Programming GTK+
Implementation of a Text Editor

Mohamed A. Aslan

February 17, 2005

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Introduction

Gtk+ The Gimp Toolkit is a library for creating GUI applications, It uses the xlib which is the main library for GUI. Gtk+ works on many UNIX-like platforms as Linux, BSD, and it also runs on Windows and MacOSX.
Gtk+ is released unders the terms of the GNU library General Public License (LGPL), which means that is free and open source library, and you can modify it as you like under the terms of LGPL license. Gtk+ ues a C-based object-oriented architecture, it also can be used by other languages rather than C, including C++, Objective-C, Guile/Scheme, Perl, Pythin, TOM, Ada95, Pascal and Eiffel.
For more information and advanced topics you should consult the Gtk+ manual reference found at : `http://www.gtk.org`

1.2 Requirements for this book

1. A UNIX-like box as Linux or BSD

2. Xlib: Xfree86 or Xorg

3. Glib, Pango, ATK, GdkPixbuf, Gdk, Gtk+

4. GCC

Mostly every Linux distribution includes those requirements.

1.3 Compiling The Examples

For compiling the examples in this book you can change to the directory containing the example and just type make, or you can compile them manually by typing the following:

```
cc `pkg-config --cflags --libs gtk+-2.0` example.c -o example
```

Note: to run this program just type:

```
./example
```

And not

```
example
```

Like Dos/Windows platform, which indicates that the executable is in the current directory, because the current directory isn't included in the path. **Note:** you should consult the Gtk+ manual reference located at: `http://developer.gnome.org/doc/API/2.0/gtk/index.html`

1.4 Outline of the book

Each chapter talks about a certain Gtk+ Widget (Object), where it starts by the description of the used functions and then the implementation of the example. In the last chapter we will write a small text editor, which you can modify its code freely and expand it according to your needs.

Chapter 2

Windows

2.1 Description

Every object in Gtk+ is called Widget and every widget has it's type.
We use gtk_set_locale() function for the Initialization of the internationalization support for Gtk+. gtk_init() will initialize everything needed to operate Gtk+ and parses some standard command line options: argc, argv. gtk_widget_show_all() which shows the widget with all of its components. And gtk_main() which runs the Gtk main loop till gtk_main_quit() is called. These functions are very important functions and will be used in all of the examples included in this book.

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

This function creates a Gtk+ toplevel window and returns it's handle.

```
void gtk_window_set_title(GtkWindow *window, const gchar *title);
```

This function sets the title of the GtkWindow.

```
void gtk_window_set_position(GtkWindow *window, GtkWindowPosition position↩
    );
```

this function is used to set the position of the Gtk+ window.

```
void gtk_window_set_default_size(GtkWindow *window, gint width, gint ↩
    height);
```

this function is used to set the default size of the Gtk+ window.

```
GTK_WINDOW()
```

A macro which casts the GtkWidget window as GtkWindow.

Gtk+ window positions

- GTK_WIN_POS_CENTER
- GTK_WIN_POS_MOUSE
- GTK_WIN_POS_CENTER_ALWAYS

The first one sets the window position to the center of the screen, the second one sets the window position under the mouse cursor, while the third keep the window centred even its size changed.

2.2 Implementation

Example 1

Listing 2.1: Window 1

```
1  #include <gtk/gtk.h>
2  GtkWidget *window;
3  int main(int argc, char **argv)
4  {
5      gtk_set_locale();
6      gtk_init(&argc, &argv);
7      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8      gtk_widget_show_all(window);
9      gtk_main();
10     return 0;
11 }
```

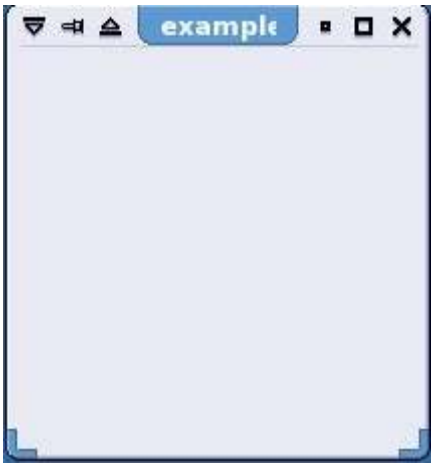


Figure 2.1: Window 1

Example 2

Listing 2.2: Window 2

```
1  #include <gtk/gtk.h>
2  GtkWidget *window;
3  int main(int argc, char **argv)
4  {
5      gtk_set_locale();
6      gtk_init(&argc, &argv);
7      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8      gtk_window_set_title(GTK_WINDOW(window), "Gtk+ Example 2");
9      gtk_widget_show_all(window);
10     gtk_main();
11     return 0;
12 }
```

Figure 2.2: Window 2

Example 3

Listing 2.3: Window 3

```
1  #include <gtk/gtk.h>
2  GtkWidget *window;
3  int main(int argc, char **argv)
4  {
5      gtk_set_locale();
6      gtk_init(&argc, &argv);
7      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8      gtk_window_set_title(GTK_WINDOW(window), "Gtk+ Example 3");
9      gtk_window_set_default_size(GTK_WINDOW(window), 500, 500);
10     gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
11     gtk_widget_show_all(window);
12     gtk_main();
13     return 0;
14 }
```

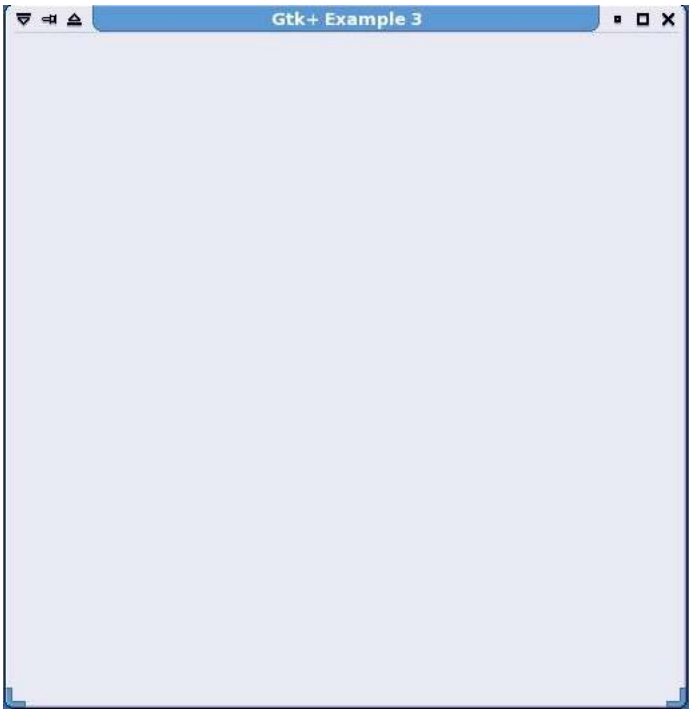


Figure 2.3: Window 3

Chapter 3

Labels

3.1 Description

Labels is a GtkWidget used to display text.

```
GtkWidget *gtk_label_new(const char *str);
```

This function creates a new label with the label text as it's argument.

```
Void gtk_label_set_text(GtkLabel *label, const char *str);
```

This function changes the text of the label.

Another important function

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

This function attaches a child widget to its parent as: a label to a window.

3.2 Implementation

Example

Listing 3.1: Label

```
1 #include <gtk/gtk.h>
2 GtkWidget *window, *label;
3 int main(int argc, char **argv)
4 {
```

```
5       gtk_set_locale();
6       gtk_init(&argc, &argv);
7       window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8       gtk_window_set_title(GTK_WINDOW(window), "Gtk+ Example4");
9       gtk_window_set_default_size(GTK_WINDOW(window), 200, 200);
10      gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
11      label=gtk_label_new("HelloWorld");
12      gtk_container_add(GTK_CONTAINER(window), label);
13      gtk_widget_show_all(window);
14      gtk_main();
15      return 0;
16  }
```

Figure 3.1: Label

Chapter 4

Entries

4.1 Description

Entries are widgets used to enter text and numbers, Entries have only one line for entering the text.

```
GtkWidget *gtk_entry_new(void);
```

This function creates a new Gtk+ entry.

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

This function is used to set the text that appears in the GtkEntry.

```
void gtk_entry_append(GtkEntry *entry, const gchar *text);
```

This function is used to append text to the end of the text of the GtkEntry.

```
Void gtk_entry_prepend(GtkEntry *entry, const gchar *text);
```

This function is used to prepend text to the beginning of the text of the GtkEntry.

```
G_CONST_RETURN gchar *gtk_entry_get_text(GtkEntry *entry);
```

This function returns the text of the GtkEntry.

