5 引数：　ボタン用のアイコンウィジェット
  第 6 引数：　ボタン用のコールバック関数
  第 7 引数：　コールバック関数に渡すデータ

  この関数の類似関数に関数 gtk_toolbar_preppend_item と関数 gtk_toolbar_insert_item があります.

- gtk_toolbar_append_element

  配置するウィジェットの種類を指定して追加する関数です.

```
GtkWidget*
gtk_toolbar_append_element(GtkToolbar          *toolbar,
                           GtkToolbarChildType type,
                           GtkWidget           *widget,
                           const char          *text,
                           const char          *tooltip_text,
```

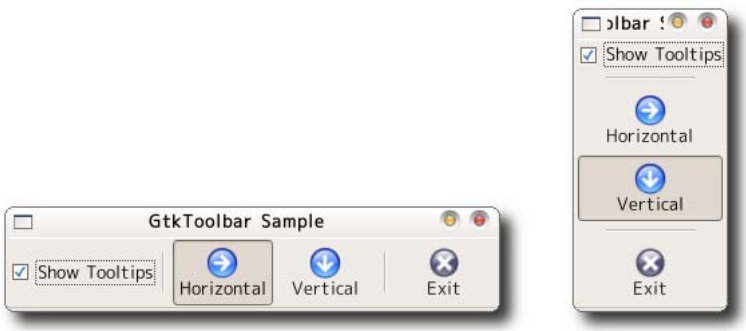


図 6.9　ツールバー

```
                                const char          *tooltip_private_text,
                                GtkWidget           *icon,
                                GtkSignalFunc       callback,
                                gpointer            user_data);
```

関数の引数は以下の通りです.

第 1 引数： ツールバーウィジェット
第 2 引数： ツールバーアイテムの種類
第 3 引数： 配置するウィジェット
第 4 引数： ラベル
第 5 引数： ツールチップテキスト
第 6 引数： ツールチップテキスト
第 7 引数： ボタン用のアイコンウィジェット
第 8 引数： ボタン用のコールバック関数
第 9 引数： コールバック関数に渡すデータ

ツールバーアイテムの種類は次の GtkToolbarChildType から選択します.

```
typedef enum
{
  GTK_TOOLBAR_CHILD_SPACE,
  GTK_TOOLBAR_CHILD_BUTTON,
  GTK_TOOLBAR_CHILD_TOGGLEBUTTON,
  GTK_TOOLBAR_CHILD_RADIOBUTTON,
  GTK_TOOLBAR_CHILD_WIDGET
} GtkToolbarChildType;
```

この関数の類似関数に関数 gtk_toolbar_preppend_element と関数 gtk_toolbar_insert_element があります.

- gtk_toolbar_append_widget
  配置するウィジェットを独立に作成し, そのウィジェットをツールバーに配置する関数です.

```
void gtk_toolbar_append_widget (GtkToolbar *toolbar,
                                GtkWidget  *widget,
                                const char *tooltip_text,
                                const char *tooltip_private_text);
```

関数の引数は以下の通りです. ウィジェットに対するコールバック関数は関数 g_signal_connect 等を使って設定する必要があります.

第 1 引数： ツールバーウィジェット
第 2 引数： 配置するウィジェット
第 3 引数： ツールチップテキスト
第 4 引数： ツールチップテキスト

この関数の類似関数に関数 gtk_toolbar_preppend_widget と関数 gtk_toolbar_insert_widget があります.

シグナルとコールバック関数

表 6.4 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;
```

表 6.4　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です.
”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です.
”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です.

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_toolbar_set_orientation (GtkToolbar      *toolbar,
                                  GtkOrientation orientation);
GtkOrientation gtk_toolbar_get_orientation (GtkToolbar *toolbar);
```

- ツールチップの表示の有無
  ツールバーに配置された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をツールチップと呼びます. ツールチップを表示するかどうか, また現在の設定を次の関数で調べることができます.

```
void gtk_toolbar_set_tooltips (GtkToolbar *toolbar, gboolean enable);
gboolean gtk_toolbar_get_tooltips (GtkToolbar *toolbar);
```

- 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 6–2–1 に示します. プログラムの実行結果は図 6.9 になります.

ソース 6–2–1 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 cb_show_tooltips (GtkWidget *widget, gpointer data) {
4   gtk_toolbar_set_tooltips (GTK_TOOLBAR(data),
```

```
5                               gtk_toggle_button_get_active
6                               (GTK_TOGGLE_BUTTON(widget)));
7 }
8
9 static void cb_set_horizontal (GtkWidget *widget, gpointer data) {
10   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON(widget))) {
11     gtk_toolbar_set_orientation (GTK_TOOLBAR(data),
12                                  GTK_ORIENTATION_HORIZONTAL);
13   }
14 }
15
16 static void cb_set_vertical (GtkWidget *widget, gpointer data) {
17   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON(widget))) {
18     gtk_toolbar_set_orientation (GTK_TOOLBAR(data),
19                                  GTK_ORIENTATION_VERTICAL);
20   }
21 }
22
23 int main (int argc, char **argv) {
24   GtkWidget     *window;
25   GtkWidget     *toolbar;
26   GtkWidget     *alignment;
27   GtkWidget     *checkbutton;
28   GtkWidget     *radiobutton;
29   GtkWidget     *button;
30   GtkWidget     *icon;
31   GSList        *radiobutton_group = NULL;
32
33   gtk_init (&argc, &argv);
34   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
35   gtk_window_set_title  (GTK_WINDOW(window), "GtkToolbar␣Sample");
36   gtk_window_set_resizable (GTK_WINDOW(window), FALSE);
37   g_signal_connect (G_OBJECT(window), "destroy",
38                     G_CALLBACK(gtk_main_quit), NULL);
39
40   toolbar = gtk_toolbar_new ();
41   gtk_container_add (GTK_CONTAINER(window), toolbar);
42   gtk_toolbar_set_style (GTK_TOOLBAR(toolbar), GTK_TOOLBAR_BOTH);
43   gtk_toolbar_set_orientation (GTK_TOOLBAR(toolbar),
44                                GTK_ORIENTATION_HORIZONTAL);
45
46   checkbutton = gtk_check_button_new_with_label ("Show␣Tooltips");
47   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON(checkbutton), TRUE);
48   g_signal_connect (G_OBJECT(checkbutton), "toggled",
49                     G_CALLBACK(cb_show_tooltips), toolbar);
50
51   gtk_toolbar_append_widget (GTK_TOOLBAR(toolbar),
52                              checkbutton,
53                              "Toggle␣where␣show␣tooltips", NULL);
54   gtk_toolbar_append_space (GTK_TOOLBAR(toolbar));
```

```
55
56    icon = gtk_image_new_from_stock ("gtk-go-forward",
57                                     gtk_toolbar_get_icon_size (GTK_TOOLBAR
58                                                                (toolbar)));
59
60    radiobutton = gtk_toolbar_append_element(GTK_TOOLBAR(toolbar),
61                                             GTK_TOOLBAR_CHILD_RADIOBUTTON,
62                                             NULL,
63                                             "Horizontal",
64                                             "Set␣the␣toolbar␣to␣horizontal",
65                                             NULL, icon,
66                                             G_CALLBACK(cb_set_horizontal),
67                                             toolbar);
68    gtk_radio_button_set_group (GTK_RADIO_BUTTON(radiobutton),
69                                radiobutton_group);
70    radiobutton_group =
71      gtk_radio_button_group (GTK_RADIO_BUTTON(radiobutton));
72    gtk_toggle_button_set_mode (GTK_TOGGLE_BUTTON(radiobutton), FALSE);
73
74    icon = gtk_image_new_from_stock ("gtk-go-down",
75                                     gtk_toolbar_get_icon_size (GTK_TOOLBAR
76                                                                (toolbar)));
77    radiobutton = gtk_toolbar_append_element(GTK_TOOLBAR(toolbar),
78                                             GTK_TOOLBAR_CHILD_RADIOBUTTON,
79                                             NULL,
80                                             "Vertical",
81                                             "Set␣the␣toolbar␣to␣vertical",
82                                             NULL, icon,
83                                             G_CALLBACK(cb_set_vertical),
84                                             toolbar);
85    gtk_radio_button_set_group (GTK_RADIO_BUTTON (radiobutton),
86                                radiobutton_group);
87    gtk_toggle_button_set_mode (GTK_TOGGLE_BUTTON(radiobutton), FALSE);
88
89    gtk_toolbar_append_space (GTK_TOOLBAR (toolbar));
90
91    button = gtk_toolbar_append_item(GTK_TOOLBAR(toolbar),
92                                     "Exit",
93                                     "Exit␣this␣program.",
94                                     NULL,
95                                     gtk_image_new_from_stock
96                                     ("gtk-quit",
97                                      gtk_toolbar_get_icon_size (GTK_TOOLBAR
98                                                                 (toolbar))),
99                                     G_CALLBACK(gtk_main_quit), NULL);
100   gtk_widget_show_all (window);
101   gtk_main ();
102
103   return 0;
104 }
```

6.2.5　ハンドルボックス

ハンドルボックスウィジェット (GtkHandleBox) は, その中に一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 6.10 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 6.10 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
    +----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 6.5 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.

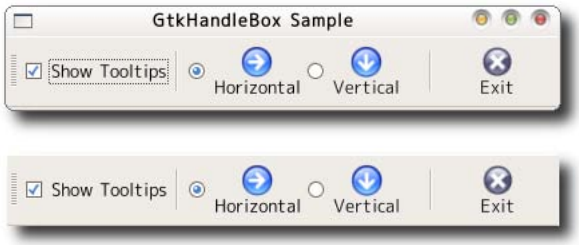


図 6.10　ハンドルボックス

表 6.5　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です.
”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です.

```
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.

- フレームの装飾

  フレームの装飾の種類は 5.4.2 節に紹介した 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 6–2–2 に示します. これはソース 6–2–1 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 6–2–2 ハンドルボックスウィジェットのサンプルプログラム

```
1  #include <gtk/gtk.h>
2
3  static void cb_show_tooltips (GtkWidget *widget, gpointer data) {
4    gtk_toolbar_set_tooltips (GTK_TOOLBAR(data),
5                              gtk_toggle_button_get_active
6                              (GTK_TOGGLE_BUTTON(widget)));
7  }
8
9  static void cb_set_horizontal (GtkWidget *widget, gpointer data) {
10   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON(widget))) {
11     gtk_toolbar_set_orientation (GTK_TOOLBAR(data),
12                                  GTK_ORIENTATION_HORIZONTAL);
13   }
14 }
15
16 static void cb_set_vertical (GtkWidget *widget, gpointer data) {
17   if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON(widget))) {
18     gtk_toolbar_set_orientation (GTK_TOOLBAR(data),
```

```
19                                  GTK_ORIENTATION_VERTICAL);
20   }
21 }
22
23 int main (int argc, char **argv) {
24   GtkWidget     *window;
25   GtkWidget     *handlebox;
26   GtkWidget     *toolbar;
27   GtkWidget     *checkbutton;
28   GtkWidget     *radiobutton;
29   GtkWidget     *button;
30   GtkWidget     *icon;
31   GSList        *radiobutton_group = NULL;
32
33   gtk_init (&argc, &argv);
34   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
35   gtk_window_set_title  (GTK_WINDOW(window), "GtkHandleBox␣Sample");
36   gtk_window_set_resizable (GTK_WINDOW(window), TRUE);
37   g_signal_connect (G_OBJECT(window), "destroy",
38                     G_CALLBACK (gtk_main_quit), NULL);
39
40   handlebox = gtk_handle_box_new ();
41   gtk_handle_box_set_handle_position (GTK_HANDLE_BOX(handlebox),
42                                       GTK_POS_LEFT);
43   gtk_container_add (GTK_CONTAINER(window), handlebox);
44
45   toolbar = gtk_toolbar_new ();
46   gtk_toolbar_set_style (GTK_TOOLBAR(toolbar), GTK_TOOLBAR_BOTH);
47   gtk_toolbar_set_orientation (GTK_TOOLBAR(toolbar),
48                                GTK_ORIENTATION_HORIZONTAL);
49   gtk_container_add (GTK_CONTAINER(handlebox), toolbar);
50
51   checkbutton = gtk_check_button_new_with_label ("Show␣Tooltips");
52   gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON(checkbutton), TRUE);
53   g_signal_connect (G_OBJECT(checkbutton), "toggled",
54                     G_CALLBACK(cb_show_tooltips), toolbar);
55
56   gtk_toolbar_append_widget (GTK_TOOLBAR(toolbar),
57                              checkbutton,
58                              "Toggle␣where␣show␣tooltips", NULL);
59
60   gtk_toolbar_append_space (GTK_TOOLBAR(toolbar));
61
62   icon = gtk_image_new_from_stock ("gtk-go-forward",
63                                    gtk_toolbar_get_icon_size (GTK_TOOLBAR
64                                                               (toolbar)));
65
66   radiobutton = gtk_toolbar_append_element(GTK_TOOLBAR(toolbar),
67                                            GTK_TOOLBAR_CHILD_RADIOBUTTON,
68                                            NULL,
```

```
69                                          "Horizontal",
70                                          "Set␣the␣toolbar␣to␣horizontal",
71                                          NULL, icon,
72                                          G_CALLBACK(cb_set_horizontal),
73                                          toolbar);
74    gtk_radio_button_set_group (GTK_RADIO_BUTTON(radiobutton),
75                                radiobutton_group);
76    radiobutton_group =
77      gtk_radio_button_group (GTK_RADIO_BUTTON(radiobutton));
78    gtk_toggle_button_set_mode (GTK_TOGGLE_BUTTON(radiobutton), TRUE);
79
80    icon = gtk_image_new_from_stock ("gtk-go-down",
81                                     gtk_toolbar_get_icon_size (GTK_TOOLBAR
82                                                                (toolbar)));
83    radiobutton = gtk_toolbar_append_element (GTK_TOOLBAR(toolbar),
84                                              GTK_TOOLBAR_CHILD_RADIOBUTTON,
85                                              NULL,
86                                              "Vertical",
87                                              "Set␣the␣toolbar␣to␣vertical",
88                                              NULL, icon,
89                                              G_CALLBACK(cb_set_vertical),
90                                              toolbar);
91    gtk_radio_button_set_group (GTK_RADIO_BUTTON(radiobutton),
92                                radiobutton_group);
93    gtk_toggle_button_set_mode (GTK_TOGGLE_BUTTON(radiobutton), TRUE);
94
95    gtk_toolbar_append_space (GTK_TOOLBAR(toolbar));
96
97    button = gtk_toolbar_append_item(GTK_TOOLBAR(toolbar),
98                                     "Exit",
99                                     "Exit␣this␣program.",
100                                    NULL,
101                                    gtk_image_new_from_stock
102                                    ("gtk-quit",
103                                     gtk_toolbar_get_icon_size (GTK_TOOLBAR
104                                                                (toolbar))),
105                                    G_CALLBACK(gtk_main_quit), NULL);
106
107   gtk_widget_show_all (window);
108   gtk_main ();
109
110   return 0;
111 }
```

6.2.6　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 6.11 を参照) です.

オブジェクトの階層構造

```
GObject
    +----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. 関数の戻り値には, 追加したページ番号 (ページ番号は 0 から開始) が帰ります.

- 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,
  ```

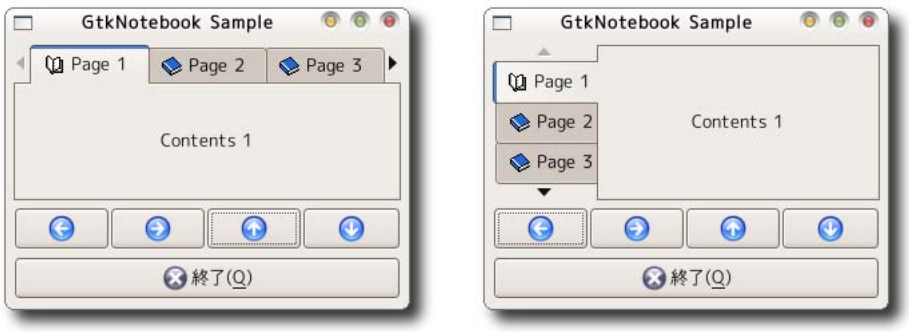


図 6.11　ノートブックウィジェット

```
                                    GtkWidget    *tab_label,
                                    GtkWidget    *menu_label);
```

この他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 6.6 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

表 6.6 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です.

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

  タブが領域内に納まらない場合に, 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 6–2–3 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 6–2–3 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 set_page_image (GtkNotebook *notebook,
7                              gint         page_num,
8                              GdkPixbuf    *pixbuf) {
9    GtkWidget     *page_widget;
10   GtkWidget     *icon;
11
12   page_widget = gtk_notebook_get_nth_page (notebook, page_num);
13   icon = (GtkWidget *)
14     g_object_get_data (G_OBJECT(page_widget), "tab_icon");
15   gtk_image_set_from_pixbuf (GTK_IMAGE(icon), pixbuf);
16   icon = (GtkWidget *)
17     g_object_get_data (G_OBJECT(page_widget), "menu_icon");
18   gtk_image_set_from_pixbuf (GTK_IMAGE(icon), pixbuf);
19 }
20
```

```
21 static void page_switch (GtkWidget       *widget,
22                          GtkNotebookPage *page,
23                          gint            page_num) {
24   GtkNotebook   *notebook = GTK_NOTEBOOK(widget);
25   gint          old_page_num;
26
27   old_page_num = gtk_notebook_get_current_page (notebook);
28   if (page_num == old_page_num) return;
29   set_page_image (notebook, page_num, book_open);
30   if (old_page_num != -1) {
31     set_page_image (notebook, old_page_num, book_close);
32   }
33 }
34
35 static void cb_tab_position_left (GtkWidget *widget, gpointer data) {
36   gtk_notebook_set_tab_pos (GTK_NOTEBOOK(data), GTK_POS_LEFT);
37 }
38
39 static void cb_tab_position_right (GtkWidget *widget, gpointer data) {
40   gtk_notebook_set_tab_pos (GTK_NOTEBOOK(data), GTK_POS_RIGHT);
41 }
42
43 static void cb_tab_position_top (GtkWidget *widget, gpointer data) {
44   gtk_notebook_set_tab_pos (GTK_NOTEBOOK(data), GTK_POS_TOP);
45 }
46
47 static void cb_tab_position_bottom (GtkWidget *widget, gpointer data) {
48   gtk_notebook_set_tab_pos (GTK_NOTEBOOK(data), GTK_POS_BOTTOM);
49 }
50
51 static GtkWidget* icon_button_new (const gchar *stock_id) {
52   GtkWidget     *button;
53   GtkWidget     *icon;
54
55   icon = gtk_image_new_from_stock (stock_id, GTK_ICON_SIZE_BUTTON);
56   button = gtk_button_new ();
57   gtk_container_add (GTK_CONTAINER(button), icon);
58
59   return button;
60 }
61
62 int main (int argc, char **argv) {
63   GtkWidget     *window;
64   GtkWidget     *vbox;
65   GtkWidget     *hbox;
66   GtkWidget     *notebook;
67   GtkWidget     *box;
68   GtkWidget     *label;
69   GtkWidget     *label_box;
70   GtkWidget     *menu_box;
```

```
71  GtkWidget      *icon;
72  GtkWidget      *button;
73  gchar          buf[1024];
74  int            n;
75
76  gtk_init (&argc, &argv);
77
78  book_open  = gdk_pixbuf_new_from_file ("stock_book_open.png",  NULL);
79  book_close = gdk_pixbuf_new_from_file ("stock_book_close.png", NULL);
80
81  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
82  gtk_window_set_title  (GTK_WINDOW(window), "GtkNotebook␣Sample");
83  gtk_window_set_resizable (GTK_WINDOW(window), TRUE);
84  gtk_widget_set_size_request (window, 300, 200);
85  gtk_container_set_border_width (GTK_CONTAINER (window), 5);
86  g_signal_connect (G_OBJECT(window), "destroy",
87                    G_CALLBACK(gtk_main_quit), NULL);
88
89  vbox = gtk_vbox_new (FALSE, 5);
90  gtk_container_add (GTK_CONTAINER(window), vbox);
91
92  notebook = gtk_notebook_new ();
93  gtk_notebook_set_scrollable (GTK_NOTEBOOK(notebook), TRUE);
94  gtk_notebook_popup_enable (GTK_NOTEBOOK(notebook));
95  g_signal_connect (G_OBJECT(notebook), "switch_page",
96                    G_CALLBACK(page_switch), NULL);
97  gtk_box_pack_start (GTK_BOX(vbox), notebook, TRUE, TRUE, 0);
98
99  for (n = 1; n <= 5; n++) {
100   box   = gtk_vbox_new (FALSE, 0);
101
102   sprintf (buf, "Contents␣%d", n);
103   label = gtk_label_new (buf);
104   gtk_box_pack_start (GTK_BOX(box), label, TRUE, TRUE, 0);
105
106   label_box = gtk_hbox_new (FALSE, 0);
107   icon = gtk_image_new_from_pixbuf (book_close);
108   g_object_set_data (G_OBJECT(box), "tab_icon", icon);
109   gtk_box_pack_start (GTK_BOX(label_box), icon, FALSE, TRUE, 0);
110   gtk_misc_set_padding (GTK_MISC(icon), 5, 1);
111
112   sprintf (buf, "Page␣%d", n);
113   label = gtk_label_new (buf);
114   gtk_box_pack_start (GTK_BOX(label_box), label, FALSE, TRUE, 0);
115
116   gtk_widget_show_all (label_box);
117
118   menu_box = gtk_hbox_new (FALSE, 0);
119   icon = gtk_image_new_from_pixbuf (book_close);
120   g_object_set_data (G_OBJECT(box), "menu_icon", icon);
```

```
121     gtk_box_pack_start (GTK_BOX(menu_box), icon, FALSE, TRUE, 0);
122     gtk_misc_set_padding (GTK_MISC(icon), 5, 1);
123
124     label = gtk_label_new (buf);
125     gtk_box_pack_start (GTK_BOX(menu_box), label, FALSE, TRUE, 0);
126
127     gtk_widget_show_all (menu_box);
128
129     gtk_notebook_append_page_menu (GTK_NOTEBOOK(notebook),
130                                    box, label_box, menu_box);
131   }
132   hbox = gtk_hbox_new (TRUE, 0);
133   gtk_box_pack_start (GTK_BOX (vbox), hbox, FALSE, FALSE, 0);
134   {
135     button = icon_button_new (GTK_STOCK_GO_BACK);
136     g_signal_connect (G_OBJECT(button), "clicked",
137                       G_CALLBACK(cb_tab_position_left), notebook);
138     gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
139
140     button = icon_button_new (GTK_STOCK_GO_FORWARD);
141     g_signal_connect (G_OBJECT(button), "clicked",
142                       G_CALLBACK(cb_tab_position_right), notebook);
143     gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
144
145     button = icon_button_new (GTK_STOCK_GO_UP);
146     g_signal_connect (G_OBJECT(button), "clicked",
147                       G_CALLBACK(cb_tab_position_top), notebook);
148     gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
149
150     button = icon_button_new (GTK_STOCK_GO_DOWN);
151     g_signal_connect (G_OBJECT(button), "clicked",
152                       G_CALLBACK(cb_tab_position_bottom), notebook);
153     gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
154   }
155   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
156   g_signal_connect (G_OBJECT(button), "clicked",
157                     G_CALLBACK(gtk_main_quit), NULL);
158   gtk_box_pack_start (GTK_BOX(vbox), button, FALSE, FALSE, 0);
159
160   gtk_widget_show_all (window);
161   gtk_main ();
162
163   return 0;
164 }
```

6.2.7　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
    +----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 6.7 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.

図 6.12　エクスパンダウィジェット

表 6.7 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です.

- 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.

```
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 6–2–4 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 6–2–4 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 main (int    argc,
4            char   **argv) {
5    GtkWidget      *window, *vbox, *hbox, *label, *button, *expander;
6
7    gtk_init (&argc, &argv);
8    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
9    gtk_window_set_title  (GTK_WINDOW(window), "GtkExpander Sample");
10   gtk_window_set_resizable (GTK_WINDOW(window), FALSE);
11   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
12   g_signal_connect (G_OBJECT(window), "destroy",
13                     G_CALLBACK(gtk_main_quit), NULL);
14
15   vbox = gtk_vbox_new (FALSE, 5);
16   gtk_container_add (GTK_CONTAINER(window), vbox);
17
18   label =
19     gtk_label_new ("Expander demo. Click on the triangle for details.");
20   gtk_box_pack_start (GTK_BOX(vbox), label, FALSE, FALSE, 0);
21
22   expander = gtk_expander_new ("Details");
```

```
23    gtk_box_pack_start (GTK_BOX(vbox), expander, FALSE, FALSE, 0);
24    gtk_expander_set_expanded (GTK_EXPANDER(expander), TRUE);
25
26    label = gtk_label_new ("Details␣can␣be␣shown␣or␣hidden.");
27    gtk_container_add (GTK_CONTAINER(expander), label);
28
29    hbox = gtk_hbox_new (FALSE, 5);
30    gtk_box_pack_start (GTK_BOX(vbox), hbox, FALSE, FALSE, 0);
31
32    button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
33    g_signal_connect (G_OBJECT(button), "clicked",
34                      G_CALLBACK(gtk_main_quit), NULL);
35
36    gtk_box_pack_end (GTK_BOX(hbox), button, FALSE, FALSE, 0);
37
38    gtk_widget_show_all (window);
39    gtk_main ();
40
41    return 0;
42 }
```

6.3　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.

6.3.1　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
    +----GtkObject
          +----GtkWidget
                +----GtkEntry
```

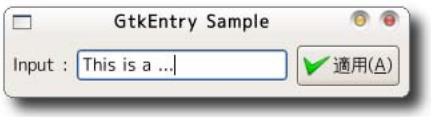


図 6.13　エントリウィジェット

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 6.8 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” シグナルの他にも ”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．その他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
  パスワードの入力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の代わりに表示する文字です．

  ```
  void gtk_entry_set_invisible_char (GtkEntry *entry, gunichar ch);
  gunichar gtk_entry_get_invisible_char (GtkEntry *entry);
  ```

- シークレット文字を表示するかどうか
  入力した文字を上記の隠し文字で表示するかどうかを設定します．

  ```
  void gtk_entry_set_visibility (GtkEntry *entry, gboolean visible);
  gboolean gtk_entry_get_visibility (GtkEntry *entry);
  ```

- 入力されているテキスト現在エントリウィジェットに入力されている文字列です．

  ```
  void gtk_entry_set_text (GtkEntry *entry, const gchar *text);
  G_CONST_RETURN gchar* gtk_entry_get_text (GtkEntry *entry);
  ```

サンプルプログラム

ソース 6–3–1 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

表 6.8 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です．

字列を端末に表示するものです.

