Moin Ahmed

Roll No. 190510

Project Supervisor : Dr. Debadatta Mishra

Documentation

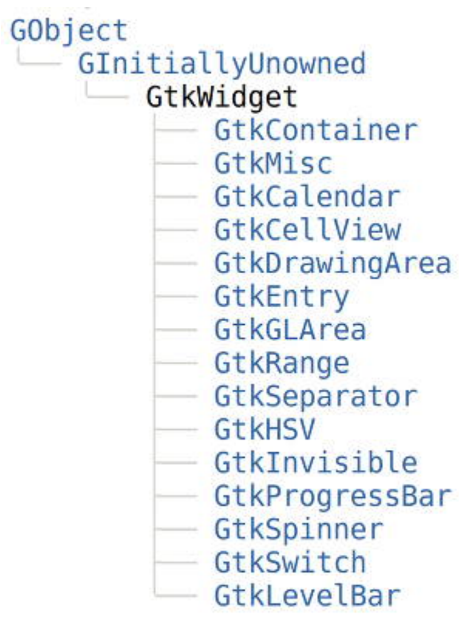
Gtk Library

Gtk+ is event-driven. The toolkit listens for events such as a click on a button, and passes the event to the application.

It is a widget toolkit. Each user interface created by Gtk+ consists of widgets. This is implemented in C using GObject, an object-oriented framework for C. Widgets are organised in a hierarchy. The window widget is the main container. The user interface is then built by adding buttons, drop-down menus, and other widgets to the window. GnomeDocumentation

Object hierarchy with GObject (Link)

Every GTK object is derived from Object. In the following image it can be seen that GObject is the parent object. GInitiallyUnowned is the child of GObject and inherits all of its properties. GtkWidget is then derived from GInitiallyUnowned and inherits from its parent object.



```
GObject
╰── GInitiallyUnowned
    ╰── GtkWidget
        ├── GtkContainer
        ├── GtkMisc
        ├── GtkCalendar
        ├── GtkCellView
        ├── GtkDrawingArea
        ├── GtkEntry
        ├── GtkGLArea
        ├── GtkRange
        ├── GtkSeparator
        ├── GtkHSV
        ├── GtkInvisible
        ├── GtkProgressBar
        ├── GtkSpinner
        ├── GtkSwitch
        ╰── GtkLevelBar
```

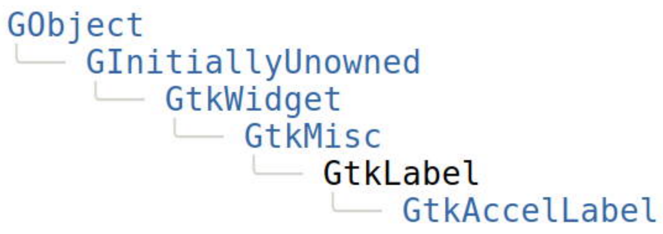
The image also shows all of the objects that are derived from or are children of GtkWidget, just like classical object oriented inheritance.

When writing GTK applications we are mostly concerned about the child widgets that are derived from GtkWidget.

GtkWidget - A control in a graphical user interface. Widgets can draw themselves and process events from the mouse and keyboard. Widget types include buttons, menus, text entry lines, and lists. Widgets can be arranged into *containers*, and these take care of assigning the *geometry* of the widgets: every widget thus has a parent except those widgets which are *toplevels*. The base class for widgets is GtkWidget.

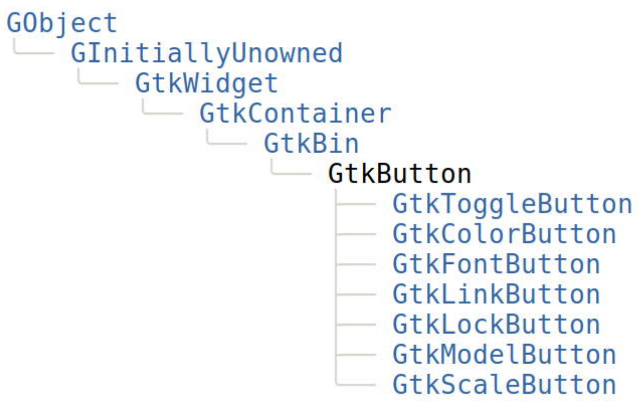
Two examples:

(1) : GtkLabel is derived from GtkMisc which is derived from GtkWidget.



GtkLabel therefore inherits all properties further up the chain. GtkAccelLabel, GtkLabel are children of GtkMisc which in turn is a derived from Gtk-Widget

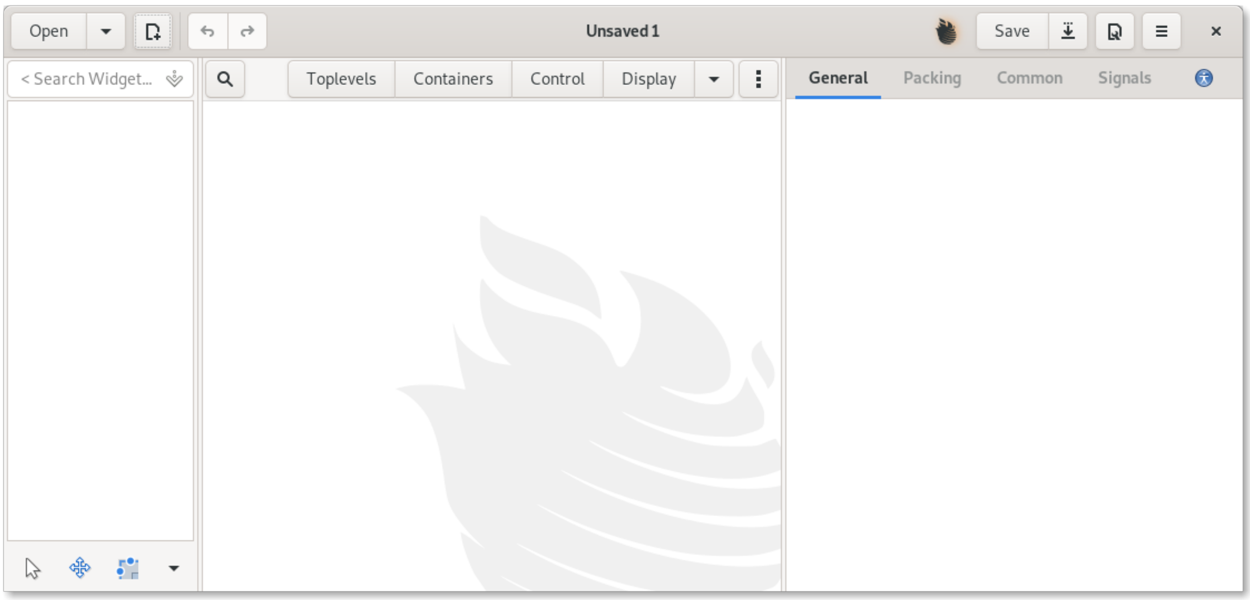
(2) : GtkButton is derived from GtkBin which in turn is derived from Gtk Container, which is a children of GtkWidget. There are several other widgets such as GtkToggleButton, GtkColorButton etc that are derived from GtkButton. These widget settings are inherited from the parent.