4.2 Implementation

Example

Listing 4.1: Entry

```
#include <gtk/gtk.h>
GtkWidget *window, *entry;
int main(int argc, char **argv)
{
    gtk_set_locale();
    gtk_init(&argc, &argv);
    window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Gtk+ Example 5");
    gtk_window_set_default_size(GTK_WINDOW(window), 200, 200);
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
    entry=gtk_entry_new();
    gtk_entry_set_text(GTK_ENTRY(entry), "HelloWorld");
    gtk_entry_append_text(GTK_ENTRY(entry), "-");
    gtk_entry_prepend_text(GTK_ENTRY(entry), ".");
    gtk_container_add(GTK_CONTAINER(window), entry);
    gtk_widget_show_all(window);
    gtk_main();
    return 0;
}
```

Figure 4.1: Entry

Chapter 5

Text View

5.1 Description

Text Views are widgets used to enter multi line text unlike text entries with just one line.

```
GtkWidget *gtk_text_view_new(void);
```

This function creates a new text view.

```
void gtk_text_view_set_buffer(GtkTextView *text_view, GtkTextBuffer *↩
    buffer);
```

Buffer is a place for storing the text of the text view in it.
This function sets the buffer of the text view.

```
GtkTextBuffer *gtk_text_buffer_new(GtkTextTagTable *table);
```

This function creates a new text buffer which can be used with a text view.

```
void gtk_text_buffer_set_text(GtkTextBuffer *buffer, const gchar *text, ↩
    gint len);
```

This function is used to set the text stored in the text buffer.

```
gchar *gtk_text_buffer_get_text(GtkTextbuffer *buffer, const GtkTextIter *↩
    start, GtkText Iter *end, gboolean include_hidden_chars);
```

This function is used to get the text stored in a text buffer, where GtkTextIter is widget used to store a position in a text buffer, so start points to the

start of buffer and end points to the end of the buffer which we get text from it.

Another two important functions

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

This function gets the start iter of a GtkTextBuffer.

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

This function gets the end iter of a GtkTextbuffer.

5.2 Implementation

Example

Listing 5.1: Text View

```
1  #include <gtk/gtk.h>
2  GtkWidget *window, *textview;
3  GtkTextBuffer *buffer;
4  int main(int argc, char **argv)
5  {
6      gtk_set_locale();
7      gtk_init(&argc, &argv);
8      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
9      gtk_window_set_title(GTK_WINDOW(window), "Gtk+ Example6");
10     gtk_window_set_default_size(GTK_WINDOW(window), 400, 400);
11     gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
12     buffer=gtk_text_buffer_new(NULL);
13     gtk_text_buffer_set_text(buffer, "HelloWorld", 10);
14     textview=gtk_text_view_new();
15     gtk_text_view_set_buffer(GTK_TEXT_VIEW(textview), buffer);
16     gtk_container_add(GTK_CONTAINER(window), textview);
17     gtk_widget_show_all(window);
18     gtk_main();
19     return 0;
20 }
```

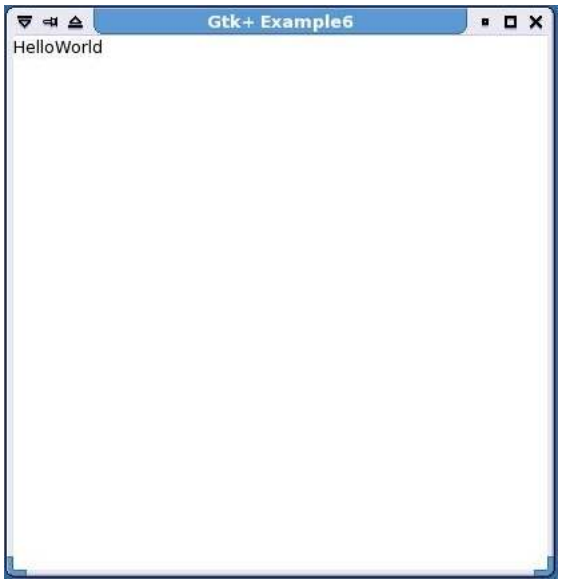


Figure 5.1: Text View

Chapter 6

Buttons

6.1 Description

Buttons are widgets that you click on them to proceed something.

```
GtkWidget *gtk_button_new(void);
```

This function creates a new button.

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

This function creates a new button with label.
Note: label can be changed later.

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

This function creates a new button, by coping it from the Gtk stock, this button usually contains a predefined image.

```
GtkWidget *gtk\_button_new_with_mnemonic(const gchar *label);
```

This function creates a new button with label but if the label is preceded by underscore the following later will be underlined, this underlined character represents a keyboard accelerator called a mnemonic. Pressing Alt and that key activates the button.

6.2 Implementation

Example 1

Listing 6.1: Button 1

```
#include <gtk/gtk.h>
GtkWidget *window, *button;
int main(int argc, char **argv)
{
    gtk_set_locale();
    gtk_init(&argc, &argv);
    window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Gtk+ Example 7");
    gtk_window_set_default_size(GTK_WINDOW(window), 200, 200);
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
    button=gtk_button_new_with_label("HelloWorld");
    gtk_container_add(GTK_CONTAINER(window), button);
    gtk_widget_show_all(window);
    gtk_main();
    return 0;
}
```

Figure 6.1: Button 1

Example 2

Listing 6.2: Button 2

```
#include <gtk/gtk.h>
GtkWidget *window, *button;
int main(int argc, char **argv)
{
    gtk_set_locale();
    gtk_init(&argc, &argv);
    window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Gtk+ Example 8");
```

```
9       gtk_window_set_default_size(GTK_WINDOW(window), 200, 200);
10      gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
11      button=gtk_button_new_from_stock(GTK_STOCK_OK);
12      gtk_container_add(GTK_CONTAINER(window), button);
13      gtk_widget_show_all(window);
14      gtk_main();
15      return 0;
16  }
```

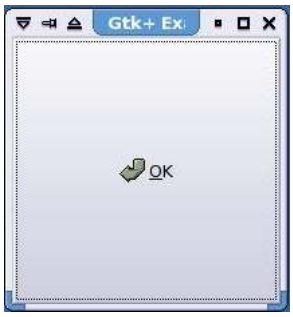


Figure 6.2: Button 2

The Gtk+ Stock List

1. GTK_STOCK_ADD
2. GTK_STOCK_APPLY
3. GTK_STOCK_BOLD
4. GTK_STOCK_CANCEL
5. GTK_STOCK_CDROM
6. GTK_STOCK_CLEAR
7. GTK_STOCK_CLOSE
8. GTK_STOCK_COLOR_PICKER
9. GTK_STOCK_CONVERT
10. GTK_STOCK_COPY
11. GTK_STOCK_CUT
12. GTK_STOCK_DELETE

13. GTK_STOCK_DIALOG_ERROR
14. GTK_STOCK_DIALOG_INFO
15. GTK_STOCK_DIALOG_QUESTION
16. GTK_STOCK_DIALOG_WARNING
17. GTK_STOCK_DND
18. GTK_STOCK_DND_MULTIPLE
19. GTK_STOCK_EXECUTE
20. GTK_STOCK_FIND
21. GTK_STOCK_FIND_AND_REPLACE
22. GTK_STOCK_FLOPPY
23. GTK_STOCK_GOTO_BOTTOM
24. GTK_STOCK_GOTO_FIRST
25. GTK_STOCK_GOTO_LAST
26. GTK_STOCK_GOTO_TOP
27. GTK_STOCK_GO_BACK
28. GTK_STOCK_GO_DOWN
29. GTK_STOCK_GO_FORWARD
30. GTK_STOCK_GO_UP
31. GTK_STOCK_HELP
32. GTK_STOCK_HOME
33. GTK_STOCK_INDEX
34. GTK_STOCK_ITALIC
35. GTK_STOCK_JUMP_TO
36. GTK_STOCK_JUSTIFY_CENTER
37. GTK_STOCK_JUSTIFY_FILL

38. GTK_STOCK_JUSTIFY_LEFT
39. GTK_STOCK_JUSTIFY_RIGHT
40. GTK_STOCK_MISSING_IMAGE
41. GTK_STOCK_NEW
42. GTK_STOCK_NO
43. GTK_STOCK_OK
44. GTK_STOCK_OPEN
45. GTK_STOCK_PASTE
46. GTK_STOCK_PREFERENCES
47. GTK_STOCK_PRINT
48. GTK_STOCK_PRINT_PREVIEW
49. GTK_STOCK_PROPERTIES
50. GTK_STOCK_QUIT
51. GTK_STOCK_REDO
52. GTK_STOCK_REFRESH
53. GTK_STOCK_REMOVE
54. GTK_STOCK_REVERT_TO_SAVED
55. GTK_STOCK_SAVE
56. GTK_STOCK_SAVE_AS
57. GTK_STOCK_SELECT_COLOR
58. GTK_STOCK_SELECT_FONT
59. GTK_STOCK_SORT_ASCENDING
60. GTK_STOCK_SORT_DESCENDING
61. GTK_STOCK_SPELL_CHECK
62. GTK_STOCK_STOP

63. GTK_STOCK_STRINGTHROUGH
64. GTK_STOCK_UNDELETE
65. GTK_STOCK_UNDERLINE
66. GTK_STOCK_UNDO
67. GTK_STOCK_YES
68. GTK_STOCK_ZOOM_100
69. GTK_STOCK_ZOOM_FIT
70. GTK_STOCK_ZOOM_IN
71. GTK_STOCK_ZOOM_OUT

Chapter 7

Status Bars

7.1 Description

Status Bars are Gtk+ widgets used to pop messages.

```
GtkWidget *gtk_statusbar_new(void);
```

This function creates a new Gtk+ status bar.

```
Guint gtk_statusbar_push(GtkStatusbar *statusbar, guint context_id, const ↩
    gchar *text);
```

This function pushes text to the status bar, where context_id is the message id.