ソース 6–3–1 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 cb_entry (GtkEntry *entry, gpointer data) {
4    g_print ("%s\n", gtk_entry_get_text (entry));
5  }
6
7  static void cb_button (GtkButton *button, gpointer data) {
8    g_print ("%s\n", gtk_entry_get_text (GTK_ENTRY(data)));
9  }
10
11 int main (int argc, char **argv) {
12   GtkWidget     *window, *hbox, *label, *entry, *button;
13
14   gtk_init (&argc, &argv);
15   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
16   gtk_window_set_title  (GTK_WINDOW(window), "GtkEntry␣Sample");
17   gtk_window_set_resizable (GTK_WINDOW(window), FALSE);
18   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
19   g_signal_connect (G_OBJECT(window), "destroy",
20                     G_CALLBACK(gtk_main_quit), NULL);
21   hbox = gtk_hbox_new (FALSE, 5);
22   gtk_container_add (GTK_CONTAINER(window), hbox);
23
24   label = gtk_label_new ("Input␣:");
25   gtk_box_pack_start (GTK_BOX(hbox), label, FALSE, FALSE, 0);
26
27   entry = gtk_entry_new ();
28   g_signal_connect (G_OBJECT(entry), "activate",
29                     G_CALLBACK(cb_entry), NULL);
30   gtk_box_pack_start (GTK_BOX(hbox), entry, TRUE, TRUE, 0);
31
32   button = gtk_button_new_from_stock (GTK_STOCK_APPLY);
33   g_signal_connect (G_OBJECT(button), "clicked",
34                     G_CALLBACK(cb_button), (gpointer) entry);
35   gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
36
37   gtk_widget_show_all (window);
38   gtk_main ();
39
40   return 0;
41 }
```

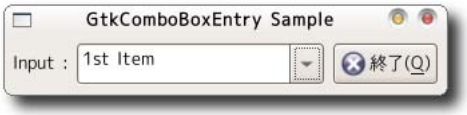


図 6.14 コンボボックスエントリウィジェット

6.3.2 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. コンボボックスは予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を先に読まれることをお薦めします.

オブジェクトの階層構造

```
GObject
    +----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 6.9 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, 標準値として予めアイテムを選択 (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次のようにします．

ソース 6–3–2 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    gtk_tree_model_get (model, &iter, 0, &label, -1);
8  }
```

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．

表 6.9　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です．

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

サンプルプログラム

ソース 6–3–3 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予め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を越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．

ソース 6–3–3 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

```
1  #include <gtk/gtk.h>
2
3  #define COMBO_LIST_LIMIT 5
4
5  static GList    *combolist = NULL;
6
7  static GList* append_item (GtkComboBox  *combobox,
8                             GList        *combolist,
9                             const gchar  *item_label) {
10   gtk_combo_box_append_text (combobox, g_strdup (item_label));
11   combolist = g_list_append (combolist, g_strdup (item_label));
12
13   return combolist;
14 }
15
16 static GList* remove_item (GtkComboBox *combobox,
17                            GList       *combolist,
18                            gint         index) {
19   gtk_combo_box_remove_text (combobox, index);
20   g_free (g_list_nth_data (combolist, index));
21   combolist = g_list_remove_link(combolist, g_list_nth (combolist, index));
22
23   return combolist;
24 }
25
26 static void cb_combo_changed (GtkComboBox *combobox,
27                               gpointer    data) {
28   GtkTreeModel  *model;
29   GtkTreeIter   iter;
30   gchar         *text;
31
32   model = gtk_combo_box_get_model (combobox);
33   if (gtk_combo_box_get_active_iter (combobox, &iter)) {
34     gtk_tree_model_get (model, &iter, 0, &text, -1);
```

```
35      g_print ("Item␣number␣is␣%d␣(%s)\n",
36               gtk_combo_box_get_active (combobox), text);
37    }
38  }
39
40  static void cb_combo_entry_activate (GtkEntry *entry,
41                                       gpointer data) {
42    GtkComboBox    *combobox;
43    GList          *list;
44    gboolean       exist_flag = FALSE;
45    const gchar    *new_text;
46
47    new_text = gtk_entry_get_text (entry);
48    if (!new_text || strcmp (new_text, "") == 0) return;
49
50    for (list = combolist; list; list = g_list_next (list)) {
51      if (strcmp (new_text, (gchar *) list->data) == 0) {
52        exist_flag = TRUE;
53        break;
54      }
55    }
56    if (!exist_flag) {
57      combobox = GTK_COMBO_BOX(user_data);
58      combolist = append_item (combobox, combolist, new_text);
59      if (g_list_length (combolist) > COMBO_LIST_LIMIT) {
60        combolist = remove_item (combobox, combolist, 0);
61      }
62      g_print ("Append␣a␣new␣item␣label␣'%s'.\n", new_text);
63    }
64  }
65
66  int main (int argc, char **argv) {
67    GtkWidget      *window, *hbox, *label, *combo, *button;
68
69    gtk_init (&argc, &argv);
70    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
71    gtk_window_set_title  (GTK_WINDOW(window), "GtkComboBoxEntry␣Sample");
72    gtk_window_set_resizable (GTK_WINDOW(window), FALSE);
73    gtk_container_set_border_width (GTK_CONTAINER(window), 5);
74    g_signal_connect (G_OBJECT(window), "destroy",
75                      G_CALLBACK(gtk_main_quit), NULL);
76    hbox = gtk_hbox_new (FALSE, 5);
77    gtk_container_add (GTK_CONTAINER(window), hbox);
78
79    label = gtk_label_new ("Input␣:");
80    gtk_box_pack_start (GTK_BOX(hbox), label, FALSE, FALSE, 0);
81
82    combo = gtk_combo_box_entry_new_text ();
83    combolist = append_item (GTK_COMBO_BOX(combo), combolist, "1st␣Item");
84    combolist = append_item (GTK_COMBO_BOX(combo), combolist, "2nd␣Item");
```

```
85    gtk_combo_box_set_active (GTK_COMBO_BOX(combo), 0);
86
87    g_signal_connect (G_OBJECT(combo), "changed",
88                      G_CALLBACK(cb_combo_changed), NULL);
89    g_signal_connect (G_OBJECT(GTK_BIN(combo)->child), "activate",
90                      G_CALLBACK(cb_combo_entry_activate), combo);
91
92    gtk_box_pack_start (GTK_BOX(hbox), combo, TRUE, TRUE, 0);
93
94    button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
95    g_signal_connect (G_OBJECT(button), "clicked",
96                      G_CALLBACK(gtk_main_quit), NULL);
97    gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
98
99    gtk_widget_show_all (window);
100   gtk_main ();
101
102   return 0;
103 }
```

6.3.3 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
    +----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.

- 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);
```

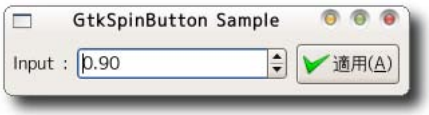


図 6.15 スピンボタンウィジェット

ソース 6–3–4 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．GtkAdjustment での指定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 6–3–4 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 6.10 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，取得したりすることができる．

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

表 6.10　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です．

```
                                    gdouble       *step,
                                    gdouble       *page);
```

サンプルプログラム

ソース 6–3–5 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するものです．

ソース 6–3–5 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

```
1  #include <gtk/gtk.h>
2
3  static void cb_value_changed (GtkSpinButton *spinbutton, gpointer data) {
4    g_print ("value␣=␣%f\n", gtk_spin_button_get_value (spinbutton));
5  }
6
7  static void cb_button (GtkButton *button, gpointer data) {
8    g_print ("value␣=␣%f\n",
9             gtk_spin_button_get_value (GTK_SPIN_BUTTON(data)));
10 }
11
12 int main (int argc, char **argv) {
13   GtkWidget     *window;
14   GtkWidget     *hbox;
15   GtkWidget     *label;
16   GtkWidget     *spinbutton;
17   GtkWidget     *button;
18   GtkObject     *adjustment;
19   gdouble       min = 0.0, max = 100.0, step = 0.1;
20
21   gtk_init (&argc, &argv);
22   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
23   gtk_window_set_title  (GTK_WINDOW(window), "GtkSpinButton␣Sample");
24   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
25   gtk_widget_set_size_request (window, 300, -1);
26   g_signal_connect (G_OBJECT(window), "destroy",
27                     G_CALLBACK(gtk_main_quit), NULL);
28
29   hbox = gtk_hbox_new (FALSE, 5);
30   gtk_container_add (GTK_CONTAINER(window), hbox);
31
32   label = gtk_label_new ("Input␣:");
33   gtk_box_pack_start (GTK_BOX(hbox), label, FALSE, FALSE, 0);
34
35   spinbutton = gtk_spin_button_new_with_range (min, max, step);
36   gtk_spin_button_set_digits (GTK_SPIN_BUTTON(spinbutton), 2);
37   gtk_spin_button_set_wrap (GTK_SPIN_BUTTON(spinbutton), TRUE);
38
```

```
39   g_signal_connect (G_OBJECT(spinbutton), "value_changed",
40                     G_CALLBACK(cb_value_changed), NULL);
41   gtk_box_pack_start (GTK_BOX(hbox), spinbutton, 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) spinbutton);
46   gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
47
48   gtk_widget_show_all (window);
49   gtk_main ();
50
51   return 0;
52 }
```

6.3.4　テキストビュー

テキストビューウィジェット (GtkTextView) は複数行に渡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
    +----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 を使用しましょう. 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.

図 6.16　テキストビューウィジェット

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プログラマが関数 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関数があります．

- gtk_text_buffer_get_iter_at_line
  指定した行番号の GtkTextIter を取得します．行番号は 0 から開始します．

```
void gtk_text_buffer_get_iter_at_line (GtkTextBuffer *buffer,
                                       GtkTextIter   *iter,
                                       gint          line_number);
```

- gtk_text_buffer_get_start_iter
  開始行の GtkTextIter を取得します．

```
void gtk_text_buffer_get_start_iter (GtkTextBuffer *buffer,
                                     GtkTextIter   *iter);
```

- gtk_text_buffer_get_end_iter
  最終行の GtkTextIter を取得します．

```
void gtk_text_buffer_get_end_iter (GtkTextBuffer *buffer,
                                   GtkTextIter   *iter);
```

サンプルプログラム

ソース 6–3–6 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全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 6–3–6 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

```
1  #include <gtk/gtk.h>
2
3  static void set_text (GtkTextView *textview, const gchar *text) {
4    GtkTextBuffer *buffer;
5    gchar         *utf8_text;
6
7    utf8_text = g_locale_to_utf8 (text, -1, NULL, NULL, NULL);
8    buffer = gtk_text_view_get_buffer (textview);
9    gtk_text_buffer_set_text (buffer, (utf8_text) ? utf8_text : text, -1);
10   if (utf8_text) g_free (utf8_text);
11 }
12
13 static void print_text (GtkWidget *widget, gpointer data) {
14   GtkTextBuffer *buffer;
15   GtkTextIter   start, end;
16   gchar         *utf8_text;
17
18   buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW(data));
19   gtk_text_buffer_get_start_iter (buffer, &start);
20   gtk_text_buffer_get_end_iter (buffer, &end);
21   utf8_text = gtk_text_buffer_get_text (buffer, &start, &end, TRUE);
22   g_print ("%s\n", utf8_text);
23   g_free (utf8_text);
24 }
25
26 int main (int argc, char **argv) {
27   GtkWidget     *window;
28   GtkWidget     *vbox, *hbox;
29   GtkWidget     *scrolledwindow;
30   GtkWidget     *textview;
31   GtkWidget     *button;
32
33   gtk_init (&argc, &argv);
34   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
35   gtk_window_set_title (GTK_WINDOW(window), "GtkTextView␣Sample");
36   gtk_widget_set_size_request (window, 300, 100);
37   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
38   g_signal_connect (G_OBJECT(window), "destroy",
39                     G_CALLBACK(gtk_main_quit), NULL);
40
41   vbox = gtk_vbox_new (FALSE, 5);
42   gtk_container_add (GTK_CONTAINER(window), vbox);
43
44   scrolledwindow = gtk_scrolled_window_new (NULL, NULL);
45   gtk_scrolled_window_set_shadow_type (GTK_SCROLLED_WINDOW(scrolledwindow),
46                                        GTK_SHADOW_ETCHED_OUT);
```

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

6.4 メニューウィジェット

6.4.1 メニューバー

メニューバーウィジェットは, ウィンドウの上部などに配置して, 様々な操作を支援するウィジェットです (図 6.17).

オブジェクトの階層構造

```
GObject
   +----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．

- 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);
    ```

- 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．

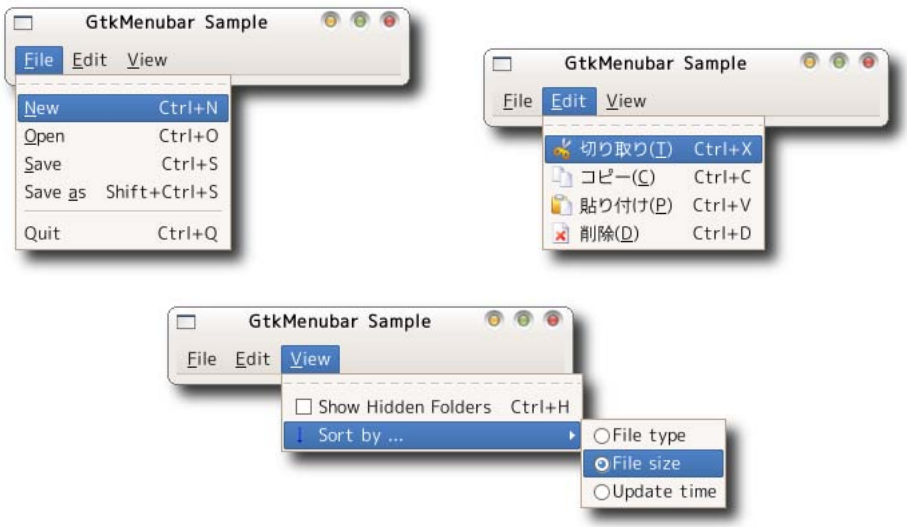


図 6.17　メニューバー

```
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ますので, 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);
```

- 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);
```

- 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, そのアイテムを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, そのアイテムを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);
```

- 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多いです. 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設定の例を次に示します. 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.

```
GtkWidget     *item;
GtkAccelGroup *accel_group;

accel_group = gtk_accel_group_new ();
item = gtk_menu_item_new_with_mnemonic ("_Open");
gtk_widget_add_accelerator (item, "activate", accel_group,
                            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, /* display in GtkAccelLabel? */
  GTK_ACCEL_LOCKED  = 1 << 1, /* is it removable? */
  GTK_ACCEL_MASK    = 0x07
} GtkAccelFlags;
```

階層的なメニューの作成

一つのメニューの中に更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

図 6.17 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 6–4–1 メニューバーのサンプルプログラム：gtkmenubar-sample.c

```
1  #include <gtk/gtk.h>
2  #include <gdk/gdkkeysyms.h>
3
4  static void cb_quit (GtkWidget *widget, gpointer data) {
5    gtk_main_quit ();
6  }
7
8  static GtkWidget* create_menu (void) {
9    GtkWidget     *menubar;
10   GtkWidget     *menu;
11   GtkWidget     *item;
12   GtkWidget     *image;
13   GtkAccelGroup *accel_group;
14   GSList        *group = NULL;
15
16   menubar = gtk_menu_bar_new ();
17   accel_group = gtk_accel_group_new ();
18
19   item = gtk_menu_item_new_with_mnemonic ("_File");
20   gtk_container_add (GTK_CONTAINER(menubar), item);
21   menu = gtk_menu_new ();
22   gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), menu);
23   {
24     item = gtk_tearoff_menu_item_new ();
25     gtk_container_add (GTK_CONTAINER(menu), item);
26
27     item = gtk_menu_item_new_with_mnemonic ("_New");
28     gtk_container_add (GTK_CONTAINER(menu), item);
29     gtk_widget_add_accelerator(item, "activate", accel_group,
30                                GDK_N, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
31
32     item = gtk_menu_item_new_with_mnemonic ("_Open");
```

```
33      gtk_container_add (GTK_CONTAINER(menu), item);
34      gtk_widget_add_accelerator(item, "activate", accel_group,
35                                 GDK_O, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
36
37      item = gtk_menu_item_new_with_mnemonic ("_Save");
38      gtk_container_add (GTK_CONTAINER(menu), item);
39      gtk_widget_add_accelerator(item, "activate", accel_group,
40                                 GDK_S, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
41
42      item = gtk_menu_item_new_with_mnemonic ("Save␣_as");
43      gtk_container_add (GTK_CONTAINER(menu), item);
44      gtk_widget_add_accelerator(item, "activate", accel_group,
45                                 GDK_S, GDK_CONTROL_MASK | GDK_SHIFT_MASK,
46                                 GTK_ACCEL_VISIBLE);
47
48      item = gtk_separator_menu_item_new ();
49      gtk_container_add (GTK_CONTAINER(menu), item);
50
51      item = gtk_menu_item_new_with_mnemonic ("_Quit");
52      gtk_container_add (GTK_CONTAINER(menu), item);
53      gtk_widget_add_accelerator(item, "activate", accel_group,
54                                 GDK_Q, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
55      g_signal_connect (G_OBJECT(item), "activate", G_CALLBACK(cb_quit), NULL
            );
56    }
57    item = gtk_menu_item_new_with_mnemonic ("_Edit");
58    gtk_container_add (GTK_CONTAINER(menubar), item);
59    menu = gtk_menu_new ();
60    gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), menu);
61    {
62      item = gtk_tearoff_menu_item_new ();
63      gtk_container_add (GTK_CONTAINER(menu), item);
64
65      item = gtk_image_menu_item_new_from_stock(GTK_STOCK_CUT, accel_group);
66      gtk_container_add (GTK_CONTAINER(menu), item);
67
68      item = gtk_image_menu_item_new_from_stock(GTK_STOCK_COPY, accel_group);
69      gtk_container_add (GTK_CONTAINER(menu), item);
70
71      item = gtk_image_menu_item_new_from_stock(GTK_STOCK_PASTE, accel_group)
            ;
72      gtk_container_add (GTK_CONTAINER(menu), item);
73
74      item = gtk_image_menu_item_new_from_stock(GTK_STOCK_DELETE, accel_group
            );
75      gtk_container_add (GTK_CONTAINER(menu), item);
76      gtk_widget_add_accelerator(item, "activate", accel_group,
77                                 GDK_D, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
78    }
79    item = gtk_menu_item_new_with_mnemonic ("_View");
```

```
80    gtk_container_add (GTK_CONTAINER(menubar), item);
81    menu = gtk_menu_new ();
82    gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), menu);
83    {
84      item = gtk_tearoff_menu_item_new ();
85      gtk_container_add (GTK_CONTAINER(menu), item);
86
87      item = gtk_check_menu_item_new_with_label ("Show␣Hidden␣Folders");
88      gtk_container_add (GTK_CONTAINER(menu), item);
89      gtk_widget_add_accelerator(item, "activate", accel_group,
90                                 GDK_H, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
91
92      item = gtk_image_menu_item_new_with_label ("Sort␣by␣...");
93      image = gtk_image_new_from_stock (GTK_STOCK_SORT_ASCENDING,
94                                        GTK_ICON_SIZE_MENU);
95      gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM(item), image);
96      gtk_container_add (GTK_CONTAINER(menu), item);
97      menu = gtk_menu_new ();
98      gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), menu);
99      {
100       item = gtk_radio_menu_item_new_with_label (group, "File␣type");
101       gtk_container_add (GTK_CONTAINER(menu), item);
102       gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM(item), TRUE);
103
104       group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM(item));
105       item = gtk_radio_menu_item_new_with_label (group, "File␣size");
106       gtk_container_add (GTK_CONTAINER(menu), item);
107
108       group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM(item));
109       item = gtk_radio_menu_item_new_with_label (group, "Update␣time");
110       gtk_container_add (GTK_CONTAINER(menu), item);
111     }
112   }
113   return menubar;
114 }
115
116 int main (int argc, char **argv) {
117   GtkWidget     *window;
118   GtkWidget     *menubar;
119
120   gtk_init (&argc, &argv);
121   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
122   gtk_window_set_title  (GTK_WINDOW(window), "GtkMenubar␣Sample");
123   gtk_widget_set_size_request (window, 300, -1);
124   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
125   g_signal_connect (G_OBJECT(window), "destroy",
126                     G_CALLBACK(gtk_main_quit), NULL);
127
128   menubar = create_menu ();
129   gtk_container_add (GTK_CONTAINER(window), menubar);
```

```
130
131   gtk_widget_show_all (window);
132   gtk_main ();
133
134   return 0;
135 }
```

6.4.2　ポップアップメニュー

ポップアップメニューは, 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 6.18).

オブジェクトの階層構造

```
GObject
   +----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, 前節で説明したメニューの作成と同様です. 違いは, メニューバーウィジェットにメニューを配置するのではなく, マウスボタンのクリック等の動作によってメニューを表示する点です.

ポップアップメニューの表示

ポップアップメニューを表示するには, 関数 gtk_menu_popup を使用します.

```
void gtk_menu_popup (GtkMenu             *menu,
                     GtkWidget           *parent_menu_shell,
                     GtkWidget           *parent_menu_item,
                     GtkMenuPositionFunc func,
                     gpointer            data,
                     guint               button,
                     guint32             activate_time);
```

第 1 引数には表示するポップアップメニューを指定します. 第 2 引数から第 5 引数までは通常 NULL を指定しておけば大丈夫です. 第 6 引数は任意の値を与えても動作に違いがありません. 通常は 0 でいいでしょう.

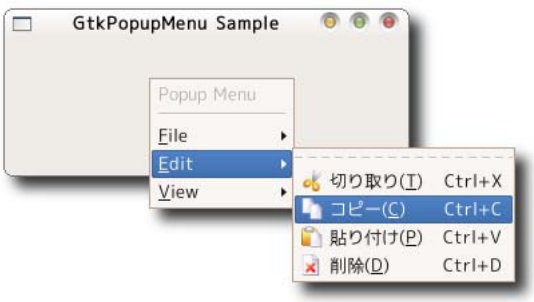


図 6.18　ポップアップメニュー

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 6–4–2 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 6–4–2 ポップアップメニューのサンプルプログラム : gtkpopupmenu-sample.c

```
1  #include <gtk/gtk.h>
2  #include <gdk/gdkkeysyms.h>
3
4  static void cb_quit (GtkWidget *widget, gpointer data) {
5    gtk_main_quit ();
6  }
7
8  static gboolean cb_popup_menu (GtkWidget        *widget,
9                                 GdkEventButton   *event,
10                                gpointer         user_data) {
11   GtkMenu       *popupmenu = GTK_MENU(user_data);
12
13   if (event->button == 3) { /* 右ボタンクリック */
14     gtk_menu_popup (popupmenu, NULL, NULL, NULL, NULL, 0, event->time);
15   }
16   return FALSE;
17 }
18
19 static GtkWidget* create_popupmenu (GtkWindow *parent) {
20   GtkWidget     *popupmenu;
21   GtkWidget     *menu;
22   GtkWidget     *item;
23   GtkWidget     *image;
24   GtkAccelGroup *accel_group;
25   GSList        *group = NULL;
26
27   popupmenu = gtk_menu_new ();
28   accel_group = gtk_accel_group_new ();
29   gtk_window_add_accel_group (parent, accel_group);
30
31   item = gtk_menu_item_new_with_label ("Popup␣Menu");
32   gtk_widget_set_sensitive (item, FALSE);
33   gtk_menu_shell_append (GTK_MENU_SHELL(popupmenu), item);
```

```
34
35  item = gtk_separator_menu_item_new ();
36  gtk_menu_shell_append (GTK_MENU_SHELL(popupmenu), item);
37
38  item = gtk_menu_item_new_with_mnemonic ("_File");
39  gtk_menu_shell_append (GTK_MENU_SHELL(popupmenu), item);
40  menu = gtk_menu_new ();
41  gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), menu);
42  {
43    item = gtk_tearoff_menu_item_new ();
44    gtk_container_add (GTK_CONTAINER(menu), item);
45
46    item = gtk_menu_item_new_with_mnemonic ("_New");
47    gtk_container_add (GTK_CONTAINER(menu), item);
48    gtk_widget_add_accelerator(item, "activate", accel_group,
49                               GDK_N, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
50
51    item = gtk_menu_item_new_with_mnemonic ("_Open");
52    gtk_container_add (GTK_CONTAINER(menu), item);
53    gtk_widget_add_accelerator(item, "activate", accel_group,
54                               GDK_O, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
55
56    item = gtk_menu_item_new_with_mnemonic ("_Save");
57    gtk_container_add (GTK_CONTAINER(menu), item);
58    gtk_widget_add_accelerator(item, "activate", accel_group,
59                               GDK_S, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
60
61    item = gtk_menu_item_new_with_mnemonic ("Save␣_as");
62    gtk_container_add (GTK_CONTAINER(menu), item);
63    gtk_widget_add_accelerator(item, "activate", accel_group,
64                               GDK_S, GDK_CONTROL_MASK | GDK_SHIFT_MASK,
65                               GTK_ACCEL_VISIBLE);
66
67    item = gtk_separator_menu_item_new ();
68    gtk_container_add (GTK_CONTAINER(menu), item);
69
70    item = gtk_menu_item_new_with_mnemonic ("_Quit");
71    gtk_container_add (GTK_CONTAINER(menu), item);
72    gtk_widget_add_accelerator(item, "activate", accel_group,
73                               GDK_Q, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
74    g_signal_connect (G_OBJECT(item),
75                      "activate", G_CALLBACK(cb_quit), NULL);
76  }
77  item = gtk_menu_item_new_with_mnemonic ("_Edit");
78  gtk_menu_shell_append (GTK_MENU_SHELL(popupmenu), item);
79  menu = gtk_menu_new ();
80  gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), menu);
81  {
82    item = gtk_tearoff_menu_item_new ();
83    gtk_container_add (GTK_CONTAINER(menu), item);
```

```
84
85      item = gtk_image_menu_item_new_from_stock(GTK_STOCK_CUT, accel_group);
86      gtk_container_add (GTK_CONTAINER(menu), item);
87
88      item = gtk_image_menu_item_new_from_stock(GTK_STOCK_COPY, accel_group);
89      gtk_container_add (GTK_CONTAINER(menu), item);
90
91      item = gtk_image_menu_item_new_from_stock(GTK_STOCK_PASTE,accel_group);
92      gtk_container_add (GTK_CONTAINER(menu), item);
93
94      item = gtk_image_menu_item_new_from_stock(GTK_STOCK_DELETE,
95                                                accel_group);
96      gtk_container_add (GTK_CONTAINER(menu), item);
97      gtk_widget_add_accelerator(item, "activate", accel_group,
98                                 GDK_D, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
99    }
100   item = gtk_menu_item_new_with_mnemonic ("_View");
101   gtk_menu_shell_append (GTK_MENU_SHELL(popupmenu), item);
102   menu = gtk_menu_new ();
103   gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), menu);
104   {
105     item = gtk_tearoff_menu_item_new ();
106     gtk_container_add (GTK_CONTAINER(menu), item);
107
108     item = gtk_check_menu_item_new_with_label ("Show␣Hidden␣Folders");
109     gtk_container_add (GTK_CONTAINER(menu), item);
110     gtk_widget_add_accelerator(item, "activate", accel_group,
111                                GDK_H, GDK_CONTROL_MASK, GTK_ACCEL_VISIBLE);
112
113     item = gtk_image_menu_item_new_with_label ("Sort␣by␣...");
114     image = gtk_image_new_from_stock (GTK_STOCK_SORT_ASCENDING,
115                                       GTK_ICON_SIZE_MENU);
116     gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM(item), image);
117     gtk_container_add (GTK_CONTAINER(menu), item);
118     menu = gtk_menu_new ();
119     gtk_menu_item_set_submenu (GTK_MENU_ITEM(item), menu);
120     {
121       item = gtk_radio_menu_item_new_with_label (group, "File␣type");
122       gtk_container_add (GTK_CONTAINER(menu), item);
123       gtk_check_menu_item_set_active (GTK_CHECK_MENU_ITEM(item), TRUE);
124
125       group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM(item));
126       item = gtk_radio_menu_item_new_with_label (group, "File␣size");
127       gtk_container_add (GTK_CONTAINER(menu), item);
128
129       group = gtk_radio_menu_item_get_group (GTK_RADIO_MENU_ITEM(item));
130       item = gtk_radio_menu_item_new_with_label (group, "Update␣time");
131       gtk_container_add (GTK_CONTAINER(menu), item);
132     }
133   }
```

```
134   return popupmenu;
135 }
136
137 int main (int argc, char **argv) {
138   GtkWidget      *window;
139   GtkWidget      *eventbox;
140   GtkWidget      *popupmenu;
141
142   gtk_init (&argc, &argv);
143   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
144   gtk_window_set_title  (GTK_WINDOW(window), "GtkPopupMenu␣Sample");
145   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
146   gtk_widget_set_size_request (window, 300, 100);
147   g_signal_connect (G_OBJECT(window), "destroy",
148                     G_CALLBACK(gtk_main_quit), NULL);
149
150   popupmenu = create_popupmenu (GTK_WINDOW(window));
151   gtk_widget_show_all (popupmenu);
152
153   eventbox = gtk_event_box_new ();
154   gtk_container_add (GTK_CONTAINER(window), eventbox);
155   g_signal_connect (G_OBJECT(eventbox), "button_press_event",
156                     G_CALLBACK(cb_popup_menu), popupmenu);
157
158   gtk_widget_show_all (window);
159   gtk_main ();
160
161   return 0;
162 }
```

6.4.3　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 6.17 は UI マネージャを利用して作成したメニューの例です. 最終的にできあ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方がはるかに簡単です.

オブジェクトの階層構造

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

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

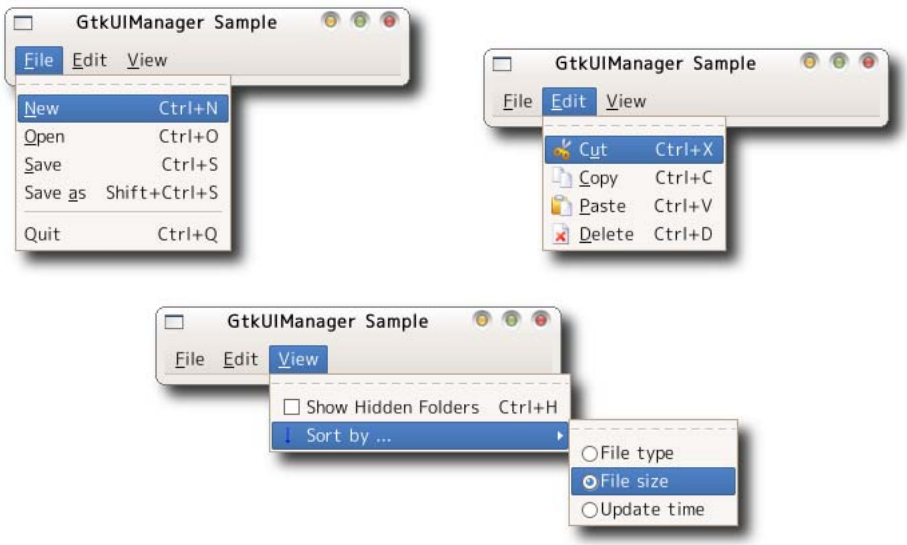


図 6.19 UI マネージャ

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.