GLADE

It is a tool for markup description language. The user interfaces designed in Glade are saved as XML, and by using GtkBuilder GTK object these can be loaded by applications dynamically as needed. The GTK+ library has various building blocks such as text boxes, dialog labels, numeric entries, check boxes, and menus these are known as widgets and glade is use to place the widgets in a GUI

Code sketchers are software applications that help a user create source code from a GladeXML file. Most code sketchers create source code which uses libglade and a GladeXML file to create the GUI.

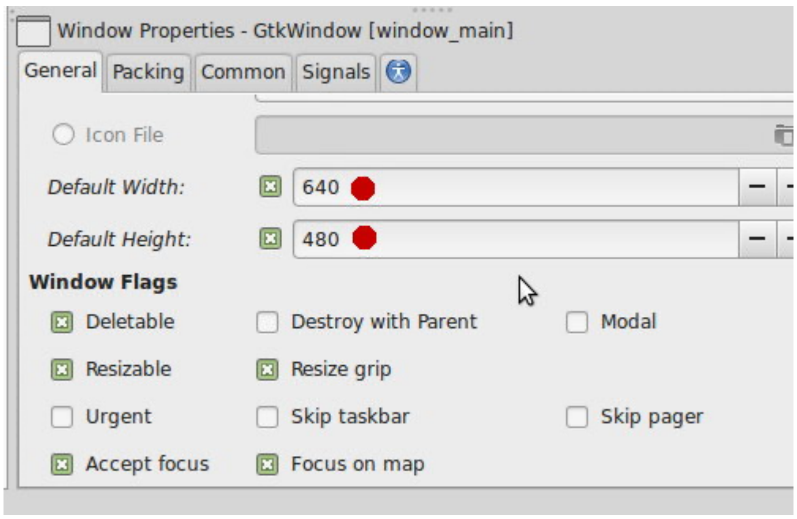


Menubar - The menus on the menubar contain all of the commands you need to work with files in Glade.

Toolbar - The toolbar contains a subset of the commands that you can access from the menubar.

Design Area - The design are is where a user interface can be visually edited.

Window - When the window that is currently being designed in Glade is displayed by the C program, it will emit a **destroy** signal when the window is closed. A callback function needs to be connected to the destroy signal that will run when the window is closed.



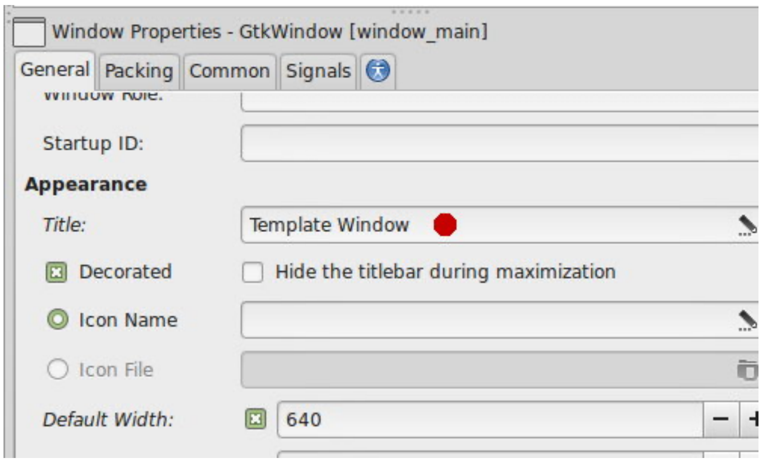
○ Icon File

Default Width: ☒ 640

Default Height: ☒ 480

Window Flags

☒ Deletable	☐ Destroy with Parent	☐ Modal
☒ Resizable	☒ Resize grip	
☐ Urgent	☐ Skip taskbar	☐ Skip pager
☒ Accept focus	☒ Focus on map	



Startup ID:

Appearance

Title: Template Window

☒ Decorated ☐ Hide the titlebar during maximization

◉ Icon Name

○ Icon File

Default Width: ☒ 640

Palette - Palette contains the widgets that can be used to build a user interface.

Inspector - The inspector displays information about the widgets in a project.

Property Editor - The property editor is used to manipulate the properties of widgets, as well as adding to source code.

Status Bar - The status bar displays information about current Glade activity and contextual information about the menu items.

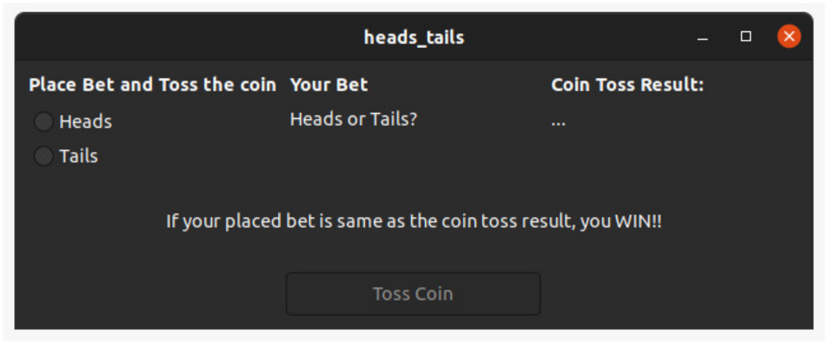
Creating a simple game using GTK/Glade

Following is the implementation of a simple game of heads and tails. I was getting fa miliarised with using GTK library of C language for creating the source code and Glade for GUI.

The logic of the game is trivial, using radio buttons and GtkLabels I implemented a sim ple heads and tails game.

The code makes use of the gbooleanrandom function to generate a random boolean output which is then used for the game logic.

