Gtk4 tutorial for beginners

Toshio Sekiya

abstract

Contenido de este repositorio Este tutorial ilustra como escribir programas en C con la biblioteca Gtk4 se enfoca en prinpcipiantes, asi que el contenido esta limitado a los temas básicos. La tabla de contenido se encuentra al final de esta introducción.

- Sección 3 a la 21 describe las bases, con el ejemplo de un editor simple `tfe` (Text File Editor).
- Sección 22 a la 25 describe como usar GtkDrawingArea.
- Sección 26 a la 29 describe el modelo de lista y la vista lista (GtkListView, GtkGridView y GtkColumnView). también describe GtkExpression.

La última versión original de este tutorial (en inglés) se encuentra en Gtk4-tutorial github repository. Puedes leerlo desde ahí directamente sin tener que descargar nada.

Documentación Gtk4 Lee Gtk API Reference y and Gnome Developer Documentation Website para mayor información.

Estos sitios son recientes (Agosto 2021) La vieja documentación se encuentra en Gtk Reference Manual y Gnome Developer Center. El nuevo sitio se encuentra en progreso actualmente, asi que a veces tendrás que revisar la vieja version

Si deseas conocer mas acerca de GObject y el sistema de tipos, por favor lee GObject tutorial. Los detalles de GObject son fáciles de entender y necesarios para escribir programas en Gtk4.

Contribución Este tutorial se encuentra bajo desarrollo y es inestable. Incluso aunque los ejemplos han sido probados en Gtk4 versión 4.0, pueden existir algunos errores. Si encuentras algún bug, o errores en el tutorial y los ejemplos de C, por favor haznoslo notar. Lo puedes publicar en (sitio original en inglés) github issues. También puedes publicar los archivos corregidos como un commit a pull request. Cuando hagas correcciones, corrige el archivo fuente, que se encuentra en la carpeta 'src', y ejecuta `rake`para crear el archivo de salida. Los archivos GFM dentro de la carpeta 'gfm' se actualizan de manera automática.

Si tienes alguna duda, puedes publicarlo como un issue dentro del sitio. Todas las preguntas son útiles y harán este tutorial mejor.

Como obtener una versión HTML o PDF Si deseas obtener una versión HTML o PDF, la puedes hacer com `rake`, que es una versión en ruby de make. Escribe `rake html` para HTML. Escribe `rake pdf` para PDF. El apéndice "Cómo construir el Tutorial Gtk4" describe como hacerlo.

Contents

1 Prerequisite and License

1.1 Prerequisite

1.1.1 Gtk4 on a Linux OS

This tutorial is about Gtk4 libraries. It is originally used on Linux with C compiler, but now it is used more widely, on Windows and MacOS, with Vala, Python and so on. However, this tutorial describes only *C programs on Linux*.

If you want to try the examples in the tutorial, you need:

- PC with Linux distribution like Ubuntu, Debian and so on.
- Gcc.
- Gtk4. The stable version of Gtk on Linux distributions is version three at present. You need to install Gtk4 to your computer. See Section 3 for the installation of Gtk4.

1.1.2 Ruby and rake for making the document

This repository includes Ruby programs. They are used to make Markdown files, HTML files, Latex files and a PDF file.

You need:

- Linux distribution like Ubuntu.
- Ruby programming language. There are two ways to install it. One is installing the distribution's package. The other is using rbenv and ruby-build. If you want to use the latest version of ruby, use rbenv.
- Rake. It is a gem, which is a library written in Ruby. You can install it as a package of your distribution or use gem command.

1.2 License

2 Installing Gtk4 into Linux distributions

This section describes how to install Gtk4 into Linux distributions.

This tutorial is without any warranty. If you want to install Gtk4 to your computer, do it at your own risk.

The information in this section is the one on April/27/2022. The words 'at present' and/or 'now' in this section means 'April/27/2022'.

There are three possible way to install Gtk4.

- Install it from the distribution packages.
- Build it from the source file.
- Install a Gnome 40 distribution with the gnome-boxes.

2.1 Installation from the distribution packages

The first way is easy to install. It is a recommended way. I've installed Gtk4 packages in Ubuntu 21.04. (Now, my Ubuntu version is 21.10).

```
$ sudo apt-get install libgtk-4-bin libgtk-4-common libgtk-4-dev libgtk-4-doc
```

Fedora, Arch, Debian and OpenSUSE are also possible. See Installing GTK from packages. The following table shows the distributions which support Gtk4.

Distribution	version	Gtk4	Gnome40
Fedora	36	4.4.2	Gnome42
Ubuntu	22.04lts	4.4	Gnome41(4.6.2)
Debian	bookworm(testing)	4.6.5	Gnome42
Arch	rolling release	4.6.5	Gnome42
Gentoo	rolling release	4.6.5	Gnome42
OpenSUSE	Tumbleweed(rolling release)	4.6.5	Gnome42

If you've installed Gtk4 from the packages, you don't need to read the rest of this section.

2.2 Installation from the source file

If your operating system doesn't have Gtk4 packages, you need to build it from the source. Or, if you want the latest version of Gtk4 (4.6.3), you also need to build it from the source.

I installed Gtk4 from the source in January 2021. So, the following information is old, especially for the version of each software. For the latest information, see Gtk API Reference, Building GTK.

2.2.1 Prerequisites for Gtk4 installation

- Linux operating system. For example, Ubuntu 20.10 or 20.04LTS. Other distributions might be OK.
- Packages for development such as gcc, meson, ninja, git, wget and so on.
- Dev package is necessary for each software below.

2.2.2 Installation target

I installed Gtk4 under the directory `$HOME/local`. This is a private user area.

If you want to install it in the system area, `/opt/gtk4` is one of good choices. Gtk API Reference, Building GTK gives an installation example to `/opt/gtk4`.

Don't install it to `/usr/local` which is the default. It is used by Ubuntu applications which are not build on Gtk4. Therefore, the risk is high and probably bad things will happen. Actually I did it and I needed to reinstall Ubuntu.

2.2.3 Installation to Ubuntu 20.10

Most of the necessary libraries are included by Ubuntu 20.10. Therefore, they can be installed with `apt-get` command. You don't need to install them from the source tarballs. You can skip the subsections below about prerequisite library installation (Glib, Pango, Gdk-pixbuf and Gtk-doc).

2.2.4 Glib installation

If your Ubuntu is 20.04LTS, you need to install prerequisite libraries from the tarballs. Check the version of your library and if it is lower than the necessary version, install it from the source.

For example,

```
$ pkg-config --modversion glib-2.0
2.64.6
```

The necessary version is 2.66.0 or higher. Therefore, the example above shows that you need to install Glib.

I installed 2.67.1 which was the latest version at that time (January 2021). Download Glib source files from the repository, then decompress and extract files.

```
$ wget https://download.gnome.org/sources/glib/2.67/glib-2.67.1.tar.xz
$ tar -Jxf glib-2.67.1.tar.xz
```

Some packages are required to build Glib. You can find them if you run meson.

```
$ meson --prefix $HOME/local _build
```

Use apt-get and install the prerequisites. For example,

```
$ sudo apt-get install -y  libpcre2-dev libffi-dev
```

After that, compile Glib.

```
$ rm -rf _build
$ meson --prefix $HOME/local _build
$ ninja -C _build
$ ninja -C _build install
```

Set several environment variables so that the Glib libraries installed can be used by build tools. Make a text file below and save it as `env.sh`

```
# compiler
CPPFLAGS="-I$HOME/local/include"
LDFLAGS="-L$HOME/local/lib"
PKG_CONFIG_PATH="$HOME/local/lib/pkgconfig:$HOME/local/lib/x86_64-linux-gnu/pkgconfig"
export CPPFLAGS LDFLAGS PKG_CONFIG_PATH
# linker
LD_LIBRARY_PATH="$HOME/local/lib/x86_64-linux-gnu/"
PATH="$HOME/local/bin:$PATH"
export LD_LIBRARY_PATH PATH
# gsetting
export GSETTINGS_SCHEMA_DIR=$HOME/local/share/glib-2.0/schemas
```

Then, use . (dot) or source command to include these commands to the current bash.

```
$ . env.sh
```

or

```
$ source env.sh
```

This command carries out the commands in `env.sh` and changes the environment variables above in the current shell.

2.2.5 Pango installation

Download and untar.

```
$ wget https://download.gnome.org/sources/pango/1.48/pango-1.48.0.tar.xz
$ tar -Jxf pango-1.48.0.tar.xz
```

Try meson and check the required packages. Install all the prerequisites. Then, compile and install Pango.

```
$ meson --prefix $HOME/local _build
$ ninja -C _build
$ ninja -C _build install
```

It installs Pango-1.0.gir under `$HOME/local/share/gir-1.0`. If you installed Pango without `--prefix` option, then it would be located at `/usr/local/share/gir-1.0`. This directory (/usr/local/share) is used by applications. They find the directory by the environment variable `XDG_DATA_DIRS`. It is a text file which keep the list of 'share' directories like `/usr/share`, `usr/local/share` and so on. Now `$HOME/local/share` needs to be added to `XDG_DATA_DIRS`, or error will occur in the later compilation.

```
$ export XDG_DATA_DIRS=$HOME/local/share:$XDG_DATA_DIRS
```

2.2.6 Gdk-pixbuf and Gtk-doc installation

Download and untar.

```
$ wget https://download.gnome.org/sources/gdk-pixbuf/2.42/gdk-pixbuf-2.42.2.tar.xz
$ tar -Jxf gdk-pixbuf-2.42.2.tar.xz
$ wget https://download.gnome.org/sources/gtk-doc/1.33/gtk-doc-1.33.1.tar.xz
$ tar -Jxf gtk-doc-1.33.1.tar.xz
```

Same as before, install prerequisite packages, then compile and install them.

The installation of Gtk-doc put `gtk-doc.pc` under `$HOME/local/share/pkgconfig`. This file is used by pkg-config, which is one of the build tools. The directory needs to be added to the environment variable `PKG_CONFIG_PATH`

```
$ export PKG_CONFIG_PATH="$HOME/local/share/pkgconfig:$PKG_CONFIG_PATH"
```

2.2.7 Gtk4 installation

If you want the latest development version of Gtk4, use git and clone the repository.

```
$ git clone https://gitlab.gnome.org/GNOME/gtk.git
```

If you want a stable version of Gtk4, then download it from Gnome source website. The latest version is 4.3.1 (13/June/2021).

Compile and install it.

```
$ meson --prefix $HOME/local _build
$ ninja -C _build
$ ninja -C _build install
```

If you want to know more information, refer to Gtk4 API Reference, Building GTK.

2.2.8 Modify env.sh

Because environment variables disappear when you log out, you need to add them again. Modify `env.sh`.

```
# compiler
CPPFLAGS="-I$HOME/local/include"
LDFLAGS="-L$HOME/local/lib"
PKG_CONFIG_PATH="$HOME/local/lib/pkgconfig:$HOME/local/lib/x86_64-linux-gnu/pkgconfig:
$HOME/local/share/pkgconfig"
export CPPFLAGS LDFLAGS PKG_CONFIG_PATH
# linker
LD_LIBRARY_PATH="$HOME/local/lib/x86_64-linux-gnu/"
PATH="$HOME/local/bin:$PATH"
export LD_LIBRARY_PATH PATH
# gir
XDG_DATA_DIRS=$HOME/local/share:$XDG_DATA_DIRS
export XDG_DATA_DIRS
# gsetting
export GSETTINGS_SCHEMA_DIR=$HOME/local/share/glib-2.0/schemas
# girepository-1.0
export GI_TYPELIB_PATH=$HOME/local/lib/x86_64-linux-gnu/girepository-1.0
```

Include this file by . (dot) command before using Gtk4 libraries.

You may think you can add them in your `.profile`. But it's a wrong decision. Never write them to your `.profile`. The environment variables above are necessary only when you compile and run Gtk4 applications. Otherwise it's not necessary. If you changed the environment variables above and run Gtk3 applications, it probably causes serious damage.

2.2.9 Compiling Gtk4 applications

Before you compile Gtk4 applications, define environment variables above.

```
$ . env.sh
```

After that you can compile them without anything. For example, to compile `sample.c`, type the following.

```
$ gcc `pkg-config --cflags gtk4` sample.c `pkg-config --libs gtk4`
```

To know how to compile Gtk4 applications, refer to the section 3 (GtkApplication and GtkApplicationWindow) and after.

2.3 Installing Fedora 34 with gnome-boxes

The last part of this section is about Gnome40 and gnome-boxes. Gnome 40 is a new version of Gnome desktop system. And Gtk4 is installed in the distribution. See Gnome 40 website first.

However, Gnome40 is not necessary to compile and run Gtk4 applications.

There are seven choices at present.

- Gnome OS
- Arch Linux
- Gentoo Linux
- Fedora 36
- openSUSE Tumbleweed
- Ubuntu 22.04
- Debian bookworm

I've installed Fedora 34 with gnome-boxes. My OS was Ubuntu 21.04 at that time. Gnome-boxes creates a virtual machine in Ubuntu and Fedora will be installed to that virtual machine.

The instruction is as follows.

1. Download Fedora 34 iso file. There is an link at the end of Gnome 40 website.
2. Install gnome-boxes with apt-get command.

```
$ sudo apt-get install gnome-boxes
```

3. Run gnome-boxes.
4. Click on `+` button on the top left corner and launch a box creation wizard by clicking `Create a Virtual Machine ...`. Then a dialog appears. Click on `Operationg System Image File` and select the iso file you have downloaded.
5. Then, the Fedora's installer is executed. Follow the instructions by the installer. At the end of the installation, the installer instructs to reboot the system. Click on the right of the title bar and select reboot or shutdown.
6. Your display is back to the initial window of gnome-boxes, but there is a button `Fedora 34 Workstation` on the upper left of the window. Click on the button then Fedora will be executed.
7. A setup dialog appears. Setup Fedora according to the wizard.

Now you can use Fedora. It includes Gtk4 libraries already. But you need to install the Gtk4 development package. Use `dnf` to install `gtk4.x86_64` package.

```
$ sudo dnf install gtk4.x86_64
```

2.3.1 Gtk4 compilation test

You can test the Gtk4 development packages by compiling files which are based on Gtk4. I've tried compiling `tfe` text editor, which is written in section 21.

1. Run Firefox.
2. Open this website (Gtk4-Tutorial).
3. Click on the green button labeled `Code`.
4. Select `Download ZIP` and download the codes from the repository.
5. Unzip the file.

6. Change your current directory to `src/tfe7`.
7. Compile it.

```
$ meson _build
bash: meson: command not found...
Install package 'meson' to provide command 'meson'? [N/y] y

 * Waiting in queue...
The following packages have to be installed:
 meson-0.56.2-2.fc34.noarch    High productivity build system
 ninja-build-1.10.2-2.fc34.x86_64    Small build system with a focus on speed
 vim-filesystem-2:8.2.2787-1.fc34.noarch    VIM filesystem layout
Proceed with changes? [N/y] y

... ...
... ...

The Meson build system
Version: 0.56.2

... ...
... ...

Project name: tfe
Project version: undefined
C compiler for the host machine: cc (gcc 11.0.0 "cc (GCC) 11.0.0 20210210 (Red Hat
    11.0.0-0)")
C linker for the host machine: cc ld.bfd 2.35.1-38
Host machine cpu family: x86_64
Host machine cpu: x86_64
Found pkg-config: /usr/bin/pkg-config (1.7.3)
Run-time dependency gtk4 found: YES 4.2.0
Found pkg-config: /usr/bin/pkg-config (1.7.3)
Program glib-compile-resources found: YES (/usr/bin/glib-compile-resources)
Program glib-compile-schemas found: YES (/usr/bin/glib-compile-schemas)
Program glib-compile-schemas found: YES (/usr/bin/glib-compile-schemas)
Build targets in project: 4

Found ninja-1.10.2 at /usr/bin/ninja

$ ninja -C _build
ninja: Entering directory `_build'
[12/12] Linking target tfe

$ ninja -C _build install
ninja: Entering directory `_build'
[0/1] Installing files.
Installing tfe to /usr/local/bin
Installation failed due to insufficient permissions.
Attempting to use polkit to gain elevated privileges...
Installing tfe to /usr/local/bin
Installing
    /home/<username>/Gtk4-tutorial-main/src/tfe7/com.github.ToshioCP.tfe.gschema.xml
    to /usr/local/share/glib-2.0/schemas
Running custom install script '/usr/bin/glib-compile-schemas
    /usr/local/share/glib-2.0/schemas/'
```

8. Execute it.

```
$ tfe
```

Then, the window of `tfe` text editor appears. The compilation and execution have succeeded.

3 GtkApplication and GtkApplicationWindow

3.1 GtkApplication

3.1.1 GtkApplication and g_application_run

Usually people write programming code to make an application. What are applications? Applications are software that runs using libraries, which includes the OS, frameworks and so on. In Gtk4 programming, the GtkApplication is a program (or executable) that runs using Gtk libraries.

The basic way to write a GtkApplication is as follows.

- Create a GtkApplication instance.
- Run the application.

That's all. Very simple. The following is the C code representing the scenario above.

```
1  #include <gtk/gtk.h>
2
3  int
4  main (int argc, char **argv) {
5    GtkApplication *app;
6    int stat;
7
8    app = gtk_application_new ("com.github.ToshioCP.pr1", G_APPLICATION_FLAGS_NONE);
9    stat =g_application_run (G_APPLICATION (app), argc, argv);
10   g_object_unref (app);
11   return stat;
12 }
```

The first line says that this program includes the header files of the Gtk libraries. The function `main` above is a startup function in C language. The variable `app` is defined as a pointer to a GtkApplication instance. The function `gtk_application_new` creates a GtkApplication instance and returns a pointer to the instance. The GtkApplication instance is a C structure data in which the information about the application is stored. The meaning of the arguments will be explained later. The function `g_application_run` runs an application that the instance defined. (We often say that the function runs `app`. Actually, `app` is not an application but a pointer to the instance of the application. However, it is simple and short, and probably no confusion occurs.)

To compile this, the following command needs to be run. The string `pr1.c` is the filename of the C source code above.

```
$ gcc `pkg-config --cflags gtk4` pr1.c `pkg-config --libs gtk4`
```

The C compiler gcc generates an executable file, `a.out`. Let's run it.

```
$ ./a.out

(a.out:13533): GLib-GIO-WARNING **: 15:30:17.449: Your application does not implement
g_application_activate() and has no handlers connected to the "activate" signal.
It should do one of these.
$
```

Oh, it just produces an error message. This error message means that the GtkApplication object ran, without a doubt. Now, let's think about what this message means.

3.1.2 signal

The message tells us that:

1. The application GtkApplication doesn't implement `g_application_activate()`,
2. It has no handlers connected to the "activate" signal, and
3. You will need to solve at least one of these.

These two causes of the error are related to signals. So, I will explain that to you first.

A signal is emitted when something happens. For example, a window is created, a window is destroyed and so on. The signal "activate" is emitted when the application is activated, or started. If the signal is connected to a function, which is called a signal handler or simply handler, then the function is invoked when the signal emits.

The flow is like this:

1. Something happens.
2. If it's related to a certain signal, then the signal is emitted.
3. If the signal as been connected to a handler, then the handler is invoked.

Signals are defined in objects. For example, the "activate" signal belongs to the GApplication object, which is a parent object of GtkApplication object.

The GApplication object is a child object of the GObject object. GObject is the top object in the hierarchy of all the objects.

```
GObject -- GApplication -- GtkApplication
<---parent                      --->child
```

A child object inherits signals, functions, properties and so on from its parent object. So, GtkApplication also has the "activate" signal.

Now we can solve the problem in `pr1.c`. We need to connect the "activate" signal to a handler. We use a function `g_signal_connect` which connects a signal to a handler.

```
1  #include <gtk/gtk.h>
2
3  static void
4  app_activate (GApplication *app, gpointer *user_data) {
5    g_print ("GtkApplication is activated.\n");
6  }
7
8  int
9  main (int argc, char **argv) {
10   GtkApplication *app;
11   int stat;
12
13   app = gtk_application_new ("com.github.ToshioCP.pr2", G_APPLICATION_FLAGS_NONE);
14   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
15   stat =g_application_run (G_APPLICATION (app), argc, argv);
16   g_object_unref (app);
17   return stat;
18 }
```

First, we define the handler `app_activate` which simply displays a message. In the function `main`, we add `g_signal_connect` before `g_application_run`. The function `g_signal_connect` has four arguments.

1. An instance to which the signal belongs.
2. The name of the signal.
3. A handler function (also called callback), which needs to be casted by `G_CALLBACK`.
4. Data to pass to the handler. If no data is necessary, NULL should be given.

You can find the description of each signal in the API reference manual. For example, "activate" signal is in GApplication section in GIO API Reference. The handler function is described in it.

In addition, `g_signal_connect` is described in GObject API Reference. API reference manual is very important. You should see and understand it to write Gtk applications. They are located in 'GTK Documentation'.

Let's compile the source file above (`pr2.c`) and run it.

```
$ gcc `pkg-config --cflags gtk4` pr2.c `pkg-config --libs gtk4`
$ ./a.out
GtkApplication is activated.
$
```

OK, well done. However, you may have noticed that it's painful to type such a long line to compile. It is a good idea to use shell script to solve this problem. Make a text file which contains the following line.

```
gcc `pkg-config --cflags gtk4` $1.c `pkg-config --libs gtk4`
```

Then, save it under the directory $HOME/bin, which is usually /home/(username)/bin. (If your user name is James, then the directory is /home/james/bin). And turn on the execute bit of the file. If the filename is `comp`, do like this:

```
$ chmod 755 $HOME/bin/comp
$ ls -log $HOME/bin
    ...  ...  ...
-rwxr-xr-x 1   62 May 23 08:21 comp
    ...  ...  ...
```

If this is the first time that you make a $HOME/bin directory and save a file in it, then you need to logout and login again.

```
$ comp pr2
$ ./a.out
GtkApplication is activated.
$
```

3.2 GtkWindow and GtkApplicationWindow

3.2.1 GtkWindow

A message "GtkApplication is activated." was printed out in the previous subsection. It was good in terms of a test of GtkApplication. However, it is insufficient because Gtk is a framework for graphical user interface (GUI). Now we go ahead with adding a window into this program. What we need to do is:

1. Create a GtkWindow.
2. Connect it to GtkApplication.
3. Show the window.

Now rewrite the function `app_activate`.

```
1 static void
2 app_activate (GApplication *app, gpointer user_data) {
3   GtkWidget *win;
4
5   win = gtk_window_new ();
6   gtk_window_set_application (GTK_WINDOW (win), GTK_APPLICATION (app));
7   gtk_widget_show (win);
8 }
```

Widget is an abstract concept that includes all the GUI interfaces such as windows, dialogs, buttons, multi-line text, containers and so on. And GtkWidget is a base object from which all the GUI objects derive.

```
parent <-----> child
GtkWidget -- GtkWindow
```

GtkWindow includes GtkWidget at the top of its object.

The function `gtk_window_new` is defined as follows.

```
GtkWidget *
gtk_window_new (void);
```

By this definition, it returns a pointer to GtkWidget, not GtkWindow. It actually creates a new GtkWindow instance (not GtkWidget) but returns a pointer to GtkWidget. However,the pointer points the GtkWidget and at the same time it also points GtkWindow that contains GtkWidget in it.

If you want to use `win` as a pointer to the GtkWindow, you need to cast it.

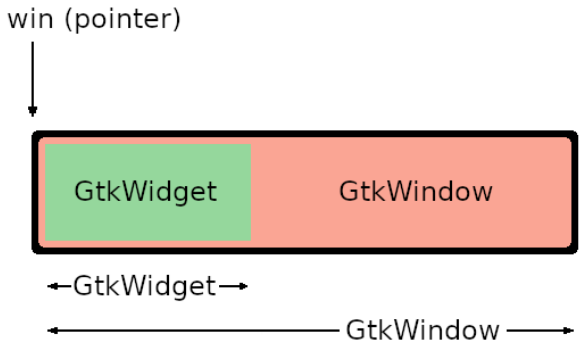

Figure 1: GtkWindow and GtkWidget

```
(GtkWindow *) win
```

Or you can use `GTK_WINDOW` macro that performs a similar function.

```
GTK_WINDOW (win)
```

This is a recommended way.

Connect it to GtkApplication. The function `gtk_window_set_application` is used to connect GtkWindow to GtkApplication.

```
gtk_window_set_application (GTK_WINDOW (win), GTK_APPLICATION (app));
```

You need to cast `win` to GtkWindow and `app` to GtkApplication. `GTK_WINDOW` and `GTK_APPLICATION` macro is appropriate for that.

GtkApplication continues to run until the related window is destroyed. If you didn't connect GtkWindow and GtkApplication, GtkApplication destroys itself immediately. Because no window is connected to GtkApplication, GtkApplication doesn't need to wait anything. As it destroys itself, the GtkWindow is also destroyed.