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一つずつの定義についての説明でした. ではメニューの構成はどうやって決定したらいいでしょう. 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全て 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 6–4–3 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 6–4–3 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 GtkActionEntry entries[] = {
4    {"FileMenu", NULL, "_File"},
5    {"EditMenu", NULL, "_Edit"},
6    {"ViewMenu", NULL, "_View"},
7    {"SortMenu", GTK_STOCK_SORT_ASCENDING, "Sort␣by␣..."},
8    {"New", NULL, "_New", "<control>N", "Create␣a␣new␣file", NULL},
9    {"Open", NULL, "_Open", "<control>O", "Open␣a␣file", NULL},
10   {"Save", NULL, "_Save", "<control>S", "Save␣a␣file", NULL},
11   {"SaveAs", NULL, "Save␣_as", "<shift><control>S", NULL, NULL},
12   {"Quit", NULL, "_Quit", "<control>Q", "Quit␣a␣program", gtk_main_quit},
13   {"Cut", GTK_STOCK_CUT, _("C_ut"), "<control>X", NULL, NULL},
14   {"Copy", GTK_STOCK_COPY, "_Copy", "<control>C", NULL, NULL},
15   {"Paste", GTK_STOCK_PASTE, "_Paste", "<control>V", NULL, NULL},
16   {"Delete", GTK_STOCK_DELETE, "_Delete", "<control>D", NULL, NULL}
17 };
18
19 static GtkToggleActionEntry toggle_entries[] = {
20   {"ShowHidden", NULL, "Show␣Hidden␣Folders", "<control>H",
21    NULL, NULL, FALSE}
22 };
23
24 enum {
25   SORT_FILE_TYPE,
26   SORT_FILE_SIZE,
27   SORT_UPDATE_TIME
28 };
29
30 static GtkRadioActionEntry radio_entries[] = {
31   {"FileType", NULL, "File␣type", NULL, NULL, SORT_FILE_TYPE},
32   {"FileSize", NULL, "File␣size", NULL, NULL, SORT_FILE_SIZE},
33   {"UpdateTime", NULL, "Update␣time", NULL, NULL, SORT_UPDATE_TIME}
34 };
35
36 static guint n_entries = G_N_ELEMENTS (entries);
37 static guint n_toggle_entries = G_N_ELEMENTS (toggle_entries);
38 static guint n_radio_entries = G_N_ELEMENTS (radio_entries);
39
40 static const gchar *ui_info =
41 "<ui>"
42 "␣␣<menubar␣name='MenuBar'>"
```

```
43 "␣␣␣␣<menu␣action='FileMenu'>"
44 "␣␣␣␣␣␣<menuitem␣action='New'/>"
45 "␣␣␣␣␣␣<menuitem␣action='Open'/>"
46 "␣␣␣␣␣␣<menuitem␣action='Save'/>"
47 "␣␣␣␣␣␣<menuitem␣action='SaveAs'/>"
48 "␣␣␣␣␣␣<separator/>"
49 "␣␣␣␣␣␣<menuitem␣action='Quit'/>"
50 "␣␣␣␣</menu>"
51 "␣␣␣␣<menu␣action='EditMenu'>"
52 "␣␣␣␣␣␣<menuitem␣action='Cut'/>"
53 "␣␣␣␣␣␣<menuitem␣action='Copy'/>"
54 "␣␣␣␣␣␣<menuitem␣action='Paste'/>"
55 "␣␣␣␣␣␣<menuitem␣action='Delete'/>"
56 "␣␣␣␣</menu>"
57 "␣␣␣␣<menu␣action='ViewMenu'>"
58 "␣␣␣␣␣␣<menuitem␣action='ShowHidden'/>"
59 "␣␣␣␣␣␣<menu␣action='SortMenu'>"
60 "␣␣␣␣␣␣␣␣<menuitem␣action='FileType'/>"
61 "␣␣␣␣␣␣␣␣<menuitem␣action='FileSize'/>"
62 "␣␣␣␣␣␣␣␣<menuitem␣action='UpdateTime'/>"
63 "␣␣␣␣␣␣</menu>"
64 "␣␣␣␣</menu>"
65 "␣␣</menubar>"
66 "</ui>";
67
68 static void activate_radio_action (GtkAction      *action,
69                                    GtkRadioAction *current) {
70   g_print ("Radio␣action␣\"%s\"␣selected\n",
71            gtk_action_get_name (GTK_ACTION (current)));
72 }
73
74 static GtkWidget* create_menu (GtkWidget *parent) {
75   GtkUIManager          *ui;
76   GtkActionGroup        *actions;
77
78   actions = gtk_action_group_new ("Actions");
79   gtk_action_group_add_actions (actions, entries, n_entries, NULL);
80   gtk_action_group_add_toggle_actions (actions,
81                                        toggle_entries, n_toggle_entries,
82                                        NULL);
83   gtk_action_group_add_radio_actions (actions,
84                                       radio_entries, n_radio_entries,
85                                       SORT_FILE_TYPE,
86                                       G_CALLBACK (activate_radio_action),
87                                       NULL);
88
89   ui = gtk_ui_manager_new ();
90   gtk_ui_manager_insert_action_group (ui, actions, 0);
91   gtk_ui_manager_set_add_tearoffs (ui, TRUE);
92   gtk_window_add_accel_group (GTK_WINDOW(parent),
```

```
93                                  gtk_ui_manager_get_accel_group (ui));
94    gtk_ui_manager_add_ui_from_string (ui, ui_info, -1, NULL);
95
96    return gtk_ui_manager_get_widget (ui, "/MenuBar");
97  }
98
99  int main (int argc, char **argv) {
100   GtkWidget      *window;
101   GtkWidget      *menubar;
102
103   gtk_init (&argc, &argv);
104   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
105   gtk_window_set_title  (GTK_WINDOW(window), "GtkUIManager␣Sample");
106   gtk_widget_set_size_request (window, 300, -1);
107   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
108   g_signal_connect (G_OBJECT(window), "destroy",
109                     G_CALLBACK(gtk_main_quit), NULL);
110
111   menubar = create_menu (window);
112   gtk_container_add (GTK_CONTAINER(window), menubar);
113
114   gtk_widget_show_all (window);
115   gtk_main ();
116
117   return 0;
118 }
```

6.5　ダイアログ

ダイアログには, ユーザが比較的自由にレイアウトを設定できる汎用的は GtkDialog から特殊な用途に使用するための GtkMessageDialog や GtkFileSelection などがあります. 本節では, これらのダイアログついて紹介します.

6.5.1　ダイアログボックス

GtkDialog はダイアログの中でも最も汎用的なウィジェットで, 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
    +----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にするには次の 2 つの関数を使用します.

- 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 ではダイアログの幾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,
```

図 6.20 ダイアログ

```
                                    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
{
  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 で設定することができます. また現在のセパレータの設定を関数 gtk_dialog_get_has_sesparator で取得することができます.

```
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 6–5–1 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 6.21 左のウィンドウが表示されます. 保存ボタンを押すと, 図 6.21 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.

ソース 6–5–1 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 Confermation",
                                        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_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;
```

図 6.21 ダイアログウィジェットのサンプルプログラム

```
38   GtkWidget      *label;
39   GtkWidget      *entry;
40   GtkWidget      *button;
41
42   gtk_init (&argc, &argv);
43   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
44   gtk_window_set_title  (GTK_WINDOW(window), "GtkDialog␣Sample");
45   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
46   g_signal_connect (G_OBJECT(window), "destroy",
47                     G_CALLBACK(gtk_main_quit), NULL);
48
49   hbox = gtk_hbox_new (FALSE, 5);
50   gtk_container_add (GTK_CONTAINER(window), hbox);
51
52   button = gtk_button_new_from_stock (GTK_STOCK_SAVE);
53   g_signal_connect (G_OBJECT(button), "clicked",
54                     G_CALLBACK(cb_button), (gpointer) window);
55   gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
56
57   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
58   g_signal_connect (G_OBJECT(button), "clicked",
59                     G_CALLBACK(gtk_main_quit), NULL);
60   gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
61
62   gtk_widget_show_all (window);
63   gtk_main ();
64
65   return 0;
66 }
```

6.5.2　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することができます. さらに GtkButtonsType によってボタンのレイアウトを簡単に設定することができます.

オブジェクトの階層構造

```
GObject
    +----GtkObject
           +----GtkWidget
                  +----GtkContainer
                         +----GtkBin
                                +----GtkWindow
                                       +----GtkDialog
                                              +----GtkMessageDialog
```

表 6.11 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します (図 6.22 左上).
GTK_MESSAGE_WARNING	警告ダイアログを作成します (図 6.22 右上).
GTK_MESSAGE_QUESTION	質問ダイアログを作成します (図 6.22 左下).
GTK_MESSAGE_ERROR	エラーダイアログを作成します (図 6.22 右下).

ウィジェットの作成

GtkMessageDialog ウィジェットを作成にするには次の 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 6.22 を参照).

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

GtkMessageType は次のように定義されています (表 6.11 参照).

```
typedef enum
{
  GTK_MESSAGE_INFO,
  GTK_MESSAGE_WARNING,
  GTK_MESSAGE_QUESTION,
  GTK_MESSAGE_ERROR
} GtkMessageType;
```

図 6.22 メッセージダイアログ

ボタンタイプは次のように定義されています.

```
typedef enum
{
  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することが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することができます.

```
void gtk_message_dialog_set_markup (GtkMessageDialog *message_dialog,
                                    const gchar      *str);
```

サンプルプログラム

ソース 6–5–2 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.

ソース 6–5–2 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 show_dialog (GtkButton *button,
6                          gpointer  data) {
7   GtkWidget             *dialog;
8   GtkButtonsType        btype[] = {GTK_BUTTONS_CLOSE,
```

図 6.23　メッセージダイアログのサンプルプログラム

```
9                                       GTK_BUTTONS_OK_CANCEL,
10                                      GTK_BUTTONS_YES_NO,
11                                      GTK_BUTTONS_OK};
12    GtkMessageType        mtype = (GtkMessageType) data;
13    gchar                 *string[] = {"GTK_MESSAGE_INFO",
14                                       "GTK_MESSAGE_WARNING",
15                                       "GTK_MESSAGE_QUESTION",
16                                       "GTK_MESSAGE_ERROE"};
17    gint                  result;
18
19    dialog =
20      gtk_message_dialog_new (GTK_WINDOW(window),
21                              GTK_DIALOG_MODAL |
22                              GTK_DIALOG_DESTROY_WITH_PARENT,
23                              mtype,
24                              btype[mtype],
25                              "This␣dialog␣is␣a␣GtkMessageDialog␣sample\n"
26                              "(GtkMessageType␣=␣%s)",
27                              string[mtype]);
28    result = gtk_dialog_run (GTK_DIALOG(dialog));
29
30    switch (result) {
31    case GTK_RESPONSE_OK:
32      g_printf ("GTK_RESPONSE_OK␣is␣recieved.\n");
33      break;
34    case GTK_RESPONSE_CANCEL:
35      g_printf ("GTK_RESPONSE_CANCEL␣is␣recieved.\n");
36      break;
37    case GTK_RESPONSE_CLOSE:
38      g_printf ("GTK_RESPONSE_CLOSE␣is␣recieved.\n");
39      break;
40    case GTK_RESPONSE_YES:
41      g_printf ("GTK_RESPONSE_YES␣is␣recieved.\n");
42      break;
43    case GTK_RESPONSE_NO:
44      g_printf ("GTK_RESPONSE_NO␣is␣recieved.\n");
45      break;
46    default:
47      g_printf ("Another␣response␣is␣recieved.\n");
48      break;
49    }
50    gtk_widget_destroy (dialog);
51  }
52
53  int main (int argc, char **argv) {
54    GtkWidget     *hbox;
55    GtkWidget     *button;
56    gchar         *stock[] = {GTK_STOCK_DIALOG_INFO,
57                              GTK_STOCK_DIALOG_WARNING,
58                              GTK_STOCK_DIALOG_QUESTION,
```

```
59                                  GTK_STOCK_DIALOG_ERROR};
60    int           n;
61
62    gtk_init (&argc, &argv);
63    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
64    gtk_window_set_title  (GTK_WINDOW(window), "GtkMessageDialog␣Sample");
65    gtk_container_set_border_width (GTK_CONTAINER(window), 5);
66    g_signal_connect (G_OBJECT(window), "destroy",
67                      G_CALLBACK(gtk_main_quit), NULL);
68
69    hbox = gtk_hbox_new (TRUE, 5);
70    gtk_container_add (GTK_CONTAINER(window), hbox);
71
72    for (n = 0; n < 4; n++) {
73      button = gtk_button_new_from_stock (stock[n]);
74      g_signal_connect (G_OBJECT(button), "clicked",
75                        G_CALLBACK(show_dialog), (gpointer) n);
76      gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
77    }
78    button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
79    g_signal_connect (G_OBJECT(button), "clicked",
80                      G_CALLBACK(gtk_main_quit), NULL);
81    gtk_box_pack_start (GTK_BOX(hbox), button, TRUE, TRUE, 0);
82
83    gtk_widget_show_all (window);
84    gtk_main ();
85
86    return 0;
87  }
```

6.5.3　ファイル選択ダイアログ

ファイル選択ダイアログ (GtkFileSelection) は, ファイルシステムからファイルを選択するためのダイアログです (図 6.24).

オブジェクトの階層構造

```
GObject
    +----GtkObject
          +----GtkWidget
                +----GtkContainer
                      +----GtkBin
                            +----GtkWindow
                                  +----GtkDialog
                                        +----GtkFileSelection
```

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_selection_new を使用します. 関数の引数にはダイアログのタイトルバーに表示するタイトルを指定します.

```
GtkWidget* gtk_file_selection_new (const gchar *title);
```

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_selection_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したり, 関数 gtk_file_selection_set_filename によって, 指定したファイル名をエントリに設定することができます.

```
G_CONST_RETURN gchar*
gtk_file_selection_get_filename (GtkFileSelection *filesel);
void gtk_file_selection_set_filename (GtkFileSelection *filesel,
                                      const gchar      *filename);
```

- 複数ファイルの選択の有無

  ダイアログ右のファイル名リストをクリックすることで, ファイルを選択することができます. このとき, 関数 gtk_file_selection_set_select_multiple によって, 複数のファイルを選択することができるかどうか設定することができます. また, 関数 gtk_file_selection_get_select_multiple によって現在の設定を取得することができます.

```
void gtk_file_selection_set_select_multiple
                                  (GtkFileSelection *filesel,
                                   gboolean         select_multiple);
gboolean
gtk_file_selection_get_select_multiple (GtkFileSelection *filesel);
```

  選択された複数のファイル名を取得するには, 関数 gtk_file_selection_get_selections を使用します.

```
gchar**
gtk_file_selection_get_selections (GtkFileSelection *filesel);
```

- オペーレーションボタンの表示の有無

図 6.24 ファイル選択ダイアログ

ファイル選択ダイアログには標準でファイルオペレーションボタン (「新しいフォルダ」, 「ファイルの削除」, 「ファイル名の変更」) が表示されます. このボタン表示/非表示を関数 gtk_file_selection_hide_fileop_buttons と, 関数 gtk_file_selection_show_fileop_buttons で設定することができます.

```
void
gtk_file_selection_hide_fileop_buttons (GtkFileSelection *filesel);
void
gtk_file_selection_show_fileop_buttons (GtkFileSelection *filesel);
```

サンプルプログラム

ファイル選択ダイアログを使用する多くのケースは, ファイル名の指定をダイアログを使って簡単に行いたい場合だと考えられます. 別のウィンドウ上のエントリにダイアログ上で選択したファイル名を設定するにはどうしたらいいでしょうか.

ダイアログの動作としては, OK ボタンが押されたら, 選択されているファイル名を取得して, 何らかの動作を実行し, キャンセルボタンが押されたら, ダイアログを閉じるという動作が一般的です. ダイアログの OK ボタンとキャンセルボタンはファイル選択ダイアログのメンバ ok_button, cancel_button として取得することができます. これらのウィジェットに対してコールバック関数を設定して, それぞれのボタンが押された場合の動作を設定します.

ソース 6–5–3 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. プログラムを実行すると, 図 6.25 上段のウィンドウが表示されます. 開くボタンを押すと, ファイル選択ダイアログ (図 6.25 中段) が表示されます. ファイルを選択, もしくは入力して OK ボタンを押すと, 選択したファイル名がメインウィンドウのエントリに反映されます (図 6.25 下段).

ソース 6–5–3 ファイル選択ダイアログウィジェットのサンプルプログラム：gtkfileselection-sample.c

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

static void filesel_ok (GtkWidget *widget, gpointer data) {
  GtkFileSelection      *filesel;
  GtkEntry              *entry;

  filesel = GTK_FILE_SELECTION(data);
  entry   = GTK_ENTRY(g_object_get_data (G_OBJECT(filesel), "entry"));

  gtk_entry_set_text (entry, gtk_file_selection_get_filename (filesel));
  gtk_window_set_modal (GTK_WINDOW(filesel), FALSE);
  gtk_widget_destroy (GTK_WIDGET(filesel));
}

static void filesel_cancel (GtkWidget *widget, gpointer data) {
  GtkFileSelection      *filesel;

  filesel = GTK_FILE_SELECTION(data);

  gtk_window_set_modal (GTK_WINDOW(filesel), FALSE);
```

```
21    gtk_widget_destroy (GTK_WIDGET(filesel));
22  }
23
24  static void cb_button (GtkButton *button, gpointer data) {
25    GtkWidget      *filesel;
26    GtkWidget      *parent;
27    GtkWidget      *entry;
28
29    parent = GTK_WIDGET(g_object_get_data (G_OBJECT(data), "parent"));
30    entry  = GTK_WIDGET(data);
31
32    filesel = gtk_file_selection_new ("File␣Selection␣Dialog");
33    g_object_set_data (G_OBJECT(filesel), "entry", (gpointer) entry);
34
35    g_signal_connect (G_OBJECT(GTK_FILE_SELECTION(filesel)->ok_button),
36                      "clicked", G_CALLBACK(filesel_ok),
37                      (gpointer) filesel);
38    g_signal_connect (G_OBJECT(GTK_FILE_SELECTION(filesel)->cancel_button),
39                      "clicked", G_CALLBACK (filesel_cancel),
40                      (gpointer) filesel);
41    gtk_window_set_transient_for (GTK_WINDOW(filesel), GTK_WINDOW(parent));
42    gtk_window_set_modal (GTK_WINDOW(filesel), TRUE);
43
44    gtk_widget_show (filesel);
45  }
```

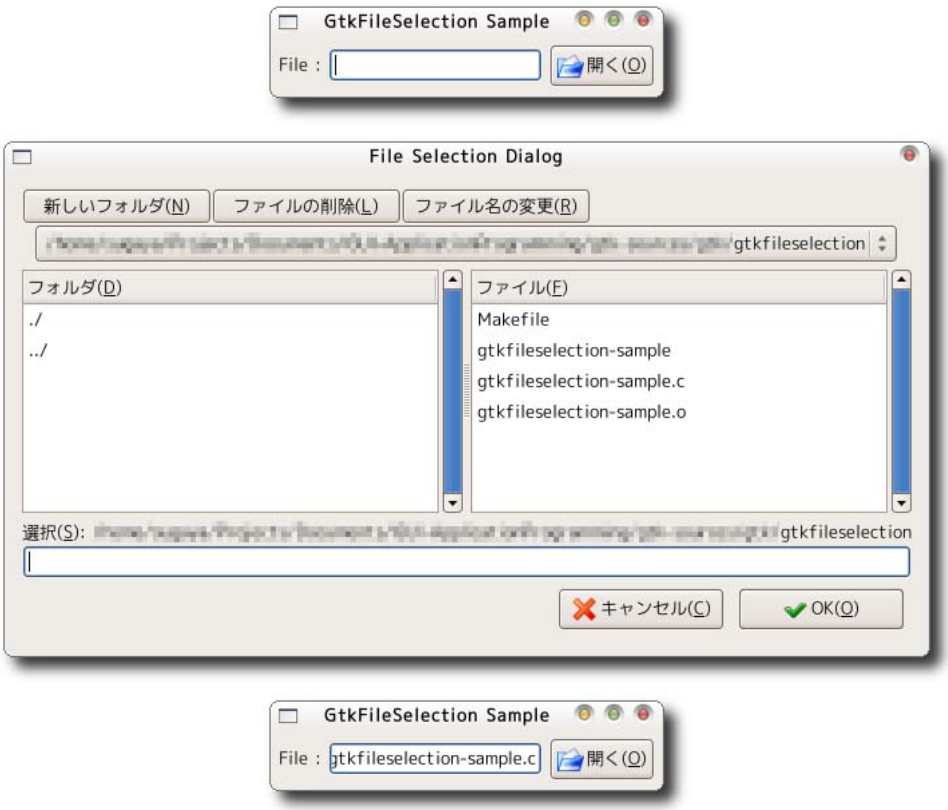


図 6.25 ファイル選択ダイアログウィジェットのサンプルプログラム

```
46
47 int main (int argc, char **argv) {
48   GtkWidget     *window;
49   GtkWidget     *hbox;
50   GtkWidget     *label;
51   GtkWidget     *entry;
52   GtkWidget     *button;
53
54   gtk_init (&argc, &argv);
55   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
56   gtk_window_set_title  (GTK_WINDOW(window), "GtkFileSelection␣Sample");
57   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
58   g_signal_connect (G_OBJECT(window), "destroy",
59                     G_CALLBACK(gtk_main_quit), NULL);
60
61   hbox = gtk_hbox_new (FALSE, 5);
62   gtk_container_add (GTK_CONTAINER(window), hbox);
63
64   label = gtk_label_new ("File␣:");
65   gtk_box_pack_start (GTK_BOX(hbox), label, FALSE, FALSE, 0);
66
67   entry = gtk_entry_new ();
68   gtk_box_pack_start (GTK_BOX(hbox), entry, TRUE, TRUE, 0);
69   g_object_set_data (G_OBJECT(entry), "parent", (gpointer) window);
70
71   button = gtk_button_new_from_stock (GTK_STOCK_OPEN);
72   g_signal_connect (G_OBJECT(button), "clicked",
73                     G_CALLBACK(cb_button), (gpointer) entry);
74   gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
75
76   gtk_widget_show_all (window);
77   gtk_main ();
78
79   return 0;
80 }
```

6.5.4　新しいファイル選択ダイアログ

GtkFileChooser はバージョン 2.4 から追加された前節で紹介したファイル選択ダイアログよりも新しいファイル選択ダイアログです (図 6.26).

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 には次に示すように 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ための引数です．

```
typedef enum
{
  GTK_FILE_CHOOSER_ACTION_OPEN,
  GTK_FILE_CHOOSER_ACTION_SAVE,
  GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER,
  GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER
 } GtkFileChooserAction;
```

第 4 引数からはダイアログに表示するボタンの指定を行います．ボタンを指定するための引数は「ボタンのラベル」と「ボタンの種類」を組として与えます．そして, 最後の引数には NULL を終端の印として与えます．ボタンの種類は GtkResponseType から指定します。

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．例のようにボタンのラベルとして GtkStockItem を使用することができます．

```
dialog =
  gtk_file_chooser_dialog_new ("File␣Open␣Dialog",
                               GTK_WIDGET(parent),
                               GTK_FILE_CHOOSER_ACTION_OPEN,
                               GTK_STOCK_CANCEL,
```

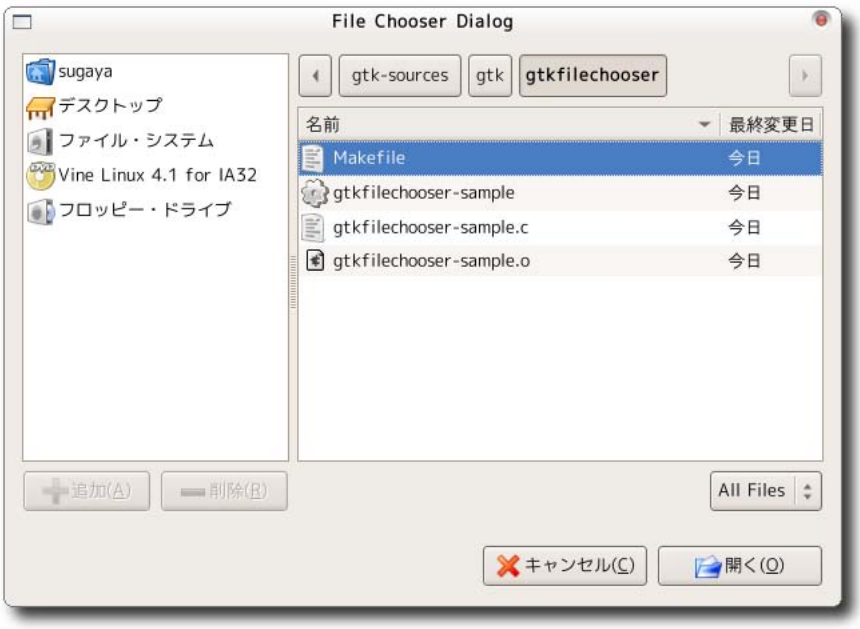


図 6.26 新しいファイル選択ダイアログ

```
                                    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したり, 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することができます.

```
gchar* gtk_file_chooser_get_filename (GtkFileChooser *chooser);

gboolean gtk_file_chooser_set_filename (GtkFileChooser *chooser,
                                        const gchar    *filename);
```

  また同様の関数に関数 gtk_file_chooser_get_uri, 関数 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:/// で始まるファイル名となります.

```
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, 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することができます. 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するには, 関数 gtk_file_chooser_get_current_folder, 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, 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することができます．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の設定を行います．

     ```
     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，具体的な使用方法についてはソース 6–5–4 を参考にして下さい．

     ```
     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 でダイアログにフィルタを登録します．

     ```
     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することができます．

```
void gtk_file_chooser_set_show_hideen (GtkFileChooser *chooser,
                                       gboolean        show_hidden);

gboolean gtk_file_chooser_get_show_hideen (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することができます．

```
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 6–5–4 に新しい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．動作はソース 6–5–3 と同様です．異なる点はフィルタを設定してダイアログに表示するファイルをコントロールできるところです．

8 行目から 59 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 23 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．そして，31 行目から 39 行目でフィルタの設定を行っています．ここでは，すべてのファイルを表示するためのフィルタと Jpeg 画像フォーマットを表示するためのフィルタを設定しています．すべてのファイルを表示するための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，41 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 6–5–4 新しいファイル選択ダイアログウィジェットのサンプルプログラム：gtkfilechooser-sample.c

```
1  #include <gtk/gtk.h>
2
3  static void filter_changed (GtkFileChooserDialog *dialog, gpointer data) {
4    g_print ("File␣filter␣changed.\n");
5  }
6
7  static void cb_button (GtkButton *button, gpointer data) {
8    GtkWidget             *dialog;
9    GtkWidget             *parent;
10   GtkEntry              *entry;
11   GtkFileFilter         *filter;
12   GtkFileChooserAction  action[] = {GTK_FILE_CHOOSER_ACTION_OPEN,
13                                     GTK_FILE_CHOOSER_ACTION_SAVE,
14                                     GTK_FILE_CHOOSER_ACTION_SELECT_FOLDER,
15                                     GTK_FILE_CHOOSER_ACTION_CREATE_FOLDER};
```

```
16  gint                    response;
17
18  parent = GTK_WIDGET(g_object_get_data (G_OBJECT(data), "parent"));
19  entry  = GTK_ENTRY(data);
20
21  dialog = gtk_file_chooser_dialog_new ("File␣Chooser␣Dialog",
22                                        GTK_WINDOW(parent),
23                                        action[0],
24                                        GTK_STOCK_CANCEL,
25                                        GTK_RESPONSE_CANCEL,
26                                        GTK_STOCK_OPEN,
27                                        GTK_RESPONSE_ACCEPT,
28                                        NULL);
29  filter = gtk_file_filter_new ();
30  gtk_file_filter_set_name (filter, "All␣Files");
31  gtk_file_filter_add_pattern (filter, "*");
32  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER(dialog), filter);
33
34  filter = gtk_file_filter_new ();
35  gtk_file_filter_set_name (filter, "JPEG");
36  gtk_file_filter_add_mime_type (filter, "image/jpeg");
37  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER(dialog), filter);
38
39  filter = gtk_file_filter_new ();
40  gtk_file_filter_set_name (filter, "PNG");
41  gtk_file_filter_add_mime_type (filter, "image/png");
42  gtk_file_chooser_add_filter (GTK_FILE_CHOOSER(dialog), filter);
43
44  g_signal_connect (dialog, "notify::filter",
45                    G_CALLBACK(filter_changed), NULL);
46
47  gtk_widget_show_all (dialog);
48
49  response = gtk_dialog_run (GTK_DIALOG(dialog));
50  if (response == GTK_RESPONSE_ACCEPT) {
51    gchar *filename;
52    gchar *folder;
53
54    filename = gtk_file_chooser_get_filename (GTK_FILE_CHOOSER(dialog));
55    folder = gtk_file_chooser_get_current_folder(GTK_FILE_CHOOSER(dialog));
56    g_print ("%s\n", folder);
57    g_free (folder);
58
59    gtk_entry_set_text (entry, filename);
60    g_free (filename);
61  } else if (response == GTK_RESPONSE_CANCEL) {
62    g_print ("Cancel␣button␣was␣pressed.\n");
63  } else {
64    g_print ("Another␣response␣was␣recieved.\n");
65  }
```

```
66   gtk_widget_destroy (dialog);
67 }
68
69 int main (int argc, char **argv) {
70   GtkWidget     *window;
71   GtkWidget     *hbox;
72   GtkWidget     *label;
73   GtkWidget     *entry;
74   GtkWidget     *button;
75
76   gtk_init (&argc, &argv);
77   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
78   gtk_window_set_title  (GTK_WINDOW(window), "GtkFileChooser␣Sample");
79   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
80   g_signal_connect (G_OBJECT(window), "destroy",
81                     G_CALLBACK(gtk_main_quit), NULL);
82
83   hbox = gtk_hbox_new (FALSE, 5);
84   gtk_container_add (GTK_CONTAINER(window), hbox);
85
86   label = gtk_label_new ("File␣:");
87   gtk_box_pack_start (GTK_BOX(hbox), label, FALSE, FALSE, 0);
88
89   entry = gtk_entry_new ();
90   gtk_box_pack_start (GTK_BOX(hbox), entry, TRUE, TRUE, 0);
91   g_object_set_data (G_OBJECT(entry), "parent", (gpointer) window);
92
93   button = gtk_button_new_from_stock (GTK_STOCK_OPEN);
94   g_signal_connect (G_OBJECT(button), "clicked",
95                     G_CALLBACK(cb_button), (gpointer) entry);
96   gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
97
98  gtk_widget_show_all (window);
99   gtk_main ();
100
101  return 0;
102 }
```

6.5.5 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 6.27). 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ためのダイアログです.

図 6.27 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に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);
```

- ウェブサイト

```
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 6–5–5 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 6–5–5 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 cb_show_dialog (GtkWidget *widget, gpointer data) {
4    GtkWidget             *dialog;
5    GtkAboutDialog        *about;
6    GdkPixbuf             *pixbuf;
7
8    const gchar   *authors[] = {"Yasuyuki␣Sugaya", NULL};
9    const gchar   *documenters[] = {"Yasuyuki␣Sugaya", NULL};
10   const gchar   *translators = "Yasuyuki␣Sugaya";
11
12   dialog = gtk_about_dialog_new ();
13   about = GTK_ABOUT_DIALOG(dialog);
14   gtk_about_dialog_set_name (about, "GtkAoubtDialog-Sample");
15   gtk_about_dialog_set_authors (about, authors);
16   gtk_about_dialog_set_documenters (about, documenters);
17   gtk_about_dialog_set_translator_credits (about, translators);
18   gtk_about_dialog_set_version (about, "1.0.0");
19   gtk_about_dialog_set_copyright (about, "Copyright␣(C)␣2007␣TEO␣Project");
20   gtk_about_dialog_set_comments(about,
21                                 "This␣is␣a␣GtkAboutDialog␣sample␣"
22                                 "program.");
23   gtk_about_dialog_set_website (about, "file:///...");
24
25   pixbuf = gdk_pixbuf_new_from_file ("gnome-tigert.png", NULL);
26   gtk_about_dialog_set_logo (about, pixbuf);
27
28   gtk_container_set_border_width (GTK_CONTAINER(dialog), 5);
29
30   gtk_widget_show_all (dialog);
31 }
32
33 int main (int argc, char **argv) {
34   GtkWidget     *window;
35   GtkWidget     *button;
36
37   gtk_init (&argc, &argv);
38   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
39   gtk_window_set_title (GTK_WINDOW(window), "GtkAboutDialog-Sample");
40   gtk_widget_set_size_request (window, 300, -1);
41   g_signal_connect (G_OBJECT(window), "destroy",
42                     G_CALLBACK(gtk_main_quit), NULL);
43
44   button = gtk_button_new_with_label ("Show␣About␣Dialog");
```

```
45    gtk_container_add (GTK_CONTAINER(window), button);
46    g_signal_connect (G_OBJECT(button), "clicked",
47                      G_CALLBACK(cb_show_dialog), NULL);
48
49    gtk_widget_show_all (window);
50    gtk_main ();
51
52    return 0;
53  }
```

6.6　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. 一つ目は, リストデータを表示する使い方です. 二つ目は, 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ますので, 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
  +----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係わらず, 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することができます.