7.2 Implementation

Example

Listing 7.1: Status Bar

```
1  #include <gtk/gtk.h>
2  GtkWidget *window, *statusbar;
3  int main(int argc, char **argv)
4  {
5      gtk_set_locale();
6      gtk_init(&argc, &argv);
7      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8      gtk_window_set_title(GTK_WINDOW(window), "Gtk+ Example 9");
9      gtk_window_set_default_size(GTK_WINDOW(window), 100, 50);
10     gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
```

```
11      statusbar=gtk_statusbar_new();
12      gtk_statusbar_push(GTK_STATUSBAR(statusbar), 1, "HelloWorld");
13      gtk_container_add(GTK_CONTAINER(window), statusbar);
14      gtk_widget_show_all(window);
15      gtk_main();
16      return 0;
17  }
```

Figure 7.1: Status Bar

Chapter 8

Tool Bars

8.1 Description

Tool Bars is Gtk+ widget, which contains child widgets as buttons.

```
GtkWidget *gtk_toolbar_new(void);
```

This function creates a new tool bar and returns its handle.

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

This function appends a new item into a GtkToolBar.
Note: we will pass null in GtkSignalFunc callback and gponter user_data, because we will use g_signals rather GtkSignals, which will describe it later in this book.

```
GtkWidget *gtk_toolbar_prepend_item(GtkToolbar *toolbar, const char *text,↩
     const char *tooltip_text, const char *tooltip_privte_text, GtkWidget ↩
    *icon, GtkSignalFunc callback, gpointer user_data);
```

This function is the same as the previous one, but it prepends the item.

```
GktWidget *gtk_toolbar_insert_stock(GtkToolBar *toolbar, const gchar *↩
    stock_id, const char *tooltip_text, const char *tooltip_private_text, ↩
    GtkSignalFunc callback, gpointer user_data, gint position);
```

This function insert Gtk+ stock item in the toolbar, if the position is -1 the item will be attached at the end of the toolbar.

```
void gtk_toolbar_set_style(GtkToolbar *toolbar, GtkToolbarStyle style);
```

This function sets the style of a toolbar whether it contains only icons or only text or both icons and text.

Gtk+ Toolbar Styles

- GTK_TOOLBAR_ICONS
- GTK_TOOLBAR_TEXT
- GTK_TOOLBAR_BOTH

8.2 Implementation

Example

Listing 8.1: Tool Bar

```
1  #include <gtk/gtk.h>
2  GtkWidget *window, *toolbar, *button1, *button2;
3  int main(int argc, char **argv)
4  {
5      gtk_set_locale();
6      gtk_init(&argc, &argv);
7      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8      gtk_window_set_default_size(GTK_WINDOW(window), 150, 50);
9      gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
10     toolbar=gtk_toolbar_new();
11     gtk_toolbar_set_style(GTK_TOOLBAR(toolbar), GTK_TOOLBAR_BOTH);
12     button1=gtk_toolbar_append_item(GTK_TOOLBAR(toolbar), "HelloWorld", "↩
           This is helloworld", NULL, NULL, NULL, NULL);
13     gtk_toolbar_append_space(GTK_TOOLBAR(toolbar));
14     button2=gtk_toolbar_insert_stock(GTK_TOOLBAR(toolbar), GTK_STOCK_OK, "↩
           This is ok", NULL, NULL, NULL, -1);
15     gtk_container_add(GTK_CONTAINER(window), toolbar);
16     gtk_widget_show_all(window);
17     gtk_main();
18     return 0;
19 }
```

Figure 8.1: Tool Bar

Chapter 9

Menus

9.1 Description

Menu bars are Gtk+ widgets that contains items called menus, that contain menu items, which we can attach a submenu to it.

```
GtkWidget *gtk_menu_bar_new(void);
```

This function creates a new menu bar.

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

This function creates a new menu item.

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

This function creates a new menu item with menmomic label.

```
GtkWidget *gtk_menu_new(void);
```

This function creates a new menu.

```
GtkWidget *gtk_menu_item_set_submenu(GtkMenuItem *menu_item, GtkWidget *↩
    submenu);
```

This function set the submenu of a menu item.

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

This function creates a menu item with image from the Gtk+ stock, where accel_group contains the keyboard accelerator group, which is the button combination used to activate the menu items.

9.2 Implementation

Example

Listing 9.1: Menu

```
1  #include <gtk/gtk.h>
2  GtkWidget *window, *menubar, *menuitem_file, *menu_file, *menuitem_quit;
3  int main(int argc, char **argv)
4  {
5      gtk_set_locale();
6      gtk_init(&argc, &argv);
7      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8      gtk_window_set_default_size(GTK_WINDOW(window), 300, 100);
9      menubar=gtk_menu_bar_new();
10     gtk_container_add(GTK_CONTAINER(window), menubar);
11     menuitem_file=gtk_menu_item_new_with_mnemonic("_File");
12     gtk_container_add(GTK_CONTAINER(menubar), menuitem_file);
13     menu_file=gtk_menu_new();
14     gtk_menu_item_set_submenu(GTK_MENU_ITEM(menuitem_file), menu_file);
15     menuitem_quit=gtk_image_menu_item_new_from_stock(GTK_STOCK_QUIT, NULL)↩
           ;
16     gtk_container_add(GTK_CONTAINER(menu_file), menuitem_quit);
17     gtk_widget_show_all(window);
18     gtk_main();
19     return 0;
20 }
```

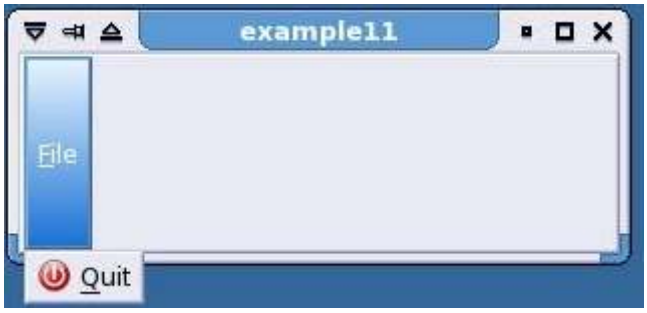


Figure 9.1: Menu

Chapter 10

Scrolled Windows

10.1 Description

Scrolled window is just a parent container for other widget, scrolled window is mostly used to contain text view, where it adds scroll bars to it.

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

This function creates a new scrolled window with the first argument is the horizontal adjustment while the second is the vertical adjustment.
Note: we pass to both arguments null to cause the scrolled window to create them for us.

10.2 Implementation

Example

Listing 10.1: Scrolled Window

```
1  #include <gtk/gtk.h>
2  GtkWidget *window, *scrolled_window, *text_view;
3  int main(int argc, char **argv)
4  {
5      gtk_set_locale();
6      gtk_init(&argc, &argv);
7      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8      gtk_window_set_default_size(GTK_WINDOW(window), 400, 300);
9      scrolled_window=gtk_scrolled_window_new(NULL, NULL);
10     gtk_container_add(GTK_CONTAINER(window), scrolled_window);
11     text_view=gtk_text_view_new();
12     gtk_container_add(GTK_CONTAINER(scrolled_window), text_view);
```

```
13      gtk_widget_show_all(window);
14      gtk_main();
15      return 0;
16  }
```

Figure 10.1: Scrolled Window

Chapter 11

Boxes

11.1 Description

What about adding more than widget to the same window ?
Boxes are widgets which enables you to add more than widget to a window.

```
GtkWidget *gtk_hbox_new(gboolean homogenous, gint spacing);
```

This function creates a horizontal box where widgets are added horizontally, setting homogeneous into true makes all children are to be given equal space allotments, the spacing is the number of pixels to place by default between children widgets.