Show the window. The function `gtk_widget_show` is used to show the window.

Gtk4 changes the default widget visibility to on, so every widget doesn't need this function to show itself. But, there's an exception. Top window (this term will be explained later) isn't visible when it is created. So you need to use the function above to show the window.

Save the program as `pr3.c` and compile and run it.

```
$ comp pr3
$ ./a.out
```

A small window appears.

Click on the close button then the window disappears and the program finishes.

3.2.2 GtkApplicationWindow

GtkApplicationWindow is a child object of GtkWindow. It has some extra functionality for better integration with GtkApplication. It is recommended to use it instead of GtkWindow when you use GtkApplication.

Now rewrite the program and use GtkApplicationWindow.

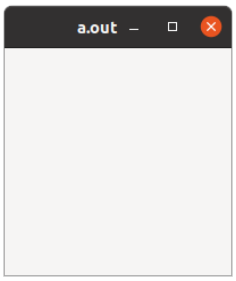

Figure 2: Screenshot of the window

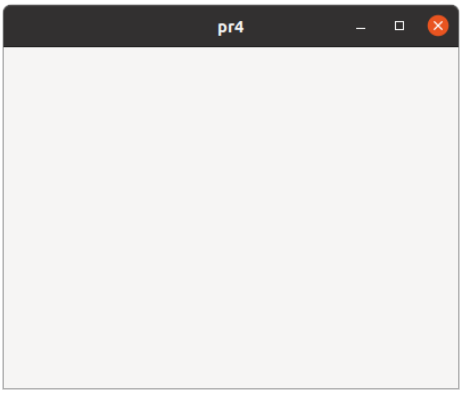

Figure 3: Screenshot of the window

```
1  static void
2  app_activate (GApplication *app, gpointer user_data) {
3    GtkWidget *win;
4
5    win = gtk_application_window_new (GTK_APPLICATION (app));
6    gtk_window_set_title (GTK_WINDOW (win), "pr4");
7    gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
8    gtk_widget_show (win);
9  }
```

When you create GtkApplicationWindow, you need to give GtkApplication instance as an argument. Then it automatically connect these two instances. So you don't need to call `gtk_window_set_application` any more.

The program sets the title and the default size of the window. Compile it and run `a.out`, then you will see a bigger window with its title "pr4".

4 Widgets (1)

4.1 GtkLabel, GtkButton and GtkBox

4.1.1 GtkLabel

In the previous section we made a window and displayed it on the screen. Now we go on to the next topic, where we add widgets to this window. The simplest widget is GtkLabel. It is a widget with text in it.

```
1  #include <gtk/gtk.h>
2
3  static void
4  app_activate (GApplication *app, gpointer user_data) {
5    GtkWidget *win;
6    GtkWidget *lab;
7
```

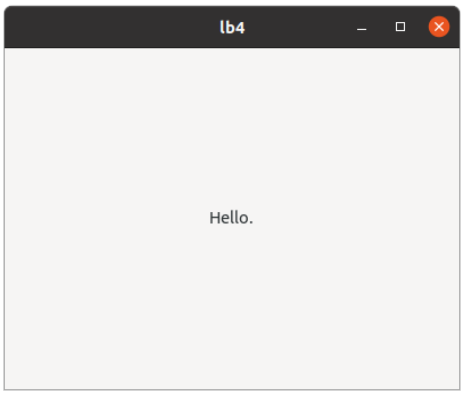

Figure 4: Screenshot of the label

```
 8    win = gtk_application_window_new (GTK_APPLICATION (app));
 9    gtk_window_set_title (GTK_WINDOW (win), "lb1");
10    gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
11
12    lab = gtk_label_new ("Hello.");
13    gtk_window_set_child (GTK_WINDOW (win), lab);
14
15    gtk_widget_show (win);
16  }
17
18  int
19  main (int argc, char **argv) {
20    GtkApplication *app;
21    int stat;
22
23    app = gtk_application_new ("com.github.ToshioCP.lb1", G_APPLICATION_FLAGS_NONE);
24    g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
25    stat =g_application_run (G_APPLICATION (app), argc, argv);
26    g_object_unref (app);
27    return stat;
28  }
```

Save this program to a file `lb1.c`. Then compile and run it.

```
$ comp lb1
$ ./a.out
```

A window with a message “Hello.” appears.

There’s only a little change between `pr4.c` and `lb1.c`. A program `diff` is good to know the difference between two files.

```
$ cd misc; diff pr4.c lb1.c
5a6
>   GtkWidget *lab;
8c9
<   gtk_window_set_title (GTK_WINDOW (win), "pr4");
---
>   gtk_window_set_title (GTK_WINDOW (win), "lb1");
9a11,14
>
>   lab = gtk_label_new ("Hello.");
>   gtk_window_set_child (GTK_WINDOW (win), lab);
>
18c23
<   app = gtk_application_new ("com.github.ToshioCP.pr4", G_APPLICATION_FLAGS_NONE);
---
>   app = gtk_application_new ("com.github.ToshioCP.lb1", G_APPLICATION_FLAGS_NONE);
```

This tells us:

- The definition of a new variable `lab` is added.
- The title of the window is changed.
- A label is created and connected to the window as a child.

The function `gtk_window_set_child (GTK_WINDOW (win), lab)` makes the label `lab` a child widget of the window `win`. Be careful. A child widget is different from a child object. Objects have parent-child relationships and widgets also have parent-child relationships. But these two relationships are totally different. Don't be confused. In the program `lb1.c`, `lab` is a child widget of `win`. Child widgets are always located in their parent widget on the screen. See how the window has appeared on the screen. The application window includes the label.

The window `win` doesn't have any parents. We call such a window top-level window. An application can have more than one top-level window.

4.1.2 GtkButton

The next widget to introduce is GtkButton. It displays a button on the screen with a label or icon on it. In this subsection, we will make a button with a label. When the button is clicked, it emits a "clicked" signal. The following program shows how to catch the signal to then do something.

```
1  #include <gtk/gtk.h>
2
3  static void
4  click_cb (GtkButton *btn, gpointer user_data) {
5    g_print ("Clicked.\n");
6  }
7
8  static void
9  app_activate (GApplication *app, gpointer user_data) {
10   GtkWidget *win;
11   GtkWidget *btn;
12
13   win = gtk_application_window_new (GTK_APPLICATION (app));
14   gtk_window_set_title (GTK_WINDOW (win), "lb2");
15   gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
16
17   btn = gtk_button_new_with_label ("Click me");
18   gtk_window_set_child (GTK_WINDOW (win), btn);
19   g_signal_connect (btn, "clicked", G_CALLBACK (click_cb), NULL);
20
21   gtk_widget_show (win);
22 }
23
24 int
25 main (int argc, char **argv) {
26   GtkApplication *app;
27   int stat;
28
29   app = gtk_application_new ("com.github.ToshioCP.lb2", G_APPLICATION_FLAGS_NONE);
30   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
31   stat =g_application_run (G_APPLICATION (app), argc, argv);
32   g_object_unref (app);
33   return stat;
34 }
```

Look at the line 17 to 19. First, it creates a GtkButton instance `btn` with a label "Click me". Then, adds the button to the window `win` as a child. Finally, connects a "clicked" signal of the button to a handler (function) `click_cb`. So, if `btn` is clicked, the function `click_cb` is invoked. The suffix "cb" means "call back".

Name the program `lb2.c` and save it. Now compile and run it.

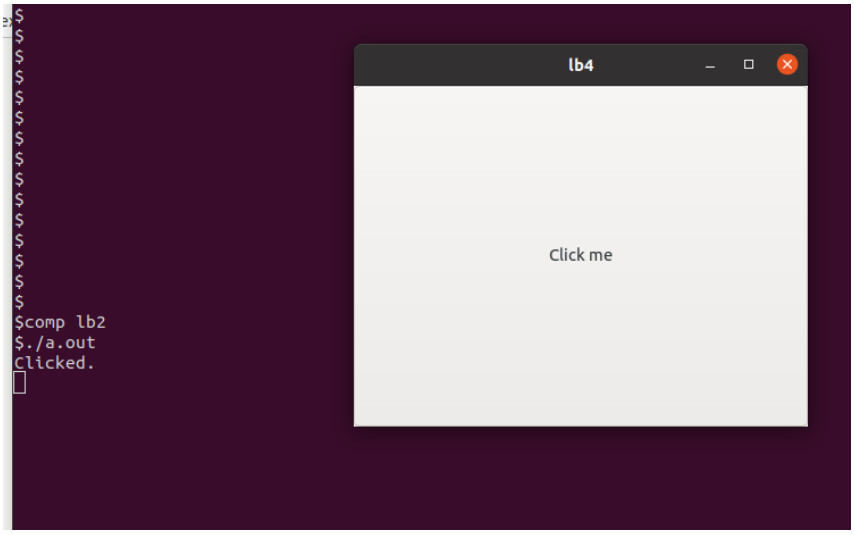

Figure 5: Screenshot of the label

A window with the button appears. Click the button (it is a large button, you can click everywhere in the window), then a string "Clicked." appears on the terminal. It shows the handler was invoked by clicking the button.

It's good that we make sure that the clicked signal was caught and the handler was invoked by using `g_print`. However, using g_print is out of harmony with Gtk which is a GUI library. So, we will change the handler. The following code is `lb3.c`.

```
1  static void
2  click_cb (GtkButton *btn, gpointer user_data) {
3    GtkWindow *win = GTK_WINDOW (user_data);
4    gtk_window_destroy (win);
5  }
6
7  static void
8  app_activate (GApplication *app, gpointer user_data) {
9    GtkWidget *win;
10   GtkWidget *btn;
11
12   win = gtk_application_window_new (GTK_APPLICATION (app));
13   gtk_window_set_title (GTK_WINDOW (win), "lb3");
14   gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
15
16   btn = gtk_button_new_with_label ("Quit");
17   gtk_window_set_child (GTK_WINDOW (win), btn);
18   g_signal_connect (btn, "clicked", G_CALLBACK (click_cb), win);
19
20   gtk_widget_show (win);
21 }
```

And the difference between `lb2.c` and `lb3.c` is as follows.

```
$ cd misc; diff lb2.c lb3.c
5c5,6
<   g_print ("Clicked.\n");
---
>   GtkWindow *win = GTK_WINDOW (user_data);
>   gtk_window_destroy (win);
14c15
<   gtk_window_set_title (GTK_WINDOW (win), "lb2");
---
>   gtk_window_set_title (GTK_WINDOW (win), "lb3");
17c18
```

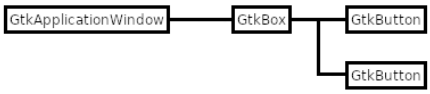

Figure 6: Parent-child relationship

```
<   btn = gtk_button_new_with_label ("Click me");
---
>   btn = gtk_button_new_with_label ("Quit");
19c20
<   g_signal_connect (btn, "clicked", G_CALLBACK (click_cb), NULL);
---
>   g_signal_connect (btn, "clicked", G_CALLBACK (click_cb), win);
29c30
<   app = gtk_application_new ("com.github.ToshioCP.lb2", G_APPLICATION_FLAGS_NONE);
---
>   app = gtk_application_new ("com.github.ToshioCP.lb3", G_APPLICATION_FLAGS_NONE);
35d35
<
```

The changes are:

- The function `g_print` in `lb2.c` was deleted and the two lines above are inserted instead.
- The label of `btn` is changed from "Click me" to "Quit".
- The fourth argument of `g_signal_connect` is changed from `NULL` to `win`.

The most important change is the fourth argument of `g_signal_connect`. This argument is described as "data to pass to handler" in the definition of `g_signal_connect` in GObject API Reference. Therefore, `win` which is a pointer to GtkApplicationWindow is passed to the handler as a second parameter `user_data`. The handler then casts it to a pointer to GtkWindow and calls `gtk_window_destroy` to destroy the top-level window. The application then quits.

4.1.3 GtkBox

GtkWindow and GtkApplicationWindow can have only one child. If you want to add two or more widgets in a window, you need a container widget. GtkBox is one of the containers. It arranges two or more child widgets into a single row or column. The following procedure shows the way to add two buttons in a window.

- Create a GtkApplicationWindow instance.
- Create a GtkBox instance and add it to the GtkApplicationWindow as a child.
- Create a GtkButton instance and append it to the GtkBox.
- Create another GtkButton instance and append it to the GtkBox.

After this, the Widgets are connected as the following diagram.

The program `lb4.c` includes these widgets. It is as follows.

```
1  #include <gtk/gtk.h>
2
3  static void
4  click1_cb (GtkButton *btn, gpointer user_data) {
5    const gchar *s;
6
7    s = gtk_button_get_label (btn);
8    if (g_strcmp0 (s, "Hello.") == 0)
9      gtk_button_set_label (btn, "Good-bye.");
10   else
11     gtk_button_set_label (btn, "Hello.");
12 }
13
14 static void
```

```
15 click2_cb (GtkButton *btn, gpointer user_data) {
16   GtkWindow *win = GTK_WINDOW (user_data);
17   gtk_window_destroy (win);
18 }
19
20 static void
21 app_activate (GApplication *app, gpointer user_data) {
22   GtkWidget *win;
23   GtkWidget *box;
24   GtkWidget *btn1;
25   GtkWidget *btn2;
26
27   win = gtk_application_window_new (GTK_APPLICATION (app));
28   gtk_window_set_title (GTK_WINDOW (win), "lb4");
29   gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
30
31   box = gtk_box_new (GTK_ORIENTATION_VERTICAL, 5);
32   gtk_box_set_homogeneous (GTK_BOX (box), TRUE);
33   gtk_window_set_child (GTK_WINDOW (win), box);
34
35   btn1 = gtk_button_new_with_label ("Hello.");
36   g_signal_connect (btn1, "clicked", G_CALLBACK (click1_cb), NULL);
37
38   btn2 = gtk_button_new_with_label ("Quit");
39   g_signal_connect (btn2, "clicked", G_CALLBACK (click2_cb), win);
40
41   gtk_box_append (GTK_BOX (box), btn1);
42   gtk_box_append (GTK_BOX (box), btn2);
43
44   gtk_widget_show (win);
45 }
46
47 int
48 main (int argc, char **argv) {
49   GtkApplication *app;
50   int stat;
51
52   app = gtk_application_new ("com.github.ToshioCP.lb4", G_APPLICATION_FLAGS_NONE);
53   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
54   stat =g_application_run (G_APPLICATION (app), argc, argv);
55   g_object_unref (app);
56   return stat;
57 }
```

Look at the function `app_activate`.

After the creation of a GtkApplicationWindow instance, a GtkBox instance is created.

```
box = gtk_box_new(GTK_ORIENTATION_VERTICAL, 5);
gtk_box_set_homogeneous (GTK_BOX (box), TRUE);
```

The first argument arranges the children of the box vertically. The second argument is the size between the children. The next function fills the box with the children, giving them the same space.

After that, two buttons `btn1` and `btn2` are created and the signal handlers are set. Then, these two buttons are appended to the box.

The handler corresponds to `btn1` toggles its label. The handler corresponds to `btn2` destroys the top-level window and the application quits.

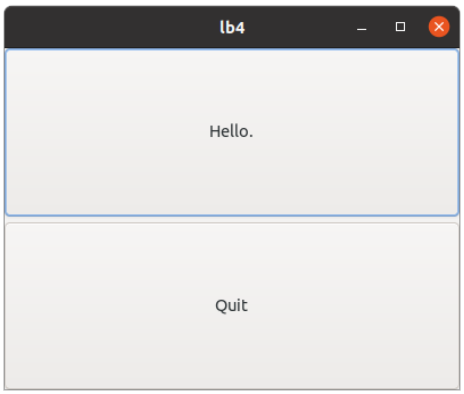

Figure 7: Screenshot of the box

5 Widgets (2)

5.1 GtkTextView, GtkTextBuffer and GtkScrolledWindow

5.1.1 GtkTextView and GtkTextBuffer

GtkTextView is a widget for multi-line text editing. GtkTextBuffer is a text buffer which is connected to GtkTextView. See the sample program `tfv1.c` below.

```
1  #include <gtk/gtk.h>
2
3  static void
4  app_activate (GApplication *app, gpointer user_data) {
5    GtkWidget *win;
6    GtkWidget *tv;
7    GtkTextBuffer *tb;
8    gchar *text;
9
10   text =
11       "Once upon a time, there was an old man who was called Taketori-no-Okina. "
12       "It is a japanese word that means a man whose work is making bamboo baskets.\n"
13       "One day, he went into a mountain and found a shining bamboo. "
14       "\"What a mysterious bamboo it is!,\" he said. "
15       "He cut it, then there was a small cute baby girl in it. "
16       "The girl was shining faintly. "
17       "He thought this baby girl is a gift from Heaven and took her home.\n"
18       "His wife was surprized at his tale. "
19       "They were very happy because they had no children. "
20       ;
21   win = gtk_application_window_new (GTK_APPLICATION (app));
22   gtk_window_set_title (GTK_WINDOW (win), "Taketori");
23   gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
24
25   tv = gtk_text_view_new ();
26   tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
27   gtk_text_buffer_set_text (tb, text, -1);
28   gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
29
30   gtk_window_set_child (GTK_WINDOW (win), tv);
31
32   gtk_widget_show (win);
33 }
34
35 int
36 main (int argc, char **argv) {
37   GtkApplication *app;
38   int stat;
```

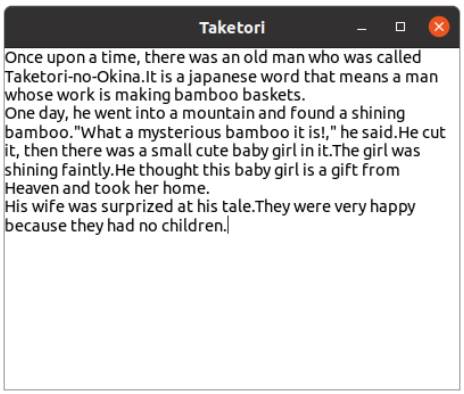

Figure 8: GtkTextView

```
39
40    app = gtk_application_new ("com.github.ToshioCP.tfv1", G_APPLICATION_FLAGS_NONE);
41    g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
42    stat = g_application_run (G_APPLICATION (app), argc, argv);
43    g_object_unref (app);
44    return stat;
45  }
```

Look at line 25. A GtkTextView instance is created and its pointer is assigned to `tv`. When the GtkTextView instance is created, a GtkTextBuffer instance is also created and connected to the GtkTextView automatically. "GtkTextBuffer instance" will be referred to simply as "GtkTextBuffer" or "buffer". In the next line, the pointer to the buffer is got and assigned to `tb`. Then, the text from line 10 to 20 is assigned to the buffer.

GtkTextView has a wrap mode. When it is set to `GTK_WRAP_WORD_CHAR`, text wraps in between words, or if that is not enough, also between graphemes.

In line 30, `tv` is added to `win` as a child.

Now compile and run it.

There's an I-beam pointer in the window. You can add or delete any characters on the GtkTextView, and your changes are kept in the GtkTextBuffer. If you add more characters beyond the limit of the window, the height increases and the window extends. If the height gets bigger than the height of the display screen, you won't be able to control the size of the window, and change it back to the original size. This is a problem and shows that there is a bug in our program. This can solve it by adding a GtkScrolledWindow between the GtkApplicationWindow and GtkTextView.

5.1.2 GtkScrolledWindow

What we need to do is:

- Create a GtkScrolledWindow and insert it as a child of the GtkApplicationWindow; and
- Insert the GtkTextView widget to the GtkScrolledWindow as a child.

Modify `tfv1.c` and save it as `tfv2.c`. The difference between these two files is small.

```
$ cd tfv; diff tfv1.c tfv2.c
5a6
>   GtkWidget *scr;
24a26,28
>   scr = gtk_scrolled_window_new ();
>   gtk_window_set_child (GTK_WINDOW (win), scr);
>
30c34
<   gtk_window_set_child (GTK_WINDOW (win), tv);
---
>   gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);
40c44
```

```
<   app = gtk_application_new ("com.github.ToshioCP.tfv1", G_APPLICATION_FLAGS_NONE);
---
>   app = gtk_application_new ("com.github.ToshioCP.tfv2", G_APPLICATION_FLAGS_NONE);
```

Here is the complete code of `tfv2.c`.

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

static void
app_activate (GApplication *app, gpointer user_data) {
  GtkWidget *win;
  GtkWidget *scr;
  GtkWidget *tv;
  GtkTextBuffer *tb;
  gchar *text;

  text =
      "Once upon a time, there was an old man who was called Taketori-no-Okina. "
      "It is a japanese word that means a man whose work is making bamboo baskets.\n"
      "One day, he went into a mountain and found a shining bamboo. "
      "\"What a mysterious bamboo it is!,\" he said. "
      "He cut it, then there was a small cute baby girl in it. "
      "The girl was shining faintly. "
      "He thought this baby girl is a gift from Heaven and took her home.\n"
      "His wife was surprized at his tale. "
      "They were very happy because they had no children. "
      ;
  win = gtk_application_window_new (GTK_APPLICATION (app));
  gtk_window_set_title (GTK_WINDOW (win), "Taketori");
  gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);

  scr = gtk_scrolled_window_new ();
  gtk_window_set_child (GTK_WINDOW (win), scr);

  tv = gtk_text_view_new ();
  tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
  gtk_text_buffer_set_text (tb, text, -1);
  gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);

  gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);

  gtk_widget_show (win);
}

int
main (int argc, char **argv) {
  GtkApplication *app;
  int stat;

  app = gtk_application_new ("com.github.ToshioCP.tfv2", G_APPLICATION_FLAGS_NONE);
  g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
  stat = g_application_run (G_APPLICATION (app), argc, argv);
  g_object_unref (app);
  return stat;
}
```

Compile and run it. Notice how this time the window doesn't extend when you type a lot of characters, it just scrolls and displays a slider.

6 String and memory management

GtkTextView and GtkTextBuffer have functions that use string parameters or return a string. The knowledge of strings and memory management is useful to understand how to use these functions.

6.1 String and memory

A String is an array of characters that is terminated with '\0'. Strings are not a C type such as char, int, float or double, but exist as a pointer to a character array. They behaves like a string type which you may be familiar from other languages. So, this pointer is often called 'a string'.

In the following, `a` and `b` defined as character arrays, and are strings.

```
char a[10], *b;

a[0] = 'H';
a[1] = 'e';
a[2] = 'l';
a[3] = 'l';
a[4] = 'o';
a[5] = '\0';

b = a;
/* *b is 'H' */
/* *(++b) is 'e' */
```

The array `a` has `char` elements and the size of ten. The first six elements are 'H', 'e', 'l', 'l', 'o' and '\0'. This array represents the string "Hello". The first five elements are character codes that correspond to the characters. The sixth element is '\0', which is the same as zero, and indicates that the string ends there. The size of the array is 10, so 4 bytes aren't used, but that's OK, they are just ignored.

The variable 'b' is a pointer to a character. Because `b` is assigned to be `a`, `a` and `b` point the same character ('H'). The variable `a` is defined as an array and it can't be changed. It always point the top address of the array. On the other hand, 'b' is a pointer, which is mutable, so `b` can be change. It is then possible to write statements like `++b`, which means take the value in b (n address), increase it by one, and store that back in `b`.

If a pointer is NULL, it points to nothing. So, the pointer is not a string. A NULL string on the other hand will be a pointer which points to a location that contains `\0`, which is a string of length 0 (or ""). Programs that use strings will include bugs if you aren't careful when using NULL pointers.

Another annoying problem is the memory that a string is allocated. There are four cases:

- The string is read only;
- The string is in static memory area;
- The string is in stack; and
- The string is in memory allocated from the heap area.

6.2 Read only string

A string literal in a C program is surrounded by double quotes and written as the following:

```
char *s;
s = "Hello"
```

"Hello" is a string literal, and is stored in program memory. A string literal is read only. In the program above, `s` points the string literal.

So, the following program is illegal.

```
*(s+1) = 'a';
```

The result is undefined. Probably a bad thing will happen, for example, a segmentation fault.

NOTE: The memory of the literal string is allocated when the program is compiled. It is possible to view all the literal strings defined in your program by using the `string` command.

6.3 Strings defined as arrays

If a string is defined as an array, it's in either stored in the static memory area or stack. This depends on the class of the array. If the array's class is `static`, then it's placed in static memory area. This allocation and memory address is fixed for the life of the program. This area can be changed and is writable.

If the array's class is `auto`, then it's placed in stack. If the array is defined inside a function, its default class is `auto`. The stack area will disappear when the function exits and returns to the caller. Arrays defined on the stack are writable.