```
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.

6.6.1　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 6.28) をもとに解説します. ソースコードはソース 6–6–1 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

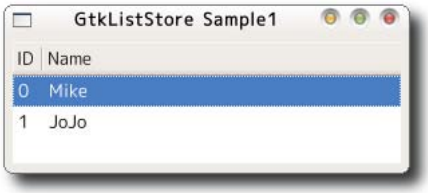


図 6.28 リストの作成

リストモデルの作成 (37–38 **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. このことについては, 次の例で詳しく説明します.

列の作成と属性の設定 (42–48 **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で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 , 関数 gtk_tree_view_column_pack_start , 関数 gtk_tree_view_column_set_attributes を呼び出す必要があります.

1. 列のタイトル設定 (44 行目)

```
void gtk_tree_view_column_set_title (GtkTreeViewColumn *tree_column,
                                     const gchar       *title);
```

2. GtkTreeViewColumn への GtkCellRenderer の登録 (45 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 (46–47 行目)

   関数 gtk_tree_view_column_set_attributes を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 6.12 に示します. この他にも列に表示するデータの種類によって様々な属性があります.

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幾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 (51–53 行目).

```
GtkTreeViewColumn*
gtk_tree_view_column_new_with_attributes (const gchar     *title,
                                          GtkCellRenderer *cell,
                                          ...);
```

表 6.12　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します.
”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します.
”pixbuf”	アイコンを表示する列に対して設定します.

列のツリービューへの追加 (48 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 (24–27 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 (24 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 (25–27 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 6–6–1 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    COLUMN_ID,
5    COLUMN_NAME,
6    N_COLUMNS
7  };
8
9  typedef struct _ListData {
10   guint          id;
11   gchar          *name;
12 } ListData;
13
14 static ListData data[] = { {0, "Mike"}, {1, "JoJo"} };
15
16 static void add_data (GtkTreeView *treeview) {
17   GtkListStore  *store;
18   GtkTreeIter   iter;
19   int           n;
20
21   store = GTK_LIST_STORE(gtk_tree_view_get_model (treeview));
22
```

```
23   for (n = 0; n < sizeof(data) / sizeof(data[0]); n++) {
24     gtk_list_store_append (store, &iter);
25     gtk_list_store_set (store, &iter,
26                         COLUMN_ID,   data[n].id,
27                         COLUMN_NAME, data[n].name, -1);
28   }
29 }
30
31 static GtkWidget* create_list_model (void) {
32   GtkWidget             *treeview;
33   GtkListStore          *liststore;
34   GtkCellRenderer       *renderer;
35   GtkTreeViewColumn     *column;
36
37   liststore = gtk_list_store_new (N_COLUMNS,
38                                   G_TYPE_UINT, G_TYPE_STRING);
39   treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL(liststore));
40   g_object_unref (liststore);
41
42   renderer = gtk_cell_renderer_text_new ();
43   column = gtk_tree_view_column_new ();
44   gtk_tree_view_column_set_title (column, "ID");
45   gtk_tree_view_column_pack_start (column, renderer, FALSE);
46   gtk_tree_view_column_set_attributes (column, renderer,
47                                        "text", COLUMN_ID, NULL);
48   gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
49
50   renderer = gtk_cell_renderer_text_new ();
51   column =
52     gtk_tree_view_column_new_with_attributes ("Name", renderer,
53                                               "text", COLUMN_NAME, NULL);
54   gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
55
56   return treeview;
57 }
58
59 int main (int argc, char **argv) {
60   GtkWidget     *window;
61   GtkWidget     *treeview;
62
63   gtk_init (&argc, &argv);
64   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
65   gtk_window_set_title  (GTK_WINDOW(window), "GtkListStore␣Sample1");
66   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
67   g_signal_connect (G_OBJECT(window), "destroy",
68                     G_CALLBACK(gtk_main_quit), NULL);
69   gtk_widget_set_size_request (window, 300, 100);
70
71   treeview = create_list_model ();
72   gtk_container_add (GTK_CONTAINER(window), treeview);
```

```
73
74    add_data (GTK_TREE_VIEW(treeview));
75
76    gtk_widget_show_all (window);
77    gtk_main ();
78
79    return 0;
80  }
```

6.6.2 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 6.29).

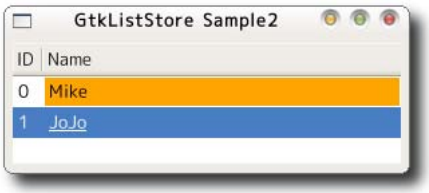


図 6.29 さまざまな列属性の設定

モデルの作成 (46–50 行目)

今回は表示する ID と名前データの他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ちますので G_TYPE_STRING と G_TYPE_UINT のデータを追加しています.

属性の設定 (64–67 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 6.13 のようになっています.

表 6.13 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;
```

テキスト属性の設定の例をソース 6–6–2 に示します.

ソース 6–6–2 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,
```

```
47                                      G_TYPE_UINT,
48                                      G_TYPE_STRING,
49                                      G_TYPE_STRING,
50                                      G_TYPE_UINT);
51    treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL(liststore));
52    g_object_unref (liststore);
53
54    renderer = gtk_cell_renderer_text_new ();
55    column =
56      gtk_tree_view_column_new_with_attributes ("ID", renderer,
57                                                "text", COLUMN_ID, NULL);
58    gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
59
60    renderer = gtk_cell_renderer_text_new ();
61    column =
62      gtk_tree_view_column_new_with_attributes ("Name", renderer,
63                                                "text", COLUMN_NAME, NULL);
64    gtk_tree_view_column_add_attribute (column, renderer,
65                                        "background", COLUMN_NAME_COLOR);
66    gtk_tree_view_column_add_attribute (column, renderer,
67                                        "underline", COLUMN_NAME_LINE);
68
69    gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
70
71    return treeview;
72 }
73
74 int main (int argc, char **argv) {
75    GtkWidget      *window;
76    GtkWidget      *treeview;
77
78    gtk_init (&argc, &argv);
79    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
80    gtk_window_set_title  (GTK_WINDOW(window), "GtkListStore␣Sample2");
81    gtk_container_set_border_width (GTK_CONTAINER(window), 5);
82    g_signal_connect (G_OBJECT(window), "destroy",
83                      G_CALLBACK(gtk_main_quit), NULL);
84    gtk_widget_set_size_request (window, 300, 100);
85
86    treeview = create_list_model ();
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 }
```

6.6.3　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次の例を使って説明します (ソース 6–6–3). 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

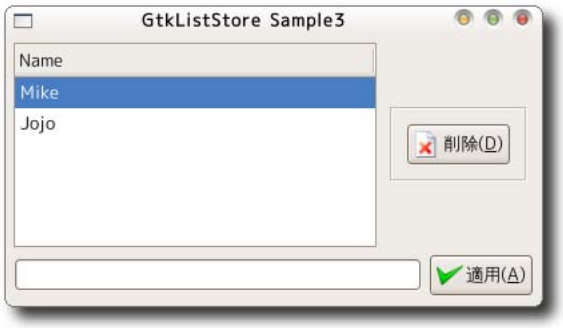


図 6.30　リストデータの削除

選択データ情報の取得 (7 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 型変数へのポインタです. 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 (11 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 (12 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 6–6–3 列データの削除：gtkliststore-sample3.c から一部抜粋

```
1  static void delete_list (GtkTreeView *treeview) {
2    GtkListStore          *store;
3    GtkTreeSelection      *selection;
4    GtkTreeIter           iter;
5    gboolean              success;
6
7    selection = gtk_tree_view_get_selection (treeview);
8    if (!selection) return;
9
10   store = GTK_LIST_STORE(gtk_tree_view_get_model (treeview));
11   success = gtk_tree_selection_get_selected (selection, NULL, &iter);
12   if (success) gtk_list_store_remove (store, &iter);
13 }
```

6.6.4 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全てのデータに対して順番にアクセスする方法を 2 種類紹介します．図 6.31 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．

関数 gtk_tree_model_iter_next を使用する方法 (26–31 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することができます．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);
```

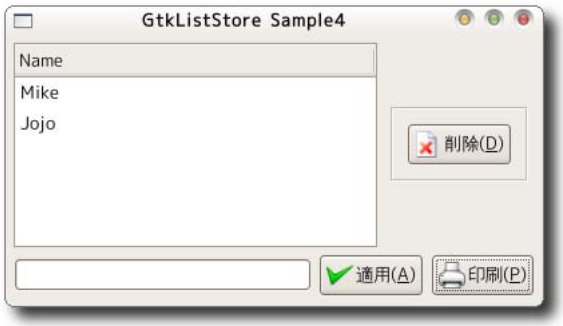


図 6.31 リストデータへの一括アクセス

関数 gtk_tree_model_foreach を使用する方法 (1–12, 33 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することができます.

```
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 6–6–4 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 list_print_func (GtkTreeModel   *model,
2                                   GtkTreePath    *path,
3                                   GtkTreeIter    *iter,
4                                   gpointer       user_data) {
5    gchar *name;
6
7    gtk_tree_model_get (model, iter, COLUMN_NAME, &name, -1);
8    g_print ("%s\n", name);
9    g_free (name);
10
11   return FALSE;
12 }
13
14 static void cb_button_print (GtkButton *button,
15                              gpointer  user_data) {
16   GtkTreeView   *treeview;
17   GtkTreeModel  *model;
18   GtkTreeIter   iter;
19   gboolean      success;
20   gchar         *name;
21
22   treeview =
23     GTK_TREE_VIEW(g_object_get_data (G_OBJECT(user_data), "treeview"));
24   model = GTK_TREE_MODEL(gtk_tree_view_get_model (treeview));
25 #if 1
26   success = gtk_tree_model_get_iter_first (model, &iter);
27   while (success) {
```

```
28      gtk_tree_model_get (model, &iter, COLUMN_NAME, &name, -1);
29      g_print ("%s\n", name);
30      success = gtk_tree_model_iter_next (model, &iter);
31    }
32  #else
33    gtk_tree_model_foreach (model, list_print_func, NULL);
34  #endif
35  }
```

6.6.5 行の入れ換え

選択された行と前後の行を入れ換えるためには, 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, その次の行を関数 gtk_tree_model_iter_next で取得します (47, 50 行目).

反対に選択された行の前の行を取得するには, 現在の行の前の行を取得する関数がありませんので, 15 行目から 33 行目のようにループを回して, 取得した行の次の行が選択された行かどうかを調べる必要があります.

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.

```
void gtk_list_store_swap (GtkListStore *store,
                          GtkTreeIter  *a,
                          GtkTreeIter  *b);
```

ソース 6–6–5 行の入れ換え：gtkliststore-sample5.c から一部抜粋

```
1  static void up_list (GtkTreeView        *treeview) {
2    GtkListStore          *store;
3    GtkTreeSelection      *selection;
4    GtkTreeIter           iter, pre_iter;
5    gboolean              success, selected;
6
7    selection = gtk_tree_view_get_selection (treeview);
8    if (!selection) return;
9
```

図 6.32 行の入れ換え

```
10    store = GTK_LIST_STORE(gtk_tree_view_get_model (treeview));
11    success = gtk_tree_model_get_iter_first (GTK_TREE_MODEL(store), &iter);
12    if (gtk_tree_selection_iter_is_selected (selection, &iter)) return;
13
14    while (success) {
15      if (gtk_tree_selection_iter_is_selected (selection, &iter)) {
16        gtk_list_store_swap (store, &pre_iter, &iter);
17        break;
18      } else {
19        pre_iter = iter;
20        success = gtk_tree_model_iter_next (GTK_TREE_MODEL(store), &iter);
21      }
22    }
23  }
24
25  static void down_list (GtkTreeView       *treeview) {
26    GtkListStore          *store;
27    GtkTreeSelection      *selection;
28    GtkTreeIter           iter, next_iter;
29    gboolean              success, selected;
30
31    selection = gtk_tree_view_get_selection (treeview);
32    if (!selection) return;
33
34    store = GTK_LIST_STORE(gtk_tree_view_get_model (treeview));
35    success = gtk_tree_selection_get_selected (selection, NULL, &iter);
36    if (success) {
37      next_iter = iter;
38      if (gtk_tree_model_iter_next (GTK_TREE_MODEL(store), &next_iter)) {
39        gtk_list_store_swap (store, &iter, &next_iter);
40      }
41    }
42  }
```

6.6.6　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 6.14, 6.15 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．

表 6.14　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ったときに発生するシグナルです．

表 6.15　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です.

```
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);
```

GtkCellRendererText のコールバック関数

まず 50–51 行目でコールバック関数の設定をしていますが, 関数内で GtkTreeView からデータを取得したいので, 関数へ引き渡すユーザデータとして変数 treeview を指定しています.

コールバック関数は 1–17 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ますので, 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 19–35 行目になります. 31 行目で現在の状態を取得し, 32–33 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 6–6–6　GtkCellRenderer に対する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 cb_name_edited (GtkCellRendererText *renderer,
2                             gchar               *path_string,
3                             gchar               *new_text,
4                             gpointer            user_data) {
```

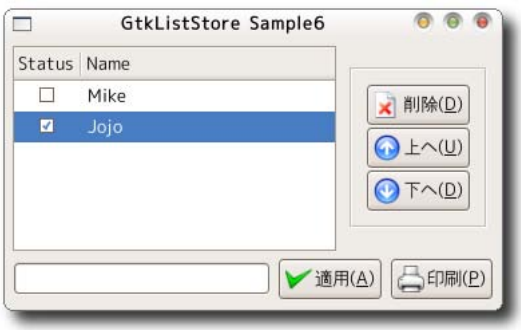


図 6.33　GtkCellRenderer に対するコールバック関数の例

```
5   GtkTreeModel  *model;
6   GtkTreeIter   iter;
7   GtkTreePath   *path;
8   gboolean      status;
9
10  model = gtk_tree_view_get_model (GTK_TREE_VIEW(user_data));
11  path  = gtk_tree_path_new_from_string (path_string);
12
13  gtk_tree_model_get_iter (model, &iter, path);
14  gtk_list_store_set (GTK_LIST_STORE(model), &iter,
15                      COLUMN_NAME, new_text, -1);
16  gtk_tree_path_free (path);
17 }
18
19 static void cb_status_toggled (GtkCellRendererToggle    *renderer,
20                                gchar                    *path_string,
21                                gpointer                 user_data) {
22  GtkTreeModel  *model;
23  GtkTreeIter   iter;
24  GtkTreePath   *path;
25  gboolean      status;
26
27  model = gtk_tree_view_get_model (GTK_TREE_VIEW(user_data));
28  path  = gtk_tree_path_new_from_string (path_string);
29
30  gtk_tree_model_get_iter (model, &iter, path);
31  gtk_tree_model_get (model, &iter, COLUMN_STATUS, &status, -1);
32  gtk_list_store_set (GTK_LIST_STORE(model), &iter,
33                      COLUMN_STATUS, !status, -1);
34  gtk_tree_path_free (path);
35 }
36
37 static GtkWidget* list_new (void) {
38  GtkWidget             *treeview;
39  GtkListStore          *liststore;
40  GtkCellRenderer       *renderer;
41  GtkTreeViewColumn     *column;
42
43  liststore = gtk_list_store_new (N_COLUMNS,
44                                  G_TYPE_BOOLEAN,
45                                  G_TYPE_STRING,
46                                  G_TYPE_BOOLEAN);
47  treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL(liststore));
48
49  renderer = gtk_cell_renderer_toggle_new ();
50  g_signal_connect (renderer, "toggled",
51                    G_CALLBACK(cb_status_toggled), treeview);
52  column =
53    gtk_tree_view_column_new_with_attributes ("Status", renderer,
54                                              "active", COLUMN_STATUS,
```

```
55                                                  NULL);
56   gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
57
58   renderer = gtk_cell_renderer_text_new ();
59   g_signal_connect (renderer, "edited",
60                     G_CALLBACK(cb_name_edited), treeview);
61   column =
62     gtk_tree_view_column_new_with_attributes ("Name", renderer,
63                                               "text", COLUMN_NAME,
64                                               "editable", COLUMN_EDITABLE,
65                                               NULL);
66   gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
67   return treeview;
68 }
```

6.6.7 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するためには, 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 6–6–7 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.

```
void
gtk_tree_view_column_set_sort_column_id(GtkTreeViewColumn *tree_column,
                                        gint               sort_column_id);
```

ソース 6–6–7 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", COLUMN_EDITABLE,
8                                               NULL);
9   gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
10  gtk_tree_view_column_set_sort_column_id (column, COLUMN_NAME);
```

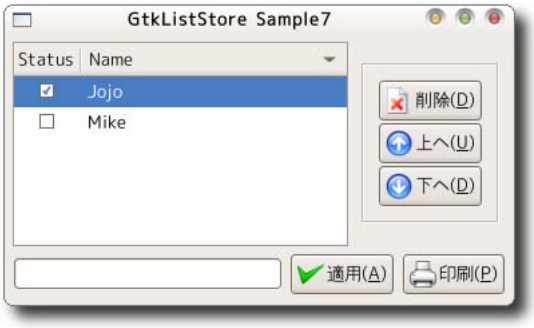


図 6.34 データのソート

6.6.8　ツリーデータの表示

ソース 6–6–8 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 6.35　ツリーデータの表示

ツリーモデルの作成 (65–66 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 (72–76 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 (42–46 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 (43 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 引数には一つ上のレベルの 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 (44–46 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 (102 **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 6–6–8 ツリーデータの表示：gtktreestore-sample1.c

```
1  #include <gtk/gtk.h>
2
3  enum {
4    COLUMN_ICON,
5    COLUMN_LABEL,
6    N_COLUMNS
7  };
8
9  typedef struct _TreeData TreeData;
10 struct _TreeData {
11   gchar          *iconname;
12   gchar          *label;
13   TreeData       *child;
14 };
15
16 static TreeData sublevel1[] = {
17   {"file.png", "file␣1-1", NULL}, {"file.png", "file␣1-2", NULL}, NULL
18 };
19
20 static TreeData sublevel2[] = {
21   {"file.png", "file␣2-1", NULL}, {"file.png", "file␣2-2", NULL}, NULL
22 };
23
24 static TreeData toplevel[] = {
25   {"folder.png", "Folder␣1", sublevel1},
26   {"folder.png", "Folder␣2", sublevel2},
27   NULL
28 };
29
```

```
30 static void add_data (GtkTreeView *treeview) {
31   GtkTreeStore  *store;
32   GtkTreeIter   iter, child_iter;
33   GdkPixbuf     *pixbuf;
34   TreeData      *top, *child;
35   int           n;
36
37   store = GTK_TREE_STORE(gtk_tree_view_get_model (treeview));
38
39   top = toplevel;
40   while (top->iconname) {
41     child = top->child;
42     pixbuf = gdk_pixbuf_new_from_file (top->iconname, NULL);
43     gtk_tree_store_append (store, &iter, NULL);
44     gtk_tree_store_set (store, &iter,
45                         COLUMN_ICON,  pixbuf,
46                         COLUMN_LABEL, top->label, -1);
47     while (child->iconname) {
48       pixbuf = gdk_pixbuf_new_from_file (child->iconname, NULL);
49       gtk_tree_store_append (store, &child_iter, &iter);
50       gtk_tree_store_set (store, &child_iter,
51                           COLUMN_ICON,  pixbuf,
52                           COLUMN_LABEL, child->label, -1);
53       child++;
54     }
55     top++;
56   }
57 }
58
59 static GtkWidget* create_tree_model (void) {
60   GtkWidget             *treeview;
61   GtkTreeStore          *treestore;
62   GtkCellRenderer       *renderer;
63   GtkTreeViewColumn     *column;
64
65   treestore = gtk_tree_store_new (N_COLUMNS,
66                                   GDK_TYPE_PIXBUF, G_TYPE_STRING);
67   treeview = gtk_tree_view_new_with_model (GTK_TREE_MODEL(treestore));
68   g_object_unref (treestore);
69
70   column = gtk_tree_view_column_new ();
71
72   renderer = gtk_cell_renderer_pixbuf_new ();
73   gtk_tree_view_column_set_title (column, "Folder");
74   gtk_tree_view_column_pack_start (column, renderer, FALSE);
75   gtk_tree_view_column_add_attribute (column, renderer, "pixbuf",
76                                       COLUMN_ICON);
77
78   renderer = gtk_cell_renderer_text_new ();
79   gtk_tree_view_column_pack_start (column, renderer, TRUE);
```

```
80    gtk_tree_view_column_add_attribute (column, renderer, "text",
81                                        COLUMN_LABEL);
82
83    gtk_tree_view_append_column (GTK_TREE_VIEW(treeview), column);
84
85    return treeview;
86  }
87
88  int main (int    argc,
89            char   **argv) {
90    GtkWidget      *window;
91    GtkWidget      *treeview;
92
93    gtk_init (&argc, &argv);
94    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
95    gtk_window_set_title  (GTK_WINDOW(window), "GtkTreeStore␣Sample1");
96    gtk_container_set_border_width (GTK_CONTAINER(window), 5);
97    g_signal_connect (G_OBJECT(window), "destroy",
98                      G_CALLBACK(gtk_main_quit), NULL);
99    gtk_widget_set_size_request (window, 280, 240);
100
101   treeview = create_tree_model ();
102   gtk_tree_view_set_headers_visible (GTK_TREE_VIEW(treeview), FALSE);
103   gtk_container_add (GTK_CONTAINER(window), treeview);
104
105   add_data (GTK_TREE_VIEW(treeview));
106
107   gtk_widget_show_all (window);
108   gtk_main ();
109
110   return 0;
111 }
```

6.6.9 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 6–6–9 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 (71–72 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 (18–34 行目)．

ソース 6–6–9 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 get_expanded_path (GtkTreeView  *treeview,
2                                 GtkTreePath  *path,
3                                 gpointer     user_data) {
4    GtkTreeModel  *model;
5    GtkTreeIter   iter;
6    GList         **list;
7    gchar         *label;
8
9    list  = user_data;
10   model = gtk_tree_view_get_model (treeview);
11
12   gtk_tree_model_get_iter (model, &iter, path);
13   gtk_tree_model_get (model, &iter, COLUMN_LABEL, &label, -1);
14
15   *list = g_list_append (*list, label);
16 }
17
18 static gboolean is_expanded (GtkTreeView *treeview, const gchar *label) {
19   GList         *list = NULL, *node;
20   gboolean      success = FALSE;
21
22   gtk_tree_view_map_expanded_rows (treeview, get_expanded_path, &list);
23   for (node = list; node; node = g_list_next (node)) {
24     if (strcmp ((char *) node->data, label) == 0) {
25       success = TRUE;
26       break;
27     }
28   }
29   g_list_foreach (list, (GFunc) g_free, NULL);
30   g_list_free (list);
31
32   return success;
33 }
34
35 static void cb_double_clicked (GtkTreeView       *treeview,
36                                GtkTreePath       *path,
37                                GtkTreeViewColumn *column,
38                                gpointer          data) {
```

```
39   GtkTreeModel  *model;
40   GtkTreeIter   iter;
41   gchar         *label;
42
43   model = gtk_tree_view_get_model (treeview);
44   if (gtk_tree_model_get_iter (model, &iter, path)) {
45     gtk_tree_model_get (model, &iter, COLUMN_LABEL, &label, -1);
46     if (is_expanded (treeview, label)) {
47       gtk_tree_view_collapse_row (treeview, path);
48     } else {
49       gtk_tree_view_expand_row (treeview, path, FALSE);
50     }
51     g_free (label);
52   }
53 }
54
55 int main (int argc, char **argv) {
56   GtkWidget     *window;
57   GtkWidget     *treeview;
58
59   gtk_init (&argc, &argv);
60   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
61   gtk_window_set_title  (GTK_WINDOW(window), "GtkTreeStore␣Sample1");
62   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
63   g_signal_connect (G_OBJECT(window), "destroy",
64                     G_CALLBACK(gtk_main_quit), NULL);
65   gtk_widget_set_size_request (window, 280, 240);
66
67   treeview = create_tree_model ();
68   g_signal_connect (treeview, "row-activated",
69                     G_CALLBACK(cb_double_clicked), NULL);
70   gtk_tree_view_set_headers_visible (GTK_TREE_VIEW(treeview), FALSE);
71   gtk_container_add (GTK_CONTAINER(window), treeview);
72
73   add_data (GTK_TREE_VIEW(treeview));
74
75   gtk_widget_show_all (window);
76   gtk_main ();
77
78   return 0;
79 }
```

6.6.10 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);
```

6.7　その他の特殊なウィジェット

6.7.1　ツールチップ

ツールチップ (GtkTooltips) 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. GtkTooltips 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るようになります (図 6.36).

ウィジェットの作成とチップスの設定

GtkTooltips を作成するには関数 gtk_tooltips_new を使用します.

```
GtkTooltips* gtk_tooltips_new (void);
```

上記の関数で作成した GtkTooltips 型の変数を利用して, 関数 gtk_tooltips_set_tip により, ウィジェットにチップスを登録します.

```
void gtk_tooltips_set_tip (GtkTooltips *tooltips,
                           GtkWidget   *widget,
                           const gchar *tip_text,
                           const gchar *tip_private);
```

第 1 引数と第 2 引数には, GtkTooltips 型の変数とチップスを表示したいウィジェットを与えます. 第 3 引数と第 4 引数には, 表示するチップスを文字列で与えます.

サンプルプログラム

ソース 6–7–1 では, 普通のボタンウィジェットに対して, ツールチップスを設定しています. 図 6.36 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, チップスが表示されます.