- The user inputs using the two GtkRadioButtons, by reference of GtkLabels for GUI.


heads_tails
Place Bet and Toss the coin
Your Bet
Coin Toss Result:
Heads
Tails
Heads or Tails?
...
If your placed bet is same as the coin toss result, you WIN!!
Toss Coin

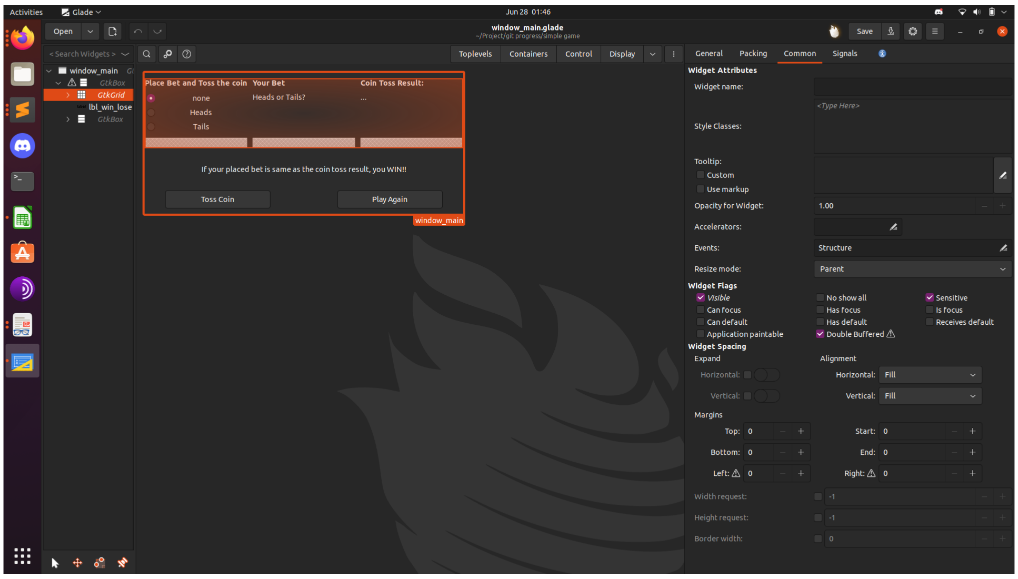
•If the input value by the user matches the gbooleanrandom value, the user wins and the label changes, else, the label displays lose message.

•The labels are as follows:
•Place bet and Toss the coin
•Your Bet
•Coin Toss Result.

- The Radio buttons are as follows:
  - rb_none : as an auxiliary button for the setting up the alternating button operation.
  - rb_heads : with label Heads
  - rb_tails : with label Tails

- The Glade file contains the XML code for the application.

<u>The following is the Glade design of the applica- tion.</u>

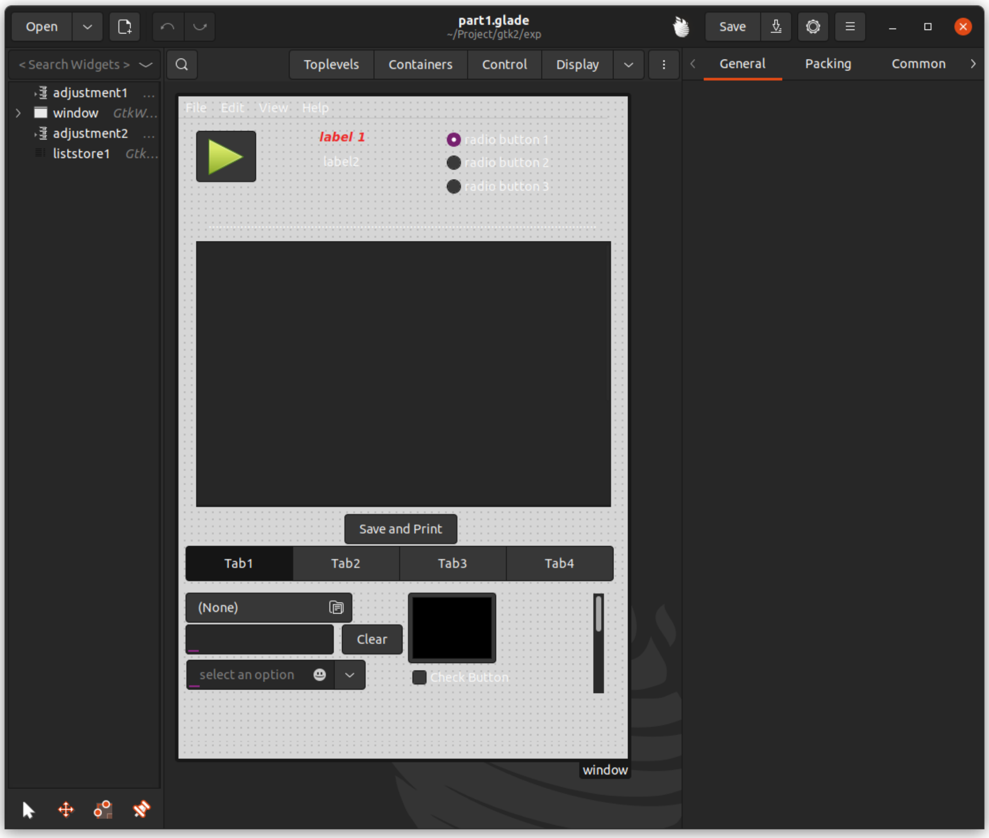


Getting Familiar with Glade

I followed the tutorials by Professor Kevin C. O’kane of University of Northern Iowa.

I followed tutorials 1 to 35 from this *Link.*
And, created the following GUI of basic applications using Gtk and Glade.