```
static char a[] = {'H', 'e', 'l', 'l', 'o', '\0'};

void
print_strings (void) {
  char b[] = "Hello";

  a[1] = 'a'; /* Because the array is static, it's writable. */
  b[1] = 'a'; /* Because the array is auto, it's writable. */

  printf ("%s\n", a); /* Hallo */
  printf ("%s\n", b); /* Hallo */
}
```

The array `a` is defined externally to a function and is global in its scope. Such variables are placed in static memory area even if the `static` class is left out. The compiler calculates the number of the elements in the right hand side (six), and then creates code that allocates six bytes in the static memory area and copies the data to this memory.

The array `b` is defined inside the function so its class is `auto`. The compiler calculates the number of the elements in the string literal. It has six elements as the zero termination character is also included. The compiler creates code which allocates six bytes memory in the stack and copies the data to the memory.

Both `a` and `b` are writable.

The memory is managed by the executable program. You don't need your program to allocate or free the memory for `a` and `b`. The array `a` is created then the program is first run and remains for the life of the program. The array `b` is created on the stack then the function is called, disappears when the function returns.

6.4 Strings in the heap area

You can also get, use and release memory from the heap area. The standard C library provides `malloc` to get memory and `free` to put back memory. GLib provides the functions `g_new` and `g_free` to do the same thing, with support for some additional Glib functionality.

```
g_new (struct_type, n_struct)
```

`g_new` is a macro to allocate memory for an array.

- `struct_type` is the type of the element of the array.
- `n_struct` is the size of the array.
- The return value is a pointer to the array. Its type is a pointer to `struct_type`.

For example,

```
char *s;
s = g_new (char, 10);
/* s points an array of char. The size of the array is 10. */

struct tuple {int x, y;} *t;
t = g_new (struct tuple, 5);
/* t points an array of struct tuple. */
/* The size of the array is 5. */
```

`g_free` frees memory.

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

If `mem` is NULL, `g_free` does nothing. `gpointer` is a type of general pointer. It is the same as `void *`. This pointer can be casted to any pointer type. Conversely, any pointer type can be casted to `gpointer`.

```
g_free (s);
/* Frees the memory allocated to s. */

g_free (t);
/* Frees the memory allocated to t. */
```

If the argument doesn't point allocated memory it will cause an error, specifically, a segmentation fault.

Some Glib functions allocate memory. For example, `g_strdup` allocates memory and copies a string given as an argument.

```
char *s;
s = g_strdup ("Hello");
g_free (s);
```

The string literal "Hello" has 6 bytes because the string has '\0' at the end it. `g_strdup` gets 6 bytes from the heap area and copies the string to the memory. `s` is assigned the top address of the memory. `g_free` returns the memory to the heap area.

`g_strdup` is described in GLib API Reference. The following is extracted from the reference.

> The returned string should be freed with `g_free()` when no longer needed.

The function reference will describe if the returned value needs to be freed. If you forget to free the allocated memory it will remain allocated. Repeated use will cause more memory to be allocated to the program, which will grow over time. This is called a memory leak, and the only way to address this bug is to close the program (and restart it), which will automatically release all of the programs memory back to the system.

Some GLib functions return a string which mustn't be freed by the caller.

```
const char *
g_quark_to_string (GQuark quark);
```

This function returns `const char*` type. The qualifier `const` means that the returned value is immutable. The characters pointed by the returned value aren't be allowed to be changed or freed.

If a variable is qualified with `const`, the variable can't be assigned except during initialization.

```
const int x = 10; /* initialization is OK. */

x = 20; /* This is illegal because x is qualified with const */
```

7 Widgets (3)

7.1 Open signal

7.1.1 G_APPLICATION_HANDLES_OPEN flag

The GtkTextView, GtkTextBuffer and GtkScrolledWindow widgets have given us a minimum editor in the previous section. We will now add a function to read a file and rework the program into a file viewer. There are many ways to implement the function and because this is a tutorial for beginners, we'll take the easiest one.

When the program starts, we will give the filename to open as an argument.

```
$ ./a.out filename
```

It will open the file and insert its contents into the GtkTextBuffer.

To do this, we need to know how GtkApplication (or GApplication) recognizes arguments. This is described in the GIO API Reference, Application.

When GtkApplication is created, a flag (with the type GApplicationFlags) is provided as an argument.

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

This tutorial explains only two flags, `G_APPLICATION_FLAGS_NONE` and `G_APPLICATION_HANDLES_OPEN`. If you want to handle command line arguments, the `G_APPLICATION_HANDLES_COMMAND_LINE` flag is what you need. How to use the new application method is described in GIO API Reference, g_application_run, and the flag is described in the GIO API Reference, ApplicationFlags.

```
GApplicationFlags' Members

G_APPLICATION_FLAGS_NONE  Default. (No argument allowed)
  ... ... ...
G_APPLICATION_HANDLES_OPEN  This application handles opening files (in the primary
    instance).
  ... ... ...
```

There are ten flags in total, but we only need two of them so far. We've already used `G_APPLICATION_FLAGS_NONE`, as it is the simplest option, and no arguments are allowed. If you provide arguments when running the application, an error will occur.

The flag `G_APPLICATION_HANDLES_OPEN` is the second simplest option. It allows arguments but only files. The application assumes all the arguments are filenames and we will use this flag when creating our GtkApplication.

```
app = gtk_application_new ("com.github.ToshioCP.tfv3", G_APPLICATION_HANDLES_OPEN);
```

7.1.2 open signal

Now, when the application starts, two signals can be emitted.

- activate signal — This signal is emitted when there's no argument.
- open signal — This signal is emitted when there is at least one argument.

The handler of the "open" signal is defined as follows.

```
void user_function (GApplication *application,
                    gpointer      files,
                    gint          n_files,
                    gchar        *hint,
                    gpointer      user_data)
```

The parameters are:

- `application` — the application (usually GtkApplication)
- `files` — an array of GFiles. [array length=n_files] [element-type GFile]
- `n_files` — the number of the elements of `files`
- `hint` — a hint provided by the calling instance (usually it can be ignored)
- `user_data` — user data set when the signal handler was connected.

How to read a specified file (GFile) will be described next.

7.2 Making a file viewer

7.2.1 What is a file viewer?

A file viewer is a program that displays the text file that is given as an argument. Our file viewer will work as follows.

- When arguments are given, it treats the first argument as a filename and opens it.
- If opening the file succeeds, it reads the contents of the file and inserts it to GtkTextBuffer and then shows the window.
- If it fails to open the file, it will show an error message and quit.
- If there's no argument, it will shows an error message and quit.
- If there are two or more arguments, the second one and any others are ignored.

The program which does this is shown below.

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

static void
app_activate (GApplication *app, gpointer user_data) {
  g_print ("You need a filename argument.\n");
}

static void
app_open (GApplication *app, GFile ** files, gint n_files, gchar *hint, gpointer
    user_data) {
  GtkWidget *win;
  GtkWidget *scr;
  GtkWidget *tv;
  GtkTextBuffer *tb;
  char *contents;
  gsize length;
  char *filename;

  win = gtk_application_window_new (GTK_APPLICATION (app));
  gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);

  scr = gtk_scrolled_window_new ();
  gtk_window_set_child (GTK_WINDOW (win), scr);

  tv = gtk_text_view_new ();
  tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
  gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
  gtk_text_view_set_editable (GTK_TEXT_VIEW (tv), FALSE);
  gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);

  if (g_file_load_contents (files[0], NULL, &contents, &length, NULL, NULL)) {
    gtk_text_buffer_set_text (tb, contents, length);
    g_free (contents);
    if ((filename = g_file_get_basename (files[0])) != NULL) {
      gtk_window_set_title (GTK_WINDOW (win), filename);
      g_free (filename);
    }
    gtk_widget_show (win);
  } else {
    if ((filename = g_file_get_path (files[0])) != NULL) {
      g_print ("No such file: %s.\n", filename);
      g_free (filename);
    }
    gtk_window_destroy (GTK_WINDOW (win));
  }
}

int
main (int argc, char **argv) {
  GtkApplication *app;
  int stat;

  app = gtk_application_new ("com.github.ToshioCP.tfv3", G_APPLICATION_HANDLES_OPEN);
  g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
  g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);
  stat = g_application_run (G_APPLICATION (app), argc, argv);
  g_object_unref (app);
  return stat;
}
```

Save it as `tfv3.c`. Then compile and run it.

```
$ comp tfv3
$ ./a.out tfv3.c
```

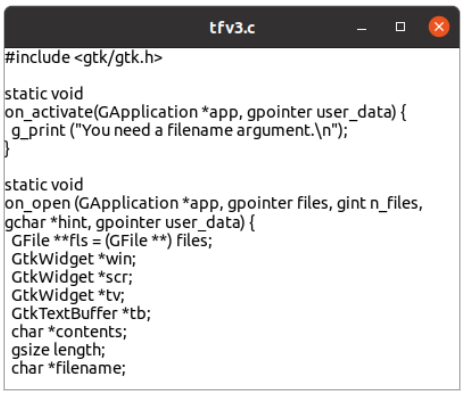

Figure 9: File viewer

Let's explain how the program `tfv3.c` works. First, the function `main` has only two changes from the previous version.

- `G_APPLICATION_FLAGS_NONE` is replaced by `G_APPLICATION_HANDLES_OPEN`; and
- `g_signal_connect (app, "open", G_CALLBACK (on_open), NULL)` is added.

Next, the handler `app_activate` is added and is very simple. It just outputs the error message and the application quits immediately because no window is created.

The main functionality is the in the handler `app_open`. It

- Creates GtkApplicationWindow, GtkScrolledWindow, GtkTextView and GtkTextBuffer and connects them together;
- Sets wrap mode to `GTK_WRAP_WORD_CHAR` in GtktextView;
- Sets GtkTextView to non-editable because the program isn't an editor but only a viewer;
- Reads the file and inserts the text into GtkTextBuffer (this will be explained in detail later); and
- If the file is not opened then outputs an error message and destroys the window. This makes the application quit.

The following is the important file reading part of the program and is shown again below.

```
if (g_file_load_contents (files[0], NULL, &contents, &length, NULL, NULL)) {
  gtk_text_buffer_set_text (tb, contents, length);
  g_free (contents);
  if ((filename = g_file_get_basename (files[0])) != NULL) {
    gtk_window_set_title (GTK_WINDOW (win), filename);
    g_free (filename);
  }
  gtk_widget_show (win);
} else {
  if ((filename = g_file_get_path (files[0])) != NULL) {
    g_print ("No such file: %s.\n", filename);
    g_free (filename);
  }
  gtk_window_destroy (GTK_WINDOW (win));
}
```

The function `g_file_load_contents` loads the file contents into a buffer, which is automatically allocated and sets `contents` to point that buffer. The length of the buffer is set to `length`. It returns `TRUE` if the file's contents are successfully loaded and `FALSE` if an error occurs.

If this function succeeds, it inserts the contents into GtkTextBuffer, frees the buffer pointed by `contents`, sets the title of the window, frees the memories pointed by `filename` and then shows the window. If it fails, it outputs an error message and destroys the window, causing the program to quit.

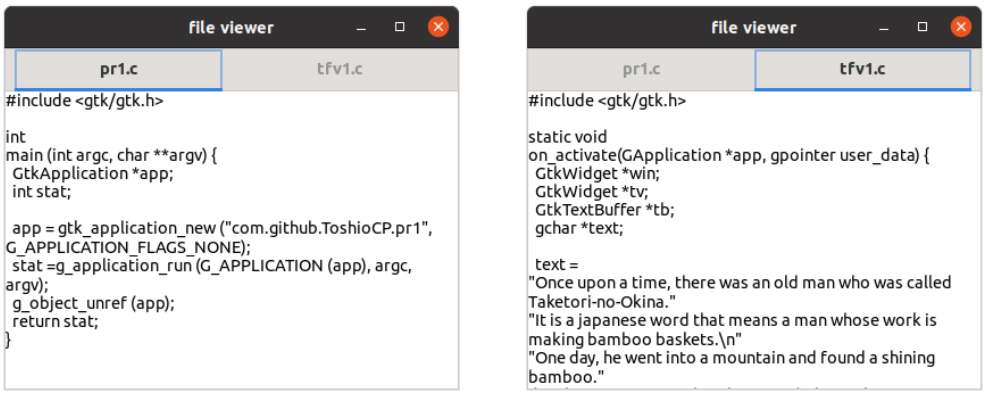

Figure 10: GtkNotebook

7.3 GtkNotebook

GtkNotebook is a container widget that uses tabs and contains multiple children. The child that is displayed depends on which tab has been selected.

Looking at the screenshots above, the left one is the window at the startup. It shows the file `pr1.c` and the filename is in the left tab. After clicking on the right tab, the contents of the file `tfv1.c` are shown instead. This is shown in the right screenshot.

The GtkNotebook widget is inserted as a child of GtkApplicationWindow and contains a GtkScrolledWindow for each file that is being displayed. The code to do this is given in `tfv4.c` and is:

```
1  #include <gtk/gtk.h>
2
3  static void
4  app_activate (GApplication *app, gpointer user_data) {
5    g_print ("You need a filename argument.\n");
6  }
7
8  static void
9  app_open (GApplication *app, GFile ** files, gint n_files, gchar *hint, gpointer
       user_data) {
10   GtkWidget *win;
11   GtkWidget *nb;
12   GtkWidget *lab;
13   GtkNotebookPage *nbp;
14   GtkWidget *scr;
15   GtkWidget *tv;
16   GtkTextBuffer *tb;
17   char *contents;
18   gsize length;
19   char *filename;
20   int i;
21
22   win = gtk_application_window_new (GTK_APPLICATION (app));
23   gtk_window_set_title (GTK_WINDOW (win), "file viewer");
24   gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
25   gtk_window_maximize (GTK_WINDOW (win));
26
27   nb = gtk_notebook_new ();
28   gtk_window_set_child (GTK_WINDOW (win), nb);
29
30   for (i = 0; i < n_files; i++) {
31     if (g_file_load_contents (files[i], NULL, &contents, &length, NULL, NULL)) {
32       scr = gtk_scrolled_window_new ();
33       tv = gtk_text_view_new ();
34       tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
35       gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
```

```
36        gtk_text_view_set_editable (GTK_TEXT_VIEW (tv), FALSE);
37        gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);
38
39        gtk_text_buffer_set_text (tb, contents, length);
40        g_free (contents);
41        if ((filename = g_file_get_basename (files[i])) != NULL) {
42          lab = gtk_label_new (filename);
43          g_free (filename);
44        } else
45          lab = gtk_label_new ("");
46        gtk_notebook_append_page (GTK_NOTEBOOK (nb), scr, lab);
47        nbp = gtk_notebook_get_page (GTK_NOTEBOOK (nb), scr);
48        g_object_set (nbp, "tab-expand", TRUE, NULL);
49      } else if ((filename = g_file_get_path (files[i])) != NULL) {
50          g_print ("No such file: %s.\n", filename);
51          g_free (filename);
52      } else
53          g_print ("No valid file is given\n");
54    }
55    if (gtk_notebook_get_n_pages (GTK_NOTEBOOK (nb)) > 0)
56      gtk_widget_show (win);
57    else
58      gtk_window_destroy (GTK_WINDOW (win));
59  }
60
61  int
62  main (int argc, char **argv) {
63    GtkApplication *app;
64    int stat;
65
66    app = gtk_application_new ("com.github.ToshioCP.tfv4", G_APPLICATION_HANDLES_OPEN);
67    g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
68    g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);
69    stat = g_application_run (G_APPLICATION (app), argc, argv);
70    g_object_unref (app);
71    return stat;
72  }
```

Most of the changes are in the function `app_open`. The numbers at the left of the following items are line numbers in the source code.

- 11-13: Variables `nb`, `lab` and `nbp` are defined and will point to a new GtkNotebook, GtkLabel and GtkNotebookPage widget respectively.
- 23: The window's title is set to "file viewer".
- 25: The size of the window is set to maximum because a big window is appropriate for file viewers.
- 27-28 GtkNotebook is created and inserted to the GtkApplicationWindow as a child.
- 30-59 For-loop. Each loop corresponds to a filename argument, and `files[i]` is GFile object containing the i-th argument.
- 32-37 GtkScrollledWindow, GtkTextView are created and GtkTextBuffer found from the new GtkTextView. GtkTextView is connected to GtkScrolledWindow as a child. Each file gets their own copy of these widgets, so they are created inside the for-loop.
- 39-40 inserts the contents of the file into GtkTextBuffer and frees the memory pointed by `contents`.
- 41-43: Gets the filename and creates GtkLabel with the filename and then frees `filename`.
- 44-45: If `filename` is NULL, creates GtkLabel with the empty string.
- 46: Appends GtkScrolledWindow as a page, with the tab GtkLabel, to GtkNotebook. At this time a GtkNoteBookPage widget is created automatically. The GtkScrolledWindow widget is connected to the GtkNotebookPage. Therefore, the structure is like this:

  ```
  GtkNotebook -- GtkNotebookPage -- GtkScrolledWindow
  ```

- 47: Gets GtkNotebookPage widget and sets `nbp` to point to this GtkNotebookPage.
- 48: GtkNotebookPage has a property set called "tab-expand". If it is set to TRUE then the tab expands horizontally as long as possible. If it is FALSE, then the width of the tab is determined by

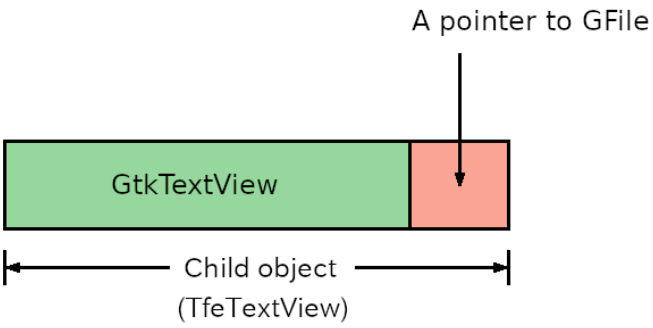

Figure 11: Child object of GtkTextView

the size of the label. `g_object_set` is a general function to set properties in objects. See GObject API Reference, g_object_set.

- 49-51: If the file cannot be read, "No such file" message is displayed and the `filename` buffer is freed.
- 52-53: If `filename` is NULL, the "No valid file is given" message is outputted.
- 55-58: If at least one file was read, then the number of GtkNotebookPage is greater than zero. If it's true, it shows the window. If it's false, it destroys the window, which causes the program to quit.

8 Defining a Child object

8.1 A Very Simple Editor

In the previous section we made a very simple file viewer. Now we go on to rewrite it and turn it into very simple editor. Its source file is in tfe1.c (text file editor 1).

GtkTextView has a feature for editing multiple lines. Therefore, we don't need to write the program from scratch, we just add two things to the file viewer:

- Memory to store a pointer to the GFile instance.
- A function to write the file.

There are a couple of ways to store the details of GFile.

- Use global variables; or
- Make a child object, which can extend the instance memory for the GFile object.

Using global variables is easy to implement. Define a sufficient size array of pointers to GFile. For example,

```
GFile *f[20];
```

The variable `f[i]` corresponds to the file associated to the i-th GtkNotebookPage. There are however two problems with this. The first concerns the size of the array. If a user gives too many arguments (more than 20 in the example above), it is impossible to store the additional pointers to the GFile instances. The second is the increasing difficulty for maintenance of the program. We have a small program so far, but however, if you continue developing it, the size of the program will grow. Generally speaking, the bigger the program size, the more difficult it is to keep track of and maintain global variables. Global variables can be used and changed anywhere throughout the entire program.

Making a child object is a good idea in terms of maintenance. One thing you need to be careful of is the difference between "child object" and "child widget". Here we are describing a "child object". A child object includes, and expands on its parent object, as a child object derives everything from the parent object.

We will define TfeTextView as a child object of GtkTextView. It has everything that GtkTextView has. Specifically, TfeTextView has a GtkTextbuffer which corresponds to the GtkTextView inside TfeTextView. The additional important thing is that TfeTextView can also keep an additional pointer to GFile.

In general, this is how GObjects work. Understanding the general theory about Gobject's is difficult, particularly for beginners. So, I will just show you the way how to write the code and avoid the theoretical side. If you want to know about GObject system, refer to another tutorial.

8.2 How to Define a Child Object of GtkTextView

Let's define the TfeTextView object, which is a child object of GtkTextView. First, look at the program below.

```
#define TFE_TYPE_TEXT_VIEW tfe_text_view_get_type ()
G_DECLARE_FINAL_TYPE (TfeTextView, tfe_text_view, TFE, TEXT_VIEW, GtkTextView)

struct _TfeTextView
{
  GtkTextView parent;
  GFile *file;
};

G_DEFINE_TYPE (TfeTextView, tfe_text_view, GTK_TYPE_TEXT_VIEW);

static void
tfe_text_view_init (TfeTextView *tv) {
}

static void
tfe_text_view_class_init (TfeTextViewClass *class) {
}

void
tfe_text_view_set_file (TfeTextView *tv, GFile *f) {
  tv -> file = f;
}

GFile *
tfe_text_view_get_file (TfeTextView *tv) {
  return tv -> file;
}

GtkWidget *
tfe_text_view_new (void) {
  return GTK_WIDGET (g_object_new (TFE_TYPE_TEXT_VIEW, NULL));
}
```

If you are curious about the background theory of this program, that's good, because knowing the theory is very important if you want to program GTK applications. Look at GObject API Reference. All you need is described there, or refer to GObject tutorial. It's a tough journey especially for beginners so for now, you don't need to know about this difficult theory. It is enough to just remember the instructions below.

- TfeTextView is divided into two parts. Tfe and TextView. Tfe is called the prefix, namespace or module. TextView is called the object.
- There are three differnet identifier patterns. TfeTextView (camel case), tfe_text_view (this is used to write functions) and TFE_TEXT_VIEW (This is used to cast a pointer to point TfeTextView type).
- First, define TFE_TYPE_TEXT_VIEW macro as tfe_text_view_get_type (). The name is always (prefix)_TYPE_(object) and the letters are upper case. And the replacement text is always (prefix)_(object)_get_type () and the letters are lower case.
- Next, use G_DECLARE_FINAL_TYPE macro. The arguments are the child object name in camel case, lower case with underscore, prefix (upper case), object (upper case with underscore) and parent object name (camel case).
- Declare the structure _TfeTextView. The underscore is necessary. The first member is the parent object. Notice this is not a pointer but the object itself. The second member and after are members of the child object. TfeTextView structure has a pointer to a GFile instance as a member.
- Use G_DEFINE_TYPE macro. The arguments are the child object name in camel case, lower case with underscore and parent object type (prefix)_TYPE_(module).
- Define instance init function (tfe_text_view_init). Usually you don't need to do anything.
- Define class init function (tfe_text_view_class_init). You don't need to do anything in this object.
- Write function codes you want to add (tfe_text_view_set_file and tfe_text_view_get_file). `tv` is a pointer to the TfeTextView object instance which is a C-structure declared with the tag _TfeTextView.

So, the structure has a member `file` as a pointer to a GFile instance. `tv->file = f` is an assignment of `f` to a member `file` of the structure pointed by `tv`. This is an example how to use the extended memory in a child widget.

- Write a function to create an instance. Its name is (prefix)_(object)_new. If the parent object function needs parameters, this function also need them. You sometimes might want to add some parameters. It's your choice. Use g_object_new function to create the instance. The arguments are (prefix)_TYPE_(object), a list to initialize properties and NULL. In this code no property needs to be initialized. And the return value is casted to GtkWidget.

This program is not perfect. It has some problems. It will be modified later.

8.3 Close-request signal

Imagine that you are using this editor. First, you run the editor with arguments. The arguments are filenames. The editor reads the files and shows the window with the text of files in it. Then you edit the text. After you finish editing, you exit the editor. The editor updates files just before the window closes.

GtkWindow emits the "close-request" signal before it closes. We connect the signal and the handler `before_close`. A handler is a C function. When a function is connected to a certain signal, we call it a handler. The function `before_close` is invoked when the signal "close-request" is emitted.

```
g_signal_connect (win, "close-request", G_CALLBACK (before_close), NULL);
```

The argument `win` is a GtkApplicationWindow, in which the signal "close-request" is defined, and `before_close` is the handler. `G_CALLBACK` cast is necessary for the handler. The program of `before_close` is as follows.

```
1  static gboolean
2  before_close (GtkWindow *win, gpointer user_data) {
3    GtkWidget *nb = GTK_WIDGET (user_data);
4    GtkWidget *scr;
5    GtkWidget *tv;
6    GFile *file;
7    char *pathname;
8    GtkTextBuffer *tb;
9    GtkTextIter start_iter;
10   GtkTextIter end_iter;
11   char *contents;
12   unsigned int n;
13   unsigned int i;
14
15   n = gtk_notebook_get_n_pages (GTK_NOTEBOOK (nb));
16   for (i = 0; i < n; ++i) {
17     scr = gtk_notebook_get_nth_page (GTK_NOTEBOOK (nb), i);
18     tv = gtk_scrolled_window_get_child (GTK_SCROLLED_WINDOW (scr));
19     file = tfe_text_view_get_file (TFE_TEXT_VIEW (tv));
20     tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
21     gtk_text_buffer_get_bounds (tb, &start_iter, &end_iter);
22     contents = gtk_text_buffer_get_text (tb, &start_iter, &end_iter, FALSE);
23     if (! g_file_replace_contents (file, contents, strlen (contents), NULL, TRUE,
           G_FILE_CREATE_NONE, NULL, NULL, NULL)) {
24       pathname = g_file_get_path (file);
25       g_print ("ERROR : Can't save %s.", pathname);
26       g_free (pathname);
27     }
28     g_free (contents);
29   }
30   return FALSE;
31 }
```

The numbers on the left of items are line numbers in the source code.