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

This function creates a vertical box where widgets are added vertically.

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

This function adds a child widget to the box, if expand is set to true the child will be given extra space allocated to box, this space will be divided evenly between all children of box, if fill is set to true the space given to child by the expand option is actually allocated to child, rather than just padding it. This parameter has no effect if expand is set to FALSE, padding is the extra space in pixels to put between this child and its neighbours.

11.2 Implementation

Example 1: HBox

Listing 11.1: Box 1

```
1  #include <gtk/gtk.h>
2  GtkWidget *window, *hbox, *button1, *button2;
3  int main(int argc, char **argv)
4  {
5      gtk_set_locale();
6      gtk_init(&argc, &argv);
7      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
8      gtk_window_set_default_size(GTK_WINDOW(window), 100, 100);
9      gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
10     button1=gtk_button_new_from_stock(GTK_STOCK_OK);
11     button2=gtk_button_new_with_label("HellWorld");
12     hbox=gtk_hbox_new(FALSE, 0);
13     gtk_box_pack_start(GTK_BOX(hbox), button1, FALSE, FALSE, 0);
14     gtk_box_pack_start(GTK_BOX(hbox), button2, FALSE, FALSE, 0);
15     gtk_container_add(GTK_CONTAINER(window), hbox);
16     gtk_widget_show_all(window);
17     gtk_main();
18     return 0;
19 }
```

Figure 11.1: Box 1

Example 2: VBox

Listing 11.2: Box 2

```
#include <gtk/gtk.h>
GtkWidget *window, *vbox, *button1, *button2;
int main(int argc, char **argv)
{
    gtk_set_locale();
    gtk_init(&argc, &argv);
    window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(window), 100, 50);
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
    button1=gtk_button_new_from_stock(GTK_STOCK_OK);
    button2=gtk_button_new_with_label("HelloWorld");
    vbox=gtk_vbox_new(FALSE, 0);
    gtk_box_pack_start(GTK_BOX(vbox), button1, FALSE, FALSE, 0);
    gtk_box_pack_start(GTK_BOX(vbox), button2, FALSE, FALSE, 0);
    gtk_container_add(GTK_CONTAINER(window), vbox);
    gtk_widget_show_all(window);
    gtk_main();
    return 0;
}
```

Figure 11.2: Box 2

Chapter 12

Tables

12.1 Description

Tables are widgets used to attack child widgets to parent ones, tables differ from boxes where tables can add widgets beside each other and upwards each others , which means table is able to add widgets horizontally and vertically at the same time.

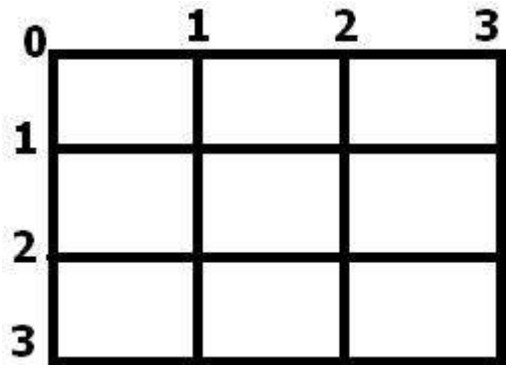
```
GtkWidget *gtk_table_new(guint rows, guint columns, gboolean homogeneous);
```

This function creates a new table where rows is the number of rows and columns is the number of columns of this table, while if homogeneous is set into true which makes all children are to be given equal space allotments, the spacing is the number of pixels to place by default between children widgets.

```
void gtk_table_attach_defaults(GtkTable *table, GtkWidget *widget, guint ↩
    left_attach, guint right_attach, guint top_attach, guint bottom_attach↩
    );
```

This function attaches child widgets to a table.

- left_attach: The column number to attach the left side of the child widget to.
- right_attach: The column number to attach the right side of the child widget to.
- top_attach: The row number to attach the top of the child widget to.
- bottom_attach: The row number to attach the bottom of the child widget to.



12.2 Implementation

Example

Listing 12.1: Table

```
#include <gtk/gtk.h>
GtkWidget *window, *table, *button1, *button2, *button3, *button4;
int main(int argc, char **argv)
{
    gtk_set_locale();
    gtk_init(&argc, &argv);
    window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_default_size(GTK_WINDOW(window), 300, 300);
    gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);
    button1=gtk_button_new_with_label("Foo");
    button2=gtk_button_new_with_label("Bar");
    button3=gtk_button_new_from_stock(GTK_STOCK_OK);
    button4=gtk_button_new_from_stock(GTK_STOCK_QUIT);
    table=gtk_table_new(2, 2, TRUE);
    gtk_table_attach_defaults(GTK_TABLE(table), button1, 0, 1, 0, 1);
    gtk_table_attach_defaults(GTK_TABLE(table), button2, 1, 2, 0, 1);
    gtk_table_attach_defaults(GTK_TABLE(table), button3, 0, 1, 1, 2);
    gtk_table_attach_defaults(GTK_TABLE(table), button4, 1, 2, 1, 2);
    gtk_container_add(GTK_CONTAINER(window), table);
    gtk_widget_show_all(window);
    gtk_main();
    return 0;
}
```

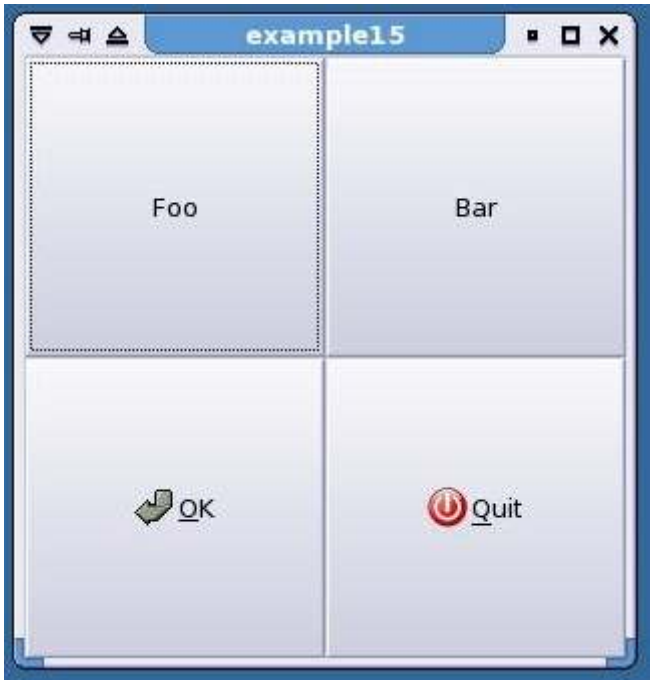


Figure 12.1: Table

Chapter 13

Signals

13.1 Description

What happens if you click on button, it emits a signal which call what is called callback function.

```
#define g_signal_connect(instance, detailed_signal, c_handler, data);
```

- instance: The instance to connect to
- detailed_signal: A string of the signal name
- c_handler: The GCallback to connect
- data: The handler id, which is usually null

The Callback Function

```
void user_function(GtkWidget *widget, gpointer user_data);
```

The call back function is the function called by Gtk+ when a certain event happens, this function should be connected to the widget and the signal name first.

Some Signals

- activate : As clicked but for menu items
- clicked
- pressed
- released

13.2 Implementation

Example

Listing 13.1: Signals

```
1  #include <gtk/gtk.h>
2  GtkWidget *window, *button;
3  void do_exit(GtkWidget *, gpointer);
4  int main(int argc, char **argv)
5  {
6      gtk_set_locale();
7      gtk_init(&argc, &argv);
8      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
9      gtk_window_set_default_size(GTK_WINDOW(window), 100, 100);
10     button=gtk_button_new_with_label("Exit");
11     gtk_container_add(GTK_CONTAINER(window), button);
12     g_signal_connect(G_OBJECT(button), "clicked", G_CALLBACK(do_exit), ↩
           NULL);
13     gtk_widget_show_all(window);
14     gtk_main();
15     return 0;
16 }
17 void do_exit(GtkWidget *widget, gpointer data)
18 {
19     exit(0);
20 }
```

Figure 13.1: Signals

Chapter 14

File Selection

14.1 Description

File selection is a dialog used to select files either for opening or saving.

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

This function creates a new file selection with title, but the file selection need it's on gtk_widget_show_all()

```
G_CONST_RETURN gchar *gtk_file_selection_get_filename(GtkFileSelection *↩
    filesel);
```

This function gets the complete path and name of the selected file.