It has the basic implementation of various functions.
Namely :



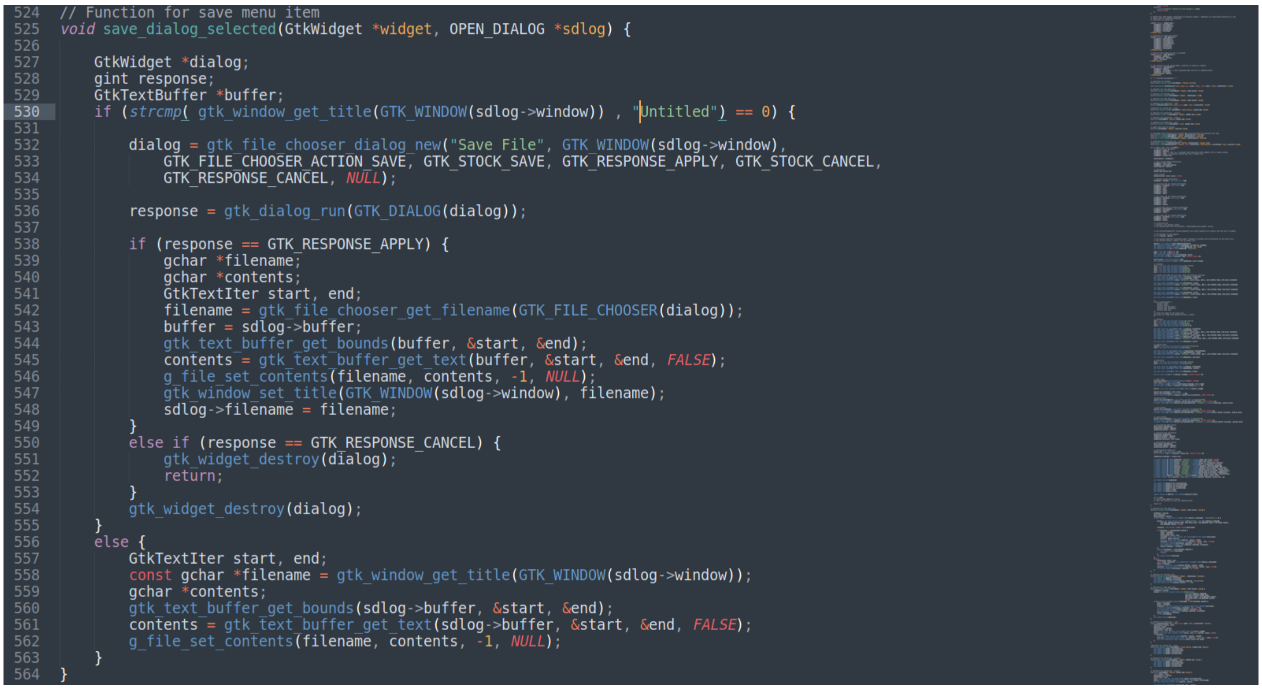
- **GtkButton**
- GtkRadioButton
- GtkCheckButton
- GtkComboBox
- GtkFileChooser
- GtkFontButton
- GtkEntry
- **GtkMenuBar, MenuItem**
- GtkProgressBar
- **TextEditingWindow**
- **DynamicScrollableGrid**

The source code : link

Text Editor

Next, I implemented the save file, open file, find search operation using gtk_tex-t_iter_get, replace and find and replace.

Save Function : After opening and adding placeholder texts, when the user selects **save** button from the drop down menu, the following sub routine is instantiated.



```c
// Function for save menu item
void save_dialog_selected(GtkWidget *widget, OPEN_DIALOG *sdlog) {

    GtkWidget *dialog;
    gint response;
    GtkTextBuffer *buffer;
    if (strcmp( gtk_window_get_title(GTK_WINDOW(sdlog->window)) , "Untitled") == 0) {

        dialog = gtk_file_chooser_dialog_new("Save File", GTK_WINDOW(sdlog->window),
            GTK_FILE_CHOOSER_ACTION_SAVE, GTK_STOCK_SAVE, GTK_RESPONSE_APPLY, GTK_STOCK_CANCEL,
            GTK_RESPONSE_CANCEL, NULL);

        response = gtk_dialog_run(GTK_DIALOG(dialog));

        if (response == GTK_RESPONSE_APPLY) {
            gchar *filename;
            gchar *contents;
            GtkTextIter start, end;
            filename = gtk_file_chooser_get_filename(GTK_FILE_CHOOSER(dialog));
            buffer = sdlog->buffer;
            gtk_text_buffer_get_bounds(buffer, &start, &end);
            contents = gtk_text_buffer_get_text(buffer, &start, &end, FALSE);
            g_file_set_contents(filename, contents, -1, NULL);
            gtk_window_set_title(GTK_WINDOW(sdlog->window), filename);
            sdlog->filename = filename;
        }
        else if (response == GTK_RESPONSE_CANCEL) {
            gtk_widget_destroy(dialog);
            return;
        }
        gtk_widget_destroy(dialog);
    }
    else {
        GtkTextIter start, end;
        const gchar *filename = gtk_window_get_title(GTK_WINDOW(sdlog->window));
        gchar *contents;
        gtk_text_buffer_get_bounds(sdlog->buffer, &start, &end);
        contents = gtk_text_buffer_get_text(sdlog->buffer, &start, &end, FALSE);
        g_file_set_contents(filename, contents, -1, NULL);
    }
}
```

The function ***gtk_window_get_title****()* returns the const *gchar and retrieves the title of the window.
We compare the string returned by the function with “Untitled” which is the default name of unsaved files. And, if it returns 0, that means we can set the title of the file as “Untitled”.

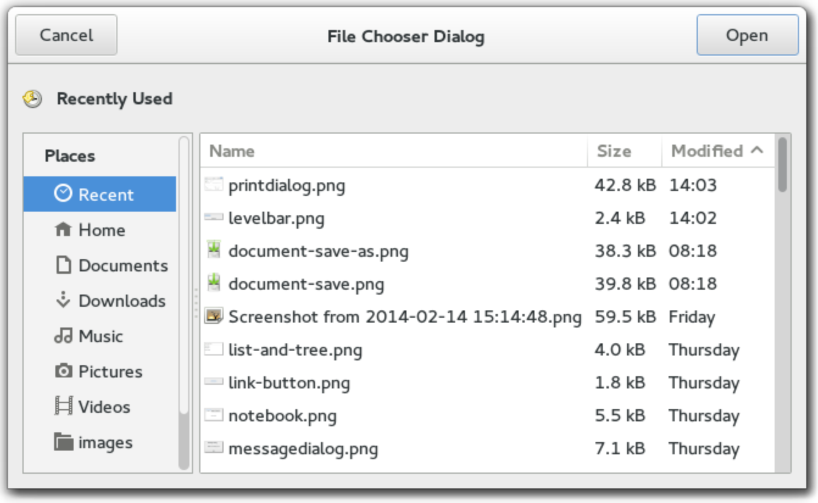
gtk_file_choose_dialog_new() - This function open a GtkFileChooserDialog box which is a dialog box suitable for use with “File/Open” or “File/Save as” commands.

Object Hierarchy

```
GObject
 └── GInitiallyUnowned
      └── GtkWidget
           └── GtkContainer
                └── GtkBin
                     └── GtkWindow
                          └── GtkDialog
                               └── GtkFileChooserDialog
```

This widget works by putting a GtkFile-ChooserWidget inside a GtkDialogBox.