- 15: Gets the number of pages `nb` has.
- 16-29: For loop with regard to the index to each pages.

- 17-19: Gets GtkScrolledWindow, TfeTextView and a pointer to GFile. The pointer was stored when `app_open` handler had run. It will be shown later.
- 20-22: Gets GtkTextBuffer and contents. `start_iter` and `end_iter` are iterators of the buffer. I don't want to explain them now because it would take a lot of time. Just remember these lines for the present.
- 23-27: Writes the contents to the file. If it fails, it outputs an error message.
- 28: Frees `contents`.

8.4 Source code of tfe1.c

The following is the complete source code of `tfe1.c`.

```
1  #include <gtk/gtk.h>
2
3  /* Define TfeTextView Widget which is the child object of GtkTextView */
4
5  #define TFE_TYPE_TEXT_VIEW tfe_text_view_get_type ()
6  G_DECLARE_FINAL_TYPE (TfeTextView, tfe_text_view, TFE, TEXT_VIEW, GtkTextView)
7
8  struct _TfeTextView
9  {
10   GtkTextView parent;
11   GFile *file;
12 };
13
14 G_DEFINE_TYPE (TfeTextView, tfe_text_view, GTK_TYPE_TEXT_VIEW);
15
16 static void
17 tfe_text_view_init (TfeTextView *tv) {
18 }
19
20 static void
21 tfe_text_view_class_init (TfeTextViewClass *class) {
22 }
23
24 void
25 tfe_text_view_set_file (TfeTextView *tv, GFile *f) {
26   tv -> file = f;
27 }
28
29 GFile *
30 tfe_text_view_get_file (TfeTextView *tv) {
31   return tv -> file;
32 }
33
34 GtkWidget *
35 tfe_text_view_new (void) {
36   return GTK_WIDGET (g_object_new (TFE_TYPE_TEXT_VIEW, NULL));
37 }
38
39 /* ---------- end of the definition of TfeTextView ---------- */
40
41 static gboolean
42 before_close (GtkWindow *win, gpointer user_data) {
43   GtkWidget *nb = GTK_WIDGET (user_data);
44   GtkWidget *scr;
45   GtkWidget *tv;
46   GFile *file;
47   char *pathname;
48   GtkTextBuffer *tb;
49   GtkTextIter start_iter;
50   GtkTextIter end_iter;
51   char *contents;
52   unsigned int n;
```

```
53    unsigned int i;
54
55    n = gtk_notebook_get_n_pages (GTK_NOTEBOOK (nb));
56    for (i = 0; i < n; ++i) {
57      scr = gtk_notebook_get_nth_page (GTK_NOTEBOOK (nb), i);
58      tv = gtk_scrolled_window_get_child (GTK_SCROLLED_WINDOW (scr));
59      file = tfe_text_view_get_file (TFE_TEXT_VIEW (tv));
60      tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
61      gtk_text_buffer_get_bounds (tb, &start_iter, &end_iter);
62      contents = gtk_text_buffer_get_text (tb, &start_iter, &end_iter, FALSE);
63      if (! g_file_replace_contents (file, contents, strlen (contents), NULL, TRUE,
            G_FILE_CREATE_NONE, NULL, NULL, NULL)) {
64        pathname = g_file_get_path (file);
65        g_print ("ERROR : Can't save %s.", pathname);
66        g_free (pathname);
67      }
68      g_free (contents);
69    }
70    return FALSE;
71  }
72
73  static void
74  app_activate (GApplication *app, gpointer user_data) {
75    g_print ("You need to give filenames as arguments.\n");
76  }
77
78  static void
79  app_open (GApplication *app, GFile ** files, gint n_files, gchar *hint, gpointer
      user_data) {
80    GtkWidget *win;
81    GtkWidget *nb;
82    GtkWidget *lab;
83    GtkNotebookPage *nbp;
84    GtkWidget *scr;
85    GtkWidget *tv;
86    GtkTextBuffer *tb;
87    char *contents;
88    gsize length;
89    char *filename;
90    int i;
91
92    win = gtk_application_window_new (GTK_APPLICATION (app));
93    gtk_window_set_title (GTK_WINDOW (win), "file editor");
94    gtk_window_maximize (GTK_WINDOW (win));
95
96    nb = gtk_notebook_new ();
97    gtk_window_set_child (GTK_WINDOW (win), nb);
98
99    for (i = 0; i < n_files; i++) {
100     if (g_file_load_contents (files[i], NULL, &contents, &length, NULL, NULL)) {
101       scr = gtk_scrolled_window_new ();
102       tv = tfe_text_view_new ();
103       tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
104       gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
105       gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);
106
107       tfe_text_view_set_file (TFE_TEXT_VIEW (tv),  g_file_dup (files[i]));
108       gtk_text_buffer_set_text (tb, contents, length);
109       g_free (contents);
110       filename = g_file_get_basename (files[i]);
111       lab = gtk_label_new (filename);
112       gtk_notebook_append_page (GTK_NOTEBOOK (nb), scr, lab);
113       nbp = gtk_notebook_get_page (GTK_NOTEBOOK (nb), scr);
114       g_object_set (nbp, "tab-expand", TRUE, NULL);
```

```
115         g_free (filename);
116       } else if ((filename = g_file_get_path (files[i])) != NULL) {
117           g_print ("No such file: %s.\n", filename);
118           g_free (filename);
119       } else
120           g_print ("No valid file is given\n");
121     }
122     if (gtk_notebook_get_n_pages (GTK_NOTEBOOK (nb)) > 0) {
123       g_signal_connect (win, "close-request", G_CALLBACK (before_close), nb);
124       gtk_widget_show (win);
125     } else
126       gtk_window_destroy (GTK_WINDOW (win));
127 }
128
129 int
130 main (int argc, char **argv) {
131   GtkApplication *app;
132   int stat;
133
134   app = gtk_application_new ("com.github.ToshioCP.tfe1", G_APPLICATION_HANDLES_OPEN);
135   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
136   g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);
137   stat =g_application_run (G_APPLICATION (app), argc, argv);
138   g_object_unref (app);
139   return stat;
140 }
```

- 107: Sets the pointer to GFile into TfeTextView. `files[i]` is a pointer to GFile structure. It will be freed by the system. So you need to copy it. `g_file_dup` duplicates the given GFile structure.
- 123: Connects "close-request" signal and `before_close` handler. The fourth argument is called user data and it is given to the signal handler. So, `nb` is given to `before_close` as the second argument.

Now compile and run it. There's a sample file in the directory `tfe`. Type `./a.out taketori.txt`. Modify the contents and close the window. Make sure that the file is modified.

Now we got a very simple editor. It's not smart. We need more features like open, save, saveas, change font and so on. We will add them in the next section and after.

9 The User Interface (UI) file and GtkBuilder

9.1 New, Open and Save button

In the last section we made the almost simplest editor possible. It reads files in the `app_open` function at start-up and writes them out when closing the window. It works but is not very good. It would be better if we had "New", "Open", "Save" and "Close" buttons. This section describes how to put those buttons into the window. Signals and handlers will be explained later.

The screenshot above shows the layout. The function `app_open` in the source code `tfe2.c` is as follows.

```
1  static void
2  app_open (GApplication *app, GFile ** files, gint n_files, gchar *hint, gpointer
       user_data) {
3    GtkWidget *win;
4    GtkWidget *nb;
5    GtkWidget *lab;
6    GtkNotebookPage *nbp;
7    GtkWidget *scr;
8    GtkWidget *tv;
9    GtkTextBuffer *tb;
10   char *contents;
11   gsize length;
12   char *filename;
13   int i;
14
```

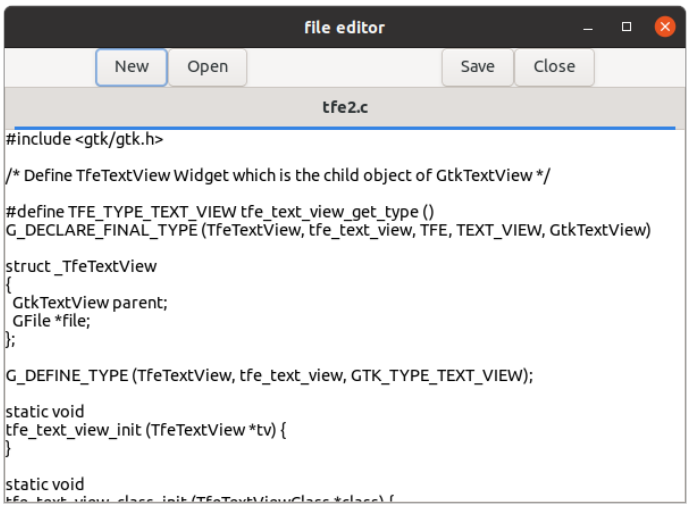

Figure 12: Screenshot of the file editor

```
15    GtkWidget *boxv;
16    GtkWidget *boxh;
17    GtkWidget *dmy1;
18    GtkWidget *dmy2;
19    GtkWidget *dmy3;
20    GtkWidget *btnn; /* button for new */
21    GtkWidget *btno; /* button for open */
22    GtkWidget *btns; /* button for save */
23    GtkWidget *btnc; /* button for close */
24
25    win = gtk_application_window_new (GTK_APPLICATION (app));
26    gtk_window_set_title (GTK_WINDOW (win), "file editor");
27    gtk_window_set_default_size (GTK_WINDOW (win), 600, 400);
28
29    boxv = gtk_box_new (GTK_ORIENTATION_VERTICAL, 0);
30    gtk_window_set_child (GTK_WINDOW (win), boxv);
31
32    boxh = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
33    gtk_box_append (GTK_BOX (boxv), boxh);
34
35    dmy1 = gtk_label_new(NULL); /* dummy label for left space */
36    gtk_label_set_width_chars (GTK_LABEL (dmy1), 10);
37    dmy2 = gtk_label_new(NULL); /* dummy label for center space */
38    gtk_widget_set_hexpand (dmy2, TRUE);
39    dmy3 = gtk_label_new(NULL); /* dummy label for right space */
40    gtk_label_set_width_chars (GTK_LABEL (dmy3), 10);
41    btnn = gtk_button_new_with_label ("New");
42    btno = gtk_button_new_with_label ("Open");
43    btns = gtk_button_new_with_label ("Save");
44    btnc = gtk_button_new_with_label ("Close");
45
46    gtk_box_append (GTK_BOX (boxh), dmy1);
47    gtk_box_append (GTK_BOX (boxh), btnn);
48    gtk_box_append (GTK_BOX (boxh), btno);
49    gtk_box_append (GTK_BOX (boxh), dmy2);
50    gtk_box_append (GTK_BOX (boxh), btns);
51    gtk_box_append (GTK_BOX (boxh), btnc);
52    gtk_box_append (GTK_BOX (boxh), dmy3);
53
54    nb = gtk_notebook_new ();
55    gtk_widget_set_hexpand (nb, TRUE);
56    gtk_widget_set_vexpand (nb, TRUE);
```

```
57    gtk_box_append (GTK_BOX (boxv), nb);
58
59    for (i = 0; i < n_files; i++) {
60      if (g_file_load_contents (files[i], NULL, &contents, &length, NULL, NULL)) {
61        scr = gtk_scrolled_window_new ();
62        tv = tfe_text_view_new ();
63        tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
64        gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
65        gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);
66
67        tfe_text_view_set_file (TFE_TEXT_VIEW (tv),  g_file_dup (files[i]));
68        gtk_text_buffer_set_text (tb, contents, length);
69        g_free (contents);
70        filename = g_file_get_basename (files[i]);
71        lab = gtk_label_new (filename);
72        gtk_notebook_append_page (GTK_NOTEBOOK (nb), scr, lab);
73        nbp = gtk_notebook_get_page (GTK_NOTEBOOK (nb), scr);
74        g_object_set (nbp, "tab-expand", TRUE, NULL);
75        g_free (filename);
76      } else if ((filename = g_file_get_path (files[i])) != NULL) {
77          g_print ("No such file: %s.\n", filename);
78          g_free (filename);
79      } else
80          g_print ("No valid file is given\n");
81    }
82    if (gtk_notebook_get_n_pages (GTK_NOTEBOOK (nb)) > 0) {
83      gtk_widget_show (win);
84    } else
85      gtk_window_destroy (GTK_WINDOW (win));
86  }
```

The aim is to build the widgets of the main application window.

- 25-27: Creates a GtkApplicationWindow instance and sets the title and default size.
- 29-30: Creates a GtkBox instance `boxv`. It is a vertical box and a child of GtkApplicationWindow. It has two children. The first child is a horizontal box. The second child is a GtkNotebook.
- 32-33: Creates a GtkBox instance `boxh` and appends it to `boxv` as a first child.
- 35-40: Creates three dummy labels. The labels `dmy1` and `dmy3` has a character width of ten. The other label `dmy2` has hexpand property which is set to be TRUE. This makes the label expands horizontally as long as possible.
- 41-44: Creates four buttons.
- 46-52: Appends these GtkLabel and GtkButton to `boxh`.
- 54-57: Creates a GtkNotebook instance and sets hexpand and vexpand properties TRUE. This makes it expand horizontally and vertically as big as possible. It is appended to `boxv` as the second child.

The number of lines to build the widgets is 33(=57-25+1). We also needed many additional variables (`boxv`, `boxh`, `dmy1`, ...), most of which weren't necessary, except for building the widgets. Are there any good solution to reduce these work?

Gtk provides GtkBuilder. It reads user interface (UI) data and builds a window. It reduces this cumbersome work.

9.2 The UI File

First, let's look at the UI file `tfe3.ui` that is used to define the widget structure.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <interface>
3    <object class="GtkApplicationWindow" id="win">
4      <property name="title">file editor</property>
5      <property name="default-width">600</property>
6      <property name="default-height">400</property>
7      <child>
8        <object class="GtkBox" id="boxv">
```

```
 9          <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
10          <child>
11            <object class="GtkBox" id="boxh">
12              <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
13              <child>
14                <object class="GtkLabel" id="dmy1">
15                  <property name="width-chars">10</property>
16                </object>
17              </child>
18              <child>
19                <object class="GtkButton" id="btnn">
20                  <property name="label">New</property>
21                </object>
22              </child>
23              <child>
24                <object class="GtkButton" id="btno">
25                  <property name="label">Open</property>
26                </object>
27              </child>
28              <child>
29                <object class="GtkLabel" id="dmy2">
30                  <property name="hexpand">TRUE</property>
31                </object>
32              </child>
33              <child>
34                <object class="GtkButton" id="btns">
35                  <property name="label">Save</property>
36                </object>
37              </child>
38              <child>
39                <object class="GtkButton" id="btnc">
40                  <property name="label">Close</property>
41                </object>
42              </child>
43              <child>
44                <object class="GtkLabel" id="dmy3">
45                  <property name="width-chars">10</property>
46                </object>
47              </child>
48            </object>
49          </child>
50          <child>
51            <object class="GtkNotebook" id="nb">
52              <property name="hexpand">TRUE</property>
53              <property name="vexpand">TRUE</property>
54            </object>
55          </child>
56        </object>
57      </child>
58    </object>
59  </interface>
```

The structure of this file is XML. Constructs that begin with `<` and end with `>` are called tags. There are two types of tags, the start tag and the end tag. For example, `<interface>` is a start tag and `</interface>` is an end tag. The UI file begins and ends with interface tags. Some tags, for example object tags, can have a class and id attributes in their start tag.

- 1: The first line is XML declaration. It specifies that the version of XML is 1.0 and the encoding is UTF-8. Even if the line is left out, GtkBuilder builds objects from the ui file. But ui files must use UTF-8 encoding, or GtkBuilder can't recognize it and a fatal error occurs.
- 3-6: An object with `GtkApplicationWindow` class and `win` id is defined. This is the top level window. And the three properties of the window are defined. `title` property is "file editor", `default-width` property is 600 and `default-height` property is 400.
- 7: child tag means a child of the widget above. For example, line 7 tells us that GtkBox object which

id is “boxv” is a child widget of `win`.

Compare this ui file and the lines 25-57 in the source code of `app_open` function. Those two describe the same structure of widgets.

You can check the ui file with `gtk4-builder-tool`.

- `gtk4-builder-tool validate <ui file name>` validates the ui file. If the ui file includes some syntactical error, `gtk4-builder-tool` prints the error.
- `gtk4-builder-tool simplify <ui file name>` simplifies the ui file and prints the result. If `--replace` option is given, it replaces the ui file with the simplified one. If the ui file specifies a value of property but it is default, then it will be removed. And some values are simplified. For example, “TRUE”and “FALSE” becomes “1” and “0” respectively. However, “TRUE” or “FALSE” is better for maintenance.

It is a good idea to check your ui file before compiling.

9.3 GtkBuilder

GtkBuilder builds widgets based on the ui file.

```
GtkBuilder *build;

build = gtk_builder_new_from_file ("tfe3.ui");
win = GTK_WIDGET (gtk_builder_get_object (build, "win"));
gtk_window_set_application (GTK_WINDOW (win), GTK_APPLICATION (app));
nb = GTK_WIDGET (gtk_builder_get_object (build, "nb"));
```

The function `gtk_builder_new_from_file` reads the file given as an argument. Then, it builds the widgets and creates GtkBuilder object. The function `gtk_builder_get_object (build, "win")` returns the pointer to the widget `win`, which is the id in the ui file. All the widgets are connected based on the parent-children relationship described in the ui file. We only need `win` and `nb` for the program after this, so we don’t need to take out any other widgets. This reduces lines in the C source file.

```
$ cd tfe; diff tfe2.c tfe3.c
58a59
>   GtkBuilder *build;
60,103c61,65
<   GtkWidget *boxv;
<   GtkWidget *boxh;
<   GtkWidget *dmy1;
<   GtkWidget *dmy2;
<   GtkWidget *dmy3;
<   GtkWidget *btnn; /* button for new */
<   GtkWidget *btno; /* button for open */
<   GtkWidget *btns; /* button for save */
<   GtkWidget *btnc; /* button for close */
<
<   win = gtk_application_window_new (GTK_APPLICATION (app));
<   gtk_window_set_title (GTK_WINDOW (win), "file editor");
<   gtk_window_set_default_size (GTK_WINDOW (win), 600, 400);
<
<   boxv = gtk_box_new (GTK_ORIENTATION_VERTICAL, 0);
<   gtk_window_set_child (GTK_WINDOW (win), boxv);
<
<   boxh = gtk_box_new (GTK_ORIENTATION_HORIZONTAL, 0);
<   gtk_box_append (GTK_BOX (boxv), boxh);
<
<   dmy1 = gtk_label_new(NULL); /* dummy label for left space */
<   gtk_label_set_width_chars (GTK_LABEL (dmy1), 10);
<   dmy2 = gtk_label_new(NULL); /* dummy label for center space */
<   gtk_widget_set_hexpand (dmy2, TRUE);
<   dmy3 = gtk_label_new(NULL); /* dummy label for right space */
<   gtk_label_set_width_chars (GTK_LABEL (dmy3), 10);
<   btnn = gtk_button_new_with_label ("New");
<   btno = gtk_button_new_with_label ("Open");
```

```
<   btns = gtk_button_new_with_label ("Save");
<   btnc = gtk_button_new_with_label ("Close");
<
<   gtk_box_append (GTK_BOX (boxh), dmy1);
<   gtk_box_append (GTK_BOX (boxh), btnn);
<   gtk_box_append (GTK_BOX (boxh), btno);
<   gtk_box_append (GTK_BOX (boxh), dmy2);
<   gtk_box_append (GTK_BOX (boxh), btns);
<   gtk_box_append (GTK_BOX (boxh), btnc);
<   gtk_box_append (GTK_BOX (boxh), dmy3);
<
<   nb = gtk_notebook_new ();
<   gtk_widget_set_hexpand (nb, TRUE);
<   gtk_widget_set_vexpand (nb, TRUE);
<   gtk_box_append (GTK_BOX (boxv), nb);
<
---
>   build = gtk_builder_new_from_file ("tfe3.ui");
>   win = GTK_WIDGET (gtk_builder_get_object (build, "win"));
>   gtk_window_set_application (GTK_WINDOW (win), GTK_APPLICATION (app));
>   nb = GTK_WIDGET (gtk_builder_get_object (build, "nb"));
>   g_object_unref(build);
138c100
<   app = gtk_application_new ("com.github.ToshioCP.tfe2",
    G_APPLICATION_HANDLES_OPEN);
---
>   app = gtk_application_new ("com.github.ToshioCP.tfe3",
    G_APPLICATION_HANDLES_OPEN);
```

`60,103c61,65` means 44 (=103-60+1) lines are changed to 5 (=65-61+1) lines. Therefore, 39 lines are reduced. Using ui file not only shortens C source files, but also makes the widgets' structure clear.

Now I'll show you `app_open` function in the C file `tfe3.c`.

```
1  static void
2  app_open (GApplication *app, GFile ** files, gint n_files, gchar *hint, gpointer
       user_data) {
3    GtkWidget *win;
4    GtkWidget *nb;
5    GtkWidget *lab;
6    GtkNotebookPage *nbp;
7    GtkWidget *scr;
8    GtkWidget *tv;
9    GtkTextBuffer *tb;
10   char *contents;
11   gsize length;
12   char *filename;
13   int i;
14   GtkBuilder *build;
15
16   build = gtk_builder_new_from_file ("tfe3.ui");
17   win = GTK_WIDGET (gtk_builder_get_object (build, "win"));
18   gtk_window_set_application (GTK_WINDOW (win), GTK_APPLICATION (app));
19   nb = GTK_WIDGET (gtk_builder_get_object (build, "nb"));
20   g_object_unref(build);
21   for (i = 0; i < n_files; i++) {
22     if (g_file_load_contents (files[i], NULL, &contents, &length, NULL, NULL)) {
23       scr = gtk_scrolled_window_new ();
24       tv = tfe_text_view_new ();
25       tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
26       gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
27       gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);
28
29       tfe_text_view_set_file (TFE_TEXT_VIEW (tv),  g_file_dup (files[i]));
30       gtk_text_buffer_set_text (tb, contents, length);
```

```
31        g_free (contents);
32        filename = g_file_get_basename (files[i]);
33        lab = gtk_label_new (filename);
34        gtk_notebook_append_page (GTK_NOTEBOOK (nb), scr, lab);
35        nbp = gtk_notebook_get_page (GTK_NOTEBOOK (nb), scr);
36        g_object_set (nbp, "tab-expand", TRUE, NULL);
37        g_free (filename);
38      } else if ((filename = g_file_get_path (files[i])) != NULL) {
39          g_print ("No such file: %s.\n", filename);
40          g_free (filename);
41      } else
42          g_print ("No valid file is given\n");
43    }
44    if (gtk_notebook_get_n_pages (GTK_NOTEBOOK (nb)) > 0) {
45      gtk_widget_show (win);
46    } else
47      gtk_window_destroy (GTK_WINDOW (win));
48  }
```

The whole source code of `tfe3.c` is stored in src/tfe directory. If you want to see it, click the link above.

9.3.1 Using ui string

GtkBuilder can build widgets using string. Use the function `gtk_builder_new_from_string` instead of `gtk_builder_new_from_file`.

```
char *uistring;

uistring =
"<interface>"
  "<object class="GtkApplicationWindow" id="win">"
    "<property name=\"title\">file editor</property>"
    "<property name=\"default-width\">600</property>"
    "<property name=\"default-height\">400</property>"
    "<child>"
      "<object class=\"GtkBox\" id=\"boxv\">"
        "<property name="orientation">GTK_ORIENTATION_VERTICAL</property>"
... ... ...
... ... ...
"</interface>";

build = gtk_builder_new_from_stringfile (uistring);
```

This method has an advantage and disadvantage. The advantage is that the ui string is written in the source code. So ui file is not necessary on runtime. The disadvantage is that writing C string is a bit bothersome because of the double quotes. If you want to use this method, you should write a script that transforms ui file into C-string.

- Add backslash before each double quote.
- Add double quotes at the left and right of the string in each line.

9.3.2 Using Gresource

Using Gresource is similar to using string. But Gresource is compressed binary data, not text data. And there's a compiler that compiles ui file into Gresource. It can compile not only text files but also binary files such as images, sounds and so on. And after compilation, it bundles them up into one Gresource object.

An xml file is necessary for the resource compiler `glib-compile-resources`. It describes resource files.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <gresources>
3   <gresource prefix="/com/github/ToshioCP/tfe3">
4     <file>tfe3.ui</file>
5   </gresource>
6 </gresources>
```

- 2: `gresources` tag can include multiple gresources (gresource tags). However, this xml has only one gresource.
- 3: The gresource has a prefix `/com/github/ToshioCP/tfe3`.
- 4: The gresource has `tfe3.ui`. And it is pointed by `/com/github/ToshioCP/tfe3/tfe3.ui` because it needs prefix. If you want to add more files, then insert them between line 4 and 5.