14.2 Implementation

Example

Listing 14.1: File Selection

```
1 #include <gtk/gtk.h>
2 GtkWidget *window, *vbox, *label, *button, *file_select, *file_select_ok, ↩
      *file_select_cancel;
3 void get_filename(GtkWidget *, gpointer);
4 void open_file_select(GtkWidget *, gpointer);
5 void    file_select_exit(GtkWidget *, gpointer);
6 int main(int argc, char **argv)
7 {
8     gtk_set_locale();
9     gtk_init(&argc, &argv);
```

```
10      window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
11      gtk_window_set_default_size(GTK_WINDOW(window), 200, 100);
12      vbox=gtk_vbox_new(FALSE, 0);
13      gtk_container_add(GTK_CONTAINER(window), vbox);
14      label=gtk_label_new(NULL);
15      gtk_box_pack_start(GTK_BOX(vbox), label, FALSE, FALSE, 0);
16      button=gtk_button_new_with_label("Select File");
17      gtk_box_pack_start(GTK_BOX(vbox), button, FALSE, FALSE, 0);
18      g_signal_connect(G_OBJECT(button), "clicked", G_CALLBACK(↩
            open_file_select), NULL);
19      gtk_widget_show_all(window);
20      gtk_main();
21      return 0;
22  }
23  void open_file_select(GtkWidget *widget, gpointer data)
24  {
25      file_select=gtk_file_selection_new("Select A File ");
26      file_select_ok=GTK_FILE_SELECTION(file_select)->ok_button;
27      file_select_cancel=GTK_FILE_SELECTION(file_select)->cancel_button;
28      g_signal_connect(G_OBJECT(file_select_ok), "clicked", G_CALLBACK(↩
            get_filename), NULL);
29      g_signal_connect(G_OBJECT(file_select_cancel), "clicked", G_CALLBACK(↩
            file_select_exit), NULL);
30      gtk_widget_show_all(file_select);
31  }
32  void get_filename(GtkWidget *widget, gpointer data)
33  {
34      const gchar *filename;
35      filename=gtk_file_selection_get_filename(GTK_FILE_SELECTION(↩
            file_select));
36      gtk_label_set_text(GTK_LABEL(label), filename);
37      gtk_widget_destroy(file_select);
38  }
39  void file_select_exit(GtkWidget *widget, gpointer data)
40  {
41      gtk_widget_destroy(file_select);
42  }
```

Figure 14.1: File Selection Before Selection

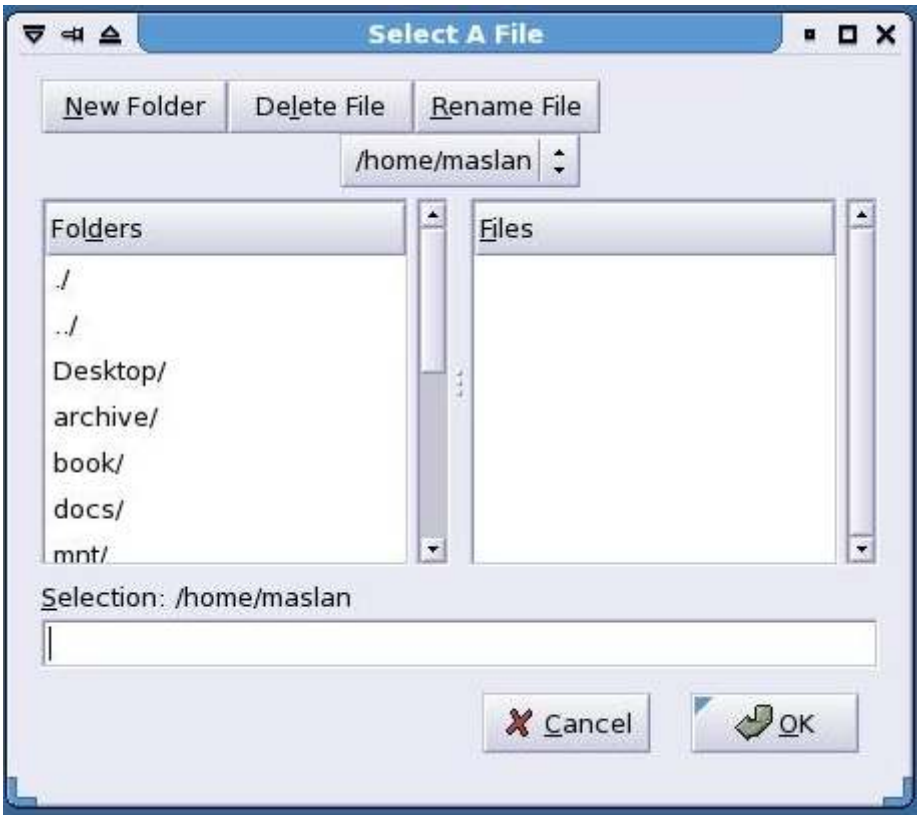


Figure 14.2: File Selection Dialog

Figure 14.3: File Selection After Selection

Chapter 15

Font Selection

15.1 Description

Font selection dialog is a widget like file selection, but it is used to select fonts not files.

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

This function creates font selection dialog with title.

```
gchar *gtk_font_selection_dialog_get_font_name(GtkFontSelectionDialog *fsd↩
    );
```

This function gets the name of the selected font.

```
PangoFontDescription *pango_font_description_from_string(const char *str);
```

This function returns a pango font description from the string containing the font name, pango is one of Gtk+ required libraries responsible for font stuff.

```
void gtk_widget_modify_font(GtkWidget *widget, PangoFontDescription ↩
    font_desc);
```

This function changes the font of a widget using pango font description.

15.2 Implementation

Example

Listing 15.1: Font Selection

```
1  #include <gtk/gtk.h>
2  GtkWidget *window, *vbox, *text_view, *button, *font_select_dlg;
3  GtkWidget *font_select_dlg_ok, *font_select_dlg_apply, *↩
       font_select_dlg_cancel;
4  void get_fontname(GtkWidget *, gpointer);
5  void apply_fontname(GtkWidget *, gpointer);
6  void open_font_select_dlg(GtkWidget *, gpointer);
7  void font_select_dlg_exit(GtkWidget *, gpointer);
8  int main(int argc, char **argv)
9  {
10     gtk_set_locale();
11     gtk_init(&argc, &argv);
12     window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
13     gtk_window_set_default_size(GTK_WINDOW(window), 300, 200);
14     vbox=gtk_vbox_new(FALSE, 0);
15     gtk_container_add(GTK_CONTAINER(window), vbox);
16     text_view=gtk_text_view_new();
17     gtk_box_pack_start(GTK_BOX(vbox), text_view, TRUE, TRUE, 0);
18     button=gtk_button_new_with_label("Select Font");
19     gtk_box_pack_start(GTK_BOX(vbox), button, FALSE, FALSE, 0);
20     g_signal_connect(G_OBJECT(button), "clicked", G_CALLBACK(↩
           open_font_select_dlg), NULL);
21     gtk_widget_show_all(window);
22     gtk_main();
23     return 0;
24 }
25 void open_font_select_dlg(GtkWidget *widget, gpointer data)
26 {
27     font_select_dlg=gtk_font_selection_dialog_new("Select Font");
28     font_select_dlg_ok=GTK_FONT_SELECTION_DIALOG(font_select_dlg)->↩
           ok_button;
29     font_select_dlg_apply=GTK_FONT_SELECTION_DIALOG(font_select_dlg)->↩
           apply_button;
30     font_select_dlg_cancel=GTK_FONT_SELECTION_DIALOG(font_select_dlg)->↩
           cancel_button;
31     g_signal_connect(G_OBJECT(font_select_dlg_ok), "clicked", G_CALLBACK(↩
           get_fontname), NULL);
32     g_signal_connect(G_OBJECT(font_select_dlg_apply), "clicked", ↩
           G_CALLBACK(apply_fontname), NULL);
33     g_signal_connect(G_OBJECT(font_select_dlg_cancel), "clicked", ↩
           G_CALLBACK(font_select_dlg_exit), NULL);
34     gtk_widget_show_all(font_select_dlg);
35 }
36 void get_fontname(GtkWidget *widget, gpointer data)
37 {
38     gchar *fontname;
39     PangoFontDescription *font_desc;
40     fontname=gtk_font_selection_dialog_get_font_name(↩
           GTK_FONT_SELECTION_DIALOG(font_select_dlg));
41     font_desc=pango_font_description_from_string(fontname);
42     gtk_widget_modify_font(text_view, font_desc);
43     gtk_widget_destroy(font_select_dlg);
44 }
45 void apply_fontname(GtkWidget *widget, gpointer data)
46 {
47     gchar *fontname;
48     PangoFontDescription *font_desc;
49     fontname=gtk_font_selection_dialog_get_font_name(↩
           GTK_FONT_SELECTION_DIALOG(font_select_dlg));
50     font_desc=pango_font_description_from_string(fontname);
```

```
51      gtk_widget_modify_font(text_view, font_desc);
52  }
53  void font_select_dlg_exit(GtkWidget *widget, gpointer data)
54  {
55      gtk_widget_destroy(font_select_dlg);
56  }
```