It exposes the GtkFileChooser interface, so you can use all of the GtkFileChooser func-tions on the file chooser dialog as well as those for GtkDialog.



Name	Size	Modified
printdialog.png	42.8 kB	14:03
levelbar.png	2.4 kB	14:02
document-save-as.png	38.3 kB	08:18
document-save.png	39.8 kB	08:18
Screenshot from 2014-02-14 15:14:48.png	59.5 kB	Friday
list-and-tree.png	4.0 kB	Thursday
link-button.png	1.8 kB	Thursday
notebook.png	5.5 kB	Thursday
messagedialog.png	7.1 kB	Thursday

The function instantiates the File Chooser Dialog.

This is a typical implementation of the above function.

• If the returned response is GTK_RESPONSE_APPLY,

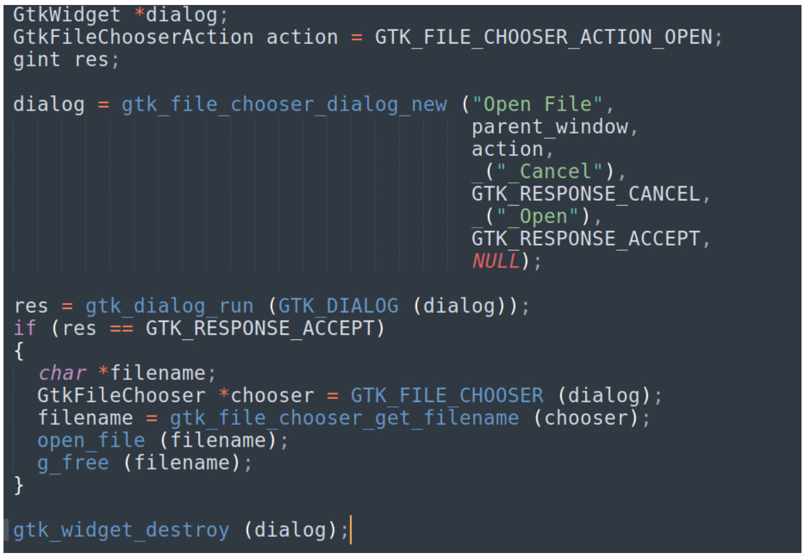
• We proceed and set the filename as entered in the GTK_FILE_-CHOOSER_DIALOG.

• Next, buffer stores the contents in sdlog buffer structure.

• If the response is GTK_RESPON-SE_CANCEL the function gtk_wid-get_destroy is called which collaps-es the window.

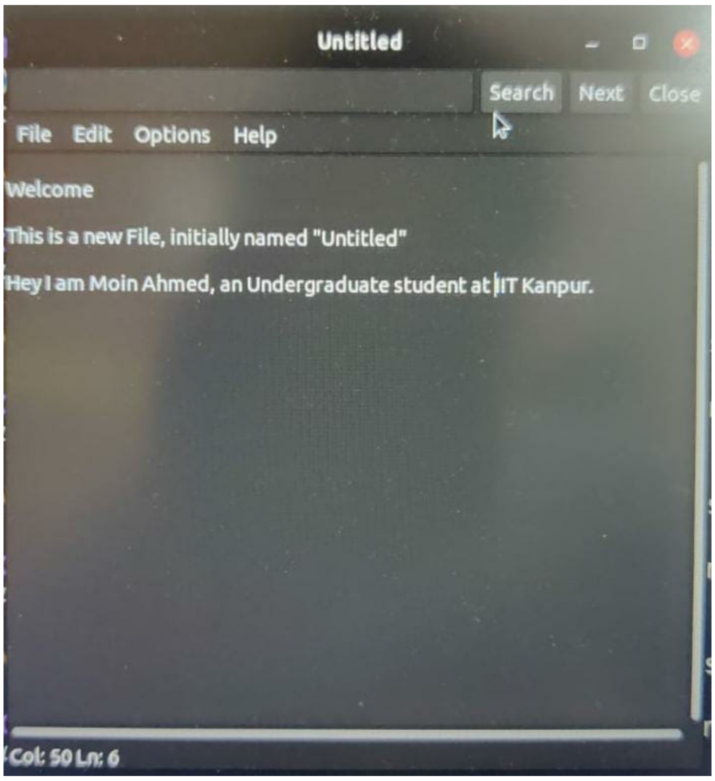
• Also, if window name “Untitled” is not possible for some reason, we obtain the title of the window and set the same.

• Next the function **g_file_set_con-tents()** is called which writes all of the contents to a file named “file-name”. It is a boolean return type function.



```
GtkWidget *dialog;
GtkFileChooserAction action = GTK_FILE_CHOOSER_ACTION_OPEN;
gint res;

dialog = gtk_file_chooser_dialog_new ("Open File",
                                      parent_window,
                                      action,
                                      _("_Cancel"),
                                      GTK_RESPONSE_CANCEL,
                                      _("_Open"),
                                      GTK_RESPONSE_ACCEPT,
                                      NULL);

res = gtk_dialog_run (GTK_DIALOG (dialog));
if (res == GTK_RESPONSE_ACCEPT)
{
  char *filename;
  GtkFileChooser *chooser = GTK_FILE_CHOOSER (dialog);
  filename = gtk_file_chooser_get_filename (chooser);
  open_file (filename);
  g_free (filename);
}

gtk_widget_destroy (dialog);
```

FIND : This function can be called using the “find” operation from the menubar. After typing in the query in the search bar. The function highlights the occurrences of the query in the text_view.



Welcome

This is a new File, initially named "Untitled"

Hey I am Moin Ahmed, an Undergraduate student at IIT Kanpur.