Save this xml text to `tfe3.gresource.xml`. The gresource compiler `glib-compile-resources` shows its usage with the argument `--help`.

```
$ LANG=C glib-compile-resources --help
Usage:
  glib-compile-resources [OPTION?] FILE

Compile a resource specification into a resource file.
Resource specification files have the extension .gresource.xml,
and the resource file have the extension called .gresource.

Help Options:
  -h, --help                    Show help options

Application Options:
  --version                     Show program version and exit
  --target=FILE                 Name of the output file
  --sourcedir=DIRECTORY         The directories to load files referenced in FILE from
      (default: current directory)
  --generate                    Generate output in the format selected for by the
      target filename extension
  --generate-header             Generate source header
  --generate-source             Generate source code used to link in the resource
      file into your code
  --generate-dependencies       Generate dependency list
  --dependency-file=FILE        Name of the dependency file to generate
  --generate-phony-targets      Include phony targets in the generated dependency file
  --manual-register             Don?t automatically create and register resource
  --internal                    Don?t export functions; declare them G_GNUC_INTERNAL
  --external-data               Don?t embed resource data in the C file; assume it's
      linked externally instead
  --c-name                      C identifier name used for the generated source code
  -C, --compiler                The target C compiler (default: the CC environment
      variable)
```

Now run the compiler.

```
$ glib-compile-resources tfe3.gresource.xml --target=resources.c --generate-source
```

Then a C source file `resources.c` is generated. Modify `tfe3.c` and save it as `tfe3_r.c`.

```
#include "resources.c"
... ... ...
... ... ...
build = gtk_builder_new_from_resource ("/com/github/ToshioCP/tfe3/tfe3.ui");
... ... ...
... ... ...
```

Then, compile and run it. The window appears and it is the same as the screenshot at the beginning of this page.

10 Build system

10.1 What do we need to think about to manage big source files?

We've compiled a small editor so far. But Some bad signs are beginning to appear.

- We've had only one C source file and put everything into it. We need to sort it out.

- There are two compilers, `gcc` and `glib-compile-resources`. We should control them by one building tool.

These ideas are useful to manage big source files.

10.2 Divide a C source file into two parts.

When you divide C source file into several parts, each file should contain only one thing. For example, our source has two things, the definition of TfeTextView and functions related to GtkApplication and GtkApplicationWindow. It is a good idea to separate them into two files, `tfetextview.c` and `tfe.c`.

- `tfetextview.c` includes the definition and functions of TfeTextView.
- `tfe.c` includes functions like `main`, `app_activate`, `app_open` and so on, which relate to GtkApplication and GtkApplicationWindow

Now we have three source files, `tfetextview.c`, `tfe.c` and `tfe3.ui`. The `3` of `tfe3.ui` is like a version number. Managing version with filenames is one possible idea but it may make bothersome problem. You need to rewrite filename in each version and it affects to contents of source files that refer to filenames. So, we should take `3` away from the filename.

In `tfe.c` the function `tfe_text_view_new` is invoked to create a TfeTextView instance. But it is defined in `tfetextview.c`, not `tfe.c`. The lack of the declaration (not definition) of `tfe_text_view_new` makes error when `tfe.c` is compiled. The declaration is necessary in `tfe.c`. Those public information is usually written in header files. It has `.h` suffix like `tfetextview.h` And header files are included by C source files. For example, `tfetextview.h` is included by `tfe.c`.

All the source files are listed below.

`tfetextview.h`

```
1  #include <gtk/gtk.h>
2
3  #define TFE_TYPE_TEXT_VIEW tfe_text_view_get_type ()
4  G_DECLARE_FINAL_TYPE (TfeTextView, tfe_text_view, TFE, TEXT_VIEW, GtkTextView)
5
6  void
7  tfe_text_view_set_file (TfeTextView *tv, GFile *f);
8
9  GFile *
10 tfe_text_view_get_file (TfeTextView *tv);
11
12 GtkWidget *
13 tfe_text_view_new (void);
```

`tfetextview.c`

```
1  #include <gtk/gtk.h>
2  #include "tfetextview.h"
3
4  struct _TfeTextView
5  {
6    GtkTextView parent;
7    GFile *file;
8  };
9
10 G_DEFINE_TYPE (TfeTextView, tfe_text_view, GTK_TYPE_TEXT_VIEW);
11
12 static void
13 tfe_text_view_init (TfeTextView *tv) {
14 }
15
16 static void
17 tfe_text_view_class_init (TfeTextViewClass *class) {
18 }
19
20 void
```

```
21 tfe_text_view_set_file (TfeTextView *tv, GFile *f) {
22   tv -> file = f;
23 }
24
25 GFile *
26 tfe_text_view_get_file (TfeTextView *tv) {
27   return tv -> file;
28 }
29
30 GtkWidget *
31 tfe_text_view_new (void) {
32   return GTK_WIDGET (g_object_new (TFE_TYPE_TEXT_VIEW, NULL));
33 }
```

tfe.c

```
1  #include <gtk/gtk.h>
2  #include "tfetextview.h"
3
4  static void
5  app_activate (GApplication *app, gpointer user_data) {
6    g_print ("You need a filename argument.\n");
7  }
8
9  static void
10 app_open (GApplication *app, GFile ** files, gint n_files, gchar *hint, gpointer
       user_data) {
11   GtkWidget *win;
12   GtkWidget *nb;
13   GtkWidget *lab;
14   GtkNotebookPage *nbp;
15   GtkWidget *scr;
16   GtkWidget *tv;
17   GtkTextBuffer *tb;
18   char *contents;
19   gsize length;
20   char *filename;
21   int i;
22   GtkBuilder *build;
23
24   build = gtk_builder_new_from_resource ("/com/github/ToshioCP/tfe3/tfe.ui");
25   win = GTK_WIDGET (gtk_builder_get_object (build, "win"));
26   gtk_window_set_application (GTK_WINDOW (win), GTK_APPLICATION (app));
27   nb = GTK_WIDGET (gtk_builder_get_object (build, "nb"));
28   g_object_unref (build);
29   for (i = 0; i < n_files; i++) {
30     if (g_file_load_contents (files[i], NULL, &contents, &length, NULL, NULL)) {
31       scr = gtk_scrolled_window_new ();
32       tv = tfe_text_view_new ();
33       tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
34       gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
35       gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);
36
37       tfe_text_view_set_file (TFE_TEXT_VIEW (tv),  g_file_dup (files[i]));
38       gtk_text_buffer_set_text (tb, contents, length);
39       g_free (contents);
40       filename = g_file_get_basename (files[i]);
41       lab = gtk_label_new (filename);
42       gtk_notebook_append_page (GTK_NOTEBOOK (nb), scr, lab);
43       nbp = gtk_notebook_get_page (GTK_NOTEBOOK (nb), scr);
44       g_object_set (nbp, "tab-expand", TRUE, NULL);
45       g_free (filename);
46     } else if ((filename = g_file_get_path (files[i])) != NULL) {
47         g_print ("No such file: %s.\n", filename);
48         g_free (filename);
```

```
49      } else
50          g_print ("No valid file is given\n");
51    }
52    if (gtk_notebook_get_n_pages (GTK_NOTEBOOK (nb)) > 0) {
53      gtk_widget_show (win);
54    } else
55      gtk_window_destroy (GTK_WINDOW (win));
56  }
57
58  int
59  main (int argc, char **argv) {
60    GtkApplication *app;
61    int stat;
62
63    app = gtk_application_new ("com.github.ToshioCP.tfe", G_APPLICATION_HANDLES_OPEN);
64    g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
65    g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);
66    stat =g_application_run (G_APPLICATION (app), argc, argv);
67    g_object_unref (app);
68    return stat;
69  }
```

The ui file `tfe.ui` is the same as `tfe3.ui` in the previous section.

`tfe.gresource.xml`

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <gresources>
3    <gresource prefix="/com/github/ToshioCP/tfe3">
4      <file>tfe.ui</file>
5    </gresource>
6  </gresources>
```

10.3 Make

Dividing a file makes it easy to maintain source files. But now we are faced with a new problem. The building step increases.

- Compiling the ui file `tfe.ui` into `resources.c`.
- Compiling `tfe.c` into `tfe.o` (object file).
- Compiling `tfetextview.c` into `tfetextview.o`.
- Compiling `resources.c` into `resources.o`.
- Linking all the object files into application `tfe`.

Now build tool is necessary to manage it. Make is one of the build tools. It was created in 1976. It is an old and widely used program.

Make analyzes Makefile and executes compilers. All instructions are written in Makefile.

```
sample.o: sample.c
    gcc -o sample.o sample.c
```

The sample of Malefile above consists of three elements, `sample.o`, `sample.c` and `gcc -o sample.o sample.c`.

- `sample.o` is called target.
- `sample.c` is prerequisite.
- `gcc -o sample.o sample.c` is recipe. Recipes follow tab characters, not spaces. (It is very important. Use tab not space, or make won't work as you expected).

The rule is:

If a prerequisite modified later than a target, then make executes the recipe.

In the example above, if `sample.c` is modified after the generation of `sample.o`, then make executes gcc and compile `sample.c` into `sample.o`. If the modification time of `sample.c` is older then the generation of `sample.o`, then no compiling is necessary, so make does nothing.

The Makefile for `tfe` is as follows.

```
all: tfe

tfe: tfe.o tfetextview.o resources.o
    gcc -o tfe tfe.o tfetextview.o resources.o `pkg-config --libs gtk4`

tfe.o: tfe.c tfetextview.h
    gcc -c -o tfe.o `pkg-config --cflags gtk4` tfe.c
tfetextview.o: tfetextview.c tfetextview.h
    gcc -c -o tfetextview.o `pkg-config --cflags gtk4` tfetextview.c
resources.o: resources.c
    gcc -c -o resources.o `pkg-config --cflags gtk4` resources.c

resources.c: tfe.gresource.xml tfe.ui
    glib-compile-resources tfe.gresource.xml --target=resources.c --generate-source

.Phony: clean

clean:
    rm -f tfe tfe.o tfetextview.o resources.o resources.c
```

You only need to type `make`.

```
$ make
gcc -c -o tfe.o `pkg-config --cflags gtk4` tfe.c
gcc -c -o tfetextview.o `pkg-config --cflags gtk4` tfetextview.c
glib-compile-resources tfe.gresource.xml --target=resources.c --generate-source
gcc -c -o resources.o `pkg-config --cflags gtk4` resources.c
gcc -o tfe tfe.o tfetextview.o resources.o `pkg-config --libs gtk4`
```

I used only very basic rules to write this Makefile. There are many more convenient methods to make it more compact. But it will be long to explain it. So I want to finish explaining make and move on to the next topic.

10.4 Rake

Rake is a similar program to make. It is written in Ruby code. If you don't use Ruby, you don't need to read this subsection. However, Ruby is really sophisticated and recommendable script language.

- Rakefile controls the behavior of `rake`.
- You can write any Ruby code in Rakefile.

Rake has task and file task, which is similar to target, prerequisite and recipe in make.

```
require 'rake/clean'

targetfile = "tfe"
srcfiles = FileList["tfe.c", "tfetextview.c", "resources.c"]
rscfile = srcfiles[2]
objfiles = srcfiles.gsub(/.c$/, '.o')

CLEAN.include(targetfile, objfiles, rscfile)

task default: targetfile

file targetfile => objfiles do |t|
  sh "gcc -o #{t.name} #{t.prerequisites.join(' ')} `pkg-config --libs gtk4`"
end

objfiles.each do |obj|
  src = obj.gsub(/.o$/,'.c')
  file obj => src do |t|
    sh "gcc -c -o #{t.name} `pkg-config --cflags gtk4` #{t.source}"
  end
```

```
21 end
22
23 file rscfile => ["tfe.gresource.xml", "tfe.ui"] do |t|
24   sh "glib-compile-resources #{t.prerequisites[0]} --target=#{t.name}
       --generate-source"
25 end
```

The contents of the `Rakefile` is almost same as the `Makefile` in the previous subsection.

- 3-6: Defines target file, source file and so on.
- 1, 8: Loads clean library. And defines CLEAN file list. The files included by CLEAN will be removed when `rake clean` is typed on the command line.
- 10: The default target depends on `targetfile`. The task `default` is the final goal of tasks.
- 12-14: `targetfile` depends on `objfiles`. The variable `t` is a task object.
  - `t.name` is a target name
  - `t.prerequisites` is an array of prerequisites.
  - `t.source` is the first element of prerequisites.
- `sh` is a method to give the following string to shell as an argument and executes the shell.
- 16-21: There's a loop by each element of the array of `objfiles`. Each object depends on corresponding source file.
- 23-25: Resource file depends on xml file and ui file.

Rakefile might seem to be difficult for beginners. But, you can use any Ruby syntax in Rakefile, so it is really flexible. If you practice Ruby and Rakefile, it will be highly productive tools.

10.5 Meson and ninja

Meson is one of the most popular building tool despite the developing version. And ninja is similar to make but much faster than make. Several years ago, most of the C developers used autotools and make. But now the situation has changed. Many developers are using meson and ninja now.

To use meson, you first need to write `meson.build` file.

```
1  project('tfe', 'c')
2
3  gtkdep = dependency('gtk4')
4
5  gnome=import('gnome')
6  resources = gnome.compile_resources('resources','tfe.gresource.xml')
7
8  sourcefiles=files('tfe.c', 'tfetextview.c')
9
10 executable('tfe', sourcefiles, resources, dependencies: gtkdep)
```

- 1: The function `project` defines things about the project. The first parameter is the name of the project and the second is the programming language.
- 2: `dependency` function defines a dependency that is taken by `pkg-config`. We put `gtk4` as an argument.
- 5: `import` function imports a module. In line 5, the gnome module is imported and assigned to the variable `gnome`. The gnome module provides helper tools to build GTK programs.
- 6: `.compile_resources` is a method of the gnome module and compiles files to resources under the instruction of xml file. In line 6, the resource filename is `resources`, which means `resources.c` and `resources.h`, and xml file is `tfe.gresource.xml`. This method generates C source file by default.
- 8: Defines source files.
- 10: Executable function generates a target file by compiling source files. The first parameter is the filename of the target. The following parameters are source files. The last parameter is an option `dependencies`. `gtkdep` is used in the compilation.

Now run meson and ninja.

```
$ meson _build
$ ninja -C _build
```

Then, the executable file `tfe` is generated under the directory `_build`.

```
$ _build/tfe tfe.c tfetextview.c
```

Then the window appears. And two notebook pages are in the window. One notebook is `tfe.c` and the other is `tfetextview.c`.

I've shown you three build tools. I think meson and ninja is the best choice for the present.

We divided a file into some categorized files and used a build tool. This method is used by many developers.

11 Initialization and destruction of instances

A new version of the text file editor (`tfe`) will be made in this section and the following four sections. It is `tfe5`. There are many changes from the prior version. All the sources are listed in Section 16. They are located in two directories, src/tfe5 and src/tfetextview.

11.1 Encapsulation

We've divided C source file into two parts. But it is not enough in terms of encapsulation.

- `tfe.c` includes everything other than TfeTextView. It should be divided at least into two parts, `tfeapplication.c` and `tfenotebook.c`.
- Header files also need to be organized.

However, first of all, I'd like to focus on the object TfeTextView. It is a child object of GtkTextView and has a new member `file` in it. The important thing is to manage the Gfile object pointed by `file`.

- What is necessary to GFile when creating (or initializing) TfeTextView?
- What is necessary to GFile when destructing TfeTextView?
- TfeTextView should read/write a file by itself or not?
- How it communicates with objects outside?

You need to know at least class, instance and signals before thinking about them. I will explain them in this section and the next section. After that I will explain:

- Organizing functions.
- How to use GtkFileChooserDialog

11.2 GObject and its children

GObject and its children are objects, which have both class and instance. First, think about instance of objects. Instance is structured memory. THe structure is described as C language structure. The following is a structure of TfeTextView.

```
/* This typedef statement is automatically generated by the macro
    G_DECLARE_FINAL_TYPE */
typedef struct _TfeTextView TfeTextView;

struct _TfeTextView {
  GtkTextView parent;
  GFile *file;
};
```

The members of the structure are:

- The type of `parent` is GtkTextView which is C structure. It is declared in `gtktextview.h`. GtkTextView is the parent of TfeTextView.
- `file` is a pointer to GFile. It can be NULL if no file corresponds to the TfeTextView object.

Notice the program above is the declaration of the structure, not the definition. So, no memories are allocated at this moment. They are to be allocated when `tfe_text_view_new` function is invoked. The memory allocated with `tfe_text_view_new` is an instance of TfeTextView object. Therefore, There can be multiple TfeTextView instances if `tfe_text_view_new` is called multiple times.

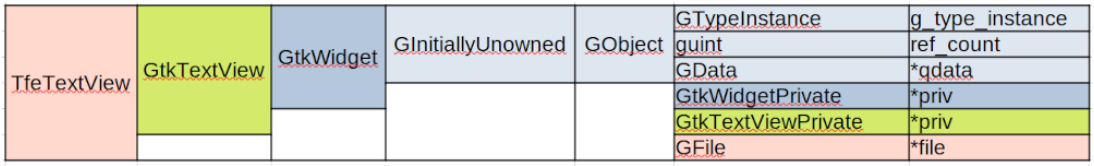

Figure 13: The structure of the instance TfeTextView

You can find the declaration of the ancestors of TfeTextView in the source files of GTK and GLib. The following is extracts from the source files (not exactly the same).

```
typedef struct _GObject GObject;
typedef struct _GObject GInitiallyUnowned;
struct  _GObject
{
  GTypeInstance  g_type_instance;
  volatile guint ref_count;
  GData         *qdata;
};

typedef struct _GtkWidget GtkWidget;
struct _GtkWidget
{
  GInitiallyUnowned parent_instance;
  GtkWidgetPrivate *priv;
};

typedef struct _GtkTextView GtkTextView;
struct _GtkTextView
{
  GtkWidget parent_instance;
  GtkTextViewPrivate *priv;
};
```

In each structure, its parent is declared at the top of the members. So, every ancestors is included in the child instance. This is very important. It guarantees a child widget to inherit all the features from ancestors. The structure of `TfeTextView` is like the following diagram.

11.3 Initialization of a TfeTextView instance

The function `tfe_text_view_new` creates a new TfeTextView instance.

```
1 GtkWidget *
2 tfe_text_view_new (void) {
3   return GTK_WIDGET (g_object_new (TFE_TYPE_TEXT_VIEW, NULL));
4 }
```

When this function is involed, a TfeTextView instance is created and initialized. The initialization process is as follows.

1. Initializes GObject (GInitiallyUnowned) part in TfeTextView instance.
2. Initializes GtkWidget part in TfeTextView instance.
3. Initializes GtkTextView part in TfeTextView instance.
4. Initializes TfeTextView part in TfeTextView instance.

The step one through three is done by `g_object_init`, `gtk_widget_init` and `gtk_text_view_init`. They are called by the system automatically and you don't need to care about them. Step four is done by the function `tfe_text_view_init` in `tfetextview.c`.

```
1 static void
2 tfe_text_view_init (TfeTextView *tv) {
3   tv->file = NULL;
4 }
```

This function just initializes `tv->file` to be `NULL`.

11.4 Functions and Classes

In Gtk, all objects derived from GObject have class and instance (except abstract object). An instance is memory of C structure, which is described in the previous two subsections. Each object can have more than one instance. Those instances have the same structure. An instance just keeps status of the instance. Therefore, it is insufficient to define its behavior. We need at least two things. One is functions and the other is class.

You've already seen many functions. For example, `tfe_text_view_new` is a function to create a TfeTextView instance. These functions are similar to public object methods in object oriented languages such as Java or Ruby. Functions are public, which means that they are expected to be used by other objects.

Class comprises mainly pointers to functions. Those functions are used by the object itself or its descendant objects. For example, GObject class is declared in `gobject.h` in GLib source files.

```
1  typedef struct _GObjectClass             GObjectClass;
2  typedef struct _GObjectClass             GInitiallyUnownedClass;
3
4  struct  _GObjectClass
5  {
6    GTypeClass   g_type_class;
7    /*< private >*/
8    GSList      *construct_properties;
9    /*< public >*/
10   /* seldom overridden */
11   GObject*   (*constructor)     (GType                  type,
12                                  guint                  n_construct_properties,
13                                  GObjectConstructParam *construct_properties);
14   /* overridable methods */
15   void       (*set_property)    (GObject        *object,
16                                  guint           property_id,
17                                  const GValue   *value,
18                                  GParamSpec     *pspec);
19   void       (*get_property)    (GObject        *object,
20                                  guint           property_id,
21                                  GValue         *value,
22                                  GParamSpec     *pspec);
23   void       (*dispose)         (GObject        *object);
24   void       (*finalize)        (GObject        *object);
25   /* seldom overridden */
26   void       (*dispatch_properties_changed) (GObject      *object,
27                                              guint         n_pspecs,
28                                              GParamSpec  **pspecs);
29   /* signals */
30   void      (*notify)           (GObject    *object,
31                                  GParamSpec *pspec);
32   /* called when done constructing */
33   void      (*constructed)      (GObject    *object);
34   /*< private >*/
35   gsize  flags;
36   /* padding */
37   gpointer pdummy[6];
38 };
```

I'd like to explain some of the members. There's a pointer to the function `dispose` in line 23.

```
void (*dispose) (GObject *object);
```

The declaration is a bit complicated. The asterisk before the identifier `dispose` means pointer. So, the pointer `dispose` points to a function which has one parameter, which points a GObject structure, and returns no value. In the same way, line 24 says `finalize` is a pointer to the function which has one parameter, which points a GObject structure, and returns no value.

```
void (*finalize) (GObject *object);
```

Look at the declaration of `_GObjectClass` so that you would find that most of the members are pointers to functions.

- 11: A function pointed by `constructor` is called when the instance is generated. It completes the initialization of the instance.
- 23: A function pointed by `dispose` is called when the instance destructs itself. Destruction process is divided into two phases. The first one is called disposing. In this phase, the instance releases all the references to other instances. The second phase is finalizing.
- 24: A function pointed by `finalize` finishes the destruction process.
- The other pointers point to functions which are called while the instance lives.

These functions are called class methods. The methods are open to its descendants. But not open to the objects which are not the descendants.

11.5 TfeTextView class

TfeTextView class is a structure and it includes all its ancestors' class in it.

```
typedef _TfeTextView TfeTextView;
struct _TfeTextView {
  GtkTextView parent;
  GFile *file;
};
```

TfeTextView structure has GtkTextView type as the first member. In the same way, GtkTextView has its parent type (GtkWidget) as the first member. GtkWidget has its parent type (GtkInitiallyUnowned) as the first member. The structure of GtkInitiallyUnowned is the same as GObject. Therefore, TFeTextView includes GObject, GtkWidget and GtkTextView in itself.

```
GObject -- GInitiallyUnowned -- GtkWidget -- GtkTextView -- TfeTextView
```

The following is extracts from the source files (not exactly the same).