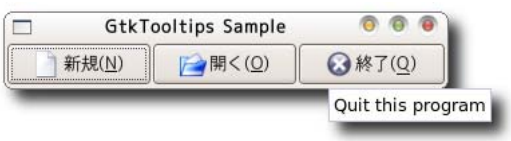


図 6.36　ツールチップス

ソース 6–7–1 ツールチップスのサンプルプログラム：gtktooltips-sample.c

```
1  #include <gtk/gtk.h>
2
3  int main (int argc, char **argv) {
4    GtkWidget     *window;
5    GtkWidget     *hbox;
6    GtkTooltips   *tooltips;
7    GtkWidget     *button;
8
9    gtk_init (&argc, &argv);
10   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
11   gtk_window_set_title  (GTK_WINDOW (window), "GtkTooltips␣Sample");
12   g_signal_connect (G_OBJECT(window), "destroy",
13                     G_CALLBACK(gtk_main_quit, NULL);
14   hbox = gtk_hbox_new (FALSE, 0);
15   gtk_container_add (GTK_CONTAINER (window), hbox);
16
17   tooltips = gtk_tooltips_new ();
18
19   button = gtk_button_new_from_stock (GTK_STOCK_NEW);
20   gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
21   gtk_tooltips_set_tip (GTK_TOOLTIPS(tooltips), button,
22                         "Run␣new␣program", "Run␣new␣program");
23
24   button = gtk_button_new_from_stock (GTK_STOCK_OPEN);
25   gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
26   gtk_tooltips_set_tip (GTK_TOOLTIPS(tooltips), button,
27                         "Open␣a␣file", "Open␣a␣file");
28
29   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
30   gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
31   gtk_tooltips_set_tip (GTK_TOOLTIPS(tooltips), button,
32                         "Quit␣this␣program", "Quit␣this␣program");
33   g_signal_connect (G_OBJECT(button), "clicked",
34                     G_CALLBACK(gtk_main_quit), NULL);
35
36   gtk_widget_show_all (window);
37   gtk_main ();
38
39   return 0;
40 }
```

6.7.2　プログレスバー

プログレスバー (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 6.37 上段左のような, 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, 図 6.37 上段右のような, 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 6.37 下段のように, バーは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
   +----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することができます.

```
void gtk_progress_bar_pulse (GtkProgressBar *pbar);
```

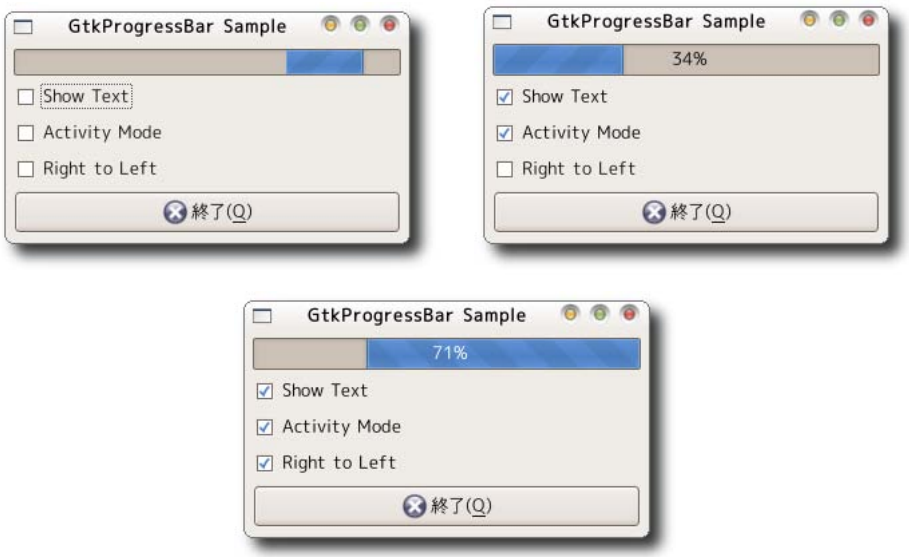


図 6.37　プログレスバー

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することができます. 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, 値を更新することができます.

```
gdouble gtk_progress_bar_get_fraction (GtkProgressBar *pbar);
```

- プログレスバーの表示方向
  プログレスバーの表示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することができます. また, 関数 gtk_progress_bar_get_orientation によって, 現在の設定を取得することができます.

```
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することができます. 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 % と表示するとよりいっそう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 6–7–2 にプログレスー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 6.37 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することができます．

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．

ソース 6–7–2 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 progressbar_update (gpointer data) {
8    GtkProgressBar        *progressbar = GTK_PROGRESS_BAR(data);
9    gdouble               new_val;
10   const gchar           *text;
11   gchar                 label[256];
12
13   if (!activity_mode) {
14     gtk_progress_bar_pulse(progressbar);
15   } else {
16     new_val = gtk_progress_bar_get_fraction (progressbar) + 0.01;
17     if (new_val > 1.0) new_val = 0.0;
18     gtk_progress_bar_set_fraction (progressbar, new_val);
19     if (show_text) {
20       sprintf (label, "%3d%s", (int) (new_val * 100.0), "%");
21       gtk_progress_bar_set_text (progressbar, label);
22     }
23   }
24   return TRUE;
25 }
26
27 static void cb_show_text (GtkToggleButton *widget, gpointer data) {
28   if (show_text) {
29     gtk_progress_bar_set_text (GTK_PROGRESS_BAR(data), "");
30     show_text = FALSE;
31   } else {
32     show_text = TRUE;
33   }
34 }
35
36 static void cb_activity_mode (GtkToggleButton *widget, gpointer data) {
37   activity_mode = gtk_toggle_button_get_active (widget);
```

```
38    if (activity_mode) {
39      gtk_progress_bar_set_fraction (GTK_PROGRESS_BAR(data), 0.0);
40    } else {
41      gtk_progress_bar_pulse (GTK_PROGRESS_BAR(data));
42    }
43  }
44
45  static void cb_orientation (GtkToggleButton *widget, gpointer data) {
46    GtkProgressBarOrientation     orientation;
47
48    orientation =
49      gtk_progress_bar_get_orientation (GTK_PROGRESS_BAR(data));
50    switch (orientation) {
51    case GTK_PROGRESS_LEFT_TO_RIGHT:
52      gtk_progress_bar_set_orientation (GTK_PROGRESS_BAR(data),
53                                        GTK_PROGRESS_RIGHT_TO_LEFT);
54      break;
55    case GTK_PROGRESS_RIGHT_TO_LEFT:
56      gtk_progress_bar_set_orientation (GTK_PROGRESS_BAR(data),
57                                        GTK_PROGRESS_LEFT_TO_RIGHT);
58      break;
59    default:
60      break;
61    }
62  }
63
64  static void cb_quit (GtkButton *widget, gpointer data) {
65    g_source_remove (timer);
66    gtk_main_quit ();
67  }
68
69  int main (int argc, char **argv) {
70    GtkWidget     *window;
71    GtkWidget     *vbox;
72    GtkWidget     *progressbar;
73    GtkWidget     *button;
74
75    gtk_init (&argc, &argv);
76    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
77    gtk_window_set_title  (GTK_WINDOW (window), "GtkProgressBar␣Sample");
78    gtk_widget_set_size_request (window, 300, -1);
79    gtk_container_set_border_width (GTK_CONTAINER(window), 5);
80    g_signal_connect (G_OBJECT(window), "destroy",
81                      G_CALLBACK(cb_quit), NULL);
82
83    vbox = gtk_vbox_new (FALSE, 5);
84    gtk_container_add (GTK_CONTAINER (window), vbox);
85
86    progressbar = gtk_progress_bar_new ();
87    gtk_box_pack_start (GTK_BOX(vbox), progressbar, FALSE, FALSE, 0);
```

```
88
89    button = gtk_check_button_new_with_label ("Show␣Text");
90    gtk_box_pack_start (GTK_BOX(vbox), button, FALSE, FALSE, 0);
91    g_signal_connect (G_OBJECT(button), "clicked",
92                      G_CALLBACK(cb_show_text), progressbar);
93
94    button = gtk_check_button_new_with_label ("Activity␣Mode");
95    gtk_box_pack_start (GTK_BOX(vbox), button, FALSE, FALSE, 0);
96    g_signal_connect (G_OBJECT(button), "clicked",
97                      G_CALLBACK(cb_activity_mode), progressbar);
98
99    button = gtk_check_button_new_with_label ("Right␣to␣Left");
100   gtk_box_pack_start (GTK_BOX(vbox), button, FALSE, FALSE, 0);
101   g_signal_connect (G_OBJECT(button), "clicked",
102                     G_CALLBACK(cb_orientation), progressbar);
103
104   button = gtk_button_new_from_stock (GTK_STOCK_QUIT);
105   gtk_box_pack_start (GTK_BOX(vbox), button, FALSE, FALSE, 0);
106   g_signal_connect (G_OBJECT(button), "clicked",
107                     G_CALLBACK(gtk_main_quit), NULL);
108
109   timer = g_timeout_add (100, progressbar_update, progressbar);
110
111   gtk_widget_show_all (window);
112   gtk_main ();
113
114   return 0;
115 }
```

6.7.3 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
   +----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);
```

6.7.4　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 6.38).

オブジェクトの階層構造

```
GObject
   +----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 で数値の範囲等の設定をして, 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 で数値の範囲等の設定をして, 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.

図 6.38　スケール

```
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もっともよく使用されるシグナルは”value-changed” シグナルです．このシグナルはスケールの値が変化したときに発生します．

”value-changed” シグナルに対するコールバック関数のプロトタイプ宣言は次のようになります．第 1 引数の型を見てもわかるように GtkScale は GtkRange のサブクラスで，シグナルも GtkRange のシグナルを継承しています．

```
gboolean user_function (GtkRange *range,
                        gpointer user_data);
```

ウィジェットのプロパティ設定

スケールウィジェットのプロパティには次の項目が存在します．

- 現在の値

  スケールの現在の値はレンジウィジェットの関数 gtk_range_get_value を使って取得することができます．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することができます． 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，現在の設定を取得することができます．

  ```
  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 6–7–3 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 6–7–3 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 cb_value_changed (GtkScale *scale, gpointer data) {
4    g_print ("value␣=␣%f\n", gtk_range_get_value (GTK_RANGE(scale)));
5  }
6
7  static void cb_button (GtkButton *button, gpointer data) {
8    g_print ("value␣=␣%f\n", gtk_range_get_value (GTK_RANGE(data)));
9  }
10
11 int main (int argc, char **argv) {
12   GtkWidget     *window;
13   GtkWidget     *hbox;
14   GtkWidget     *label;
15   GtkWidget     *scale;
16   GtkWidget     *button;
17   gdouble       min = 0.0, max = 100.0, step = 0.1;
18
19   gtk_init (&argc, &argv);
20   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
21   gtk_window_set_title  (GTK_WINDOW(window), "GtkScale␣Sample");
22   gtk_widget_set_size_request (window, 400, -1);
23   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
24   g_signal_connect (G_OBJECT(window), "destroy",
25                     G_CALLBACK(gtk_main_quit), NULL);
26
27   hbox = gtk_hbox_new (FALSE, 5);
28   gtk_container_add (GTK_CONTAINER(window), hbox);
29
30   label = gtk_label_new ("Input␣:");
31   gtk_box_pack_start (GTK_BOX(hbox), label, FALSE, FALSE, 0);
32
33   scale = gtk_hscale_new_with_range (min, max, step);
34   gtk_scale_set_digits (GTK_SCALE(scale), 2);
35   gtk_scale_set_draw_value (GTK_SCALE(scale), TRUE);
36   gtk_scale_set_value_pos (GTK_SCALE(scale), GTK_POS_BOTTOM);
37
38   g_signal_connect (G_OBJECT(scale), "value_changed",
39                     G_CALLBACK(cb_value_changed), NULL);
40   gtk_box_pack_start (GTK_BOX(hbox), scale, TRUE, TRUE, 0);
41
42   button = gtk_button_new_from_stock (GTK_STOCK_APPLY);
43   g_signal_connect (G_OBJECT(button), "clicked",
44                     G_CALLBACK(cb_button), (gpointer) scale);
```

```
45    gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
46
47    gtk_widget_show_all (window);
48    gtk_main ();
49
50    return 0;
51  }
```

6.7.5　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表示して一括して閲覧できるようなウィジェットです (図 6.39). ファイルブラウザや画像ブラウザなどの作成に適したウィジェットです.

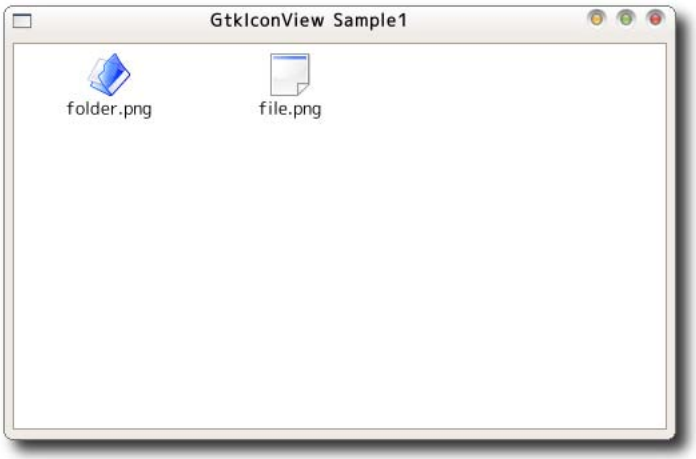


図 6.39　アイコンビューウィジェットの例

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, 関数 gtk_icon_view_new_with_model を使用します.

```
GtkWidget* gtk_icon_view_new (void);

GtkWidget* gtk_icon_view_new_with_model (GtkTreeModel *model);
```

2 つ目の関数からもわかるように, アイコンビューウィジェットは GtkTreeModel を用いて表示するデータを管理します.

表 6.16　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です.
”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です.
”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です.

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. 用途によって様々なデータを設定することが可能ですが、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中でどの項目がウィジェットに表示するラベル, 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. 詳しい説明はツリービューウィジェットの節を参照してください.

シグナルとコールバック関数

表 6.16 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 の値を取得することができます.

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 6.40 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ま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);
```

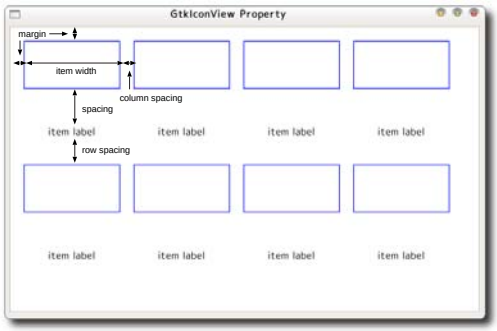


図 6.40　アイコンビューウィジェットのプロパティ

```
gint gtk_icon_view_get_column_spacing (GtkIconView *icon_view);
```

- 行の空白 (row spacing)

  アイテム間の縦の空白を設定します.

```
void gtk_icon_view_set_row_spacing (GtkIconView *icon_view,
                                    gint        row_spacing);

gint gtk_icon_view_get_row_spacing (GtkIconView *icon_view);
```

- 画像とラベルの間の空白 (spacing)

  画像とラベル間の空白を設定します.

```
void gtk_icon_view_set_spacing (GtkIconView *icon_view,
                                gint        spacing);

gint gtk_icon_view_get_spacing (GtkIconView *icon_view);
```

- ラベルの表示位置

  ラベルを画像の下に表示するか右に表示するかを設定します. 表示位置は GtkOrientation の値で指定します.

```
void gtk_icon_view_set_orientation (GtkIconView    *icon_view,
                                    GtkOrientation orientation);

GtkOrientation
gtk_icon_view_get_orientation (GtkIconView *icon_view);
```

- 並べ替えの可/不可

  アイテムが追加されたときに表示しているアイテム全体を並べ替えするかどうかを設定します.

```
void gtk_icon_view_set_reorderable (GtkIconView *icon_view,
                                    gboolean    reorderable);

gboolean gtk_icon_view_get_reorderable (GtkIconView *icon_view);
```

- 選択モード

  アイテムの選択モードを設定します. 選択モードは GtkSelectionMode の値で指定します.

```
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,               /* Nothing can be selected */
  GTK_SELECTION_SINGLE,
  GTK_SELECTION_BROWSE,
  GTK_SELECTION_MULTIPLE,
  GTK_SELECTION_EXTENDED = GTK_SELECTION_MULTIPLE /* Deprecated */
} GtkSelectionMode;
```

サンプルプログラム

ソース 6–7–4 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 6–7–4 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    COLUMN_NAME,
5    COLUMN_PIXBUF,
6    N_COLUMNS
7  };
8
9  static GdkPixbuf *file_pixbuf, *folder_pixbuf;
10 static gchar* currentdir;
11
12 static void load_pixbuf (void) {
13   file_pixbuf   = gdk_pixbuf_new_from_file ("file.png", NULL);
14   folder_pixbuf = gdk_pixbuf_new_from_file ("folder.png", NULL);
15 }
16
17 static void add_data (GtkIconView *iconview) {
18   GtkListStore  *store;
19   GDir          *dir;
20   const gchar   *name;
21   GtkTreeIter   iter;
22
23   store = GTK_LIST_STORE(gtk_icon_view_get_model (iconview));
24
25   gtk_list_store_clear (store);
26
27   gtk_list_store_append (store, &iter);
28   gtk_list_store_set (store, &iter,
29                       COLUMN_NAME, "folder.png",
30                       COLUMN_PIXBUF, folder_pixbuf, -1);
31
32   gtk_list_store_append (store, &iter);
33   gtk_list_store_set (store, &iter,
34                       COLUMN_NAME, "file.png",
35                       COLUMN_PIXBUF, file_pixbuf, -1);
36 }
37
38 static GtkWidget* create_icon_view_widget (void) {
39   GtkWidget     *iconview;
40   GtkListStore  *store;
41
```

```
42   store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING, GDK_TYPE_PIXBUF);
43   iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL(store));
44   g_object_unref (store);
45
46   return iconview;
47 }
48
49 static void cb_item_activated (GtkIconView       *iconview,
50                                GtkTreePath       *treepath,
51                                gpointer          data) {
52   GtkListStore  *store;
53   GtkTreeIter   iter;
54   gchar         *name;
55
56   store = GTK_LIST_STORE(gtk_icon_view_get_model (iconview));
57   gtk_tree_model_get_iter (GTK_TREE_MODEL(store), &iter, treepath);
58   gtk_tree_model_get (GTK_TREE_MODEL(store), &iter,
59                       COLUMN_NAME, &name, -1);
60   g_print ("item␣'%s'␣is␣clicked.\n", name);
61   g_free (name);
62 }
63
64 int main (int argc, char **argv) {
65   GtkWidget     *window;
66   GtkWidget     *scrolled_window;
67   GtkWidget     *iconview;
68
69   gtk_init (&argc, &argv);
70   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
71   gtk_window_set_title  (GTK_WINDOW(window), "GtkIconView␣Sample1");
72   gtk_widget_set_size_request (window, 500, 300);
73   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
74   g_signal_connect (G_OBJECT(window), "destroy",
75                     G_CALLBACK(gtk_main_quit), NULL);
76
77   scrolled_window = gtk_scrolled_window_new (NULL, NULL);
78   gtk_scrolled_window_set_shadow_type(GTK_SCROLLED_WINDOW(scrolled_window),
79                                       GTK_SHADOW_ETCHED_IN);
80   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW(scrolled_window),
81                                   GTK_POLICY_AUTOMATIC,
82                                   GTK_POLICY_AUTOMATIC);
83   gtk_container_add (GTK_CONTAINER(window), scrolled_window);
84
85   load_pixbuf ();
86
87   iconview = create_icon_view_widget ();
88   gtk_icon_view_set_text_column (GTK_ICON_VIEW(iconview), COLUMN_NAME);
89   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW(iconview), COLUMN_PIXBUF);
90   gtk_icon_view_set_item_width (GTK_ICON_VIEW(iconview), 128);
91   g_signal_connect (G_OBJECT(iconview), "item_activated",
```

```
92                      G_CALLBACK(cb_item_activated), NULL);
93
94    gtk_container_add (GTK_CONTAINER(scrolled_window), iconview);
95
96    add_data (GTK_ICON_VIEW(iconview));
97
98    gtk_widget_show_all (window);
99    gtk_main ();
100
101   return 0;
102 }
```

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，更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 98 行目から 102 行目でソートに関する設定を行い，64 行目から 88 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 (98–99 **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 (100–102 **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 6–7–5 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

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

enum {
  COLUMN_PATH,
  COLUMN_DISPLAY_NAME,
  COLUMN_PIXBUF,
  COLUMN_IS_DIRECTORY,
  N_COLUMNS
};

static GdkPixbuf *file_pixbuf, *folder_pixbuf;
static gchar* currentdir;

static void load_pixbuf (void) {
  file_pixbuf   = gdk_pixbuf_new_from_file ("file.png", NULL);
  folder_pixbuf = gdk_pixbuf_new_from_file ("folder.png", NULL);
}

static void add_data (GtkIconView *iconview) {
  GtkListStore  *store;
  GDir          *dir;
  const gchar   *name;
  GtkTreeIter   iter;

  store = GTK_LIST_STORE(gtk_icon_view_get_model (iconview));

  gtk_list_store_clear (store);
```

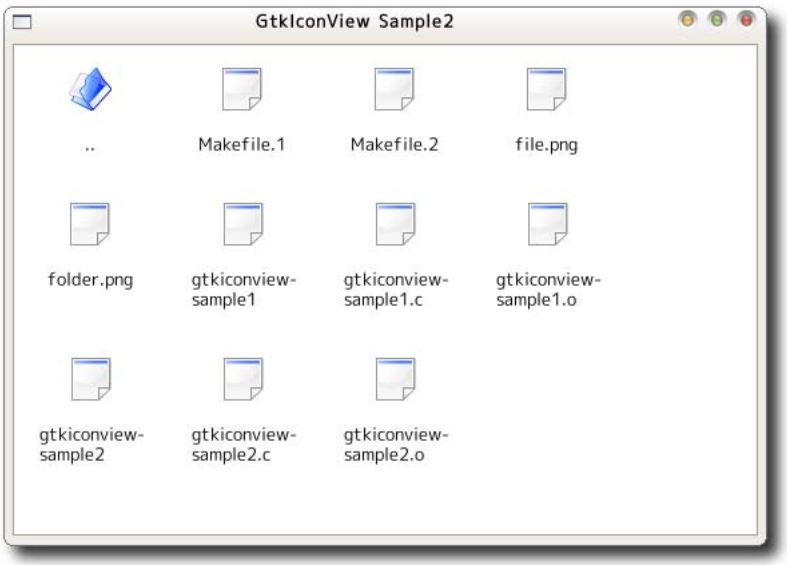


図 6.41　簡易ファイルブラウザ

```
28
29    dir = g_dir_open (currentdir, 0, NULL);
30    if (!dir) return;
31
32    while (name = g_dir_read_name (dir)) {
33      gchar       *path, *display_name;
34      gboolean    is_dir;
35
36      if (name[0] != '.') {
37        path = g_build_filename (currentdir, name, NULL);
38        is_dir = g_file_test (path, G_FILE_TEST_IS_DIR);
39        display_name = g_filename_to_utf8 (name, -1, NULL, NULL, NULL);
40
41        gtk_list_store_append (store, &iter);
42        gtk_list_store_set (store, &iter,
43                            COLUMN_PATH, path,
44                            COLUMN_DISPLAY_NAME, display_name,
45                            COLUMN_IS_DIRECTORY, is_dir,
46                            COLUMN_PIXBUF,
47                            (is_dir) ? folder_pixbuf : file_pixbuf, -1);
48        g_free (path);
49        g_free (display_name);
50      }
51    }
52    g_dir_close (dir);
53
54    if (g_utf8_collate (currentdir, "/") != 0) {
55      gtk_list_store_append (store, &iter);
56      gtk_list_store_set (store, &iter,
57                          COLUMN_PATH, g_path_get_dirname (currentdir),
58                          COLUMN_DISPLAY_NAME, "..",
59                          COLUMN_IS_DIRECTORY, TRUE,
60                          COLUMN_PIXBUF, folder_pixbuf, -1);
61    }
62  }
63
64  static gint sort_func (GtkTreeModel *model,
65                         GtkTreeIter  *a,
66                         GtkTreeIter  *b,
67                         gpointer     data) {
68    gboolean      is_dir_a, is_dir_b;
69    gchar         *name_a, *name_b;
70    int           result;
71
72    gtk_tree_model_get (model, a,
73                        COLUMN_IS_DIRECTORY, &is_dir_a,
74                        COLUMN_DISPLAY_NAME, &name_a,
75                        -1);
76    gtk_tree_model_get (model, b,
77                        COLUMN_IS_DIRECTORY, &is_dir_b,
```

```
78                        COLUMN_DISPLAY_NAME, &name_b,
79                        -1);
80    if (!is_dir_a && is_dir_b) {
81      result = 1;
82    } else if (is_dir_a && !is_dir_b) {
83      result = -1;
84    } else {
85      result = g_utf8_collate (name_a, name_b);
86    }
87    return result;
88  }
89
90  static GtkWidget* create_icon_view_widget (void) {
91    GtkWidget     *iconview;
92    GtkListStore  *store;
93
94    currentdir = g_get_current_dir ();
95    store = gtk_list_store_new(N_COLUMNS,
96                               G_TYPE_STRING, G_TYPE_STRING, GDK_TYPE_PIXBUF,
97                               G_TYPE_BOOLEAN);
98    gtk_tree_sortable_set_default_sort_func (GTK_TREE_SORTABLE(store),
99                                             sort_func, NULL, NULL);
100   gtk_tree_sortable_set_sort_column_id
101     (GTK_TREE_SORTABLE (store), GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID,
102      GTK_SORT_ASCENDING);
103
104   iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL(store));
105   g_object_unref (store);
106
107   return iconview;
108 }
109
110 static void cb_item_activated (GtkIconView *iconview,
111                                GtkTreePath *treepath,
112                                gpointer    data) {
113   GtkListStore  *store;
114   GtkTreeIter   iter;
115   gchar         *path;
116   gboolean      is_dir;
117
118   store = GTK_LIST_STORE(gtk_icon_view_get_model (iconview));
119   gtk_tree_model_get_iter (GTK_TREE_MODEL(store), &iter, treepath);
120   gtk_tree_model_get(GTK_TREE_MODEL(store), &iter,
121                      COLUMN_PATH, &path, COLUMN_IS_DIRECTORY, &is_dir, -1);
122   if (is_dir) {
123     g_free (currentdir);
124     currentdir = g_strdup (path);
125     add_data (iconview);
126   }
127   g_free (path);
```

```
128 }
129
130 int main (int argc, char **argv) {
131   GtkWidget     *window;
132   GtkWidget     *scrolled_window;
133   GtkWidget     *iconview;
134
135   gtk_init (&argc, &argv);
136   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
137   gtk_window_set_title  (GTK_WINDOW(window), "GtkIconView␣Sample2");
138   gtk_widget_set_size_request (window, 500, 300);
139   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
140   g_signal_connect (G_OBJECT(window), "destroy",
141                     G_CALLBACK(gtk_main_quit), NULL);
142
143   scrolled_window = gtk_scrolled_window_new (NULL, NULL);
144   gtk_scrolled_window_set_shadow_type(GTK_SCROLLED_WINDOW(scrolled_window),
145                                       GTK_SHADOW_ETCHED_IN);
146   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW(scrolled_window),
147                                   GTK_POLICY_AUTOMATIC,
148                                   GTK_POLICY_AUTOMATIC);
149   gtk_container_add (GTK_CONTAINER(window), scrolled_window);
150
151   load_pixbuf ();
152
153   iconview = create_icon_view_widget ();
154   gtk_icon_view_set_text_column (GTK_ICON_VIEW(iconview),
155                                  COLUMN_DISPLAY_NAME);
156   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW(iconview), COLUMN_PIXBUF);
157   gtk_icon_view_set_item_width (GTK_ICON_VIEW(iconview), 80);
158   g_signal_connect (G_OBJECT(iconview), "item_activated",
159                     G_CALLBACK(cb_item_activated), NULL);
160
161   gtk_icon_view_set_margin (GTK_ICON_VIEW(iconview), 16);
162   gtk_icon_view_set_item_width (GTK_ICON_VIEW(iconview), 80);
163   gtk_icon_view_set_spacing (GTK_ICON_VIEW(iconview), 16);
164   gtk_icon_view_set_spacing (GTK_ICON_VIEW(iconview), 16);
165   gtk_icon_view_set_column_spacing (GTK_ICON_VIEW(iconview), 32);
166   gtk_icon_view_set_row_spacing (GTK_ICON_VIEW(iconview), 32);
167
168   gtk_container_add (GTK_CONTAINER(scrolled_window), iconview);
169
170   add_data (GTK_ICON_VIEW(iconview));
171
172   gtk_widget_show_all (window);
173   gtk_main ();
174
175   return 0;
176 }
```

6.7.6　エントリ補間ウィジェット

エントリ補間ウィジェット (GtkEntryCompletion) は, エントリに入力を行う際に入力文字列の補間を行うウィジェットです. よく利用される文字列補間の例は, 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間するものです.

オブジェクトの階層構造

```
GObject
    +----GtkEntryCompletion
```

ウィジェットの作成

エントリ補間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. 更に関数 gtk_entry_completion_set_text_column を使用して何番目の項目が文字列補間用のテキストであるか設定します. 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 6–7–6 にエントリ補間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 つの文字列を補間用の文字列として予め登録しています. 入力する文字列によって図 6.42 のように補間文字列の候補が表示されます.

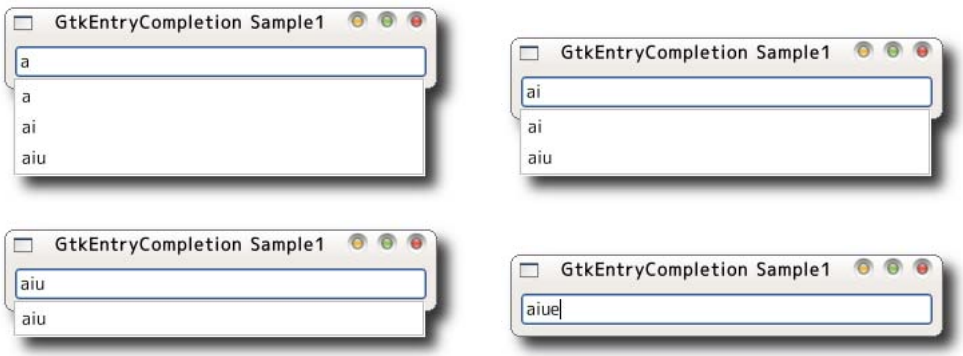


図 6.42　エントリ補間ウィジェットのサンプルプログラム 1

ソース 6–7–6 エントリ補間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    COLUMN_COMPLETION_TEXT,
5    N_COLUMNS
6  };
7
8  static GtkEntryCompletion* create_completion_widget (void) {
9    GtkEntryCompletion    *completion;
10   GtkListStore          *store;
11   GtkTreeIter           iter;
12
13   completion = gtk_entry_completion_new ();
14   store = gtk_list_store_new (N_COLUMNS, G_TYPE_STRING);
15   gtk_entry_completion_set_model (completion, GTK_TREE_MODEL(store));
16   g_object_unref (store);
17   gtk_entry_completion_set_text_column (completion, 0);
18
19   gtk_list_store_append (store, &iter);
20   gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "a", -1);
21
22   gtk_list_store_append (store, &iter);
23   gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "ai", -1);
24
25   gtk_list_store_append (store, &iter);
26   gtk_list_store_set (store, &iter, COLUMN_COMPLETION_TEXT, "aiu", -1);
27
28   return completion;
29 }
30
31 int main (int argc, char **argv) {
32   GtkWidget             *window;
33   GtkWidget             *entry;
34   GtkEntryCompletion    *completion;
35
36   gtk_init (&argc, &argv);
37   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
38   gtk_window_set_title (GTK_WINDOW(window), "GtkEntryCompletion␣Sample␣1");
39   gtk_widget_set_size_request (window, 320, -1);
40   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
41   g_signal_connect (G_OBJECT(window), "destroy",
42                     G_CALLBACK(gtk_main_quit), NULL);
43
44   entry = gtk_entry_new ();
45   gtk_container_add (GTK_CONTAINER(window), entry);
46
```