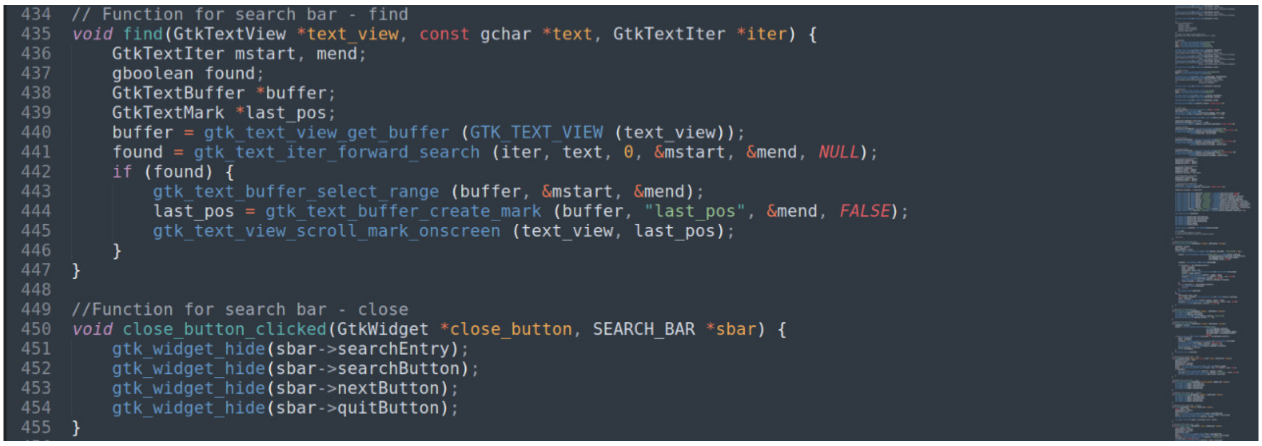
***gtk_text_view_get_buffer()* -** Returns the GtkTextBuffer being displayed by this text view. The reference count on the buffer is not incremented; the caller of this function won’t own a new reference. The return value is stored in the *GtkText-Buffer* buffer*.

gtk_text_iter_forward_search() - Searches forward for str. Any match is returned by setting match_start to the first character to the first character of the match and match_end to the first character after the match. The search will not continue past *limit. It is a linear O(N) operation.*
<u>*Caution*</u> - Set a limit to avoid locking up the UI for large inputs.

***gtk_text_buffer_select_range()* -** This function moves the “insert” and “selection_bound” marks simultaneously. We select a region temporarily between their old and new locations. This function moves the region as a unit.

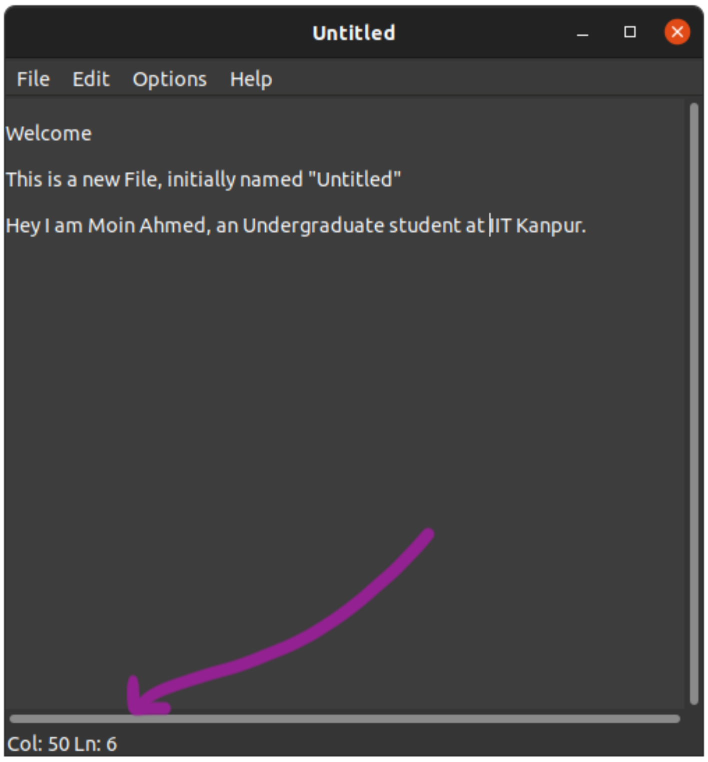
gtk_text_buffer_create_mark() - Creates a mark at position, say where. If mark_name is *NULL, the mark is anonymous, otherwise,* the mark can be retrieved by name using *gtk_text_buffer_get_mark()*. <u>We can ignore the return value if not needed.</u>

***gtk_text_view_scroll_mark_onscreen()* -** Scrolls the *text_view* the minimum distance such that mark is contained within the visible area of the widget.



```c
// Function for search bar - find
void find(GtkTextView *text_view, const gchar *text, GtkTextIter *iter) {
    GtkTextIter mstart, mend;
    gboolean found;
    GtkTextBuffer *buffer;
    GtkTextMark *last_pos;
    buffer = gtk_text_view_get_buffer (GTK_TEXT_VIEW (text_view));
    found = gtk_text_iter_forward_search (iter, text, 0, &mstart, &mend, NULL);
    if (found) {
        gtk_text_buffer_select_range (buffer, &mstart, &mend);
        last_pos = gtk_text_buffer_create_mark (buffer, "last_pos", &mend, FALSE);
        gtk_text_view_scroll_mark_onscreen (text_view, last_pos);
    }
}

//Function for search bar - close
void close_button_clicked(GtkWidget *close_button, SEARCH_BAR *sbar) {
    gtk_widget_hide(sbar->searchEntry);
    gtk_widget_hide(sbar->searchButton);
    gtk_widget_hide(sbar->nextButton);
    gtk_widget_hide(sbar->quitButton);
}
```

Status Bar : The subroutine is followed on every instance when text_view/ cursor is changed is updated. It updates the status of the pointer, For example, in the picture the pointer is at column number 50 and row 6.



Welcome

This is a new File, initially named "Untitled"

Hey I am Moin Ahmed, an Undergraduate student at IIT Kanpur.

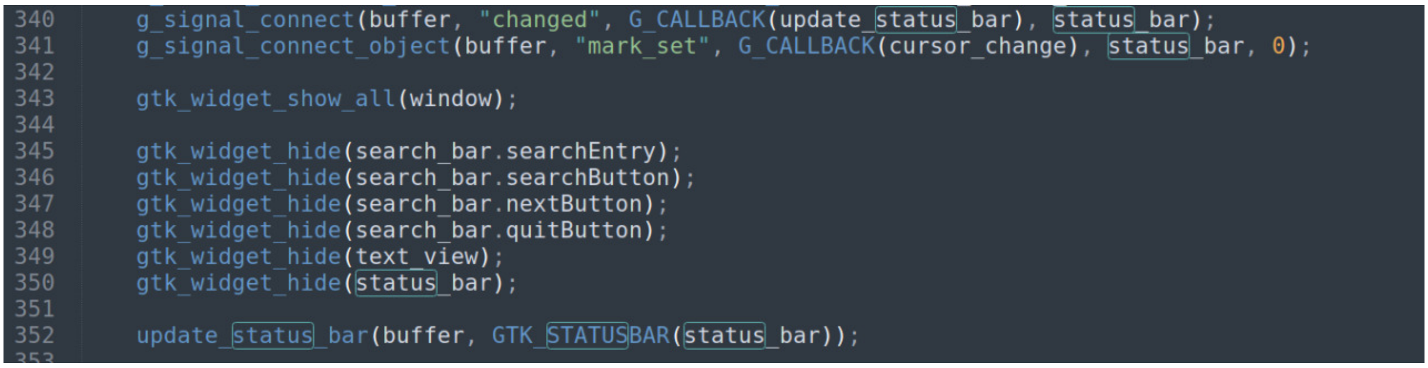
Col: 50 Ln: 6

gtk_statusbar_pop() - Removes the first message in the GtkStatusBar's stack with the given context id.