```
1  struct _GtkWidgetClass
2  {
3    GInitiallyUnownedClass parent_class;
4
5    /*< public >*/
6
7    /* basics */
8    void (* show)                (GtkWidget        *widget);
9    void (* hide)                (GtkWidget        *widget);
10   void (* map)                 (GtkWidget        *widget);
11   void (* unmap)               (GtkWidget        *widget);
12   void (* realize)             (GtkWidget        *widget);
13   void (* unrealize)           (GtkWidget        *widget);
14   void (* root)                (GtkWidget        *widget);
15   void (* unroot)              (GtkWidget        *widget);
16   void (* size_allocate)       (GtkWidget           *widget,
17                                 int                  width,
18                                 int                  height,
19                                 int                  baseline);
20   void (* state_flags_changed) (GtkWidget        *widget,
21                                 GtkStateFlags     previous_state_flags);
22   void (* direction_changed)   (GtkWidget        *widget,
23                                 GtkTextDirection  previous_direction);
24
25   /* size requests */
26   GtkSizeRequestMode (* get_request_mode)              (GtkWidget      *widget);
27   void               (* measure) (GtkWidget      *widget,
28                                   GtkOrientation  orientation,
29                                   int             for_size,
30                                   int            *minimum,
```

```
                                    int            *natural,
                                    int            *minimum_baseline,
                                    int            *natural_baseline);

  /* Mnemonics */
  gboolean (* mnemonic_activate)        (GtkWidget           *widget,
                                         gboolean             group_cycling);

  /* explicit focus */
  gboolean (* grab_focus)               (GtkWidget           *widget);
  gboolean (* focus)                    (GtkWidget           *widget,
                                         GtkDirectionType     direction);
  void     (* set_focus_child)          (GtkWidget           *widget,
                                         GtkWidget           *child);

  /* keyboard navigation */
  void     (* move_focus)               (GtkWidget           *widget,
                                         GtkDirectionType     direction);
  gboolean (* keynav_failed)            (GtkWidget           *widget,
                                         GtkDirectionType     direction);

  gboolean     (* query_tooltip)      (GtkWidget  *widget,
                                       int         x,
                                       int         y,
                                       gboolean    keyboard_tooltip,
                                       GtkTooltip *tooltip);

  void         (* compute_expand)     (GtkWidget  *widget,
                                       gboolean   *hexpand_p,
                                       gboolean   *vexpand_p);

  void         (* css_changed)                (GtkWidget            *widget,
                                               GtkCssStyleChange    *change);

  void         (* system_setting_changed)     (GtkWidget            *widget,
                                               GtkSystemSetting      settings);

  void         (* snapshot)                   (GtkWidget            *widget,
                                               GtkSnapshot          *snapshot);

  gboolean     (* contains)                   (GtkWidget *widget,
                                               double     x,
                                               double     y);

  /*< private >*/

  GtkWidgetClassPrivate *priv;

  gpointer padding[8];
};

struct _GtkTextViewClass
{
  GtkWidgetClass parent_class;

  /*< public >*/

  void (* move_cursor)          (GtkTextView      *text_view,
                                 GtkMovementStep   step,
                                 int               count,
                                 gboolean          extend_selection);
  void (* set_anchor)           (GtkTextView      *text_view);
  void (* insert_at_cursor)     (GtkTextView      *text_view,
                                 const char       *str);
```

```
95    void (* delete_from_cursor)    (GtkTextView      *text_view,
96                                    GtkDeleteType     type,
97                                    int               count);
98    void (* backspace)             (GtkTextView      *text_view);
99    void (* cut_clipboard)         (GtkTextView      *text_view);
100   void (* copy_clipboard)        (GtkTextView      *text_view);
101   void (* paste_clipboard)       (GtkTextView      *text_view);
102   void (* toggle_overwrite)      (GtkTextView      *text_view);
103   GtkTextBuffer * (* create_buffer) (GtkTextView   *text_view);
104   void (* snapshot_layer)        (GtkTextView      *text_view,
105                                   GtkTextViewLayer  layer,
106                                   GtkSnapshot      *snapshot);
107   gboolean (* extend_selection)  (GtkTextView            *text_view,
108                                   GtkTextExtendSelection  granularity,
109                                   const GtkTextIter      *location,
110                                   GtkTextIter            *start,
111                                   GtkTextIter            *end);
112   void (* insert_emoji)          (GtkTextView      *text_view);
113
114   /*< private >*/
115
116   gpointer padding[8];
117 };
118
119 /* The following definition is generated by the macro G_DECLARE_FINAL_TYPE */
120 typedef struct {
121   GtkTextView parent_class;
122 } TfeTextViewClass;
```

- 120-122: This three lines are generated by the macro `G_DECLARE_FINAL_TYPE`. So, they are not written in either `tfe_text_view.h` or `tfe_text_view.c`.
- 3, 84, 121: Each derived class puts its parent class at the first member of its structure. It is the same as instance structures.
- Class members in ancestors are open to the descendant class. So, they can be changed in `tfe_text_view_class_init` function. For example, the `dispose` pointer in GObjectClass will be overridden later in `tfe_text_view_class_init`. (Override is an object oriented programming terminology. Override is rewriting ancestors' class methods in the descendant class.)
- Some class methods are often overridden. `set_property`, `get_property`, `dispose`, `finalize` and `constructed` are such methods.

TfeTextViewClass includes its ancestors' class in it. It is illustrated in the following diagram.

11.6 Destruction of TfeTextView

Every Object derived from GObject has a reference count. If an object A refers to an object B, then A keeps a pointer to B in A and at the same time increases the reference count of B by one with the function `g_object_ref (B)`. If A doesn't need B any longer, then A discards the pointer to B (usually it is done by assigning NULL to the pointer) and decreases the reference count of B by one with the function `g_object_unref (B)`.

If two objects A and B refer to C, then the reference count of C is two. If A no longer needs C, A discards the pointer to C and decreases the reference count in C by one. Now the reference count of C is one. In the same way, if B no longer needs C, B discards the pointer to C and decreases the reference count in C by one. At this moment, no object refers to C and the reference count of C is zero. This means C is no longer useful. Then C destructs itself and finally the memories allocated to C is freed.

The idea above is based on an assumption that an object referred by nothing has reference count of zero. When the reference count drops to zero, the object starts its destruction process. The destruction process is split into two phases: disposing and finalizing. In the disposing process, the object invokes the function pointed by `dispose` in its class to release all references to other objects. In the finalizing process, it invokes the function pointed by `finalize` in its class to complete the destruction process. These functions are also called handlers or methods. For example, dispose handler or dispose method.

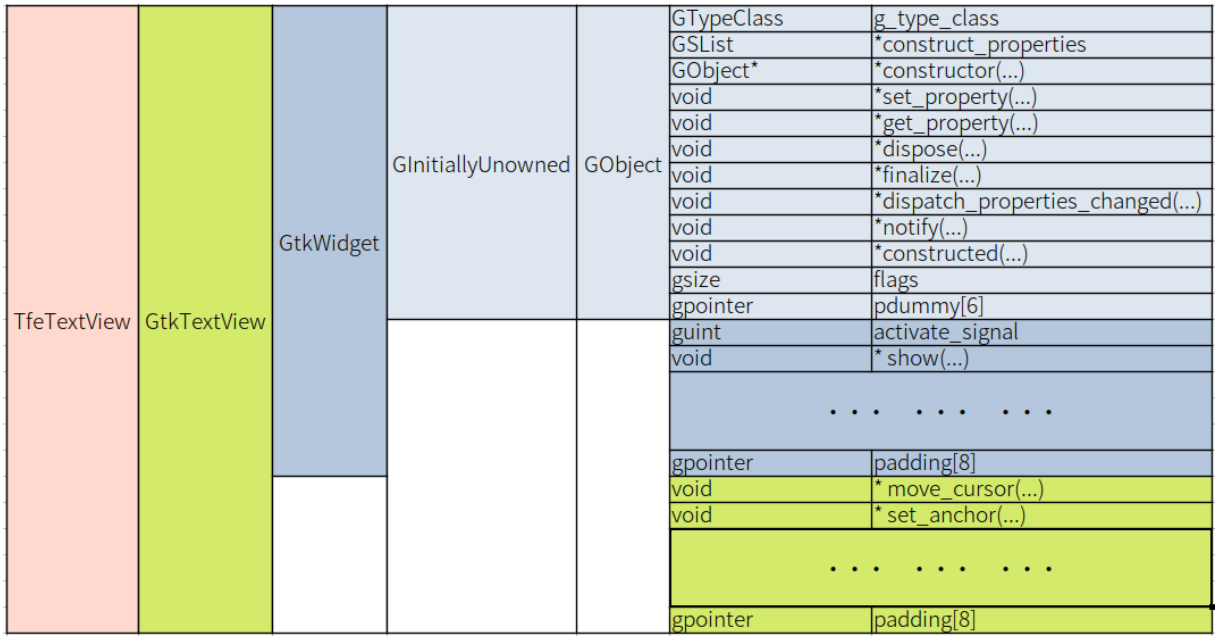

Figure 14: The structure of TfeTextView Class

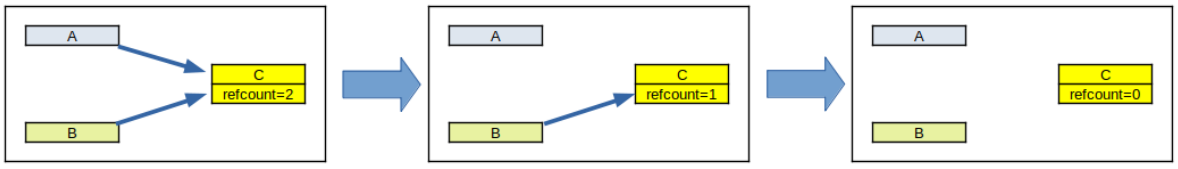

Figure 15: Reference count of B

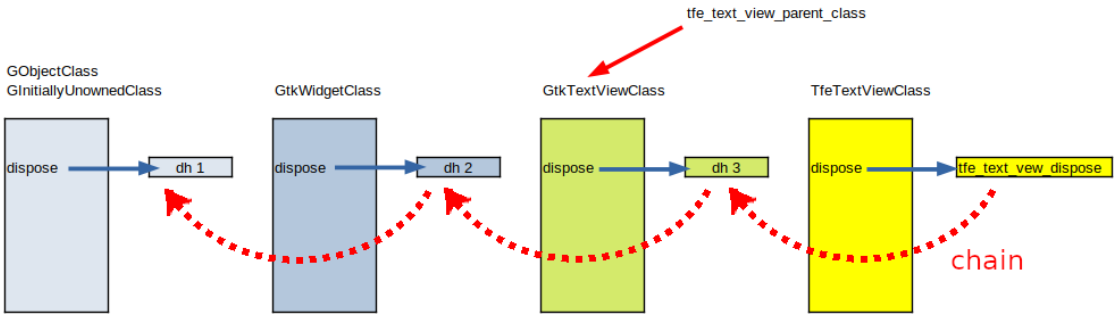

Figure 16: dispose handlers

In the destruction process of TfeTextView, the reference count of widgets related to TfeTextView is automatically decreased. But GFile pointed by `tv->file` needs to decrease its reference count by one. You must write the code in the dispose handler `tfe_text_view_dispose`.

```
1  static void
2  tfe_text_view_dispose (GObject *gobject) {
3    TfeTextView *tv = TFE_TEXT_VIEW (gobject);
4
5    if (G_IS_FILE (tv->file))
6      g_clear_object (&tv->file);
7
8    G_OBJECT_CLASS (tfe_text_view_parent_class)->dispose (gobject);
9  }
```

- 5,6: If `tv->file` points a GFile, decrease its reference count. `g_clear_object` decreases the reference count and assigns NULL to `tv->file`. In dispose handlers, we usually use `g_clear_object` rather than `g_object_unref`.
- 8: invokes parent's dispose handler. (This will be explained later.)

In the disposing process, the object uses the pointer in its class to call the handler. Therefore, `tfe_text_view_dispose` needs to be registered in the class when the TfeTextView class is initialized. The function `tfe_text_view_class_init` is the class initialization function and it is declared in the replacement produced by `G_DEFINE_TYPE` macro.

```
static void
tfe_text_view_class_init (TfeTextViewClass *class) {
  GObjectClass *object_class = G_OBJECT_CLASS (class);

  object_class->dispose = tfe_text_view_dispose;

}
```

Each ancestors' class has been created before TfeTextViewClass is created. Therefore, there are four classes and each class has a pointer to each dispose handler. Look at the following diagram. There are four classes – GObjectClass (GInitiallyUnownedClass), GtkWidgetClass, GtkTextViewClass and TfeTextViewClass. Each class has its own dispose handler – `dh1`, `dh2`, `dh3` and `tfe_text_view_dispose`.

Now, look at the `tfe_text_view_dispose` program above. It first releases the reference to GFile object pointed by `tv->file`. Then it invokes its parent's dispose handler in line 8.

```
G_OBJECT_CLASS (tfe_text_view_parent_class)->dispose (gobject);
```

`tfe_text_view_parent_class`,which is made by `G_DEFINE_TYPE` macro, is a pointer that points the parent object class. Therefore, `G_OBJECT_CLASS (tfe_text_view_parent_class)->dispose` points the handler `dh3` in the diagram above. And `gobject` is a pointer to TfeTextView instance which is casted as a GObject instance. `dh3` releases all the references to objects in the GtkTextView part (it is actually the private area pointed by `prev`) in TfeTextView instance. After that, `dh3` calls `dh2`, and `dh2` calls `dh1`. Finally all the references are released.

12 Signals

12.1 Signals

In Gtk programming, each object is encapsulated. And it is not recommended to use global variables because they tend to make the program complicated. So, we need something to communicate between objects. There are two ways to do so.

- Functions. For example, `tb = gtk_text_view_get_buffer (tv)`. The caller requests `tv` to give `tb`, which is a GtkTextBuffer instance connected to `tv` to the caller.
- Signals. For example, `activate` signal on GApplication object. When the application is activated, the signal is emitted. Then the handler, which has been connected to the signal, is invoked.

The caller of the function or the handler connected to the signal is usually out of the object. One of the difference between these two is that the object is active or passive. In functions the object passively responds to the caller. In signals the object actively sends a signal to the handler.

GObject signals are registered, connected and emitted.

1. Signals are registered with the object type on which they are emitted. The registration is done usually when the object class is initialized.
2. Signals are connected to handlers by `g_connect_signal` or its family functions. The connection is usually done out of the object.
3. When Signals are emitted, the connected handlers are invoked. Signal is emitted on the instance of the object.

12.2 Signal registration

In TfeTextView, two signals are registered.

- "change-file" signal. This signal is emitted when `tv->file` is changed.
- "open-response" signal. `tfe_text_view_open` function is not able to return the status because it uses GtkFileChooserDialog. This signal is emitted instead of the return value of the function.

A static variable or array is used to store the signal ID. A static array is used to register two or more signals.

```
enum {
  CHANGE_FILE,
  OPEN_RESPONSE,
  NUMBER_OF_SIGNALS
};

static guint tfe_text_view_signals[NUMBER_OF_SIGNALS];
```

Signals are registered in the class initialization function.

```
1  static void
2  tfe_text_view_class_init (TfeTextViewClass *class) {
3    GObjectClass *object_class = G_OBJECT_CLASS (class);
4
5    object_class->dispose = tfe_text_view_dispose;
6    tfe_text_view_signals[CHANGE_FILE] = g_signal_new ("change-file",
7                                    G_TYPE_FROM_CLASS (class),
8                                    G_SIGNAL_RUN_LAST | G_SIGNAL_NO_RECURSE |
                                        G_SIGNAL_NO_HOOKS,
9                                    0 /* class offset */,
10                                   NULL /* accumulator */,
11                                   NULL /* accumulator data */,
12                                   NULL /* C marshaller */,
13                                   G_TYPE_NONE /* return_type */,
14                                   0     /* n_params */
15                                   );
16   tfe_text_view_signals[OPEN_RESPONSE] = g_signal_new ("open-response",
17                                   G_TYPE_FROM_CLASS (class),
```

```
18                                  G_SIGNAL_RUN_LAST | G_SIGNAL_NO_RECURSE |
                                        G_SIGNAL_NO_HOOKS,
19                                  0 /* class offset */,
20                                  NULL /* accumulator */,
21                                  NULL /* accumulator data */,
22                                  NULL /* C marshaller */,
23                                  G_TYPE_NONE /* return_type */,
24                                  1     /* n_params */,
25                                  G_TYPE_INT
26                                  );
27  }
```

- 6-15: Registers "change-file" signal. `g_signal_new` function is used. The signal "change-file" has no default handler (object method handler). You usually don't need to set a default handler. If you need it, use `g_signal_new_class_handler` function. See GObject API Reference, g__signal__new__class__handler for further information.
- The return value of `g_signal_new` is the signal id. The type of signal id is guint, which is the same as unsigned int. It is used in the function `g_signal_emit`.
- 16-26: Registers "open-response" signal. This signal has a parameter.
- 24: Number of the parameters. "open-response" signal has one parameter.
- 25: The type of the parameter. `G_TYPE_INT` is a type of integer. Such fundamental types are described in GObject reference manual.

The handlers are declared as follows.

```
/* "change-file" signal handler */
void
user_function (TfeTextView *tv,
               gpointer user_data)

/* "open-response" signal handler */
void
user_function (TfeTextView *tv,
               TfeTextViewOpenResponseType response-id,
               gpointer user_data)
```

- Because "change-file" signal doesn't have parameter, the handler's parameters are a TfeTextView instance and user data.
- Because "open-response" signal has one parameter, the handler's parameters are a TfeTextView instance, the signal's parameter and user data.
- `tv` is the object instance on which the signal is emitted.
- `user_data` comes from the fourth argument of `g_signal_connect`.
- `parameter` comes from the fourth argument of `g_signal_emit`.

The values of the parameter is defined in `tfetextview.h` because they are public.

```
/* "open-response" signal response */
enum
{
  TFE_OPEN_RESPONSE_SUCCESS,
  TFE_OPEN_RESPONSE_CANCEL,
  TFE_OPEN_RESPONSE_ERROR
};
```

- The parameter is set to `TFE_OPEN_RESPONSE_SUCCESS` when `tfe_text_view_open` has successfully opened a file and read it.
- The parameter is set to `TFE_OPEN_RESPONSE_CANCEL` when the user has canceled.
- The parameter is set to `TFE_OPEN_RESPONSE_ERROR` when an error has occurred.

12.3 Signal connection

A signal and a handler are connected by the function `g_signal_connect`. There are some similar functions like `g_signal_connect_after`, `g_signal_connect_swapped` and so on. However, `g_signal_connect` is the most

common. The signals “change-file” is connected to a callback function out of the TfeTextView object. In the same way, the signals “open-response” is connected to a callback function out of the TfeTextView object. Those callback functions are defined by users.

In the program `tfe`, callback functions are defined in `tfenotebook.c`. And their names are `file_changed` and `open_response`. They will be explained later.

```
g_signal_connect (GTK_TEXT_VIEW (tv), "change-file", G_CALLBACK (file_changed), nb);

g_signal_connect (TFE_TEXT_VIEW (tv), "open-response", G_CALLBACK (open_response),
    nb);
```

12.4 Signal emission

Signals are emitted on an instance. The type of the instance is the second argument of `g_signal_new`. The relationship between the signal and object type is determined when the signal is registered.

A function `g_signal_emit` is used to emit the signal. The following lines are extracted from `tfetextview.c`. Each line comes from a different line.

```
g_signal_emit (tv, tfe_text_view_signals[CHANGE_FILE], 0);
g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
    TFE_OPEN_RESPONSE_SUCCESS);
g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
    TFE_OPEN_RESPONSE_CANCEL);
g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0, TFE_OPEN_RESPONSE_ERROR);
```

- The first argument is the instance on which the signal is emitted.
- The second argument is the signal id.
- The third argument is the detail of the signal. “change-file” signal and “open-response” signal doesn’t have details and the argument is zero when no details.
- “change-file” signal doesn’t have parameter, so there’s no fourth parameter.
- “open-response” signal has one parameter. The fourth parameter is the parameter.

13 Functions in TfeTextView

In this section I will explain functions in TfeTextView object.

13.1 tfe.h and tfetextview.h

`tfe.h` is a top header file and it includes `gtk.h` and all the header files. C source files `tfeapplication.c` and `tfenotebook.c` include `tfe.h` at the beginning.

```
1  #include <gtk/gtk.h>
2
3  #include "../tfetextview/tfetextview.h"
4  #include "tfenotebook.h"
```

`../tfetextview/tfetextview.h` is a header file which describes the public functions in `tfetextview.c`.

```
1  #ifndef __TFE_TEXT_VIEW_H__
2  #define __TFE_TEXT_VIEW_H__
3
4  #include <gtk/gtk.h>
5
6  #define TFE_TYPE_TEXT_VIEW tfe_text_view_get_type ()
7  G_DECLARE_FINAL_TYPE (TfeTextView, tfe_text_view, TFE, TEXT_VIEW, GtkTextView)
8
9  /* "open-response" signal response */
10 enum TfeTextViewOpenResponseType
11 {
12   TFE_OPEN_RESPONSE_SUCCESS,
13   TFE_OPEN_RESPONSE_CANCEL,
14   TFE_OPEN_RESPONSE_ERROR
```

```
15 };
16
17 GFile *
18 tfe_text_view_get_file (TfeTextView *tv);
19
20 void
21 tfe_text_view_open (TfeTextView *tv, GtkWindow *win);
22
23 void
24 tfe_text_view_save (TfeTextView *tv);
25
26 void
27 tfe_text_view_saveas (TfeTextView *tv);
28
29 GtkWidget *
30 tfe_text_view_new_with_file (GFile *file);
31
32 GtkWidget *
33 tfe_text_view_new (void);
34
35 #endif /* __TFE_TEXT_VIEW_H__ */
```

- 1,2,35: Thanks to these three lines, the following lines are included only once.
- 4: Includes gtk4 header files. The header file `gtk4` also has the same mechanism to avoid including it multiple times.
- 6-7: These two lines define TfeTextView type, its class structure and some useful macros.
- 9-15: A definition of the value of the parameter of "open-response" signal.
- 17-33: Declarations of public functions on TfeTextView.

13.2 Functions to create TfeTextView instances

A TfeTextView instance is created with `tfe_text_view_new` or `tfe_text_view_new_with_file`.

```
GtkWidget *tfe_text_view_new (void);
```

`tfe_text_view_new` just creates a new TfeTextView instance and returns the pointer to the new instance.

```
GtkWidget *tfe_text_view_new_with_file (GFile *file);
```

`tfe_text_view_new_with_file` is given a Gfile object as an argument and it loads the file into the GtkTextBuffer instance, then returns the pointer to the new instance. If an error occurs during the creation process, NULL is returned.

Each function is defined as follows.

```
1  GtkWidget *
2  tfe_text_view_new_with_file (GFile *file) {
3    g_return_val_if_fail (G_IS_FILE (file), NULL);
4
5    GtkWidget *tv;
6    GtkTextBuffer *tb;
7    char *contents;
8    gsize length;
9
10   if (! g_file_load_contents (file, NULL, &contents, &length, NULL, NULL)) /* read
         error */
11     return NULL;
12
13   if ((tv = tfe_text_view_new()) != NULL) {
14     tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
15     gtk_text_buffer_set_text (tb, contents, length);
16     TFE_TEXT_VIEW (tv)->file = g_file_dup (file);
17     gtk_text_buffer_set_modified (tb, FALSE);
18   }
19   g_free (contents);
```

```
20    return tv;
21  }
22
23  GtkWidget *
24  tfe_text_view_new (void) {
25    return GTK_WIDGET (g_object_new (TFE_TYPE_TEXT_VIEW, NULL));
26  }
```

- 23-25: `tfe_text_view_new` function. Just returns the value from the function `g_object_new` but casts it to the pointer to GtkWidget. Initialization is done in `tfe_text_view_init` which is called in the process of `g_object_new` function.
- 1-21: `tfe_text_view_new_with_file` function.
- 3: `g_return_val_if_fail` is described in GLib API Reference, g_return_val_if_fail. And also GLib API Reference, Message Logging. It tests whether the argument `file` is a pointer to GFile. If it's true, then the program goes on to the next line. If it's false, then it returns NULL (the second argument) immediately. And at the same time it logs out the error message (usually the log is outputted to stderr or stdout). This function is used to check the programmer's error. If an error occurs, the solution is usually to change the (caller) program and fix the bug. You need to distinguish programmer's errors and runtime errors. You shouldn't use this function to find runtime errors.
- 10-11: If an error occurs when reading the file, then the function returns NULL.
- 13: Calls the function `tfe_text_view_new`. The function creates TfeTextView instance and returns the pointer to the instance. If an error happens in `tfe_text_view_new`, it returns NULL.
- 14: Gets the pointer to GtkTextBuffer corresponds to `tv`. The pointer is assigned to `tb`
- 15: Assigns the contents read from the file to GtkTextBuffer pointed by `tb`.
- 16: Duplicates `file` and sets `tv->file` to point it.
- 17: The function `gtk_text_buffer_set_modified (tb, FALSE)` sets the modification flag of `tb` to FALSE. The modification flag indicates that the contents of the buffer is modified. It is used when the contents are saved. If the modification flag is FALSE, it doesn't need to save the contents.
- 19: Frees the memories pointed by `contents`.
- 20: Returns `tv`, which is a pointer to the newly created TfeTextView instance. If an error happens, NULL is returned.

13.3 Save and saveas functions

Save and saveas functions write the contents in the GtkTextBuffer to a file.

```
void tfe_text_view_save (TfeTextView *tv)
```

The function `tfe_text_view_save` writes the contents in the GtkTextBuffer to a file specified by `tv->file`. If `tv->file` is NULL, then it shows GtkFileChooserDialog and prompts the user to choose a file to save. Then it saves the contents to the file and sets `tv->file` to point the GFile instance for the file.

```
void tfe_text_view_saveas (TfeTextView *tv)
```

The function `saveas` uses GtkFileChooserDialog and prompts the user to select a existed file or specify a new file to save. Then, the function changes `tv->file` and save the contents to the specified file. If an error occurs, it is shown to the user through the message dialog. The error is managed only in the TfeTextView and no information is notified to the caller.