```
47    completion = create_completion_widget ();
48    gtk_entry_set_completion (GTK_ENTRY(entry), completion);
49    g_object_unref (completion);
50
51    gtk_widget_show_all (window);
52    gtk_main ();
53
54    return 0;
55 }
```

次の例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 のファイルブラウザにエントリ補間ウィジェットを追加したものです．77–78 行目でそのディレクトリ内のファイル名をエントリ補間ウィジェットに登録していますので，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，27–29 行目で関数 gtk_tree_sortable_set_sort_column_id で補間文字列でソートするように設定していますので，補間候補の文字列は自動的にソートされて表示されます (図 6.43)．

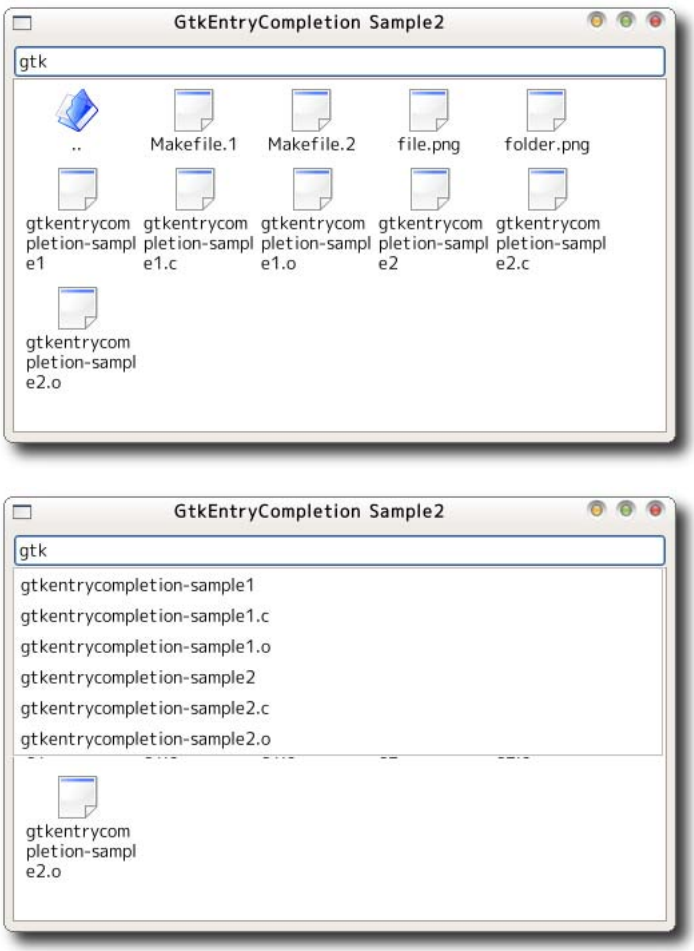


図 6.43　エントリ補間ウィジェットのサンプルプログラム 2

ソース 6–7–7 エントリ補間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    COLUMN_COMPLETION_TEXT,
5    N_COMPLETION_COLUMNS
6  };
7
8  enum {
9    COLUMN_PATH,
10   COLUMN_DISPLAY_NAME,
11   COLUMN_PIXBUF,
12   COLUMN_IS_DIRECTORY,
13   N_ICONVIEW_COLUMNS
14 };
15
16 static GdkPixbuf *file_pixbuf, *folder_pixbuf;
17 static gchar* currentdir;
18
19 static GtkEntryCompletion* create_completion_widget (void) {
20   GtkEntryCompletion    *completion;
21   GtkTreeModel          *completion_model;
22
23   completion = gtk_entry_completion_new ();
24   completion_model =
25     GTK_TREE_MODEL(gtk_list_store_new (N_COMPLETION_COLUMNS,
26                                        G_TYPE_STRING));
27   gtk_tree_sortable_set_sort_column_id
28     (GTK_TREE_SORTABLE (completion_model), COLUMN_COMPLETION_TEXT,
29      GTK_SORT_ASCENDING);
30   gtk_entry_completion_set_model (completion, completion_model);
31   g_object_unref (completion_model);
32   gtk_entry_completion_set_text_column (completion, 0);
33
34   return completion;
35 }
36
37 static void load_pixbuf (void) {
38   file_pixbuf   = gdk_pixbuf_new_from_file ("file.png", NULL);
39   folder_pixbuf = gdk_pixbuf_new_from_file ("folder.png", NULL);
40 }
41
42 static void add_data (GtkIconView        *iconview,
43                       GtkEntryCompletion *completion) {
44   GtkListStore  *iconview_store, *completion_store;
45   GDir          *dir;
46   const gchar   *name;
```

```
47   GtkTreeIter   iter;
48
49   iconview_store = GTK_LIST_STORE(gtk_icon_view_get_model (iconview));
50   completion_store
51     = GTK_LIST_STORE(gtk_entry_completion_get_model (completion));
52
53   gtk_list_store_clear (iconview_store);
54   gtk_list_store_clear (completion_store);
55
56   dir = g_dir_open (currentdir, 0, NULL);
57   if (!dir) return;
58
59   while (name = g_dir_read_name (dir)) {
60     gchar       *path, *display_name;
61     gboolean    is_dir;
62
63     if (name[0] != '.') {
64       path = g_build_filename (currentdir, name, NULL);
65       is_dir = g_file_test (path, G_FILE_TEST_IS_DIR);
66       display_name = g_filename_to_utf8 (name, -1, NULL, NULL, NULL);
67
68       gtk_list_store_append (iconview_store, &iter);
69       gtk_list_store_set (iconview_store, &iter,
70                           COLUMN_PATH, path,
71                           COLUMN_DISPLAY_NAME, display_name,
72                           COLUMN_IS_DIRECTORY, is_dir,
73                           COLUMN_PIXBUF,
74                           (is_dir) ? folder_pixbuf : file_pixbuf, -1);
75
76       gtk_list_store_append (completion_store, &iter);
77       gtk_list_store_set (completion_store, &iter,
78                           COLUMN_COMPLETION_TEXT, display_name, -1);
79       g_free (path);
80       g_free (display_name);
81     }
82   }
83   g_dir_close (dir);
84
85   if (g_utf8_collate (currentdir, "/") != 0) {
86     gtk_list_store_append (iconview_store, &iter);
87     gtk_list_store_set (iconview_store, &iter,
88                         COLUMN_PATH, g_path_get_dirname (currentdir),
89                         COLUMN_DISPLAY_NAME, "..",
90                         COLUMN_IS_DIRECTORY, TRUE,
91                         COLUMN_PIXBUF, folder_pixbuf, -1);
92   }
93 }
94
95 static gint sort_func (GtkTreeModel *model,
96                        GtkTreeIter  *a,
```

```
97                          GtkTreeIter  *b,
98                          gpointer     data) {
99    gboolean      is_dir_a, is_dir_b;
100   gchar         *name_a, *name_b;
101   int           result;
102
103   gtk_tree_model_get (model, a,
104                       COLUMN_IS_DIRECTORY, &is_dir_a,
105                       COLUMN_DISPLAY_NAME, &name_a,
106                       -1);
107   gtk_tree_model_get (model, b,
108                       COLUMN_IS_DIRECTORY, &is_dir_b,
109                       COLUMN_DISPLAY_NAME, &name_b,
110                       -1);
111   if (!is_dir_a && is_dir_b) {
112     result = 1;
113   } else if (is_dir_a && !is_dir_b) {
114     result = -1;
115   } else {
116     result = g_utf8_collate (name_a, name_b);
117   }
118   return result;
119 }
120
121 static GtkWidget* create_icon_view_widget (void) {
122   GtkWidget     *iconview;
123   GtkListStore  *store;
124
125   currentdir = g_get_current_dir ();
126   store = gtk_list_store_new(N_ICONVIEW_COLUMNS,
127                              G_TYPE_STRING, G_TYPE_STRING, GDK_TYPE_PIXBUF,
128                              G_TYPE_BOOLEAN);
129   gtk_tree_sortable_set_default_sort_func (GTK_TREE_SORTABLE(store),
130                                            sort_func, NULL, NULL);
131   gtk_tree_sortable_set_sort_column_id
132     (GTK_TREE_SORTABLE (store), GTK_TREE_SORTABLE_DEFAULT_SORT_COLUMN_ID,
133      GTK_SORT_ASCENDING);
134
135   iconview = gtk_icon_view_new_with_model (GTK_TREE_MODEL(store));
136   g_object_unref (store);
137
138   return iconview;
139 }
140
141 static void cb_item_activated (GtkIconView *iconview,
142                                GtkTreePath *treepath,
143                                gpointer    data) {
144   GtkListStore          *store;
145   GtkTreeIter           iter;
146   gchar                 *path;
```

```
147   gboolean              is_dir;
148   GtkEntryCompletion    *completion;
149
150   store = GTK_LIST_STORE(gtk_icon_view_get_model (iconview));
151   gtk_tree_model_get_iter (GTK_TREE_MODEL(store), &iter, treepath);
152   gtk_tree_model_get(GTK_TREE_MODEL(store), &iter,
153                      COLUMN_PATH, &path, COLUMN_IS_DIRECTORY, &is_dir, -1);
154   if (is_dir) {
155     g_free (currentdir);
156     currentdir = g_strdup (path);
157     add_data (iconview, GTK_ENTRY_COMPLETION(data));
158   }
159   g_free (path);
160 }
161
162 int main (int argc, char **argv) {
163   GtkWidget             *window;
164   GtkWidget             *vbox;
165   GtkWidget             *entry;
166   GtkWidget             *scrolled_window;
167   GtkWidget             *iconview;
168   GtkEntryCompletion    *completion;
169
170   gtk_init (&argc, &argv);
171   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
172   gtk_window_set_title  (GTK_WINDOW(window), "GtkEntryCompletion␣Sample2");
173   gtk_widget_set_size_request (window, 500, 300);
174   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
175   g_signal_connect (G_OBJECT(window), "destroy",
176                     G_CALLBACK(gtk_main_quit), NULL);
177
178   vbox = gtk_vbox_new (FALSE, 0);
179   gtk_container_add (GTK_CONTAINER(window), vbox);
180
181   entry = gtk_entry_new ();
182   gtk_box_pack_start (GTK_BOX(vbox), entry, FALSE, FALSE, 0);
183
184   completion = create_completion_widget ();
185   gtk_entry_set_completion (GTK_ENTRY(entry), completion);
186   g_object_unref (completion);
187
188   scrolled_window = gtk_scrolled_window_new (NULL, NULL);
189   gtk_scrolled_window_set_shadow_type(GTK_SCROLLED_WINDOW(scrolled_window),
190                                       GTK_SHADOW_ETCHED_IN);
191   gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW(scrolled_window),
192                                   GTK_POLICY_AUTOMATIC,
193                                   GTK_POLICY_AUTOMATIC);
194   gtk_box_pack_start (GTK_BOX(vbox), scrolled_window, TRUE, TRUE, 0);
195
196   load_pixbuf ();
```

```
197
198   iconview = create_icon_view_widget ();
199   gtk_icon_view_set_text_column (GTK_ICON_VIEW(iconview),
200                                  COLUMN_DISPLAY_NAME);
201   gtk_icon_view_set_pixbuf_column (GTK_ICON_VIEW(iconview), COLUMN_PIXBUF);
202   gtk_icon_view_set_item_width (GTK_ICON_VIEW(iconview), 80);
203   g_signal_connect (G_OBJECT(iconview), "item_activated",
204                     G_CALLBACK(cb_item_activated), completion);
205
206   gtk_container_add (GTK_CONTAINER(scrolled_window), iconview);
207
208   add_data (GTK_ICON_VIEW(iconview), completion);
209
210   gtk_widget_show_all (window);
211   gtk_main ();
212
213   return 0;
214 }
```

第 7 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について具体例を挙げて説明します．

7.1 アイコン付きボタンウィジェットの作成

ここでは図 7.1 に示すようなアイコンとラベルが一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することができる．アイコンデータは次の 4 種類から指定することができる．
  - 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．

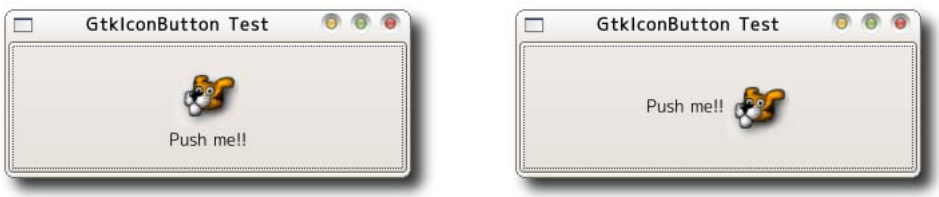


図 7.1　アイコン付きボタンウィジェット

- 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
  - 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

7.2　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ですので, 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 7–1 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** (8–10, 25–27 **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.

ウィジェット専用マクロ定義 (12–22 **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用されるマクロです. これらは全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.

ソース 7–1 GtkButton ウィジェットのヘッダファイル：gtkbutton.h から一部抜粋

```
#ifndef __GTK_BUTTON_H__
#define __GTK_BUTTON_H__

#include <gdk/gdk.h>
#include <gtk/gtkbin.h>
#include <gtk/gtkenums.h>

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

#define GTK_TYPE_BUTTON             (gtk_button_get_type ())
#define GTK_BUTTON(obj)             (G_TYPE_CHECK_INSTANCE_CAST ((obj), \
                                     GTK_TYPE_BUTTON, GtkButton))
#define GTK_BUTTON_CLASS(klass)     (G_TYPE_CHECK_CLASS_CAST ((klass), \
                                     GTK_TYPE_BUTTON, GtkButtonClass))
#define GTK_IS_BUTTON(obj)          (G_TYPE_CHECK_INSTANCE_TYPE ((obj), \
                                     GTK_TYPE_BUTTON))
#define GTK_IS_BUTTON_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE ((klass), \
                                     GTK_TYPE_BUTTON))
#define GTK_BUTTON_GET_CLASS(obj)   (G_TYPE_INSTANCE_GET_CLASS ((obj), \
                                     GTK_TYPE_BUTTON, GtkButtonClass))
 ...

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* __GTK_BUTTON_H__ */
```

これに加えて, GtkIconButton で使用する列挙体を定義します. 一つはラベルテキストの配置位置を表す列挙体, もう一つはアイコンデータのソースタイプを表す列挙体です.

```
enum {
  GTK_ICON_BUTTON_TEXT_TOP,
  GTK_ICON_BUTTON_TEXT_BOTTOM,
  GTK_ICON_BUTTON_TEXT_LEFT,
  GTK_ICON_BUTTON_TEXT_RIGHT
};

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

修正を行ったヘッダファイル gtkiconbutton.h の一部をソース 7–2 に示します.

ソース 7–2 GtkIconButton のヘッダファイル：gtkiconbutton.h から一部抜粋

```
1  #ifndef __GTK_ICON_BUTTON_H__
2  #define __GTK_ICON_BUTTON_H__
3
4  #include <gtk/gtk.h>
5
6  #ifdef __cplusplus
7  extern "C" {
8  #endif
9
10 #define GTK_TYPE_ICON_BUTTON            (gtk_icon_button_get_type ())
11 #define GTK_ICON_BUTTON(obj)            (G_TYPE_CHECK_INSTANCE_CAST((obj),\
12                                          GTK_TYPE_ICON_BUTTON,\
13                                          GtkIconButton))
14 #define GTK_ICON_BUTTON_CLASS(klass)    (G_TYPE_CHECK_CLASS_CAST((klass),\
15                                          GTK_TYPE_ICON_BUTTON,\
16                                          GtkIconButtonClass))
17 #define GTK_IS_ICON_BUTTON(obj)         (G_TYPE_CHECK_INSTANCE_TYPE((obj),\
18                                          GTK_TYPE_ICON_BUTTON))
19 #define GTK_IS_ICON_BUTTON_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE((klass),\
20                                          GTK_TYPE_ICON_BUTTON))
21 #define GTK_ICON_BUTTON_GET_CLASS(obj)  (G_TYPE_INSTANCE_GET_CLASS((obj),\
22                                          GTK_TYPE_ICON_BUTTON,\
23                                          GtkIconButtonClass))
24
25 enum {
26   GTK_ICON_BUTTON_TEXT_TOP,
27   GTK_ICON_BUTTON_TEXT_BOTTOM,
28   GTK_ICON_BUTTON_TEXT_LEFT,
29   GTK_ICON_BUTTON_TEXT_RIGHT
30 };
31
32 enum {
33   GTK_ICON_BUTTON_SOURCE_FILE,
34   GTK_ICON_BUTTON_SOURCE_INLINE,
35   GTK_ICON_BUTTON_SOURCE_IMAGE,
36   GTK_ICON_BUTTON_SOURCE_PIXBUF
37 };
38
39  ...
40
41 #ifdef __cplusplus
42 }
43 #endif
44
45 #endif  /* __GTK_ICON_BUTTON_H__ */
```

7.3 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 としますので，構造体の一番始めのメンバは 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 7–3 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    GtkButton      button;
6    guint          text_position;
7    guint          icon_source;
8    GdkPixbuf      *pixbuf;
9  };
10
11 struct _GtkIconButtonClass {
12   GtkButtonClass        parent_class;
13 };
```

7.4 クラス情報の登録関数

ヘッダファイルの記述が終わりました (関数のプロトタイプ宣言は除きます) ので，次にクラス情報の登録関数 gtk_icon_button_get_type を実装します．この関数は初めて呼び出されるとクラス情報を登録して，このクラスに唯一の ID を割り当てます．2 回目以降にこの関数が呼び出されたときには，この ID を返します．

GTypeInfo は次のように定義されています．

```
struct _GTypeInfo {
  guint16               class_size;
  GBaseInitFunc         base_init;
  GBaseFinalizeFunc     base_finalize;
  GClassInitFunc        class_init;
  GClassFinalizeFunc    clss_finalize;
  gconstpointer         class_data;
  guint16               instance_size;
  guint16               n_preallocs;
  GInstanceInitFunc     instalce_init;
  const GTypeValueTable *value_table;
};
```

ソース 7–4 に関数 gtk_icon_button_get_type を示します．これは関数 gtk_button_get_type 内の変数名や関数名を GtkIconButton 用に修正したものです．

ソース 7–4 関数 gtk_icon_button_get_type

```
1  GType
2  gtk_icon_button_get_type (void) {
3    static GType  icon_button_type = 0;
4
5    if (!icon_button_type) {
6      static const GTypeInfo icon_button_info = {
7        sizeof (GtkIconButtonClass),
8        NULL, /* base_init */
9        NULL, /* base_finalize */
10       (GClassInitFunc) gtk_icon_button_class_init,
11       NULL, /* class_finalize */
12       NULL, /* class_data */
13       sizeof (GtkIconButton),
14       16, /* n_preallocs */
15       (GInstanceInitFunc) gtk_icon_button_init,
16     };
17     icon_button_type
18       = g_type_register_static (GTK_TYPE_BUTTON, "GtkIconButton",
19                                 &icon_button_info, 0);
20   }
21   return icon_button_type;
22 }
```

7.5　初期化関数

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親クラスの割り当てを行うだけとします．ウィジェット初期化関数ではアイコンデータの初期化のみを行います．

ソース 7–5 初期化関数

```
1  static void
2  gtk_icon_button_class_init (GtkIconButtonClass *klass) {
3    parent_class = g_type_class_peek_parent (klass);
4  }
5
6  static void
7  gtk_icon_button_init (GtkIconButton *button) {
8    button->pixbuf = NULL;
9  }
```

7.6 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始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．そこで全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 7–6)．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 7–6 ウィジェット作成関数

```
1 1static GtkWidget*
2 _gtk_icon_button_new (const gchar       *label_text,
3                       guint             text_position,
4                       guint             use_underline,
5                       guint             icon_source,
6                       gpointer          data) {
7   GtkIconButton *ibutton;
8   GtkWidget     *widget;
9   GtkButton     *button;
10
11  widget  = GTK_WIDGET(gtk_type_new (gtk_icon_button_get_type ()));
12  ibutton = GTK_ICON_BUTTON (widget);
13  button  = GTK_BUTTON(ibutton);
14
15  if (label_text) {
16    button->label_text = g_strdup (label_text);
17  } else {
18    button->label_text = NULL;
19  }
20  button->use_underline  = use_underline;
21  ibutton->text_position = text_position;
22  ibutton->icon_source   = icon_source;
23
24  if (data) _set_icon (ibutton, data);
25  gtk_icon_button_construct_child (ibutton);
26
27  return widget;
28 }
```

アイコンデータを作成する部分は別の関数_set_icon(ソース 7–7) 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 7–7 アイコンデータ作成関数

```
1  static void
2  _set_icon (GtkIconButton *ibutton,
3             gpointer       data) {
4    if (ibutton->pixbuf) g_object_unref (ibutton->pixbuf);
5
6    switch (ibutton->icon_source) {
7    case GTK_ICON_BUTTON_SOURCE_FILE:
8      ibutton->pixbuf = gdk_pixbuf_new_from_file ((gchar *) data, NULL);
9      break;
10   case GTK_ICON_BUTTON_SOURCE_INLINE:
11     ibutton->pixbuf = gdk_pixbuf_new_from_inline (-1, (guint8 *) data,
12                                                   TRUE, NULL);
13     break;
14   case GTK_ICON_BUTTON_SOURCE_IMAGE:
15     ibutton->pixbuf = gtk_image_get_pixbuf (GTK_IMAGE(data));
16     break;
17   case GTK_ICON_BUTTON_SOURCE_PIXBUF:
18     ibutton->pixbuf = gdk_pixbuf_copy ((GdkPixbuf *) data);
19     break;
20   }
21 }
```

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 7–8) 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することができますが，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することができません．GtkIconButton ウィジェットでは，19–24 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替えています．そして，45–56 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 7–8 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    GtkButton        *button;
4    GtkButtonPrivate *priv;
5    GtkWidget        *label = NULL;
6    GtkWidget        *box;
7    GtkWidget        *align;
8    GdkPixbuf        *pixbuf;
```

```
 9
10  button = GTK_BUTTON(ibutton);
11  priv   = GTK_BUTTON_GET_PRIVATE (button);
12
13  if (!button->constructed) return;
14
15  if (GTK_BIN (button)->child) {
16    gtk_container_remove (GTK_CONTAINER (button), GTK_BIN (button)->child);
17    priv->image = NULL;
18  }
19  if (ibutton->text_position == GTK_ICON_BUTTON_TEXT_TOP ||
20      ibutton->text_position == GTK_ICON_BUTTON_TEXT_BOTTOM) {
21    box = gtk_vbox_new (FALSE, 2);
22  } else {
23    box = gtk_hbox_new (FALSE, 2);
24  }
25  if (button->label_text) {
26    if (button->use_underline) {
27      label = gtk_label_new_with_mnemonic (button->label_text);
28      gtk_label_set_mnemonic_widget (GTK_LABEL(label), GTK_WIDGET(button));
29    } else {
30      label = gtk_label_new (button->label_text);
31    }
32  }
33  if (ibutton->pixbuf) {
34    priv->image = gtk_image_new_from_pixbuf (ibutton->pixbuf);
35    g_object_set (priv->image,
36                  "visible", show_image (button), "no_show_all", TRUE,
37                  NULL);
38  }
39  if (button->label_text && priv->align_set) {
40    align = gtk_alignment_new (priv->xalign, priv->yalign, 0.0, 0.0);
41    gtk_misc_set_alignment (GTK_MISC (label), priv->xalign, priv->yalign);
42  } else {
43    align = gtk_alignment_new (0.5, 0.5, 0.0, 0.0);
44  }
45  if (ibutton->text_position == GTK_ICON_BUTTON_TEXT_TOP ||
46      ibutton->text_position == GTK_ICON_BUTTON_TEXT_LEFT) {
47    if (label) gtk_box_pack_start (GTK_BOX(box), label, FALSE, FALSE, 0);
48    if (priv->image) {
49      gtk_box_pack_end (GTK_BOX(box), priv->image, FALSE, FALSE, 0);
50    }
51  } else {
52    if (priv->image) {
53      gtk_box_pack_start (GTK_BOX(box), priv->image, FALSE, FALSE, 0);
54    }
55    if (label) gtk_box_pack_end (GTK_BOX(box), label, FALSE, FALSE, 0);
56  }
57  gtk_container_add (GTK_CONTAINER (button), align);
58  gtk_container_add (GTK_CONTAINER (align), box);
```

```
59    gtk_widget_show_all (align);
60
61    return;
62  }
63
64  GtkWidget*
65  gtk_icon_button_new (const gchar *filename) {
66    return _gtk_icon_button_new (NULL, 0,
67                                 GTK_ICON_BUTTON_TEXT_RIGHT,
68                                 GTK_ICON_BUTTON_SOURCE_FILE,
69                                 (gpointer) filename);
70  }
```

7.7　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ましたので, 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.

ここではラベルテキスト, 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 7–9 のようになります.

ソース 7–9　プロパティ取得関数

```
1  G_CONST_RETURN gchar*
2  gtk_icon_button_get_label (GtkIconButton *ibutton) {
3    g_return_val_if_fail (GTK_IS_ICON_BUTTON (ibutton), NULL);
4
5    return GTK_BUTTON(ibutton)->label_text;
6  }
7
8  G_CONST_RETURN GdkPixbuf*
9  gtk_icon_button_get_icon (GtkIconButton *ibutton) {
10   g_return_val_if_fail (GTK_IS_ICON_BUTTON (ibutton), NULL);
11
12   return ibutton->pixbuf;
13 }
14
15 guint
16 gtk_icon_button_get_text_position (GtkIconButton *ibutton) {
17   g_return_val_if_fail (GTK_IS_ICON_BUTTON (ibutton), 0);
18
19   return ibutton->text_position;
20 }
```

7.8　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 7–10 プロパティ変更関数

```
1  void
2  gtk_icon_button_set_label (GtkIconButton *ibutton,
3                             const gchar   *label) {
4    GtkButton *button;
5    gchar     *new_label;
6
7    g_return_if_fail (GTK_IS_ICON_BUTTON (ibutton));
8
9    button = GTK_BUTTON(ibutton);
10
11   new_label = g_strdup (label);
12   g_free (button->label_text);
13   button->label_text = new_label;
14
15   gtk_icon_button_construct_child (ibutton);
16   g_object_notify (G_OBJECT (ibutton), "label");
17 }
18
19 void
20 gtk_icon_button_set_icon (GtkIconButton   *ibutton,
21                           const GdkPixbuf *pixbuf) {
22   g_return_if_fail (GTK_IS_ICON_BUTTON (ibutton));
23
24   if (ibutton->pixbuf) g_object_unref (ibutton->pixbuf);
25   ibutton->pixbuf = gdk_pixbuf_copy (pixbuf);
26
27   gtk_icon_button_construct_child (ibutton);
28 }
29
30 void
31 gtk_icon_button_set_text_position (GtkIconButton *ibutton,
32                                    guint         text_position) {
33   g_return_if_fail (GTK_IS_ICON_BUTTON (ibutton));
34
35   ibutton->text_position = text_position;
36
37   gtk_icon_button_construct_child (ibutton);
38 }
```

7.9 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ための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くプログラムです. 実行してみると図 7.1 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ると思います.

ソース 7–11 GtkIconButton ウィジェットのテストプログラム

```
1  #include <gtk/gtk.h>
2  #include "gtkiconbutton.h"
3
4  static void cb_click (GtkWidget *widget, gpointer data) {
5    static guint position = 0;
6
7    position++;
8    if (position == 4) position = 0;
9    gtk_icon_button_set_text_position (GTK_ICON_BUTTON(widget), position);
10 }
11
12 int main (int argc, char **argv) {
13   GtkWidget     *window;
14   GtkWidget     *ibutton;
15
16   gtk_init (&argc, &argv);
17
18   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
19   gtk_window_set_title  (GTK_WINDOW (window), "GtkIconButton␣Test");
20   gtk_widget_set_size_request (window, 300, 100);
21   g_signal_connect (G_OBJECT(window), "destroy",
22                     G_CALLBACK(gtk_main_quit), NULL);
23
24   ibutton = gtk_icon_button_new_with_label ("gnome-tigert.png",
25                                             "Push␣me!!",
26                                             GTK_ICON_BUTTON_TEXT_TOP, 5);
27   g_signal_connect (G_OBJECT(ibutton), "clicked", G_CALLBACK(cb_click),
       NULL);
28   gtk_container_add (GTK_CONTAINER(window), ibutton);
29
30   gtk_widget_show_all (window);
31   gtk_main ();
32
33   return 0;
34 }
```

第 8 章

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.

8.1 簡単な Gnome プログラムの作成

ここでは図 8.1 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 8–1 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.

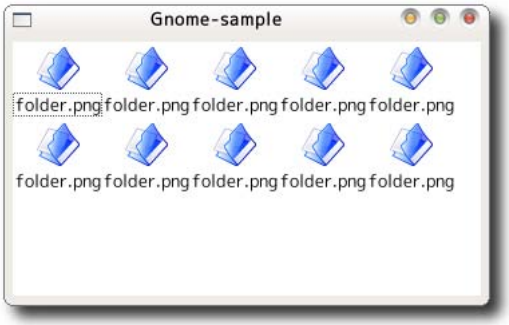


図 8.1 簡単な Gnome アプリケーション

ソース 8–1 簡単な GNOME アプリケーション：gnome-sample.c のヘッダファイル

```
1  #include <gnome.h>
2
3  int main (int argc, char **argv) {
4    GnomeProgram  *app;
5    GtkWidget     *window;
6    GtkWidget     *iconlist;
7    int           n;
8
9    app = gnome_program_init ("gnome-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), "Gnome-sample");
15   gtk_widget_set_size_request (window, 360, 200);
16
17   iconlist = gnome_icon_list_new (64, NULL, 0);
18   gtk_container_add (GTK_CONTAINER(window), iconlist);
19
20   for (n = 0; n < 10; n++) {
21     gnome_icon_list_append (GNOME_ICON_LIST(iconlist),
22                             "folder.png", "folder.png");
23   }
24   gtk_widget_show_all (window);
25   gtk_main ();
26   return 0;
27 }
```

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. Vine Linux 4.1 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** (10 **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 (13 **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 (18–24 **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. 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, 関数 gnome_icon_list_insert と関数 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` ↵
```

8.2 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.

8.2.1 ファイルエントリ

ファイルエントリウィジェット (GnomeFileEntry) 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.

```
GtkWidget*
gnome_file_entry_new (const char *history_id,
                      const char *browse_dialog_title);
```

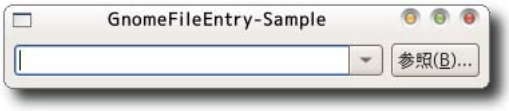


図 8.2 ファイルエントリウィジェットのサンプル

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 8–2 に示します.