gtk_text_buffer_get_iter_at_mark() - Initializes tier with the current position of mark.

gtk_text_iter_get_line() - Returns the line number containing the iterator. Lines in GtkTextBuffer are numbered beginning with 0 for the first line in the buffer.

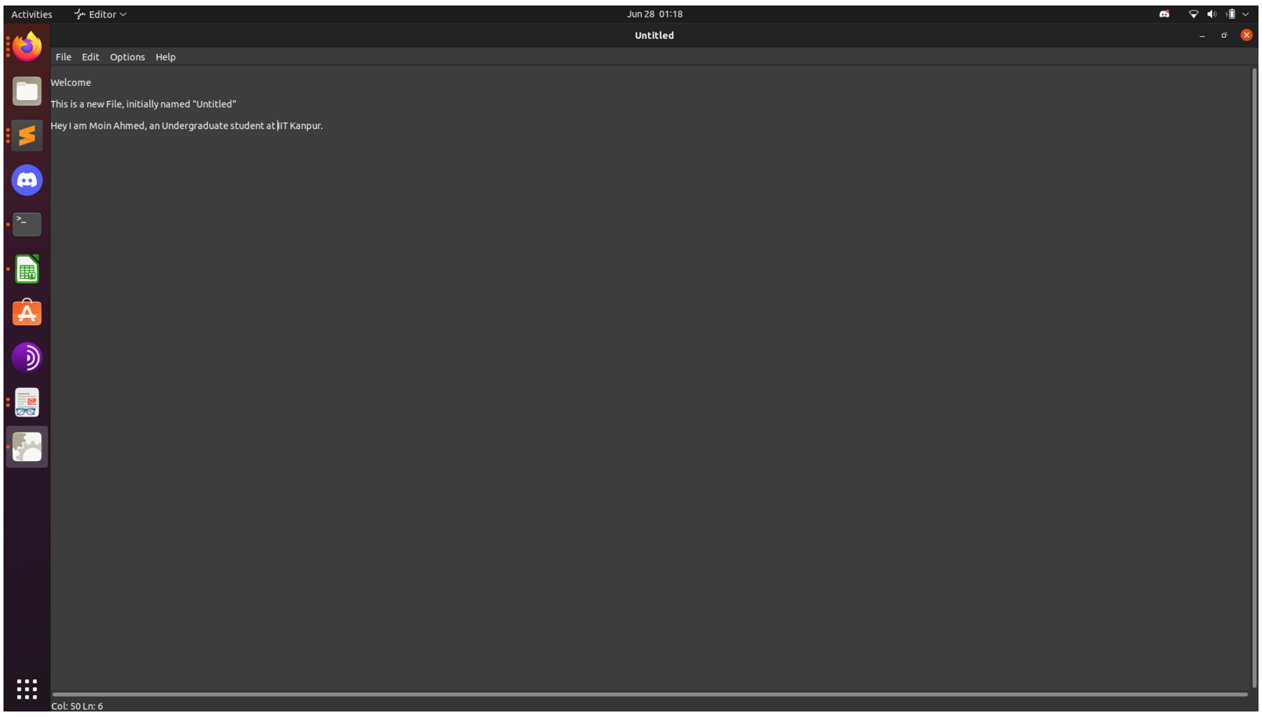
gtk_text_iter_get_line_offset() - Returns the character offset of the iterator, counting from the start of a newline-terminated line. The first character on the line has offset 0.



```
    g_signal_connect(buffer, "changed", G_CALLBACK(update_status_bar), status_bar);
    g_signal_connect_object(buffer, "mark_set", G_CALLBACK(cursor_change), status_bar, 0);

    gtk_widget_show_all(window);

    gtk_widget_hide(search_bar.searchEntry);
    gtk_widget_hide(search_bar.searchButton);
    gtk_widget_hide(search_bar.nextButton);
    gtk_widget_hide(search_bar.quitButton);
    gtk_widget_hide(text_view);
    gtk_widget_hide(status_bar);

    update_status_bar(buffer, GTK_STATUSBAR(status_bar));
```

The editor

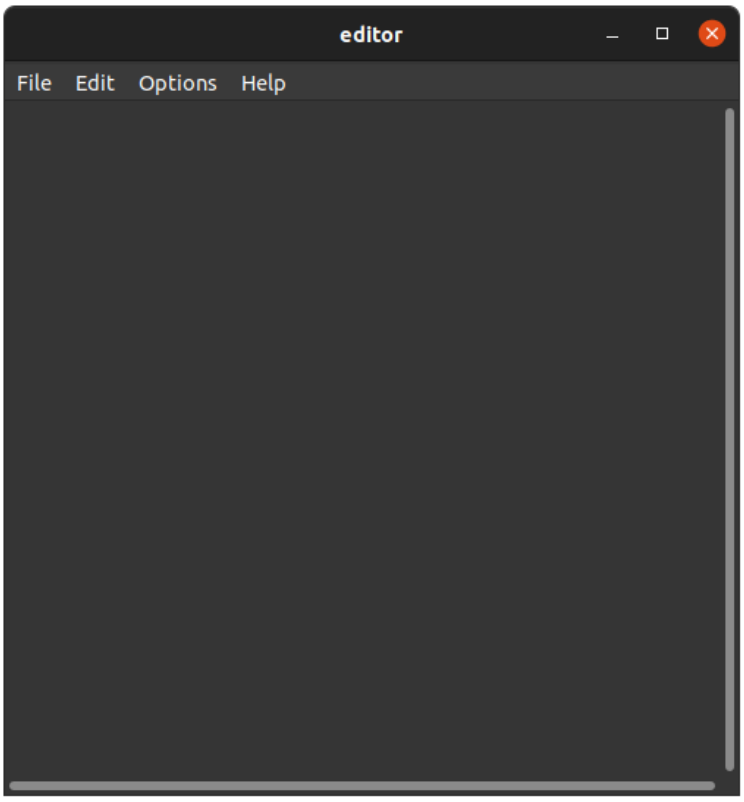
Following are some of the operations possible in the editor.