```
1   static gboolean
2   save_file (GFile *file, GtkTextBuffer *tb, GtkWindow *win) {
3     GtkTextIter start_iter;
4     GtkTextIter end_iter;
5     gchar *contents;
6     gboolean stat;
7     GtkWidget *message_dialog;
8     GError *err = NULL;
9
10    gtk_text_buffer_get_bounds (tb, &start_iter, &end_iter);
11    contents = gtk_text_buffer_get_text (tb, &start_iter, &end_iter, FALSE);
12    if (g_file_replace_contents (file, contents, strlen (contents), NULL, TRUE,
          G_FILE_CREATE_NONE, NULL, NULL, &err)) {
13      gtk_text_buffer_set_modified (tb, FALSE);
```

```
14      stat = TRUE;
15    } else {
16      message_dialog = gtk_message_dialog_new (win, GTK_DIALOG_MODAL,
17                                               GTK_MESSAGE_ERROR, GTK_BUTTONS_CLOSE,
18                                              "%s.\n", err->message);
19      g_signal_connect (message_dialog, "response", G_CALLBACK (gtk_window_destroy),
            NULL);
20      gtk_widget_show (message_dialog);
21      g_error_free (err);
22      stat = FALSE;
23    }
24    g_free (contents);
25    return stat;
26  }
27
28  static void
29  saveas_dialog_response (GtkWidget *dialog, gint response, TfeTextView *tv) {
30    GtkTextBuffer *tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
31    GFile *file;
32    GtkWidget *win = gtk_widget_get_ancestor (GTK_WIDGET (tv), GTK_TYPE_WINDOW);
33
34    if (response == GTK_RESPONSE_ACCEPT) {
35      file = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (dialog));
36      if (! G_IS_FILE (file))
37        g_warning ("TfeTextView: gtk_file_chooser_get_file returns non GFile.\n");
38      else if (save_file(file, tb, GTK_WINDOW (win))) {
39        if (G_IS_FILE (tv->file))
40          g_object_unref (tv->file);
41        tv->file = file;
42        g_signal_emit (tv, tfe_text_view_signals[CHANGE_FILE], 0);
43      } else
44        g_object_unref (file);
45    }
46    gtk_window_destroy (GTK_WINDOW (dialog));
47  }
48
49  void
50  tfe_text_view_save (TfeTextView *tv) {
51    g_return_if_fail (TFE_IS_TEXT_VIEW (tv));
52
53    GtkTextBuffer *tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
54    GtkWidget *win = gtk_widget_get_ancestor (GTK_WIDGET (tv), GTK_TYPE_WINDOW);
55
56    if (! gtk_text_buffer_get_modified (tb))
57      return; /* no need to save it */
58    else if (tv->file == NULL)
59      tfe_text_view_saveas (tv);
60    else if (! G_IS_FILE (tv->file))
61      g_error ("TfeTextView: The pointer tv->file isn't NULL nor GFile.\n");
62    else
63      save_file (tv->file, tb, GTK_WINDOW (win));
64  }
65
66  void
67  tfe_text_view_saveas (TfeTextView *tv) {
68    g_return_if_fail (TFE_IS_TEXT_VIEW (tv));
69
70    GtkWidget *dialog;
71    GtkWidget *win = gtk_widget_get_ancestor (GTK_WIDGET (tv), GTK_TYPE_WINDOW);
72
73    dialog = gtk_file_chooser_dialog_new ("Save file", GTK_WINDOW (win),
         GTK_FILE_CHOOSER_ACTION_SAVE,
74                                          "Cancel", GTK_RESPONSE_CANCEL,
75                                          "Save", GTK_RESPONSE_ACCEPT,
```

```
76                                          NULL);
77    g_signal_connect (dialog, "response", G_CALLBACK (saveas_dialog_response), tv);
78    gtk_widget_show (dialog);
79  }
```

- 1-26: `save_file` function. This function is called from `saveas_dialog_response` and `tfe_text_view_save`. This function saves the contents of the buffer to the file given as an argument. If error happens, it displays an error message. The class of this function is `static`. Therefore, only functions in this file (`tfeTetview.c`) call this function. Such static functions usally don't have `g_return_val_if_fail` function.
- 10-11: Gets the text contents from the buffer.
- 12-14: Saves the contents to the file. If no error happens, set the modified flag to be FALSE. This means that the buffer is not modified since it has been saved. And set the return status `stat` to be TRUE.
- 15-23: If it fails to save the contents, displays an error message.
- 16-18: Creates a message dialog with the error message.
- 19: Connects the "response" signal to `gtk_window_destroy`, so that the dialog disappears when a user clicked on the button.
- 20-21: Shows the window, frees `err` and set `stat` to be FLASE.
- 24: Frees `contents`.
- 25: Returns to the caller.
- 28-47: `saveas_dialog_response` function. This is a signal handler for the "response" signal on GtkFileChooserDialog instance created by `tfe_text_view_saveas` function. This handler analyzes the response and determines whether to save the contents.
- 34-45: If the response is `GTK_RESPONSE_ACCEPT`, the user has clicked on the `Save` button. So, it tries to save.
- 35: Gets the GFile `file` from GtkFileChooserDialog.
- 36-37: If it doesn't point GFile, it outputs an error message to the log.
- 38: Otherwise, it calls `save_file` to save the contents to the file.
- 39-42: If `save_file` has successfully saved the contents, `tv->file` is updated. If the old GFile pointed by `tv->file` exists, it is freed in advance. Emits "change-file" signal.
- 44: Unrefs `file`.
- 46: destroys the file chooser dialog.
- 49-64: `tfe_text_view_save` function.
- 51: `tfe_text_view_save` is public, i.e. it is open to the other files. So, it doesn't have `static` class. Public functions should check the parameter type with `g_return_if_fail` function. If `tv` is not a pointer to a TfeTextView instance, then it logs an error message and immediately returns. This function is similar to `g_return_val_if_fail`, but no value is returned because `tfe_text_view_save` doesn't return a value.
- 53-54: Gets GtkTextBuffer instance and GtkWidget instance and assignes them to `tb` and`win` respectively.
- 56-57: If the buffer hasn't modified, then it doesn't need to save it. So the function returns.
- 58-59: If `tv->file` is NULL, no file has given yet. It calls `tfe_text_view_saveas` which prompts a user to select a file or specify a new file to save.
- 60-61: If `tv->file` doesn't point GFile, somethig bad has happened. Logs an error message.
- 62-63: Calls `save_file` to save the contents to the file.
- 66-79: `tfe_text_view_saveas` function. It shows GtkFileChooserDialog and prompts the user to choose a file.
- 73-76: Creates GtkFileChooserDialog. The title is "Save file". Transient parent of the dialog is `win`, which is the top-level window. The action is save mode. The buttons are Cancel and Save.
- 77: connects the "response" signal of the dialog and `saveas_dialog_response` handler.
- 78: Shows the dialog.

When you use GtkFileChooserDialog, you need to divide the program into two parts. One is a function which creates GtkFileChooserDialog and the other is a signal handler. The function just creates and shows GtkFileChooserDialog. The rest is done by the handler. It gets Gfile from GtkFileChooserDialog and saves the buffer to the file by calling `save_file`.

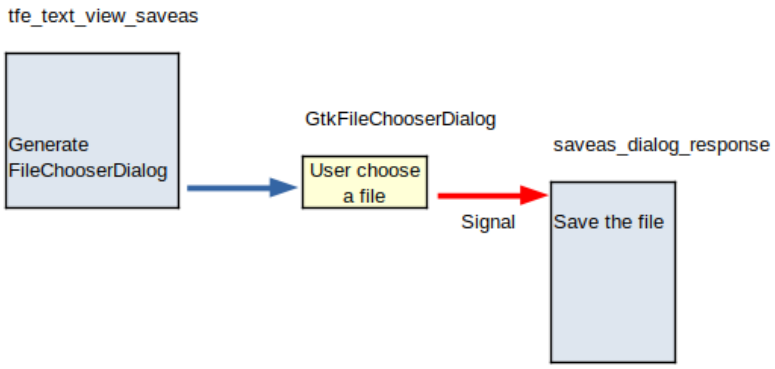

Figure 17: Saveas process

13.4 Open function

Open function shows GtkFileChooserDialog to users and prompts them to choose a file. Then it reads the file and puts the text into GtkTextBuffer.

```
void tfe_text_view_open (TfeTextView *tv, GtkWindow *win);
```

The parameter `win` is the top-level window. It will be a transient parent window of GtkFileChooserDialog when the dialog is created. This allows window managers to keep the dialog on top of the parent window, or center the dialog over the parent window. It is possible to give no parent window to the dialog. However, it is encouraged to give a parent window to dialog. This function might be called just after `tv` has been created. In that case, `tv` has not been incorporated into the widget hierarchy. Therefore it is impossible to get the top-level window from `tv`. That's why the function needs `win` parameter.

This function is usually called when the buffer of `tv` is empty. However, even if the buffer is not empty, `tfe_text_view_open` doesn't treat it as an error. If you want to revert the buffer, calling this function is appropriate. Otherwise probably bad things will happen.

```
1  static void
2  open_dialog_response(GtkWidget *dialog, gint response, TfeTextView *tv) {
3    GtkTextBuffer *tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
4    GFile *file;
5    char *contents;
6    gsize length;
7    GtkWidget *message_dialog;
8    GError *err = NULL;
9
10   if (response != GTK_RESPONSE_ACCEPT)
11     g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
           TFE_OPEN_RESPONSE_CANCEL);
12   else if (! G_IS_FILE (file = gtk_file_chooser_get_file (GTK_FILE_CHOOSER
         (dialog)))) {
13     g_warning ("TfeTextView: gtk_file_chooser_get_file returns non GFile.\n");
14     g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
           TFE_OPEN_RESPONSE_ERROR);
15   } else if (! g_file_load_contents (file, NULL, &contents, &length, NULL, &err)) {
       /* read error */
16     g_object_unref (file);
17     message_dialog = gtk_message_dialog_new (GTK_WINDOW (dialog), GTK_DIALOG_MODAL,
18                                              GTK_MESSAGE_ERROR, GTK_BUTTONS_CLOSE,
19                                             "%s.\n", err->message);
20     g_signal_connect (message_dialog, "response", G_CALLBACK (gtk_window_destroy),
           NULL);
21     gtk_widget_show (message_dialog);
22     g_error_free (err);
23     g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
           TFE_OPEN_RESPONSE_ERROR);
24   } else {
```

```
25      gtk_text_buffer_set_text (tb, contents, length);
26      g_free (contents);
27      if (G_IS_FILE (tv->file))
28        g_object_unref (tv->file);
29      tv->file = file;
30      gtk_text_buffer_set_modified (tb, FALSE);
31      g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
            TFE_OPEN_RESPONSE_SUCCESS);
32      g_signal_emit (tv, tfe_text_view_signals[CHANGE_FILE], 0);
33    }
34    gtk_window_destroy (GTK_WINDOW (dialog));
35  }
36
37  void
38  tfe_text_view_open (TfeTextView *tv, GtkWindow *win) {
39    g_return_if_fail (TFE_IS_TEXT_VIEW (tv));
40    g_return_if_fail (GTK_IS_WINDOW (win));
41
42    GtkWidget *dialog;
43
44    dialog = gtk_file_chooser_dialog_new ("Open file", win,
          GTK_FILE_CHOOSER_ACTION_OPEN,
45                                          "Cancel", GTK_RESPONSE_CANCEL,
46                                          "Open", GTK_RESPONSE_ACCEPT,
47                                          NULL);
48    g_signal_connect (dialog, "response", G_CALLBACK (open_dialog_response), tv);
49    gtk_widget_show (dialog);
50  }
```

- 37-50: `tfe_text_view_open` function.
- 44-47: Creates GtkFileChooserDialog. The title is "Open file". Transient parent window is the top-level window of the application, which is given by the caller. The action is open mode. The buttons are Cancel and Open.
- 48: connects the "response" signal of the dialog and `open_dialog_response` signal handler.
- 49: Shows the dialog.
- 1-35: `open_dialog_response` signal handler.
- 10-11: If the response from GtkFileChooserDialog is not `GTK_RESPONSE_ACCEPT`, the user has clicked on the "Cancel" button or close button on the header bar. Then, "open-response" signal is emitted. The parameter of the signal is `TFE_OPEN_RESPONSE_CANCEL`.
- 12-14: Gets the pointer to the Gfile by `gtk_file_chooser_get_file`. If it doesn't point GFile, maybe an error has occurred. Then it emits "open-response" signal with the parameter `TFE_OPEN_RESPONSE_ERROR`.
- 15-23: If an error occurs at file reading, then it decreases the reference count of the Gfile, shows a message dialog to report the error to the user and emits "open-response" signal with the parameter `TFE_OPEN_RESPONSE_ERROR`.
- 24-33: If the file has successfully been read, then the text is inserted to GtkTextBuffer, frees the temporary buffer pointed by `contents` and sets `tv->file` to point the file (no duplication is not necessary). Then, it emits "open-response" signal with the parameter `TFE_OPEN_RESPONSE_SUCCESS` and emits "change-file" signal.
- 34: destroys GtkFileCooserDialog.

Now let's think about the whole process between the caller and TfeTextView. It is shown in the following diagram and you would think that it is really complicated. Because signal is the only way for GtkFileChooserDialog to communicate with others. In Gtk3, `gtk_dialog_run` function is available. It simplifies the process. However, in Gtk4, `gtk_dialog_run` is unavailable any more.

1. A caller gets a pointer `tv` to a TfeTextView instance by calling `tfe_text_view_new`.
2. The caller connects the handler (left bottom in the diagram) and the signal "open-response".
3. It calls `tfe_text_view_open` to prompt the user to select a file from GtkFileChooserDialog.
4. The dialog emits a signal and it invokes the handler `open_dialog_response`.
5. The handler reads the file and inserts the text into GtkTextBuffer and emits a signal to inform the status as a response code.
6. The handler out of the TfeTextView receives the signal.

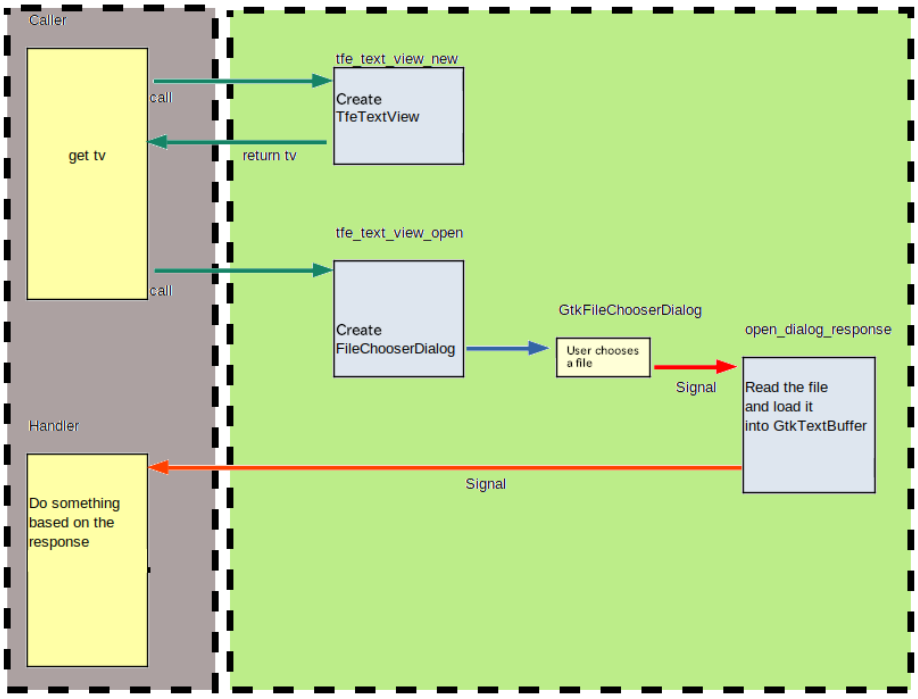

Figure 18: Caller and TfeTextView

13.5 Getting Gfile

`gtk_text_view_get_file` is a simple function shown as follows.

```
1  GFile *
2  tfe_text_view_get_file (TfeTextView *tv) {
3    g_return_val_if_fail (TFE_IS_TEXT_VIEW (tv), NULL);
4
5    if (G_IS_FILE (tv->file))
6      return g_file_dup (tv->file);
7    else
8      return NULL;
9  }
```

The important thing is to duplicate `tv->file`. Otherwise, if the caller frees the GFile object, `tv->file` is no more guaranteed to point the GFile. Another reason to use `g_file_dup` is that GFile isn't thread-safe. If you use GFile in the different thread, the duplication is necessary. See Gio API Reference, g_file_dup.

13.6 The API document and source file of tfetextview.c

Refer API document of TfeTextView in the appendix. Its original markdown file is under the directory `src/tfetextview`.

All the source files are listed in Section 16. You can find them under src/tfe5 and src/tfetextview directories.

14 Functions in GtkNotebook

GtkNotebook is a very important object in the text file editor `tfe`. It connects the application and TfeTextView objects. A set of public functions are declared in `tfenotebook.h`. The word "tfenotebook" is used only in filenames. There's no "TfeNotebook" object.

```
1  void
2  notebook_page_save(GtkNotebook *nb);
3
4  void
```

```
5  notebook_page_close (GtkNotebook *nb);
6
7  void
8  notebook_page_open (GtkNotebook *nb);
9
10 void
11 notebook_page_new_with_file (GtkNotebook *nb, GFile *file);
12
13 void
14 notebook_page_new (GtkNotebook *nb);
```

This header file describes the public functions in `tfenotebook.c`.

- 1-2: `notebook_page_save` saves the current page to the file of which the name specified in the tab. If the name is `untitled` or `untitled` followed by digits, FileChooserDialog appears and a user can choose or specify a filename.
- 4-5: `notebook_page_close` closes the current page.
- 7-8: `notebook_page_open` shows a file chooser dialog and a user can choose a file. The file is inserted to a new page.
- 10-11: `notebook_page_new_with_file` creates a new page and the file given as an argument is read and inserted into the page.
- 13-14: `notebook_page_new` creates a new empty page.

You probably find that the functions except `notebook_page_close` are higher level functions of

- `tfe_text_view_save`
- `tef_text_view_open`
- `tfe_text_view_new_with_file`
- `tfe_text_view_new`

respectively.

There are two layers. One of them is `tfe_text_view ...`, which is the lower level layer. The other is `note_book ...`, which is the higher level layer.

Now let's look at the program of each function.

14.1 notebook_page_new

```
1  static gchar*
2  get_untitled () {
3    static int c = -1;
4    if (++c == 0)
5      return g_strdup_printf("Untitled");
6    else
7      return g_strdup_printf ("Untitled%u", c);
8  }
9
10 static void
11 notebook_page_build (GtkNotebook *nb, GtkWidget *tv, char *filename) {
12   GtkWidget *scr = gtk_scrolled_window_new ();
13   GtkNotebookPage *nbp;
14   GtkWidget *lab;
15   int i;
16
17   gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
18   gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);
19   lab = gtk_label_new (filename);
20   i = gtk_notebook_append_page (nb, scr, lab);
21   nbp = gtk_notebook_get_page (nb, scr);
22   g_object_set (nbp, "tab-expand", TRUE, NULL);
23   gtk_notebook_set_current_page (nb, i);
24   g_signal_connect (GTK_TEXT_VIEW (tv), "change-file", G_CALLBACK (file_changed_cb),
         nb);
25 }
```

```
26
27 void
28 notebook_page_new (GtkNotebook *nb) {
29   g_return_if_fail(GTK_IS_NOTEBOOK (nb));
30
31   GtkWidget *tv;
32   char *filename;
33
34   if ((tv = tfe_text_view_new ()) == NULL)
35     return;
36   filename = get_untitled ();
37   notebook_page_build (nb, tv, filename);
38 }
```

- 27-38: `notebook_page_new` function.
- 29: `g_return_if_fail` is used to check the argument.
- 34: Creates TfeTextView object. If it fails, it returns to the caller.
- 36: Creates filename, which is "Untitled", "Untitled1", … .
- 1-8: `get_untitled` function.
- 3: Static variable `c` is initialized at the first call of this function. After that `c` keeps its value unless it is changed explicitly.
- 4-7: Increases `c` by one and if it is zero then it returns "Untitled". If it is a positive integer then it returns "Untitled<the integer>", for example, "Untitled1", "Untitled2", and so on. The function `g_strdup_printf` creates a string and it should be freed by `g_free` when it becomes useless. The caller of `get_untitled` is in charge of freeing the string.
- 37: calls `notebook_page_build` to build the contents of the page.
- 10- 25: `notebook_page_build` function.
- 12: Creates GtkScrolledWindow.
- 17: Sets the wrap mode of `tv` to GTK_WRAP_WORD_CHAR so that lines are broken between words or graphemes.
- 18: Inserts `tv` to GtkscrolledWindow as a child.
- 19-20: Creates GtkLabel, then appends `scr` and `lab` to the GtkNotebook instance `nb`.
- 21-22: Sets "tab-expand" property to TRUE. The function `g_object_set` sets properties on an object. The object is any object derived from GObject. In many cases, an object has its own function to set its properties, but sometimes not. In that case, use `g_object_set` to set the property.
- 23: Sets the current page of `nb` to the newly created page.
- 24: Connects "change-file" signal and `file_changed_cb` handler.

14.2 notebook_page_new_with_file

```
1  void
2  notebook_page_new_with_file (GtkNotebook *nb, GFile *file) {
3    g_return_if_fail(GTK_IS_NOTEBOOK (nb));
4    g_return_if_fail(G_IS_FILE (file));
5
6    GtkWidget *tv;
7    char *filename;
8
9    if ((tv = tfe_text_view_new_with_file (file)) == NULL)
10     return; /* read error */
11   filename = g_file_get_basename (file);
12   notebook_page_build (nb, tv, filename);
13 }
```

- 9-10: Calls `tfe_text_view_new_with_file`. If the function returns NULL, an error has happend. Then, it does nothing and returns.
- 11-12: Gets the filename and builds the contents of the page.

14.3 notebook_page_open

```
1  static void
2  open_response (TfeTextView *tv, int response, GtkNotebook *nb) {
```

```
3    GFile *file;
4    char *filename;
5
6    if (response != TFE_OPEN_RESPONSE_SUCCESS || ! G_IS_FILE (file =
         tfe_text_view_get_file (tv))) {
7      g_object_ref_sink (tv);
8      g_object_unref (tv);
9    }else {
10     filename = g_file_get_basename (file);
11     g_object_unref (file);
12     notebook_page_build (nb, GTK_WIDGET (tv), filename);
13   }
14 }
15
16 void
17 notebook_page_open (GtkNotebook *nb) {
18   g_return_if_fail(GTK_IS_NOTEBOOK (nb));
19
20   GtkWidget *tv;
21
22   if ((tv = tfe_text_view_new ()) == NULL)
23     return;
24   g_signal_connect (TFE_TEXT_VIEW (tv), "open-response", G_CALLBACK (open_response),
         nb);
25   tfe_text_view_open (TFE_TEXT_VIEW (tv), GTK_WINDOW (gtk_widget_get_ancestor
         (GTK_WIDGET (nb), GTK_TYPE_WINDOW)));
26 }
```

- 16-26: `notebook_page_open` function.
- 22-23: Creates TfeTextView object. If NULL is returned, an error has happened. Then, it returns to the caller.
- 24: Connects the signal "open-response" and the handler `open_response`.
- 25: Calls `tfe_text_view_open`. The "open-response" signal will be emitted later to inform the result of opening and reading a file.
- 1-14: `open_response` handler.
- 6-8: If the response code is NOT `TFE_OPEN_RESPONSE_SUCCESS` or `tfe_text_view_get_file` doesn't return the pointer to a GFile, it has failed to open and read a new file. Then, what `notebook_page_open` did in advance need to be canceled. The instance `tv` hasn't been a child widget of GtkScrolledWindow yet. Such instance has floating reference. Floating reference will be explained later in this subsection. You need to call `g_object_ref_sink` first. Then the floating reference is converted into an ordinary reference. Now you call `g_object_unref` to decrease the reference count by one.
- 9-13: Otherwise, everything is okay. Gets the filename, builds the contents of the page.

All the widgets are derived from GInitiallyUnowned. When an instance of GInitiallyUnowned or its descendant is created, the instance has a floating reference. The function `g_object_ref_sink` converts the floating reference into an ordinary reference. If the instance doesn't have a floating reference, `g_object_ref_sink` simply increases the reference count by one. On the other hand, when an instance of GObject (not GInitiallyUnowned) is created, no floating reference is given. And the instance has a normal reference count instead of floating reference.

If you use `g_object_unref` to an instance that has a floating reference, you need to convert the floating reference to a normal reference in advance. See GObject Reference Manual for further information.

14.4 notebook_page_close

```
1  void
2  notebook_page_close (GtkNotebook *nb) {
3    g_return_if_fail(GTK_IS_NOTEBOOK (nb));
4
5    GtkWidget *win;
6    int i;
7
8    if (gtk_notebook_get_n_pages (nb) == 1) {
```

```
9       win = gtk_widget_get_ancestor (GTK_WIDGET (nb), GTK_TYPE_WINDOW);
10      gtk_window_destroy(GTK_WINDOW (win));
11    } else {
12      i = gtk_notebook_get_current_page (nb);
13      gtk_notebook_remove_page (GTK_NOTEBOOK (nb), i);
14    }
15  }
```

This function closes the current page. If the page is the only page the notebook has, then the function destroys the top-level window and quits the application.

- 8-10: If the page is the only page the notebook has, it calls `gtk_window_destroy` to destroys the top-level window.
- 11-13: Otherwise, removes the current page.

14.5 notebook_page_save

```
1   static TfeTextView *
2   get_current_textview (GtkNotebook *nb) {
3     int i;
4     GtkWidget *scr;
5     GtkWidget *tv;
6
7     i = gtk_notebook_get_current_page (nb);
8     scr = gtk_notebook_get_nth_page (nb, i);
9     tv = gtk_scrolled_window_get_child (GTK_SCROLLED_WINDOW (scr));
10    return TFE_TEXT_VIEW (tv);
11  }
12
13  void
14  notebook_page_save (GtkNotebook *nb) {
15    g_return_if_fail(GTK_IS_NOTEBOOK (nb));
16
17    TfeTextView *tv;
18
19    tv = get_current_textview (nb);
20    tfe_text_view_save (TFE_TEXT_VIEW (tv));
21  }
```

- 13-21: `notebook_page_save`.
- 19: Gets TfeTextView belongs to the current page.
- 20: Calls `tfe_text_view_save`.
- 1-11: `get_current_textview`. This function gets the TfeTextView object belongs to the current page.
- 7: Gets the page number of the current page.
- 8: Gets the child widget `scr`, which is a GtkScrolledWindow instance, of the current page.
- 9-10: Gets the child widget of `scr`, which is a TfeTextView instance, and returns it.

14.6 file_changed_cb handler

The function `file_changed_cb` is a handler connected to "change-file" signal. If a file in a TfeTextView instance is changed, it emits this signal. This handler changes the label of GtkNotebookPage.