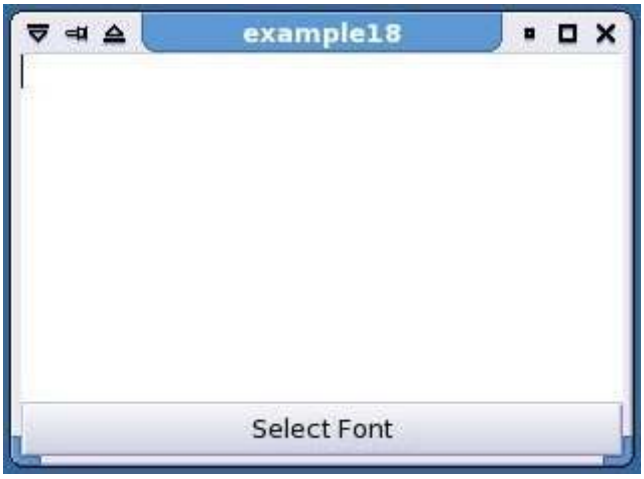


Figure 15.1: Font Selection Before Selection

Figure 15.2: Font Selection Dialog

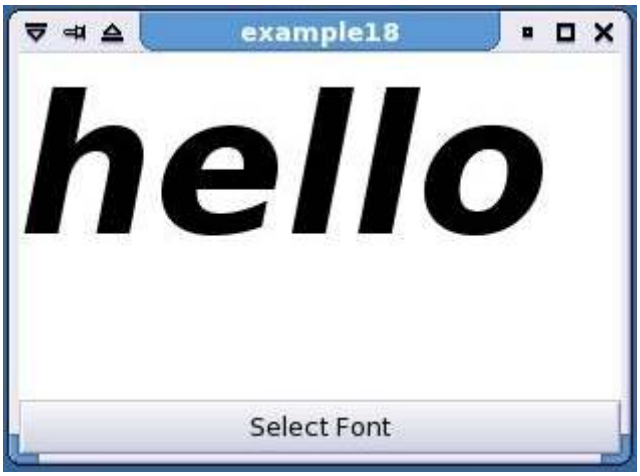


Figure 15.3: Font Selection After Selection

Chapter 16

Text Editor

16.1 Description

In this chapter we will implement a small and simple text editor to illustrate what we learned about Gtk+.

16.2 Implementation

Text Editor

Listing 16.1: textedit.h

```
1  #include <gtk/gtk.h>
2  #include <stdio.h>
3
4  #ifndef _MAX_FILE_SIZE
5  #define _MAX_FILE_SIZE 2048000  /* 4 MB */
6  #endif
7
8  /* Text Edit Functions */
9  void textedit_interface();
10 void textedit_signals();
11
12 /* Text Edit Widgets */
13 GtkWidget *window;
14 GtkWidget *vbox;
15 GtkWidget *menu_bar;
16 GtkWidget *menu_item_file;
17 GtkWidget *menu_file;
18 GtkWidget *menu_item_new;
19 GtkWidget *menu_item_open;
20 GtkWidget *menu_item_save;
21 GtkWidget *menu_item_saveas;
22 GtkWidget *menu_item_close;
23 GtkWidget *menu_item_quit;
```

```
24 GtkWidget *menu_item_edit;
25 GtkWidget *menu_edit;
26 GtkWidget *menu_item_preferences;
27 GtkWidget *tool_bar;
28 GtkWidget *button_new;
29 GtkWidget *button_open;
30 GtkWidget *button_save;
31 GtkWidget *button_saveas;
32 GtkWidget *button_close;
33 GtkWidget *button_quit;
34 GtkWidget *button_preferences;
35 GtkWidget *scrolled_window;
36 GtkWidget *text_view;
37 GtkWidget *open_file_dlg;
38 GtkWidget *open_file_dlg_ok;
39 GtkWidget *open_file_dlg_cancel;
40 GtkWidget *saveas_file_dlg;
41 GtkWidget *saveas_file_dlg_ok;
42 GtkWidget *saveas_file_dlg_cancel;
43 GtkWidget *font_dlg;
44 GtkWidget *font_dlg_ok;
45 GtkWidget *font_dlg_apply;
46 GtkWidget *font_dlg_cancel;
47
48 /* Call Backs Functions */
49 /* Menu Bar Call Backs */
50 void menu_item_new_activated(GtkWidget *, gpointer);
51 void menu_item_open_activated(GtkWidget *, gpointer);
52 void menu_item_save_activated(GtkWidget *, gpointer);
53 void menu_item_saveas_activated(GtkWidget *, gpointer);
54 void menu_item_close_activated(GtkWidget *, gpointer);
55 void menu_item_quit_activated(GtkWidget*, gpointer);
56 void menu_item_preferences_activated(GtkWidget *, gpointer);
57 /* Tool Bar Call Backs */
58 void button_new_clicked(GtkWidget *, gpointer);
59 void button_open_clicked(GtkWidget *, gpointer);
60 void button_save_clicked(GtkWidget *, gpointer);
61 void button_saveas_clicked(GtkWidget *, gpointer);
62 void button_close_clicked(GtkWidget *, gpointer);
63 void button_quit_clicked(GtkWidget *, gpointer);
64 void button_preferences_clicked(GtkWidget *, gpointer);
65 /* Open File Dialog Call Backs */
66 void open_file_dlg_ok_clicked(GtkWidget *, gpointer);
67 void open_file_dlg_cancel_clicked(GtkWidget *, gpointer);
68 /* Save As File Dialog Call Backs */
69 void saveas_file_dlg_ok_clicked(GtkWidget *, gpointer);
70 void saveas_file_dlg_cancel_clicked(GtkWidget *, gpointer);
71 /* Font Dialog Call Backs */
72 void font_dlg_ok_clicked(GtkWidget *, gpointer);
73 void font_dlg_cancel_clicked(GtkWidget *, gpointer);
74 void font_dlg_apply_clicked(GtkWidget *, gpointer);
75
76
77 /* Text Edit Functions */
78 void text_edit_new(void);
79 void text_edit_open(void);
80 void text_edit_save(void);
81 void text_edit_saveas(void);
82 void text_edit_close(void);
83 void text_edit_quit(void);
84 void text_edit_preferences(void);
85
```

```
86 /* Variables Declarations */
87 const gchar *filename;
```