Welcome

This is a new File, initially named "Untitled"

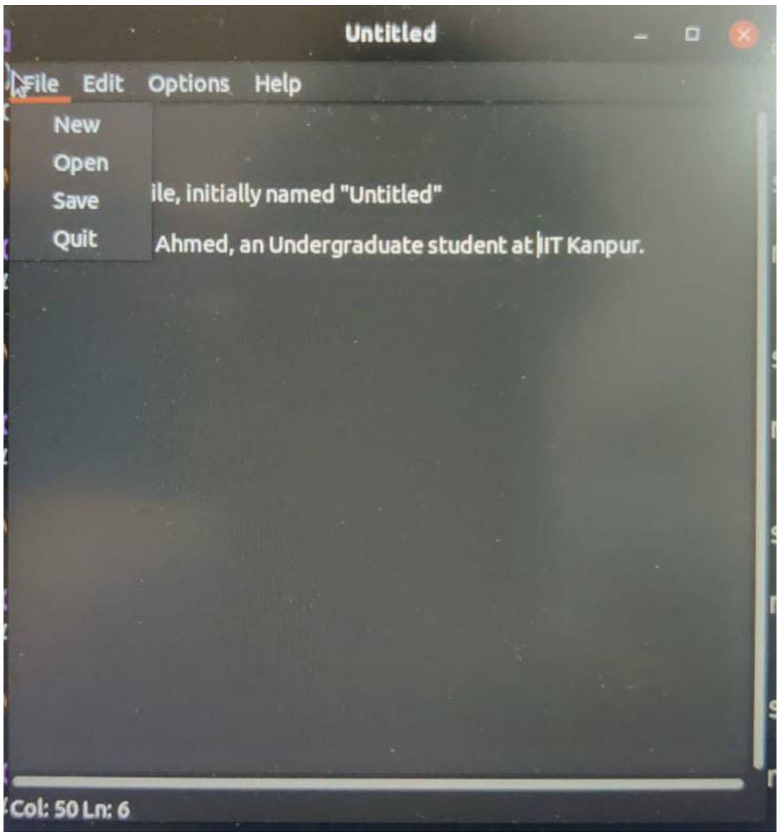
Hey I am Moin Ahmed, an Undergraduate student at IIT Kanpur.



This is the first instance of the editor, we need to roll over the menu for it to drop down.

The menu has four submenus which contain the desired operations, the four submenus are :

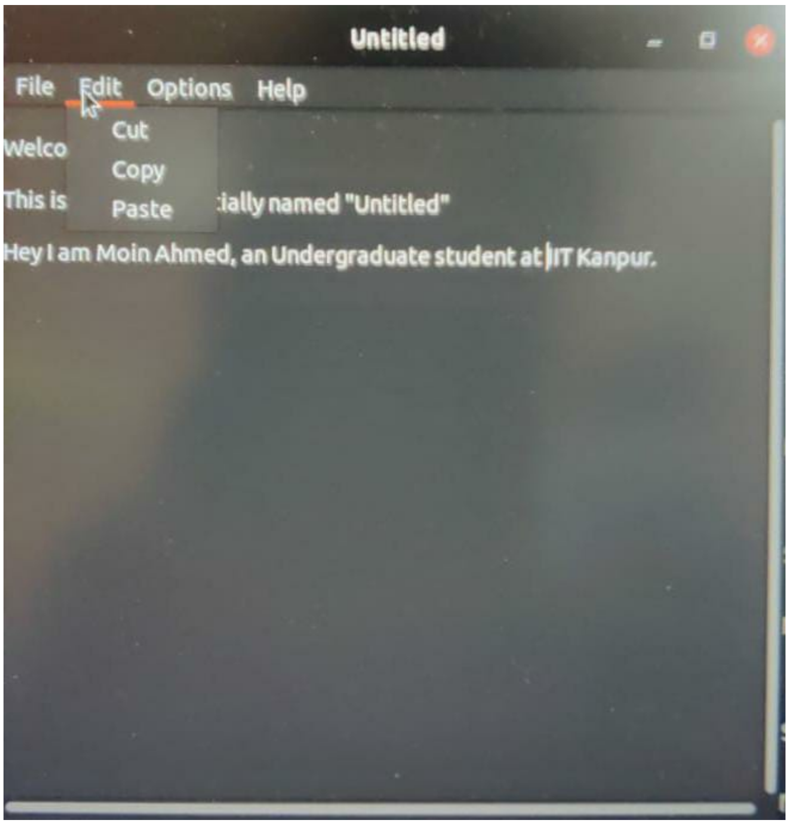
- File
- Edit
- Options
- Help



```
Untitled
File  Edit  Options  Help
New
Open
Save
Quit
ile, initially named "Untitled"
Ahmed, an Undergraduate student at IIT Kanpur.
Col: 50 Ln: 6
```

The menu has four drop down submenus which contain. The "File" contains the following operations.

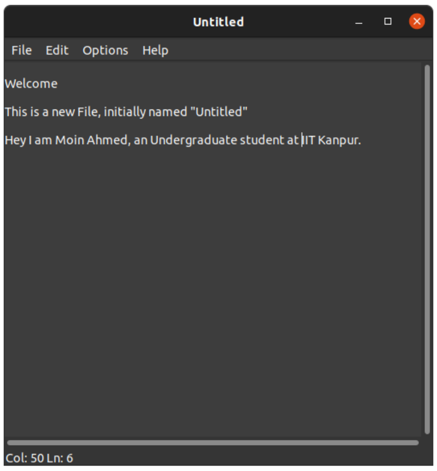
- New
- Open
- Save
- Quit



```
Untitled
File  Edit  Options  Help
Welco   Cut
        Copy
This is Paste  ially named "Untitled"
Hey I am Moin Ahmed, an Undergraduate student at IIT Kanpur.
```

The editor also has the functions of making basic clipboard functions of :

- Cut
- Copy
- Paste



```
Untitled
File  Edit  Options  Help
Welcome
This is a new File, initially named "Untitled"
Hey I am Moin Ahmed, an Undergraduate student at IIT Kanpur.
Col: 50 Ln: 6
```