ソース 8–2 ファイルエントリ : gnomefileentry-sample.c

```
1  #include <gnome.h>
2
3  int main (int argc, char **argv) {
4    GtkWidget     *window;
5    GtkWidget     *fileentry;
6
7    gnome_program_init("gnome_file_entry-sample", "1.0.0", LIBGNOMEUI_MODULE,
8                       argc, argv, NULL);
9
10   window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
11   gtk_container_set_border_width (GTK_CONTAINER(window), 5);
12   gtk_window_set_title (GTK_WINDOW(window), "GnomeFileEntry-Sample");
13   gtk_widget_set_size_request (window, 360, -1);
14   g_signal_connect (G_OBJECT(window), "destroy",
15                     G_CALLBACK(gtk_main_quit), NULL);
16
17   fileentry = gnome_file_entry_new (NULL, NULL);
18   gtk_container_add (GTK_CONTAINER(window), fileentry);
19
20   gtk_widget_show_all (window);
21   gtk_main ();
22
23   return 0;
24 }
```

8.2.2 アバウトダイアログ

アバウトダイアログ (GnomeAbout) 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.

```
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 8–3 に示します.

ソース 8–3 アバウトダイアログ：gnomeaboutdialog-sample.c

```
1 #include <gnome.h>
2
3 static void cb_show_dialog (GtkWidget *widget, gpointer data) {
```

図 8.3　アバウトダイアログのサンプル

```
4   GtkWidget     *dialog;
5   const gchar   *authors[] = {"Yasuyuki␣Sugaya", NULL};
6   const gchar   *documenters[] = {"Yasuyuki␣Sugaya", NULL};
7   gchar         *translators = "Yasuyuki␣Sugaya";
8
9   dialog = gnome_about_new ("GnomeAbout-Sample", "1.0.0",
10                            "Copyright␣(C)␣2005␣TEO␣Project",
11                            "This␣is␣a␣GnomeAbout␣dialog␣sample␣program.",
12                            authors, documenters, translators, NULL);
13  gtk_container_set_border_width (GTK_CONTAINER(dialog), 5);
14
15  gtk_widget_show_all (dialog);
16 }
17
18 int main (int argc, char **argv) {
19  GtkWidget     *window;
20  GtkWidget     *button;
21
22  gnome_program_init ("gnome_about_dialog-sample", "1.0.0",
23                      LIBGNOMEUI_MODULE, argc, argv, NULL);
24
25  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
26  gtk_container_set_border_width (GTK_CONTAINER(window), 5);
27  gtk_window_set_title (GTK_WINDOW(window), "GnomeAbout-Sample");
28  gtk_widget_set_size_request (window, 250, -1);
29  g_signal_connect (G_OBJECT(window), "destroy",
30                    G_CALLBACK(gtk_main_quit), NULL);
31
32  button = gtk_button_new_with_label ("Show␣About␣Dialog");
33  gtk_container_add (GTK_CONTAINER(window), button);
34  g_signal_connect (G_OBJECT(button), "clicked",
35                    G_CALLBACK(cb_show_dialog), NULL);
36
37  gtk_widget_show_all (window);
38  gtk_main ();
39
40  return 0;
41 }
```

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(アニュータと呼びます) 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, 作成する GUI をその場で確認しながら作成することが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することができますので, 知っておくと大変役に立つツールです.

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しますので, 一番上の [アプリケーション・ウィザード] ボタンを押して先に進みます.

図 9.1　anjuta の起動画面

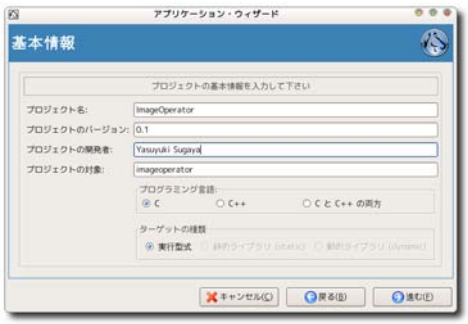
表 9.1　プロジェクトの概要

プロジェクト名	ImageOperator
バージョン	0.1
内容	画像に簡単な画像処理を施すプログラム

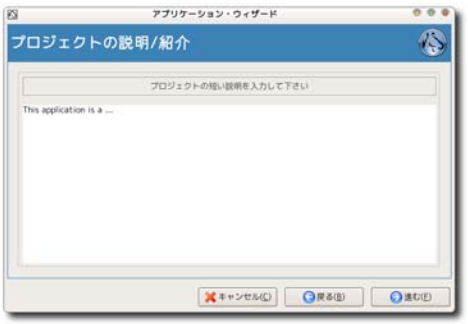
まずは, 図 9.2(a) のアプリケーションウィザード画面が表示されますので, 進むボタンを押して次に進みます. 次にプロジェクトの種類を聞いてきますので, GTK+-2.0 プロジェクトを選択して次に進みます (図 9.2(b)). プロジェクトの種類を決定すると, プロジェクト名や開発者といった, プロジェクトに関する基本情報を入力するダイアログになります. 本プロジェクトでは表 9.1 に示すアプリケーションを作成してみます. 基本情報は図 9.2(c) のように入力しました. 次にプロジェクトの簡単な説明を入力して (図 9.2(d)) , 次に進んでください. 次の追加オプションは何もしないで次に進みます. 最後にサマリが表示されますので, 適用ボタンを押して終了です.

(a) ウィザードの開始

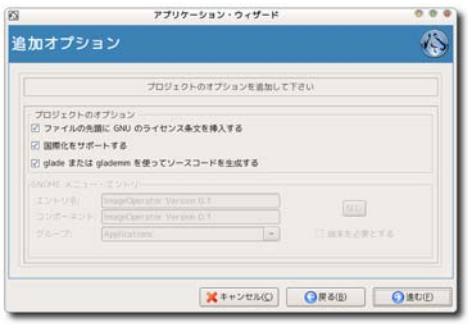
(b) プロジェクトの種類の選択



(c) 基本情報の入力



(d) プロジェクトの説明の入力



(e) 追加オプションの設定

(f) 入力情報のサマリ

図 9.2　プロジェクト作成ウィザード

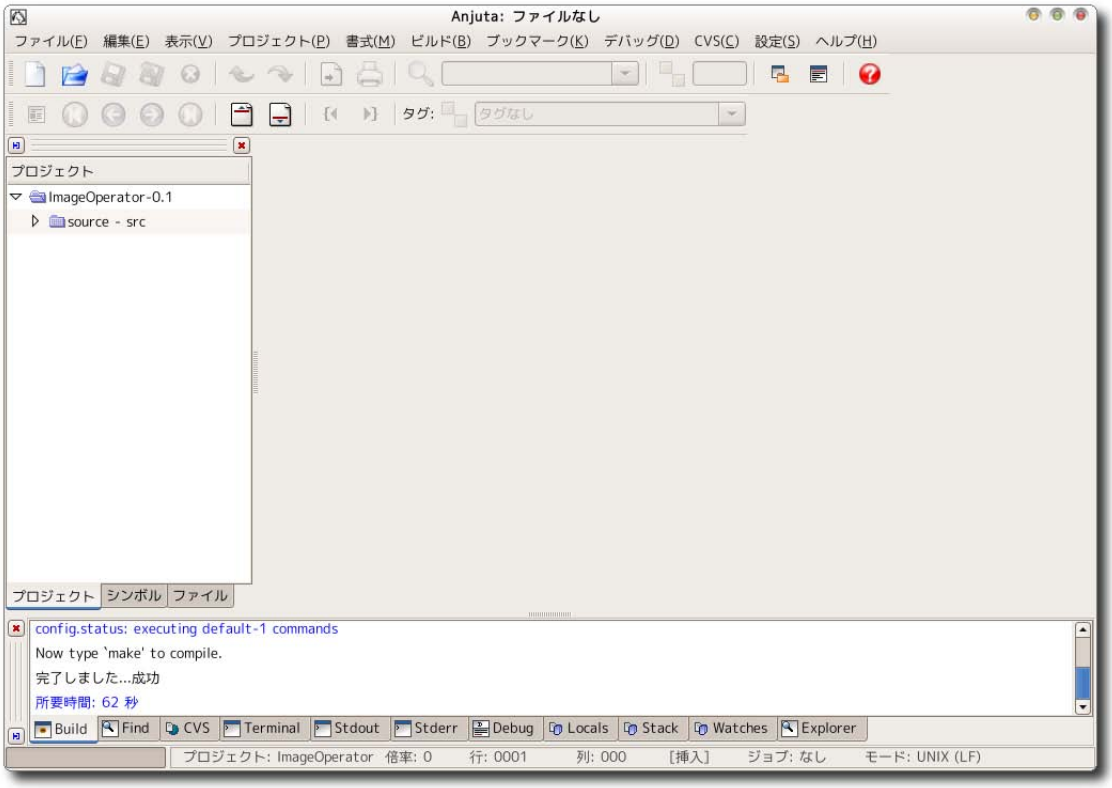


図 9.3 anjuta のメインウィンドウ

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, 自動的に必要なファイルが作成されアプリケーションを作成する準備が整います.

9.2 GUI の作成

9.2.1 glade の起動

プロジェクトを作成したら, 次にアプリケーションの GUI を作成してみましょう. GUI は glade によって作成します. anjuta から glade を起動するには, メニューの [プロジェクト]-[アプリの GUI 編集] を選択します. また glade を単独に起動するにはデスクトップのメニューの [アプリケーション]-[開発ツール]-[Glade インターフェースデザイナー] を選択します. anjuta のメニューから glade を起動すると図 9.4 の 3 つのウィンドウが表示されます.

- パレットウィンドウ
  図 9.4 の左のウィンドウは, パレットウィンドウでこのパレットから配置するウィジェットを選択して, 配置します.
- メインウィンドウ
  glade ファイルを開いたり, 保存したりするウィンドウです.
- プロパティウィンドウ
  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するウィンドウです.

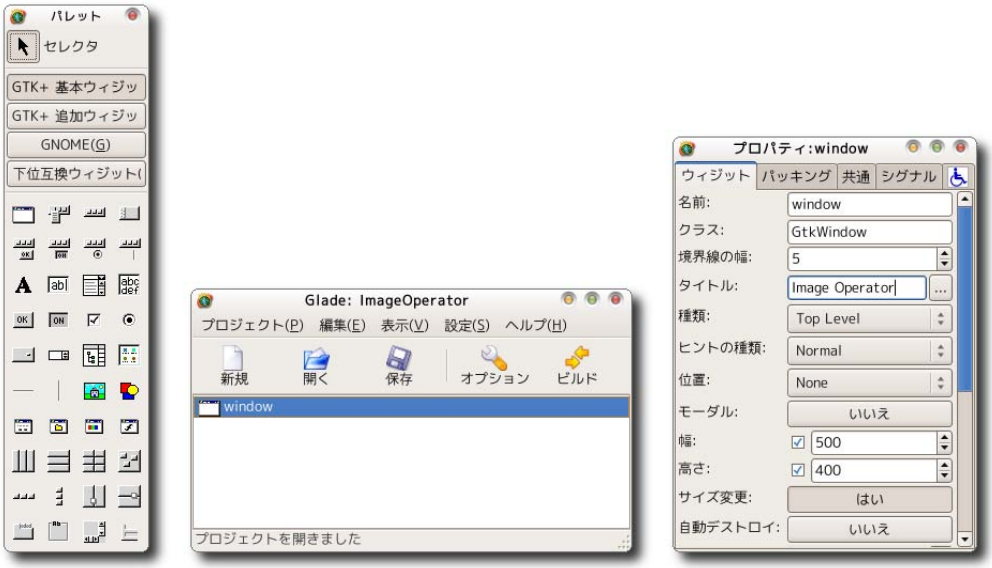


図 9.4　glade の起動画面

9.2.2　メインウィンドウのプロパティ設定

初期状態ではウィンドウが一つ存在するだけです．このウィンドウ上に様々なウィジェットを配置していきます．メインウィンドウの window1 をダブルクリックするとウィンドウが表示されます．はじめに，このウィンドウのプロパティの設定をしておきましょう．ウィンドウを選択するとプロパティウィンドウにそのウィンドウのプロパティが表示されます．

ここではウィンドウのプロパティのうち図 9.5 の赤枠の部分を図にあるように変更します．次に，ウィンドウ上に垂直ボックスを配置します．パレットの垂直ボックスのアイコンをクリックして，ウィンドウ上でクリックします．すると行数を設定するウィンドウが表示されます．行数 3 となっていますので，行数 2 に修正して OK ボタンを押してください．この後さまざまなウィジェットを配置していきますが，先に GUI の完成イメージを示しておきます (図 9.6)．

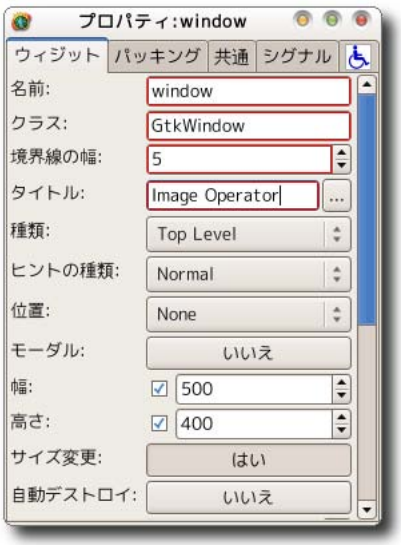


図 9.5　メインウィンドウのプロパティ設定

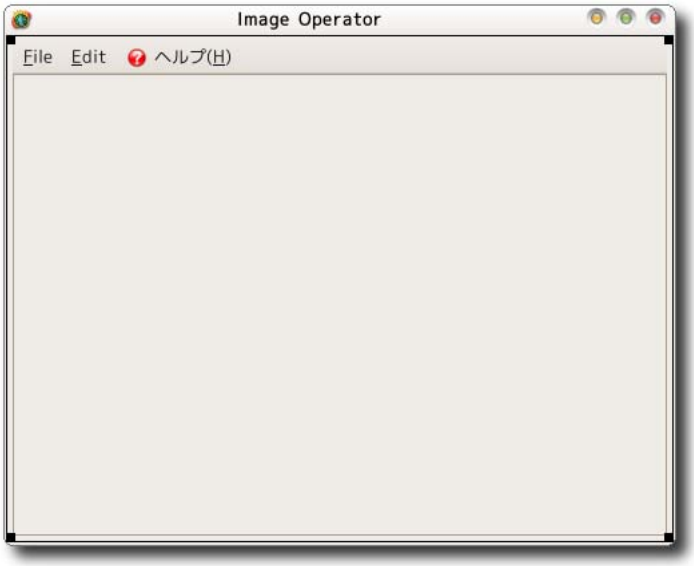


図 9.6　GUI の完成イメージ

9.2.3　メニューの生成

まずはじめに, 一番上の行にメニューバーを配置します. パレットの GTK+ 基本ウィジェットの上の段の左から 2 つ目のアイコンがメニューバーのアイコンです. このアイコンを選択して, 始めに配置した垂直ボックスの一番上の行をクリックしてください. すると glade 内で定義された標準のメニューが配置されますので, プロパティウィンドウのメニュー編集... ボタン (図 9.7(右)) を押して, メニュー編集ダイアログ (図 9.7(左)) を開きます. メニュー編集ダイアログでは, メニューのショートカットやコールバック関数の設定を行うことができます. 図 9.7(左) はメニューの完成状態です.

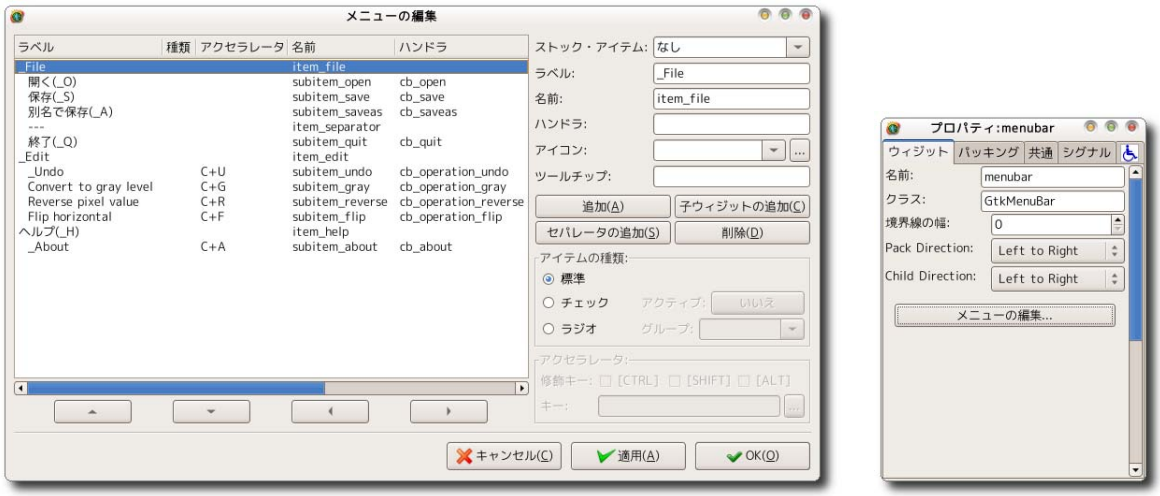


図 9.7　メニュー編集ダイアログ

9.2.4　画像表示部の作成

垂直ボックスの 2 段目には画像を表示するためのドローイングエリア (実際はスクロールウィンドウ-アラインメント-ドローイングエリア) を配置します．最終的な GUI のウィジェット配置の階層構造を図 9.8 に示します．このウィジェットツリーは glade のメインウィンドウのメニューの [表示]- [ウィジェットツリーの表示] を選択すると表示されます．

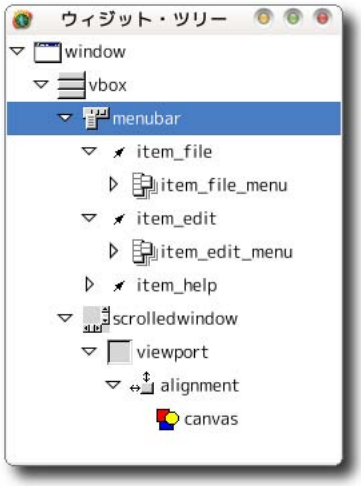


図 9.8　ウィジェット配置の階層構造

図 9.9 に 3 つのウィジェットのプロパティを示します．垂直ボックスにメニューバーとスクロールウィンドウを配置するときに, ウィンドウ内にスクロールウィンドウを拡張して表示するために, メニューバーとスクロールウィンドウのパッキング設定を表 9.2(a), (b) のように設定しています．
また, ドローイングエリアウィジェットに張り付ける画像をウィンドウの中央に配置するために, アラインメントウィジェットのプロパティを表 9.2(c) のように設定します．

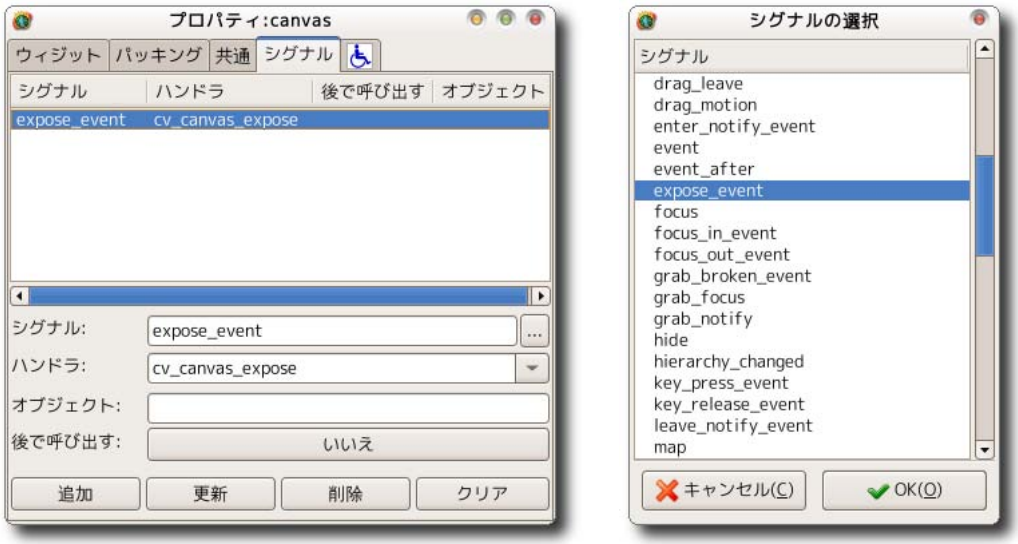
(a)　(b)　(c)

図 9.9　ウィジェットのプロパティ

表 9.2　ウィジェットのパッキング設定

	拡張	領域埋め
メニューバー	いいえ	いいえ
スクロールウィンドウ	はい	はい

図 9.10 にドローイングエリアに対するコールバック関数の設定を示します．ドローイングエリアには画像を描画しますので，”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には，プロパティダイアログのシグナルタブで行います (図 9.10)．シグナルの項目の... ボタンをクリックすると，シグナル選択ダイアログが表示されますので ”expose_event” を選択して OK ボタンをクリックします．コールバック関数の名前は”cb_canavs_expose” としました．
ここまで設定したら glade のメインウィンドウの保存ボタンと作成ボタンをクリックして作成した GUI のソースコードを出力してください．



(a) シグナルの設定　　(b) シグナル選択ウィンドウ

図 9.10　ドローイングエリアのコールバック関数

ここまでの状態でもプロジェクトをビルドすることができます．メニューバーから [ビルド]-[すべてビルド] を選択するとプロジェクトをビルドすることができます．ビルドが成功したら，メニューバーから [ビルド]-[実行](ショートカットは F3 に設定されています) を選択するとアプリケーションが起動します．

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 9.3 にまとめます．glade でコールバック関数の設定してソースコードの書き出しを行った場合にはコールバック関数のインターフェイスが自動的に作成されますので，ユーザは関数内部で行う処理を記述するだけで済みます．

表 9.3　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_about	アプリケーション情報を表示する関数
cb_operation_undo	画像処理関数 (アンドゥ)
cb_operation_gray	画像処理関数 (濃淡画像の作成)
cb_operation_reverse	画像処理関数 (画素値の反転)
cb_operation_flip	画像処理関数 (水平反転)
cb_canvas_expose	画面描画関数

9.3.1　グローバル変数の設定

様々な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．ここでは新規に image_operator.h というファイルを作成して，グローバル変数を定義します．新しいヘッダファイルを追加するにはまず，メニューバーから [ファイル]-[新規] を選択します．図 9.11 のダイアログが表示されますので，ファイル名を image_operator.h，ファイルの種類を C-C++ Header File，ヘッダファイル・テンプレートにチェックをして OK ボタンを押してください．テンプレートが挿入されたファイルが表示されますので，図 9.12 に示すように，メインウィンドウ用の変数 main_window，表示画像用の変数 pixbuf，アンドゥ用の変数 backup を定義してください．

図 9.11　ヘッダファイルの追加

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_canvas_expose 内に実装します．画像データはコールバック関数 cb_open 内で読み込みます．画像データをドローイングエリアに描画する方法は 5.3.2 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 9–1 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

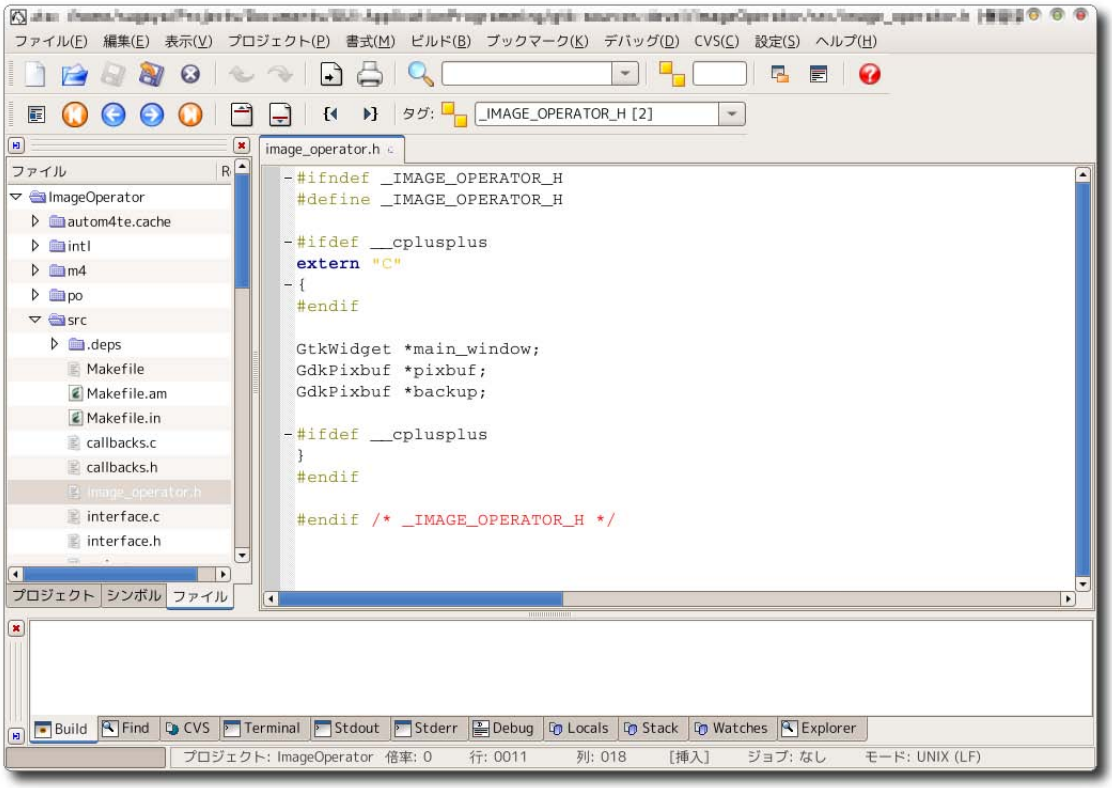


図 9.12　グローバル変数の定義

い.

ソース 9–1　関数 cb_canvas_expose

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

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. ファイル選択ダイアログから表示するための画像ファイル名を取得する方法は 6.5.3 ファイル選択ダイアログのサンプルプログラムを参考にすることにします. ファイル選択ダイアログ関連のコールバック関数など画像の読み込みに関するソースコードをソース 9–2 に示します.

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．ファイル名の選択に関する関数を画像のオープン用と画像の保存用で共有するために，グローバル変数 selection_mode を宣言しています．変数 selection_mode の値としてヘッダファイル image_operator.h に以下のように列挙型の値を定義します．同時にこの関数内で使用する変数 open_filename と save_filename を image_operator.h に追加してください．

```
enum {
  FILE_SELECTION_OPEN,
  FILE_SELECTION_SAVE
};
```

コールバック関数 cb_open が呼ばれた場合には FILE_SELECTION_OPEN を，コールバック関数 cb_save が呼ばれた場合には FILE_SELECTION_SAVE を設定します．そして，ファイル選択ダイアログを表示してファイル選択状態にします．ファイル選択ダイアログを表示した後でファイルが選択されるまで処理が次に進まないように関数 gtk_main を呼び出します．

ファイルが選択されると変数 open_filename にファイル名が入りますので，画像データを読み込んで表示します．ファイル選択ダイアログでキャンセルボタンを押してダイアログを閉じた場合には，コールバック関数 filesel_cancel が呼び出されます．キャンセルボタンを押した場合にはファイル名を選択しなかったことを表しますので，変数 open_filename のメモリ領域を開放して NULL に初期化します．

OK ボタンを押した場合は関数 gtk_file_selection_get_filename により選択されたファイル名を取得して，変数 open_filename にコピーします．ファイルメニューから保存を選択した場合には現在開いているファイル名のまま上書きしますので，変数 save_filename にも同じファイル名をコピーしておきます．

ソース 9–2 画像の読み込み関連の関数：callbacks.c から一部抜粋

```
1  static void
2  filesel_ok (GtkWidget   *widget,
3              gpointer    user_data) {
4    GtkWidget   *filesel = GTK_WIDGET(user_data);
5    const gchar *filename;
6
7    filename = gtk_file_selection_get_filename (GTK_FILE_SELECTION(filesel));
8    if (filename && strcmp (filename, "") != 0) {
9      switch (selection_mode) {
10     case FILE_SELECTION_OPEN:
11       if (open_filename) g_free (open_filename);
12       open_filename = g_strdup (filename);
13       if (save_filename) g_free (save_filename);
14       save_filename = g_strdup (filename);
15       break;
16     case FILE_SELECTION_SAVE:
17       if (save_filename) g_free (save_filename);
18       save_filename = g_strdup (filename);
19       break;
20     }
21   }
22   gtk_window_set_modal (GTK_WINDOW(filesel), FALSE);
23   gtk_widget_destroy (filesel);
```

```
24    gtk_main_quit ();
25  }
26
27  static void
28  filesel_cancel (GtkWidget        *widget,
29                  gpointer         user_data) {
30    GtkWidget *filesel = GTK_WIDGET(user_data);
31
32    switch (selection_mode) {
33    case FILE_SELECTION_OPEN:
34      if (open_filename) {
35        g_free (open_filename);
36        open_filename = NULL;
37      }
38      break;
39    case FILE_SELECTION_SAVE:
40      if (save_filename) {
41        g_free (save_filename);
42        save_filename = NULL;
43      }
44      break;
45    }
46    gtk_window_set_modal (GTK_WINDOW(filesel), FALSE);
47    gtk_widget_destroy (filesel);
48    gtk_main_quit ();
49  }
50
51  void
52  cb_open (GtkMenuItem     *menuitem,
53           gpointer        user_data) {
54    GtkWidget      *parent = GTK_WIDGET(user_data);
55    GtkWidget      *filesel;
56    GtkWidget      *canvas;
57
58    selection_mode = FILE_SELECTION_OPEN;
59
60    filesel = gtk_file_selection_new (_("Select␣image␣filename"));
61    g_signal_connect (G_OBJECT(GTK_FILE_SELECTION(filesel)->ok_button),
62                      "clicked", G_CALLBACK(filesel_ok),
63                      (gpointer) filesel);
64    g_signal_connect (G_OBJECT(GTK_FILE_SELECTION(filesel)->cancel_button),
65                      "clicked", G_CALLBACK(filesel_cancel),
66                      (gpointer) filesel);
67    gtk_window_set_transient_for (GTK_WINDOW(filesel), GTK_WINDOW(parent));
68    gtk_window_set_modal (GTK_WINDOW(filesel), TRUE);
69    gtk_widget_show (filesel);
70    gtk_main ();
71
72    if (open_filename) {
73      if (pixbuf) g_object_unref (pixbuf);
```

```
74      pixbuf = gdk_pixbuf_new_from_file (open_filename, NULL);
75      if (pixbuf) {
76        canvas = GTK_WIDGET(g_object_get_data (G_OBJECT(parent),
77                                               "canvas"));
78        gtk_widget_queue_draw (canvas);
79      }
80    }
81  }
```

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画像を保存します. 「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 9–3 に示します. 14 行目や 33 行目の文字列が_() で囲まれているのに気づいたでしょうか. これは文字列が国際化の対象となる文字列であることを表しています. メッセージの国際化については 9.6 節 メッセージの国際化で詳しく説明します.
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 9–3　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 = strrchr (filename, '.');
7    if (ext) {
8      ext++;
9      if (strcmp (ext, "png") == 0) {
10       gdk_pixbuf_save (pixbuf, filename, "png", NULL, NULL);
11     } else if (strcmp (ext, "jpg") == 0) {
12       gdk_pixbuf_save (pixbuf, filename, "jpeg", NULL, NULL);
13     } else {
14       g_printerr (_("Image␣format␣'%s'␣is␣not␣supported\n"), ext);
15     }
16   }
17 }
18
19 void
20 cb_save (GtkMenuItem     *menuitem,
21          gpointer        user_data)
22 {
```

```
23    _save (save_filename);
24  }
25
26  void
27  cb_saveas (GtkMenuItem   *menuitem,
28             gpointer      user_data)
29  {
30    GtkWidget      *parent = GTK_WIDGET(user_data);
31    GtkWidget      *filesel;
32
33    selection_mode = FILE_SELECTION_SAVE;
34
35    filesel = gtk_file_selection_new (_("Select␣image␣filename"));
36    g_signal_connect (G_OBJECT(GTK_FILE_SELECTION(filesel)->ok_button),
37                      "clicked", G_CALLBACK(filesel_ok),
38                      (gpointer) filesel);
39    g_signal_connect (G_OBJECT(GTK_FILE_SELECTION(filesel)->cancel_button),
40                      "clicked", G_CALLBACK(filesel_cancel),
41                      (gpointer) filesel);
42    gtk_window_set_transient_for (GTK_WINDOW(filesel), GTK_WINDOW(parent));
43    gtk_window_set_modal (GTK_WINDOW(filesel), TRUE);
44    gtk_widget_show (filesel);
45    gtk_main ();
46
47    if (save_filename) _save (save_filename);
48  }
```

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 GdkPixbuf データから指定した座標の画素値を取得したり, 書き込むマクロをソース 9–4 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で使用したマクロと同様です.