Listing 16.2: textedit.c

```
1  #include <gtk/gtk.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "textedit.h"
5
6  int main(int argc, char **argv)
7  {
8      gtk_set_locale();
9      gtk_init(&argc, &argv);
10     textedit_interface();
11     textedit_signals();
12     gtk_widget_show_all(window);
13     gtk_main();
14     return 0;
15 }
16
17 /* THE GUI DESIGN */
18 void textedit_interface()
19 {
20     /* Create The Main Window */
21     window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
22     gtk_window_set_title(GTK_WINDOW(window), "Text Edit");
23     gtk_window_set_default_size(GTK_WINDOW(window), 700, 500);
24     gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);
25
26     /* Create The VBox & Attach It To The Main Window */
27     vbox=gtk_vbox_new(FALSE, 0);
28     gtk_container_add(GTK_CONTAINER(window), vbox);
29
30     /* Create The Menu Bar & Attach It To The VBox */
31     menu_bar=gtk_menu_bar_new();
32     gtk_box_pack_start(GTK_BOX(vbox), menu_bar, FALSE, FALSE, 0);
33     menu_item_file=gtk_menu_item_new_with_mnemonic("_File");
34     gtk_container_add(GTK_CONTAINER(menu_bar), menu_item_file);
35     menu_file=gtk_menu_new();
36     gtk_menu_item_set_submenu(GTK_MENU_ITEM(menu_item_file), menu_file);
37     menu_item_new=gtk_image_menu_item_new_from_stock(GTK_STOCK_NEW, NULL);
38     gtk_container_add(GTK_CONTAINER(menu_file), menu_item_new);
39     menu_item_open=gtk_image_menu_item_new_from_stock(GTK_STOCK_OPEN, NULL↩
           );
40     gtk_container_add(GTK_CONTAINER(menu_file), menu_item_open);
41     menu_item_save=gtk_image_menu_item_new_from_stock(GTK_STOCK_SAVE, NULL↩
           );
42     gtk_container_add(GTK_CONTAINER(menu_file), menu_item_save);
43     menu_item_saveas=gtk_image_menu_item_new_from_stock(GTK_STOCK_SAVE_AS,↩
            NULL);
44     gtk_container_add(GTK_CONTAINER(menu_file), menu_item_saveas);
45     menu_item_close=gtk_image_menu_item_new_from_stock(GTK_STOCK_CLOSE, ↩
           NULL);
46     gtk_container_add(GTK_CONTAINER(menu_file), menu_item_close);
47     menu_item_quit=gtk_image_menu_item_new_from_stock(GTK_STOCK_QUIT, NULL↩
           );
48     gtk_container_add(GTK_CONTAINER(menu_file), menu_item_quit);
49     menu_item_edit=gtk_menu_item_new_with_mnemonic("_Edit");
```

```
50      gtk_container_add(GTK_CONTAINER(menu_bar), menu_item_edit);
51      menu_edit=gtk_menu_new();
52      gtk_menu_item_set_submenu(GTK_MENU_ITEM(menu_item_edit), menu_edit);
53      menu_item_preferences=gtk_image_menu_item_new_from_stock(↩
            GTK_STOCK_PREFERENCES, NULL);
54      gtk_container_add(GTK_CONTAINER(menu_edit), menu_item_preferences);
55
56      /* Create The Tool Bar & Attach It To The VBox */
57      tool_bar=gtk_toolbar_new();
58      gtk_box_pack_start(GTK_BOX(vbox), tool_bar, FALSE, FALSE, 0);
59      gtk_toolbar_set_style(GTK_TOOLBAR(tool_bar), GTK_TOOLBAR_BOTH);
60      button_new=gtk_toolbar_insert_stock(GTK_TOOLBAR(tool_bar), ↩
            GTK_STOCK_NEW, "New File", NULL, NULL, NULL, -1);
61      button_open=gtk_toolbar_insert_stock(GTK_TOOLBAR(tool_bar), ↩
            GTK_STOCK_OPEN, "Open File", NULL, NULL, NULL, -1);
62      button_save=gtk_toolbar_insert_stock(GTK_TOOLBAR(tool_bar), ↩
            GTK_STOCK_SAVE, "Save File", NULL, NULL, NULL, -1);
63      button_saveas=gtk_toolbar_insert_stock(GTK_TOOLBAR(tool_bar), ↩
            GTK_STOCK_SAVE_AS, "Save As File", NULL, NULL, NULL, -1);
64      button_close=gtk_toolbar_insert_stock(GTK_TOOLBAR(tool_bar), ↩
            GTK_STOCK_CLOSE, "Close File", NULL, NULL, NULL, -1);
65      button_quit=gtk_toolbar_insert_stock(GTK_TOOLBAR(tool_bar), ↩
            GTK_STOCK_QUIT, "Quit", NULL, NULL, NULL, -1);
66      gtk_toolbar_append_space(GTK_TOOLBAR(tool_bar));
67      button_preferences=gtk_toolbar_insert_stock(GTK_TOOLBAR(tool_bar), ↩
            GTK_STOCK_PREFERENCES, "Preferences", NULL, NULL, NULL, -1);
68
69      /* Create The Scrolled Window & Attach It To The VBox */
70      scrolled_window=gtk_scrolled_window_new(NULL, NULL);
71      gtk_box_pack_start(GTK_BOX(vbox), scrolled_window, TRUE, TRUE, 0);
72
73      /* Create The Text View & Attach It To The Scrolled Window */
74      GtkTextBuffer *buffer;
75      text_view=gtk_text_view_new();
76      gtk_container_add(GTK_CONTAINER(scrolled_window), text_view);
77      buffer=gtk_text_buffer_new(NULL);
78      gtk_text_view_set_buffer(GTK_TEXT_VIEW(text_view), buffer);
79  }
80
81  void textedit_signals()
82  {
83      /*  Menu Bar Items Activate Signals */
84      g_signal_connect(G_OBJECT(menu_item_new), "activate", G_CALLBACK(↩
            menu_item_new_activated), NULL);
85      g_signal_connect(G_OBJECT(menu_item_open), "activate", G_CALLBACK(↩
            menu_item_open_activated), NULL);
86      g_signal_connect(G_OBJECT(menu_item_save), "activate", G_CALLBACK(↩
            menu_item_save_activated), NULL);
87      g_signal_connect(G_OBJECT(menu_item_saveas), "activate", G_CALLBACK(↩
            menu_item_saveas_activated), NULL);
88      g_signal_connect(G_OBJECT(menu_item_close), "activate", G_CALLBACK(↩
            menu_item_close_activated), NULL);
89      g_signal_connect(G_OBJECT(menu_item_quit), "activate", G_CALLBACK(↩
            menu_item_quit_activated), NULL);
90      g_signal_connect(G_OBJECT(menu_item_preferences), "activate", ↩
            G_CALLBACK(menu_item_preferences_activated), NULL);
91      /* Tool Bar Buttons Clicked Signals */
92      g_signal_connect(G_OBJECT(button_new), "clicked", G_CALLBACK(↩
            button_new_clicked), NULL);
93      g_signal_connect(G_OBJECT(button_open), "clicked", G_CALLBACK(↩
            button_open_clicked), NULL);
```

```
94      g_signal_connect(G_OBJECT(button_save), "clicked", G_CALLBACK(↩
            button_save_clicked), NULL);
95      g_signal_connect(G_OBJECT(button_saveas), "clicked", G_CALLBACK(↩
            button_saveas_clicked), NULL);
96      g_signal_connect(G_OBJECT(button_close), "clicked", G_CALLBACK(↩
            button_close_clicked), NULL);
97      g_signal_connect(G_OBJECT(button_quit), "clicked", G_CALLBACK(↩
            button_quit_clicked), NULL);
98      g_signal_connect(G_OBJECT(button_preferences), "clicked", G_CALLBACK(↩
            button_preferences_clicked), NULL);
99  }
100
101 /* The Call Back Functions */
102
103 void menu_item_new_activated(GtkWidget *widget, gpointer data)
104 {
105     text_edit_new();
106 }
107 void menu_item_open_activated(GtkWidget *widget, gpointer data)
108 {
109     text_edit_open();
110 }
111 void menu_item_save_activated(GtkWidget *widget, gpointer data)
112 {
113     text_edit_save();
114 }
115 void menu_item_saveas_activated(GtkWidget *widget, gpointer data)
116 {
117     text_edit_saveas();
118 }
119 void menu_item_close_activated(GtkWidget *widget, gpointer data)
120 {
121     text_edit_close();
122 }
123 void menu_item_quit_activated(GtkWidget *widget, gpointer data)
124 {
125     text_edit_quit();
126 }
127 void menu_item_preferences_activated(GtkWidget *widget, gpointer data)
128 {
129     text_edit_preferences();
130 }
131 void button_new_clicked(GtkWidget *widget, gpointer data)
132 {
133     text_edit_new();
134 }
135 void button_open_clicked(GtkWidget *widget, gpointer data)
136 {
137     text_edit_open();
138 }
139 void button_save_clicked(GtkWidget *widget, gpointer data)
140 {
141     text_edit_save();
142 }
143 void button_saveas_clicked(GtkWidget *widget, gpointer data)
144 {
145     text_edit_saveas();
146 }
147 void button_close_clicked(GtkWidget *widget, gpointer data)
148 {
149     text_edit_close();
150 }
```

```
151 void button_quit_clicked(GtkWidget *widget, gpointer data)
152 {
153     text_edit_quit();
154 }
155 void button_preferences_clicked(GtkWidget *widget, gpointer data)
156 {
157     text_edit_preferences();
158 }
159 void open_file_dlg_ok_clicked(GtkWidget *widget, gpointer data)
160 {
161     int bytes_read;
162     FILE *fp;
163     gchar text[_MAX_FILE_SIZE];
164     GtkTextBuffer *buffer;
165     GtkTextIter start, end;
166     filename=gtk_file_selection_get_filename(GTK_FILE_SELECTION(↩
            open_file_dlg));
167     if((fp=fopen(filename, "r"))==NULL)
168     {
169         exit(1);
170     }
171     while(!feof(fp))
172     {
173         bytes_read=fread(&text, sizeof(gchar), _MAX_FILE_SIZE, fp);
174     }
175     fclose(fp);
176 
177     buffer=gtk_text_view_get_buffer(GTK_TEXT_VIEW(text_view));
178     gtk_text_buffer_get_start_iter(buffer,&start);
179     gtk_text_buffer_get_end_iter(buffer,&end);
180     gtk_text_buffer_delete(buffer,&start,&end);
181     gtk_text_buffer_insert(buffer, &end, text, bytes_read);
182     gtk_widget_destroy(open_file_dlg);
183 }
184 void open_file_dlg_cancel_clicked(GtkWidget *widget, gpointer data)
185 {
186     gtk_widget_destroy(open_file_dlg);
187 }
188 void saveas_file_dlg_ok_clicked(GtkWidget *widget, gpointer data)
189 {
190     FILE *fp;
191     GtkTextBuffer *buffer;
192     GtkTextIter start, end;
193     gchar *text;
194     filename=gtk_file_selection_get_filename(GTK_FILE_SELECTION(↩
            saveas_file_dlg));
195     buffer=gtk_text_view_get_buffer(GTK_TEXT_VIEW(text_view));
196     gtk_text_buffer_get_start_iter(buffer, &start);
197     gtk_text_buffer_get_end_iter(buffer, &end);
198     text=gtk_text_buffer_get_text(buffer, &start, &end, TRUE);
199     fp=fopen(filename, "w");
200     if(!fp)
201     {
202         return;
203     }
204     fwrite(text, strlen(text), 1, fp);
205     fclose(fp);
206     gtk_widget_destroy(saveas_file_dlg);
207 }
208 void saveas_file_dlg_cancel_clicked(GtkWidget *widget, gpointer data)
209 {
210     gtk_widget_destroy(saveas_file_dlg);
```

```
211 }
212 void font_dlg_ok_clicked(GtkWidget *widget, gpointer data)
213 {
214     gchar *fontname;
215     PangoFontDescription *font_desc;
216     fontname=gtk_font_selection_dialog_get_font_name(↩
            GTK_FONT_SELECTION_DIALOG(font_dlg));
217     font_desc=pango_font_description_from_string(fontname);
218     gtk_widget_modify_font(text_view, font_desc);
219     gtk_widget_destroy(font_dlg);
220 }
221 void font_dlg_cancel_clicked(GtkWidget *widget, gpointer data)
222 {
223     gtk_widget_destroy(font_dlg);
224 }
225 void font_dlg_apply_clicked(GtkWidget *widget, gpointer data)
226 {
227     gchar *fontname;
228     PangoFontDescription *font_desc;
229     fontname=gtk_font_selection_dialog_get_font_name(↩
            GTK_FONT_SELECTION_DIALOG(font_dlg));
230     font_desc=pango_font_description_from_string(fontname);
231     gtk_widget_modify_font(text_view, font_desc);
232 }
233
234 /* The Text Edit Functions */
235
236 void text_edit_new()
237 {
238     text_edit_close();
239 }
240 void text_edit_open()
241 {
242     /* First Close The Openned File */
243     text_edit_close();
244     /* Create The Open File Dialog */
245     open_file_dlg=gtk_file_selection_new("Open File...");
246     open_file_dlg_ok=GTK_FILE_SELECTION(open_file_dlg)->ok_button;
247     open_file_dlg_cancel=GTK_FILE_SELECTION(open_file_dlg)->cancel_button;
248     /* Open File Dialog Clicked Signals */
249     g_signal_connect(G_OBJECT(open_file_dlg_ok), "clicked", G_CALLBACK(↩
            open_file_dlg_ok_clicked), NULL);
250     g_signal_connect(G_OBJECT(open_file_dlg_cancel), "clicked", G_CALLBACK↩
            (open_file_dlg_cancel_clicked), NULL);
251     gtk_widget_show_all(open_file_dlg);
252 }
253 void text_edit_save()
254 {
255     if(filename==NULL)
256     {
257         text_edit_saveas();
258     }
259     else
260     {
261         FILE *fp;
262         gchar *text;
263         GtkTextBuffer *buffer;
264         GtkTextIter start, end;
265         buffer=gtk_text_view_get_buffer(GTK_TEXT_VIEW(text_view));
266         gtk_text_buffer_get_start_iter(buffer, &start);
267         gtk_text_buffer_get_end_iter(buffer, &end);
268         text=gtk_text_buffer_get_text(buffer, &start, &end, TRUE);
```

```
269         fp=fopen(filename, "w");
270         if(!fp)
271         {
272             return;
273         }
274         fwrite(text, strlen(text), 1, fp);
275         fclose(fp);
276     }
277 }
278 void text_edit_saveas()
279 {
280     /* Create The Save As File Dialog */
281     saveas_file_dlg=gtk_file_selection_new("Save File As...");
282     saveas_file_dlg_ok=GTK_FILE_SELECTION(saveas_file_dlg)->ok_button;
283     saveas_file_dlg_cancel=GTK_FILE_SELECTION(saveas_file_dlg)->↩
        cancel_button;
284     /* Save As File Dialog Clicked Signals */
285     g_signal_connect(G_OBJECT(saveas_file_dlg_ok), "clicked", G_CALLBACK(↩
        saveas_file_dlg_ok_clicked), NULL);
286     g_signal_connect(G_OBJECT(saveas_file_dlg_cancel), "clicked", ↩
        G_CALLBACK(saveas_file_dlg_cancel_clicked), NULL);
287     gtk_widget_show_all(saveas_file_dlg);
288 }
289 void text_edit_close()
290 {
291     filename=NULL;
292     GtkTextBuffer *buffer;
293     GtkTextIter start, end;
294     buffer=gtk_text_view_get_buffer(GTK_TEXT_VIEW(text_view));
295     gtk_text_buffer_get_start_iter(buffer,&start);
296     gtk_text_buffer_get_end_iter(buffer,&end);
297     gtk_text_buffer_delete(buffer,&start,&end);
298 }
299 void text_edit_quit()
300 {
301     exit(0);
302 }
303 void text_edit_preferences()
304 {
305     /* Create The Font Dialog */
306     font_dlg=gtk_font_selection_dialog_new("Preferences...");
307     font_dlg_ok=GTK_FONT_SELECTION_DIALOG(font_dlg)->ok_button;
308     font_dlg_cancel=GTK_FONT_SELECTION_DIALOG(font_dlg)->cancel_button;
309     font_dlg_apply=GTK_FONT_SELECTION_DIALOG(font_dlg)->apply_button;
310     /* Font Dialog Clicked Signals */
311     g_signal_connect(G_OBJECT(font_dlg_ok), "clicked", G_CALLBACK(↩
        font_dlg_ok_clicked), NULL);
312     g_signal_connect(G_OBJECT(font_dlg_cancel), "clicked", G_CALLBACK(↩
        font_dlg_cancel_clicked), NULL);
313     g_signal_connect(G_OBJECT(font_dlg_apply), "clicked", G_CALLBACK(↩
        font_dlg_apply_clicked), NULL);
314     gtk_widget_show_all(font_dlg);
315 }
```

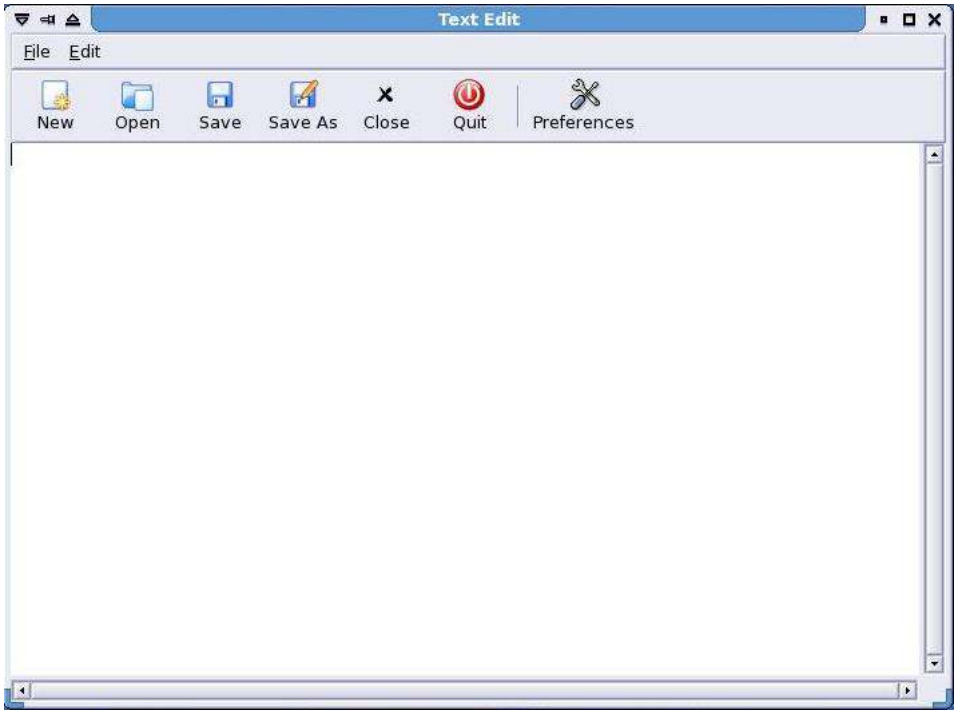


Figure 16.1: Text Editor