ソース 9–4 画素値の読み書きを行うマクロ：image_operator.h から一部抜粋

```
1  #define gdk_pixbuf_get_pixel(pixbuf,x,y,p) \
2  (*(gdk_pixbuf_get_pixels((pixbuf)) + \
3   gdk_pixbuf_get_rowstride((pixbuf)) * (y) + \
4   gdk_pixbuf_get_n_channels((pixbuf)) * (x) + (p)))
5
6  #define gdk_pixbuf_put_pixel(pixbuf,x,y,p,val) \
7  (*(gdk_pixbuf_get_pixels((pixbuf)) + \
8   gdk_pixbuf_get_rowstride((pixbuf)) * (y) + \
9   gdk_pixbuf_get_n_channels((pixbuf)) * (x) + (p)) = (val))
```

次に画像処理結果を処理前に戻す処理を実装します. 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_operation_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 9–5 に示します. データをメモリ上に保存するために GdkPixbuf 型の変数 backup をヘッダファイル image_operator.h に宣言

して, 始めに 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_operation_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, アンドゥ処理に対するアンドゥにも対応するために, 始めに変数 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 にコピーしています.

ソース 9–5 アンドゥ処理関連の関数：callbacks.c から一部抜粋

```
1  static void
2  do_backup (void) {
3    if (backup) {
4      gdk_pixbuf_copy_area (pixbuf, 0, 0,
5                            gdk_pixbuf_get_width (pixbuf),
6                            gdk_pixbuf_get_height(pixbuf),
7                            backup, 0, 0);
8    } else {
9      backup = gdk_pixbuf_copy (pixbuf);
10   }
11 }
12
13 void
14 cb_operation_undo (GtkMenuItem   *menuitem,
15                    gpointer      user_data)
16 {
17   GtkWidget      *canvas;
18   GdkPixbuf      *_backup;
19
20   if (backup) {
21     _backup = gdk_pixbuf_copy (pixbuf);
22     gdk_pixbuf_copy_area (backup, 0, 0,
23                           gdk_pixbuf_get_width(backup),
24                           gdk_pixbuf_get_height(backup),
25                           pixbuf, 0, 0);
26     gdk_pixbuf_copy_area (_backup, 0, 0,
27                           gdk_pixbuf_get_width(_backup),
28                           gdk_pixbuf_get_height(_backup),
29                           backup, 0, 0);
30     g_object_unref (_backup);
31     canvas = GTK_WIDGET(g_object_get_data(G_OBJECT(main_window),"canvas"));
32     gtk_widget_queue_draw (canvas);
33   }
34 }
```

最後に画像処理メニューの中の一つのグレースケール化の実装例をソース 9–6 に示します. まず始めに 10 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 11–21 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 23 行目で関数 gtk_widget_queue_draw を呼び出してウィジェット canvas の ”expose” シグナルを発生させます.

ソース 9–6 関数 cb_operation_gray：callbacks.c から一部抜粋

```
1  void
2  cb_operation_gray (GtkMenuItem   *menuitem,
3                     gpointer      user_data)
4  {
5    GtkWidget      *canvas;
6    int            x, y;
7    guchar         val, r, g, b;
8
9    if (pixbuf) {
10     do_backup ();
11     for (y = 0; y < gdk_pixbuf_get_height(pixbuf); y++) {
12       for (x = 0; x < gdk_pixbuf_get_width(pixbuf); x++) {
13         r = gdk_pixbuf_get_pixel(pixbuf, x, y, 0);
14         g = gdk_pixbuf_get_pixel(pixbuf, x, y, 1);
15         b = gdk_pixbuf_get_pixel(pixbuf, x, y, 2);
16         val = 0.299 * r + 0.587 * g + 0.114 * b;
17         gdk_pixbuf_put_pixel(pixbuf, x, y, 0, val);
18         gdk_pixbuf_put_pixel(pixbuf, x, y, 1, val);
19         gdk_pixbuf_put_pixel(pixbuf, x, y, 2, val);
20       }
21     }
22     canvas = GTK_WIDGET(g_object_get_data(G_OBJECT(main_window),"canvas"));
23     gtk_widget_queue_draw (canvas);
24   }
25 }
```

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 6.5.2 メッセージダイアログを参照してください．コールバック関数 cb_about をソース 9–7 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 9.13 に示します．

図 9.13　アプリケーション情報ダイアログ

ソース 9–7 関数 cb_about

```
1  void
2  cb_about (GtkMenuItem     *menuitem,
3            gpointer         user_data)
4  {
5    GtkWidget     *dialog;
6    gchar         *info;
7    gint          response;
8
9    info = g_strdup_printf ("%s␣%s\n%s\n%s␣%s\n",
10                           PACKAGE, VERSION,
11                           _("Copyright(C)␣..."),
12                           PACKAGE,
13                           _("is␣a␣simple␣image␣proccessing␣application."));
14   dialog = gtk_message_dialog_new (GTK_WINDOW(user_data),
15                                    GTK_DIALOG_MODAL |
16                                    GTK_DIALOG_DESTROY_WITH_PARENT,
17                                    GTK_MESSAGE_INFO,
18                                    GTK_BUTTONS_CLOSE,
19                                    info);
20
21   gtk_widget_show_all (dialog);
22
23   response = gtk_dialog_run (GTK_DIALOG(dialog));
24   gtk_widget_destroy (dialog);
25 }
```

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 9–8 に示します．

ソース 9–8 関数 cb_quit

```
1  void
2  cb_quit (GtkMenuItem     *menuitem,
3           gpointer        user_data)
4  {
5    if (pixbuf) g_object_unref (pixbuf);
6    if (backup) g_object_unref (backup);
7    if (open_filename) g_free (open_filename);
8    if (save_filename) g_free (save_filename);
9
10   gtk_main_quit ();
11 }
```

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．アプリケーションをビルドするにはメニューの [ビルド]-[ビルド] もしくは [ビルド]-[全てビルド] を選択します．
ビルドに成功したら図 9.14 のように Build タブのウィンドウに「完了しました... 成功」のメッセージが表示されます．

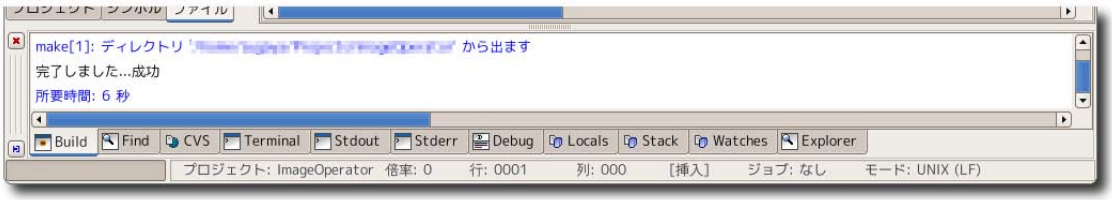


図 9.14　ビルドメッセージ

作成したアプリケーションを実行するにはメニューの [ビルド]-[実行] を選択します．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　配布可能ファイルの作成

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について説明します．

9.5.1　Makikefile.am ファイルの編集

配布パッケージを作成するにはメニューから [ビルド]-[配布可能ファイルの作成] を選択するだけです．しかし，このプロジェクトでは新しく image_operator.h を追加していますので，src ディレクトリの Makefile.am

に image_operator.h を追加しなければいけません. 図 9.16 左のファイルタブをクリックして, src ディレクトリの Makefile.am をダブルクリックすると Makefile.am がエディタに表示されます.

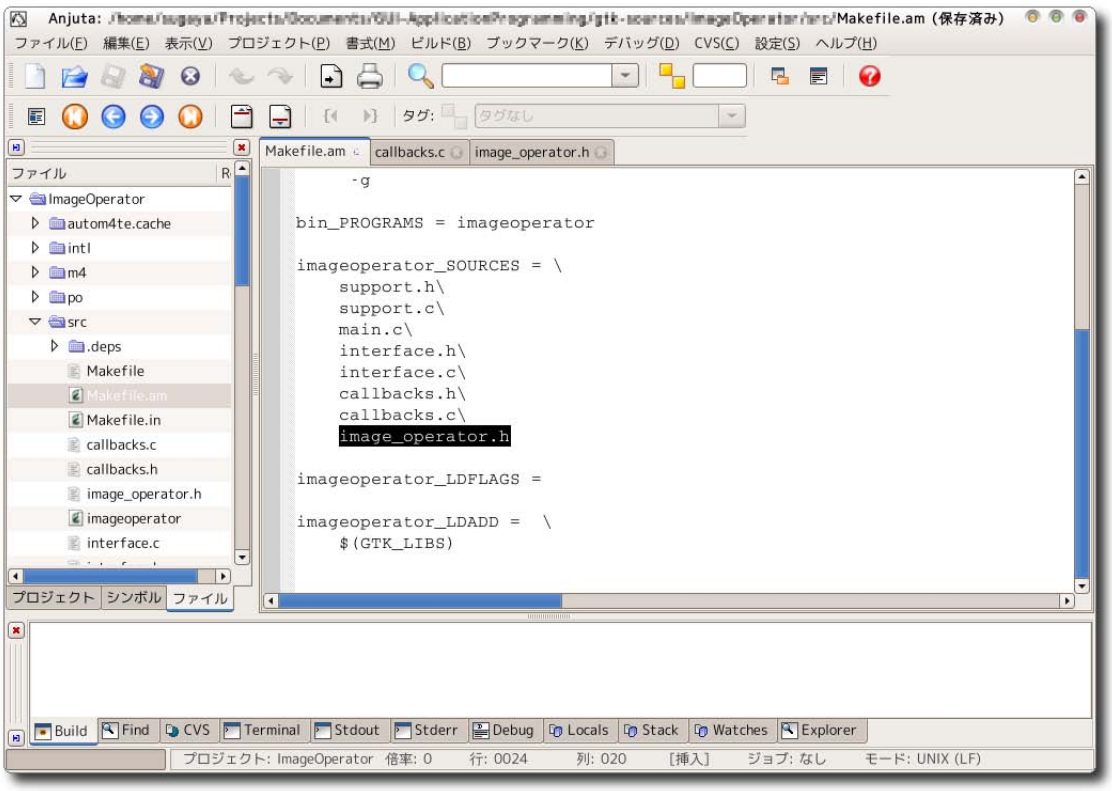


図 9.16　Makefile.am の編集

Makefile.am の内容が表示されたら図 9.16 の黒く反転した部分のように imageoperator_SOURCES の項目に image_operator.h を記述してください.

9.5.2　配布パッケージの作成

次にメニューの [ビルド]-[自動生成] を選択して, 記述した内容を Makefile に反映させます. 自動生成の処理が正常に終了したら, いよいよメニューの [ビルド]- [配布可能ファイルの生成] を選択して, 配布パッケージを生成します. エラーが起こらなければプロジェクトのファイルが保存されているディレクトリに配布パッケージが生成されます. 以下に示すようにディレクトリ内に ImageOperator-0.1.tar.gz という配布パッケージが生成されていることがわかります.

```
% cd ~/Projects/ImageOperator ↵
% ls ↵
AUTHORS                  Makefile.am      config.guess*   libtool*
COPYING@                 Makefile.in      config.h        ltmain.sh
ChangeLog                NEWS             config.h.in     missing@
INSTALL@                 README           config.log      mkinstalldirs*
ImageOperator-0.1.tar.gz TODO             config.status*  po/
```

```
ImageOperator.glade      acconfig.h       config.sub*     setup-gettext*
ImageOperator.gladep     acinclude.m4     configure*      src/
ImageOperator.prj        aclocal.m4       configure.in    stamp-h.in
ImageOperator.pws        autogen.sh*      depcomp@        stamp-h1
Makefile                 autom4te.cache/  install-sh@
```

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することができましたので, 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/var/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 ~/Projects/ImageOperator/ImageOperator-0.1.tar.gz ↵
...
% cd ImageOperator-0.1 ↵
% ./configure --prefix=/var/tmp/ImageOperator/usr ↵
% make ↵
% make install prefix=/var/tmp/ImageOperator/usr ↵
```

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 /var/tmp/ImageOperator ↵
% ls -R
.:
usr/

./usr:
bin/  doc/  share/

./usr/bin:
imageoperator*
```

```
./usr/doc:
ImageOperator/

./usr/doc/ImageOperator:
AUTHORS  COPYING  ChangeLog  INSTALL  NEWS  README TODO

./usr/share:
```

9.6　メッセージの国際化

9.6.1　翻訳メッセージの作成

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．現在は gettext を利用することで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と, 使用する環境に合わせた言語でメッセージを表示することができます．

翻訳ファイルは ImageOperator.pot という翻訳用のテンプレートファイルをもとに作成します．このファイルは一度配布ファイルを作成すると自動的にディレクトリ po 以下に作成されます．日本語メッセージの翻訳は ImageOperator.pot を ja.po という名前で同じディレクトリにコピーして行います．

```
% cd ~/Projects/ImageOperator/po ↵
% cp ImageOperator.pot ja.po ↵
```

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] を選択して起動してください．図 9.17 は gtaranslator の起動画面です．

図 9.17　gtranslator の起動画面

gtanslator の起動後, 先ほどコピーしたファイル ja.po を開きます．ファイルを開くとまずヘッダの編集ウィンドウが表示されますので図 9.18 に示すように情報を入力します．ヘッダ情報を入力したら次はメッセージの

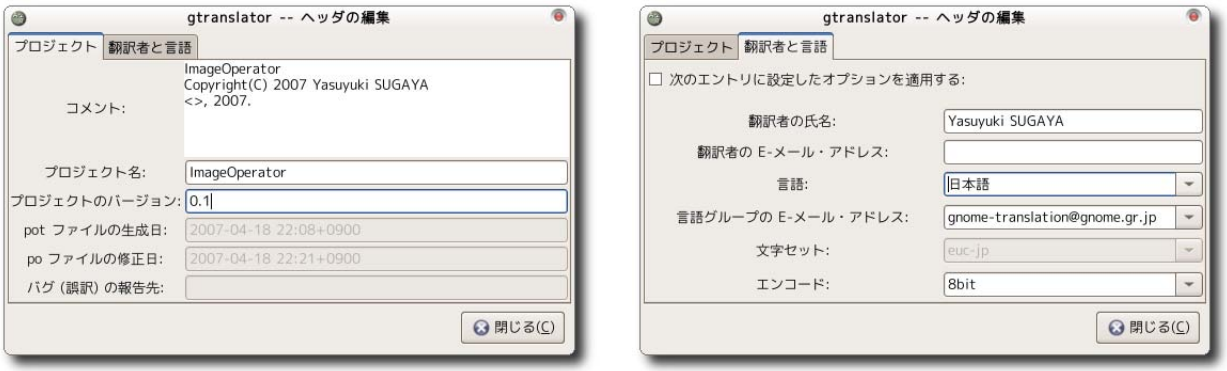


図 9.18　翻訳ファイルのヘッダ情報の入力

翻訳を開始します．ウィンドウ右の上段に翻訳元のメッセージが表示されますので，下段に日本語の翻訳メッセージを入力します．図 9.19 にメッセージの翻訳結果を示します．

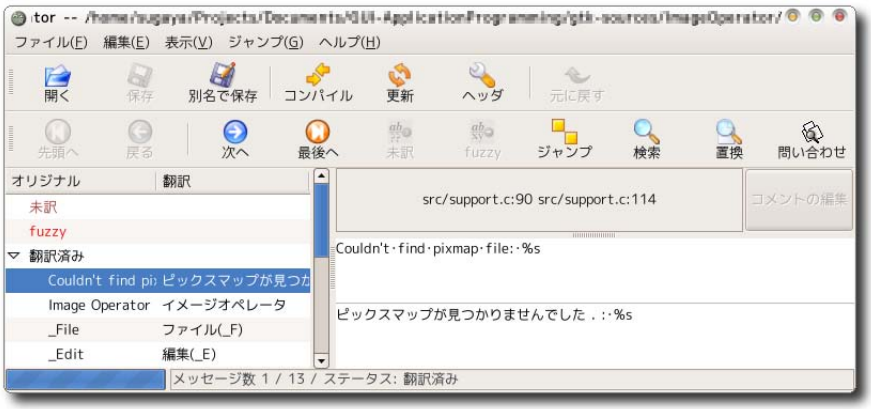


図 9.19　メッセージの翻訳結果

9.6.2　configure.in の編集

日本語メッセージファイルを作成するために configure.in を編集します．　configure.in には ALL_LINGUAS=”” という項目があります．　これを ALL_LINGUAS=”ja” のように編集します．　編集が終了したら，メニューの [ビルド]- [自動生成] を選択して編集結果を反映させます．

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 配布可能ファイルの作成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 9.20 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．

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に関する解説を終わります．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．幸いに anjuta に関する日本語訳されたドキュメントが以下の URL で公開されていますので，興味のある読者は目を通してみることをお薦めします．

図 9.20　メッセージを日本語化したアプリケーション

- 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/

付録 A

GTK+ テーマの変更

GTK+ アプリケーションの外観は GTK+ のテーマ機能によりカスタマイズすることが可能です．ユーザはウェブサイトに公開されているテーマをインストールすることで自分の好みに合わせた概観を楽しむことができます．GNOME-LOOK.org (http://gnome-look.org) というサイトでは GTK+ のテーマを始めとしてテーマ機能を持つさまざまなアプリケーションのためのテーマが公開されています．

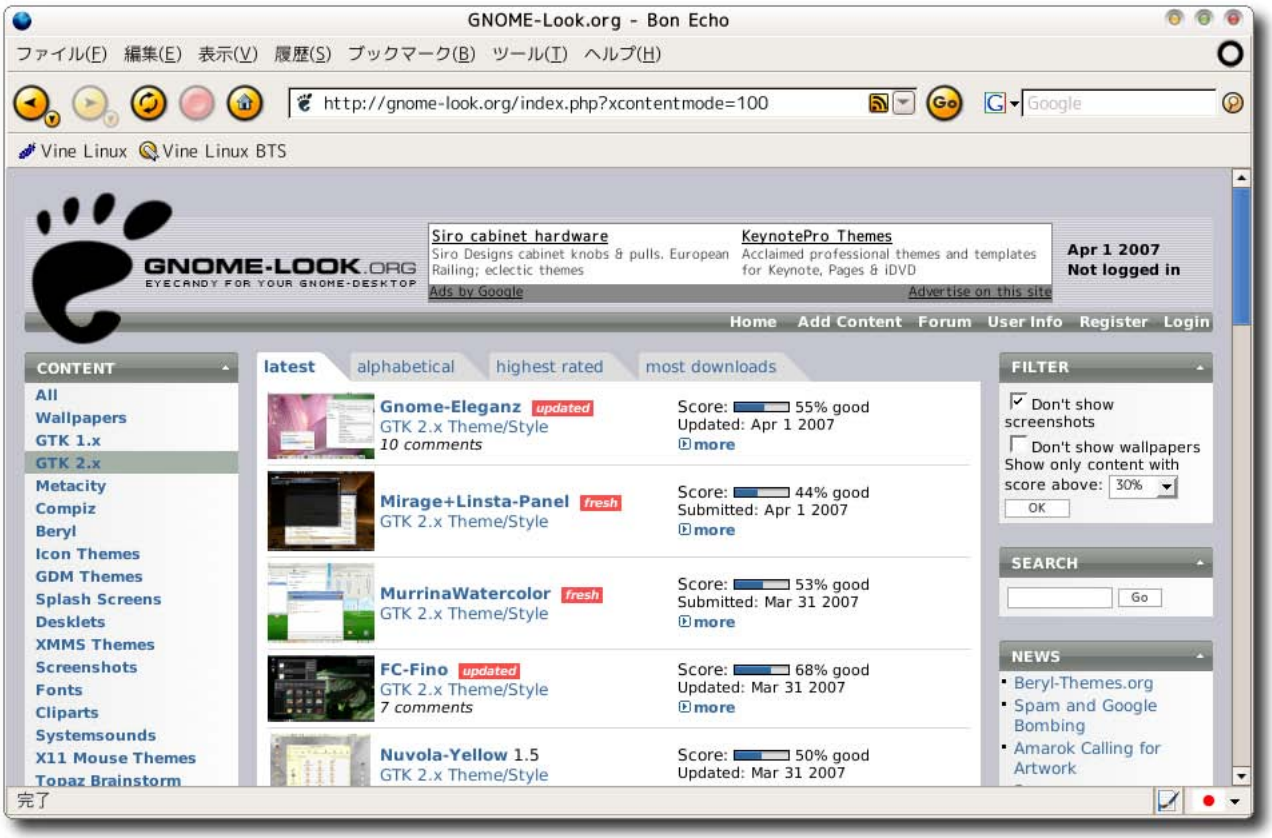


図 A.1　GNOME-LOOK.org

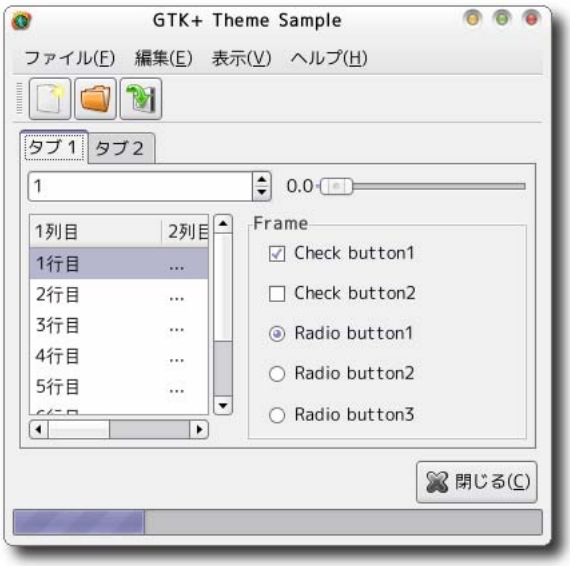
GNOME-LOOK.org には大変多くのテーマが公開されていますので, ユーザのお気に入りのテーマが見つかることでしょう. Vine Linux 4.1 にも OS をインストールした時点でいくつかのテーマがインストールされています. 図 A.2 のようにデスクトップメニューの [デスクトップ]-[設定]-[テーマ] をクリックするとテーマの設定ダイアログが表示されます.

図 A.2　テーマメニュー

図 A.3 のテーマの設定ダイアログで「テーマの詳細」ボタンをクリックすると, 右下にあるテーマの詳細ダイアログが表示されます. このダイアログのコントロールタブでテーマを選択することで GUI の概観を変更することができます.

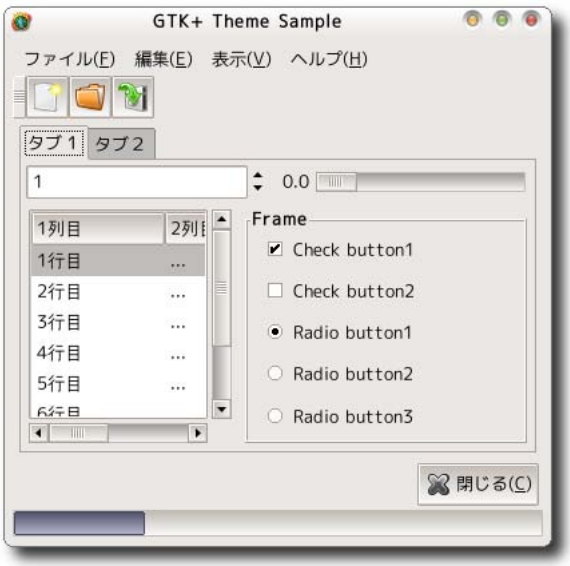
図 A.3　テーマの設定

図 A.4 に 4 つの異なるテーマを選択したときの概観を示します．(a) から (c) までは Vine Linux 4.1 に標準でインストールされているテーマです．(d) のテーマは本書で使用しているオリジナルテーマです．サンプルソースといっしょに公開していますので興味のある方は使用してみてください．

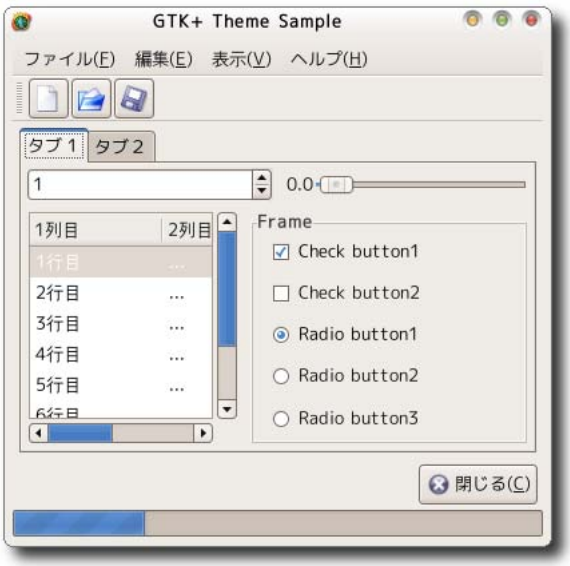


(a) ClearVine

(b) Grand-Canyon



(c) Xfce-4.0



(d) 本書で使用してるオリジナルテーマ

図 A.4　さまざまな GTK+ テーマ

付録 B

GtkStockItem 一覧

GTK+ では予め GtkStockItem と呼ばれるアイコンが定義されています．図 B.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[*1]．

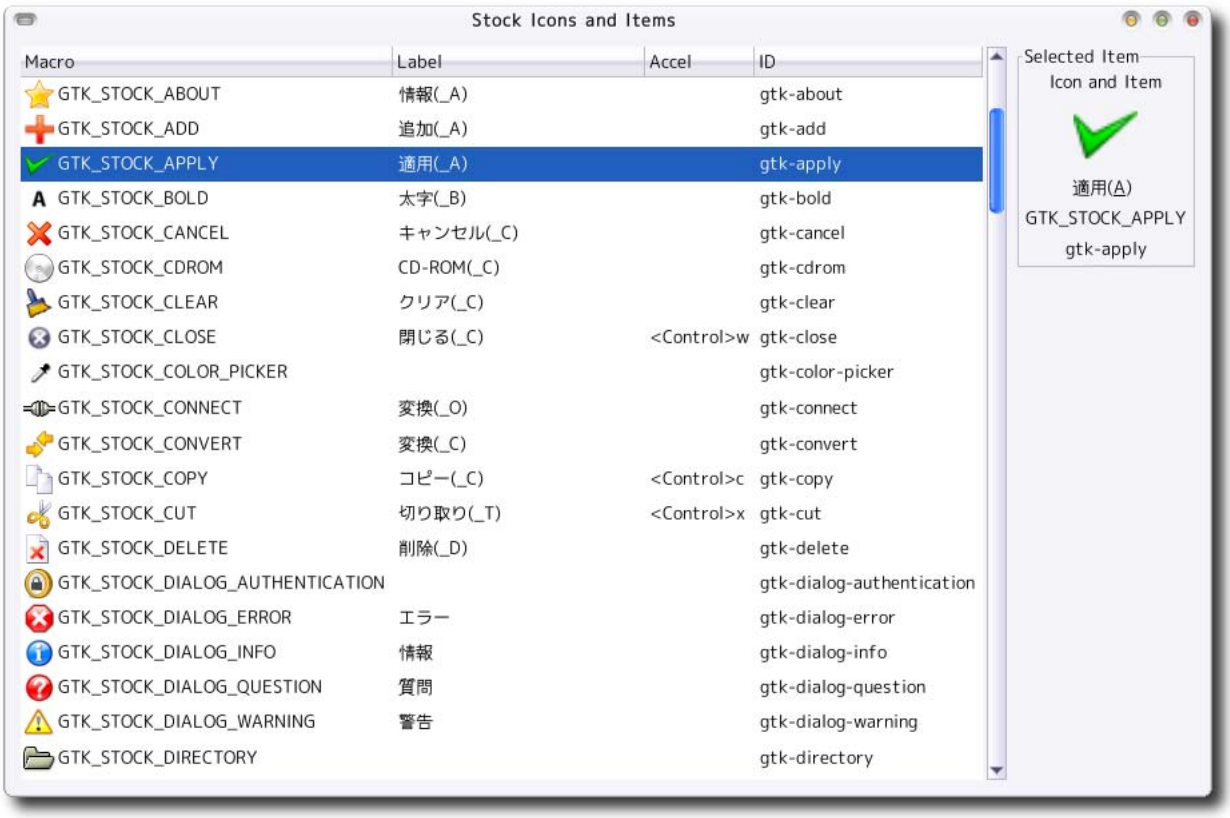


図 B.1　GtkStockItem の例

[*1] アイコンは GTK+ のテーマによって異なります．

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 C

サンプルプログラムのソースコードリスト

本書で扱ったプログラムのソースコードは http://www.iim.ics.tut.ac.jp/~sugaya/books/GUI-ApplicationProgramming/ からダウンロードすることができます．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りますのでご了承ください．

```
gtk-sources/
  +--- ImageOperator/
  +--- custom_widget/
         +--- gtkiconbutton/
  +--- gdk/
  +--- gdkpixbuf/
         +--- display/
         +--- image_application/
         +--- read/
  +--- glib/
         +--- file/
         +--- ghashtable/
         +--- glist/
         +--- timer/
         +--- utf8/
  +--- gnome/
         +--- gnome_about_dialog/
         +--- gnome_file_entry/
         +--- sample/
  +--- gtk/
         +--- 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/
+--- lesson/
        +--- hello_world/
        +--- packing/
        +--- signal/
        +--- table/
```

索引

GTK+/GNOME **による** GUI **アプリケーションプログラミング**

2006 年　3 月　 1 日　初　版

2007 年　4 月　30 日　第 2 版

著者　　菅谷 保之

E-mail　sugaya@iim.ics.tut.ac.jp