```
1   static void
2   file_changed_cb (TfeTextView *tv, GtkNotebook *nb) {
3     GtkWidget *scr;
4     GtkWidget *label;
5     GFile *file;
6     char *filename;
7
8     file = tfe_text_view_get_file (tv);
9     scr = gtk_widget_get_parent (GTK_WIDGET (tv));
10    if (G_IS_FILE (file)) {
11      filename = g_file_get_basename (file);
```

```
12      g_object_unref (file);
13    } else
14      filename = get_untitled ();
15    label = gtk_label_new (filename);
16    g_free (filename);
17    gtk_notebook_set_tab_label (nb, scr, label);
18  }
```

- 8: Gets the GFile instance from `tv`.
- 9: Gets the GkScrolledWindow instance which is the parent widget of `tv`.
- 10-12: If `file` points GFile, then assigns the filename of the GFile into `filename`. Then, unref the GFile object `file`.
- 13-14: Otherwise (file is NULL), assigns untitled string to `filename`.
- 15-16: Creates a GtkLabel instance `label` with the filename and set the label of the GtkNotebookPage with `label`.

15 tfeapplication.c

`tfeapplication.c` includes all the code other than `tfetxtview.c` and `tfenotebook.c`. It does:

- Application support, mainly handling command line arguments.
- Builds widgets using ui file.
- Connects button signals and their handlers.
- Manages CSS.

15.1 main

The function `main` is the first invoked function in C language. It connects the command line given by the user and Gtk application.

```
1  #define APPLICATION_ID "com.github.ToshioCP.tfe"
2
3  int
4  main (int argc, char **argv) {
5    GtkApplication *app;
6    int stat;
7
8    app = gtk_application_new (APPLICATION_ID, G_APPLICATION_HANDLES_OPEN);
9
10   g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
11   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
12   g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);
13
14   stat =g_application_run (G_APPLICATION (app), argc, argv);
15   g_object_unref (app);
16   return stat;
17 }
```

- 1: Defines the application id. It is easy to find the application id, and better than the id is embedded in `gtk_application_new`.
- 8: Creates GtkApplication object.
- 10-12: Connects “startup”, “activate” and “open” signals to their handlers.
- 14: Runs the application.
- 15-16: releases the reference to the application and returns the status.

15.2 startup signal handler

Startup signal is emitted just after the GtkApplication instance is initialized. What the signal handler needs to do is initialization of the application.

- Builds the widgets using ui file.
- Connects button signals and their handlers.

- Sets CSS.

The handler is as follows.

```
1  static void
2  app_startup (GApplication *application) {
3    GtkApplication *app = GTK_APPLICATION (application);
4    GtkBuilder *build;
5    GtkApplicationWindow *win;
6    GtkNotebook *nb;
7    GtkButton *btno;
8    GtkButton *btnn;
9    GtkButton *btns;
10   GtkButton *btnc;
11
12   build = gtk_builder_new_from_resource ("/com/github/ToshioCP/tfe/tfe.ui");
13   win = GTK_APPLICATION_WINDOW (gtk_builder_get_object (build, "win"));
14   nb = GTK_NOTEBOOK (gtk_builder_get_object (build, "nb"));
15   gtk_window_set_application (GTK_WINDOW (win), app);
16   btno = GTK_BUTTON (gtk_builder_get_object (build, "btno"));
17   btnn = GTK_BUTTON (gtk_builder_get_object (build, "btnn"));
18   btns = GTK_BUTTON (gtk_builder_get_object (build, "btns"));
19   btnc = GTK_BUTTON (gtk_builder_get_object (build, "btnc"));
20   g_signal_connect_swapped (btno, "clicked", G_CALLBACK (open_cb), nb);
21   g_signal_connect_swapped (btnn, "clicked", G_CALLBACK (new_cb), nb);
22   g_signal_connect_swapped (btns, "clicked", G_CALLBACK (save_cb), nb);
23   g_signal_connect_swapped (btnc, "clicked", G_CALLBACK (close_cb), nb);
24   g_object_unref(build);
25
26 GdkDisplay *display;
27
28   display = gtk_widget_get_display (GTK_WIDGET (win));
29   GtkCssProvider *provider = gtk_css_provider_new ();
30   gtk_css_provider_load_from_data (provider, "textview {padding: 10px; font-family:
         monospace; font-size: 12pt;}", -1);
31   gtk_style_context_add_provider_for_display (display, GTK_STYLE_PROVIDER
         (provider), GTK_STYLE_PROVIDER_PRIORITY_APPLICATION);
32 }
```

- 12-15: Builds widgets using ui file (resource). Connects the top-level window and the application with `gtk_window_set_application`.
- 16-23: Gets buttons and connects their signals and handlers.
- 24: Releases the reference to GtkBuilder.
- 26-31: Sets CSS. CSS in Gtk is similar to CSS in HTML. You can set margin, border, padding, color, font and so on with CSS. In this program CSS is in line 30. It sets padding, font-family and font size of GtkTextView.
- 26-28: GdkDisplay is used to set CSS. CSS will be explained in the next subsection.

15.3 CSS in Gtk

CSS is an abbreviation of Cascading Style Sheet. It is originally used with HTML to describe the presentation semantics of a document. You might have found that the widgets in Gtk is similar to a window in a browser. It implies that CSS can also be applied to Gtk windowing system.

15.3.1 CSS nodes, selectors

The syntax of CSS is as follows.

```
selector { color: yellow; padding-top: 10px; ...}
```

Every widget has CSS node. For example GtkTextView has `textview` node. If you want to set style to GtkTextView, substitute "textview" for the selector.

```
textview {color: yellow; ...}
```

Class, ID and some other things can be applied to the selector like Web CSS. Refer to Gtk4 API Reference, CSS in Gtk for further information.

In line 30, the CSS is a string.

```
textview {padding: 10px; font-family: monospace; font-size: 12pt;}
```

- padding is a space between the border and contents. This space makes the textview easier to read.
- font-family is a name of font. “monospace” is one of the generic family font keywords.
- font-size is set to 12pt.

15.3.2 GtkStyleContext, GtkCSSProvider and GdkDisplay

GtkStyleContext is an object that stores styling information affecting a widget. Each widget is connected to the corresponding GtkStyleContext. You can get the context by `gtk_widget_get_style_context`.

GtkCssProvider is an object which parses CSS in order to style widgets.

To apply your CSS to widgets, you need to add GtkStyleProvider (the interface of GtkCSSProvider) to GtkStyleContext. However, instead, you can add it to GdkDisplay of the window (usually top-level window).

Look at the source file of `startup` handler again.

- 28: The display is obtained by `gtk_widget_get_display`.
- 29: Creates a GtkCssProvider instance.
- 30: Puts the CSS into the provider.
- 31: Adds the provider to the display. The last argument of `gtk_style_context_add_provider_for_display` is the priority of the style provider. `GTK_STYLE_PROVIDER_PRIORITY_APPLICATION` is a priority for application-specific style information. `GTK_STYLE_PROVIDER_PRIORITY_USER` is also often used and it is the highest priority. So, `GTK_STYLE_PROVIDER_PRIORITY_USER` is often used to a specific widget.

It is possible to add the provider to the context of GtkTextView instead of GdkDiplay. To do so, rewrite `tfe_text_view_new`. First, get the GtkStyleContext object of a TfeTextView object. Then adds the CSS provider to the context.

```
GtkWidget *
tfe_text_view_new (void) {
  GtkWidget *tv;

  tv = gtk_widget_new (TFE_TYPE_TEXT_VIEW, NULL);

  GtkStyleContext *context;

  context = gtk_widget_get_style_context (GTK_WIDGET (tv));
  GtkCssProvider *provider = gtk_css_provider_new ();
  gtk_css_provider_load_from_data (provider, "textview {padding: 10px; font-family:
      monospace; font-size: 12pt;}", -1);
  gtk_style_context_add_provider (context, GTK_STYLE_PROVIDER (provider),
      GTK_STYLE_PROVIDER_PRIORITY_USER);

  return tv;
}
```

CSS in the context takes precedence over CSS in the display.

15.4 activate and open handler

The handler of “activate” and “open” signal are `app_activate` and `app_open` respectively. They just create a new GtkNotebookPage.

```
1 static void
2 app_activate (GApplication *application) {
3   GtkApplication *app = GTK_APPLICATION (application);
4   GtkWidget *win = GTK_WIDGET (gtk_application_get_active_window (app));
5   GtkWidget *boxv = gtk_window_get_child (GTK_WINDOW (win));
```

```
6     GtkNotebook *nb = GTK_NOTEBOOK (gtk_widget_get_last_child (boxv));
7
8     notebook_page_new (nb);
9     gtk_widget_show (GTK_WIDGET (win));
10  }
11
12  static void
13  app_open (GApplication *application, GFile ** files, gint n_files, const gchar
        *hint) {
14    GtkApplication *app = GTK_APPLICATION (application);
15    GtkWidget *win = GTK_WIDGET (gtk_application_get_active_window (app));
16    GtkWidget *boxv = gtk_window_get_child (GTK_WINDOW (win));
17    GtkNotebook *nb = GTK_NOTEBOOK (gtk_widget_get_last_child (boxv));
18    int i;
19
20    for (i = 0; i < n_files; i++)
21      notebook_page_new_with_file (nb, files[i]);
22    if (gtk_notebook_get_n_pages (nb) == 0)
23      notebook_page_new (nb);
24    gtk_widget_show (win);
25  }
```

- 1-10: `app_activate`.
- 8-10: Creates a new page and shows the window.
- 12-25: `app_open`.
- 20-21: Creates notebook pages with files.
- 22-23: If no page has created, maybe because of read error, then it creates an empty page.
- 24: Shows the window.

These codes have become really simple thanks to tfenotebook.c and tfetextview.c.

15.5 Primary instance

Only one GApplication instance can be run at a time per session. The session is a bit difficult concept and also platform-dependent, but roughly speaking, it corresponds to a graphical desktop login. When you use your PC, you probably login first, then your desktop appears until you log off. This is the session.

However, Linux is multi process OS and you can run two or more instances of the same application. Isn't it a contradiction?

When first instance is launched, then it registers itself with its application ID (for example, `com.github.ToshioCP.tfe`). Just after the registration, startup signal is emitted, then activate or open signal is emitted and the instance's main loop runs. I wrote "startup signal is emitted just after the application instance is initialized" in the prior subsection. More precisely, it is emitted just after the registration.

If another instance which has the same application ID is invoked, it also tries to register itself. Because this is the second instance, the registration of the ID has already done, so it fails. Because of the failure startup signal isn't emitted. After that, activate or open signal is emitted in the primary instance, not the second instance. The primary instance receives the signal and its handler is invoked. On the other hand, the second instance doesn't receive the signal and it immediately quits.

Try to run two instances in a row.

```
$ ./_build/tfe &
[1] 84453
$ ./build/tfe tfeapplication.c
$
```

First, the primary instance opens a window. Then, after the second instance is run, a new notebook page with the contents of `tfeapplication.c` appears in the primary instance's window. This is because the open signal is emitted in the primary instance. The second instance immediately quits so shell prompt soon appears.

15.6 a series of handlers correspond to the button signals

```
1  static void
2  open_cb (GtkNotebook *nb) {
3    notebook_page_open (nb);
4  }
5
6  static void
7  new_cb (GtkNotebook *nb) {
8    notebook_page_new (nb);
9  }
10
11 static void
12 save_cb (GtkNotebook *nb) {
13   notebook_page_save (nb);
14 }
15
16 static void
17 close_cb (GtkNotebook *nb) {
18   notebook_page_close (GTK_NOTEBOOK (nb));
19 }
```

`open_cb`, `new_cb`, `save_cb` and `close_cb` just call corresponding notebook page functions.

15.7 meson.build

```
1  project('tfe', 'c')
2
3  gtkdep = dependency('gtk4')
4
5  gnome=import('gnome')
6  resources = gnome.compile_resources('resources','tfe.gresource.xml')
7
8  sourcefiles=files('tfeapplication.c', 'tfenotebook.c',
       '../tfetextview/tfetextview.c')
9
10 executable('tfe', sourcefiles, resources, dependencies: gtkdep)
```

In this file, just the source file names are modified from the prior version.

15.8 source files

The source files of the text editor `tfe` will be shown in the next section.

You can also download the files from the repository. There are two options.

- Use git and clone.
- Run your browser and open the top page. Then click on “Code” button and click “Download ZIP” in the popup menu. After that, unzip the archive file.

If you use git, run the terminal and type the following.

```
$ git clone https://github.com/ToshioCP/Gtk4-tutorial.git
```

The source files are under /src/tfe5 directory.

16 tfe5 source files

16.1 How to compile and execute the text editor ‘tfe’.

First, source files are shown in the later subsections. How to download them is written at the end of the previous section.

The following is the instruction of compilation and execution.

- You need meson and ninja.
- Set environment variables if necessary. If you have installed gtk4 from the source and you preferred the option `--prefix $HOME/local` (see Section 2), type `. env.sh` to set the environment variables.

```
$ . env.sh
```

- change your current directory to `src/tfe5` directory.
- type `meson _build` for configuration.
- type `ninja -C _build` for compilation. Then the application `tfe` is built under the `_build` directory.
- type `_build/tfe` to execute it.

Then the window appears. There are four buttons, `New`, `Open`, `Save` and `Close`.

- Click on `Open` button, then a FileChooserDialog appears. Choose a file in the list and click on `Open` button. Then the file is read and a new Notebook Page appears.
- Edit the file and click on `Save` button, then the text is saved to the original file.
- Click `Close`, then the Notebook Page disappears.
- Click `Close` again, then the `Untitled` Notebook Page disappears and at the same time the application quits.

This is a very simple editor. It is a good practice for you to add more features.

16.2 meson.build

```
1  project('tfe', 'c')
2
3  gtkdep = dependency('gtk4')
4
5  gnome=import('gnome')
6  resources = gnome.compile_resources('resources','tfe.gresource.xml')
7
8  sourcefiles=files('tfeapplication.c', 'tfenotebook.c',
       '../tfetextview/tfetextview.c')
9
10 executable('tfe', sourcefiles, resources, dependencies: gtkdep)
```

16.3 tfe.gresource.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <gresources>
3    <gresource prefix="/com/github/ToshioCP/tfe">
4      <file>tfe.ui</file>
5    </gresource>
6  </gresources>
```

16.4 tfe.ui

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <interface>
3    <object class="GtkApplicationWindow" id="win">
4      <property name="title">file editor</property>
5      <property name="default-width">600</property>
6      <property name="default-height">400</property>
7      <child>
8        <object class="GtkBox" id="boxv">
9          <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
10         <child>
11           <object class="GtkBox" id="boxh">
12             <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
13             <child>
14               <object class="GtkLabel" id="dmy1">
15                 <property name="width-chars">10</property>
16               </object>
```

```
            </child>
            <child>
              <object class="GtkButton" id="btnn">
                <property name="label">New</property>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btno">
                <property name="label">Open</property>
              </object>
            </child>
            <child>
              <object class="GtkLabel" id="dmy2">
                <property name="hexpand">TRUE</property>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btns">
                <property name="label">Save</property>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btnc">
                <property name="label">Close</property>
              </object>
            </child>
            <child>
              <object class="GtkLabel" id="dmy3">
                <property name="width-chars">10</property>
              </object>
            </child>
          </object>
        </child>
        <child>
          <object class="GtkNotebook" id="nb">
            <property name="scrollable">TRUE</property>
            <property name="hexpand">TRUE</property>
            <property name="vexpand">TRUE</property>
          </object>
        </child>
      </object>
    </child>
  </object>
</interface>
```

16.5 tfe.h

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

#include "../tfetextview/tfetextview.h"
#include "tfenotebook.h"
```

16.6 tfeapplication.c

```
#include "tfe.h"

static void
open_cb (GtkNotebook *nb) {
  notebook_page_open (nb);
}

static void
new_cb (GtkNotebook *nb) {
```

```
10    notebook_page_new (nb);
11  }
12
13  static void
14  save_cb (GtkNotebook *nb) {
15    notebook_page_save (nb);
16  }
17
18  static void
19  close_cb (GtkNotebook *nb) {
20    notebook_page_close (GTK_NOTEBOOK (nb));
21  }
22
23  static void
24  app_activate (GApplication *application) {
25    GtkApplication *app = GTK_APPLICATION (application);
26    GtkWidget *win = GTK_WIDGET (gtk_application_get_active_window (app));
27    GtkWidget *boxv = gtk_window_get_child (GTK_WINDOW (win));
28    GtkNotebook *nb = GTK_NOTEBOOK (gtk_widget_get_last_child (boxv));
29
30    notebook_page_new (nb);
31    gtk_widget_show (GTK_WIDGET (win));
32  }
33
34  static void
35  app_open (GApplication *application, GFile ** files, gint n_files, const gchar
        *hint) {
36    GtkApplication *app = GTK_APPLICATION (application);
37    GtkWidget *win = GTK_WIDGET (gtk_application_get_active_window (app));
38    GtkWidget *boxv = gtk_window_get_child (GTK_WINDOW (win));
39    GtkNotebook *nb = GTK_NOTEBOOK (gtk_widget_get_last_child (boxv));
40    int i;
41
42    for (i = 0; i < n_files; i++)
43      notebook_page_new_with_file (nb, files[i]);
44    if (gtk_notebook_get_n_pages (nb) == 0)
45      notebook_page_new (nb);
46    gtk_widget_show (win);
47  }
48
49  static void
50  app_startup (GApplication *application) {
51    GtkApplication *app = GTK_APPLICATION (application);
52    GtkBuilder *build;
53    GtkApplicationWindow *win;
54    GtkNotebook *nb;
55    GtkButton *btno;
56    GtkButton *btnn;
57    GtkButton *btns;
58    GtkButton *btnc;
59
60    build = gtk_builder_new_from_resource ("/com/github/ToshioCP/tfe/tfe.ui");
61    win = GTK_APPLICATION_WINDOW (gtk_builder_get_object (build, "win"));
62    nb = GTK_NOTEBOOK (gtk_builder_get_object (build, "nb"));
63    gtk_window_set_application (GTK_WINDOW (win), app);
64    btno = GTK_BUTTON (gtk_builder_get_object (build, "btno"));
65    btnn = GTK_BUTTON (gtk_builder_get_object (build, "btnn"));
66    btns = GTK_BUTTON (gtk_builder_get_object (build, "btns"));
67    btnc = GTK_BUTTON (gtk_builder_get_object (build, "btnc"));
68    g_signal_connect_swapped (btno, "clicked", G_CALLBACK (open_cb), nb);
69    g_signal_connect_swapped (btnn, "clicked", G_CALLBACK (new_cb), nb);
70    g_signal_connect_swapped (btns, "clicked", G_CALLBACK (save_cb), nb);
71    g_signal_connect_swapped (btnc, "clicked", G_CALLBACK (close_cb), nb);
72    g_object_unref(build);
```

```
73
74  GdkDisplay *display;
75
76    display = gtk_widget_get_display (GTK_WIDGET (win));
77    GtkCssProvider *provider = gtk_css_provider_new ();
78    gtk_css_provider_load_from_data (provider, "textview {padding: 10px; font-family:
          monospace; font-size: 12pt;}", -1);
79    gtk_style_context_add_provider_for_display (display, GTK_STYLE_PROVIDER
          (provider), GTK_STYLE_PROVIDER_PRIORITY_APPLICATION);
80  }
81
82  #define APPLICATION_ID "com.github.ToshioCP.tfe"
83
84  int
85  main (int argc, char **argv) {
86    GtkApplication *app;
87    int stat;
88
89    app = gtk_application_new (APPLICATION_ID, G_APPLICATION_HANDLES_OPEN);
90
91    g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
92    g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
93    g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);
94
95    stat =g_application_run (G_APPLICATION (app), argc, argv);
96    g_object_unref (app);
97    return stat;
98  }
```

16.7 tfenotebook.h

```
1  void
2  notebook_page_save(GtkNotebook *nb);
3
4  void
5  notebook_page_close (GtkNotebook *nb);
6
7  void
8  notebook_page_open (GtkNotebook *nb);
9
10 void
11 notebook_page_new_with_file (GtkNotebook *nb, GFile *file);
12
13 void
14 notebook_page_new (GtkNotebook *nb);
```

16.8 tfenotebook.c

```
1  #include "tfe.h"
2
3  /* The returned string should be freed with g_free() when no longer needed. */
4  static gchar*
5  get_untitled () {
6    static int c = -1;
7    if (++c == 0)
8      return g_strdup_printf("Untitled");
9    else
10     return g_strdup_printf ("Untitled%u", c);
11 }
12
13 static TfeTextView *
14 get_current_textview (GtkNotebook *nb) {
15   int i;
```

```
16    GtkWidget *scr;
17    GtkWidget *tv;
18
19    i = gtk_notebook_get_current_page (nb);
20    scr = gtk_notebook_get_nth_page (nb, i);
21    tv = gtk_scrolled_window_get_child (GTK_SCROLLED_WINDOW (scr));
22    return TFE_TEXT_VIEW (tv);
23  }
24
25  static void
26  file_changed_cb (TfeTextView *tv, GtkNotebook *nb) {
27    GtkWidget *scr;
28    GtkWidget *label;
29    GFile *file;
30    char *filename;
31
32    file = tfe_text_view_get_file (tv);
33    scr = gtk_widget_get_parent (GTK_WIDGET (tv));
34    if (G_IS_FILE (file)) {
35      filename = g_file_get_basename (file);
36      g_object_unref (file);
37    } else
38      filename = get_untitled ();
39    label = gtk_label_new (filename);
40    g_free (filename);
41    gtk_notebook_set_tab_label (nb, scr, label);
42  }
43
44  void
45  notebook_page_save (GtkNotebook *nb) {
46    g_return_if_fail(GTK_IS_NOTEBOOK (nb));
47
48    TfeTextView *tv;
49
50    tv = get_current_textview (nb);
51    tfe_text_view_save (TFE_TEXT_VIEW (tv));
52  }
53
54  void
55  notebook_page_close (GtkNotebook *nb) {
56    g_return_if_fail(GTK_IS_NOTEBOOK (nb));
57
58    GtkWidget *win;
59    int i;
60
61    if (gtk_notebook_get_n_pages (nb) == 1) {
62      win = gtk_widget_get_ancestor (GTK_WIDGET (nb), GTK_TYPE_WINDOW);
63      gtk_window_destroy(GTK_WINDOW (win));
64    } else {
65      i = gtk_notebook_get_current_page (nb);
66      gtk_notebook_remove_page (GTK_NOTEBOOK (nb), i);
67    }
68  }
69
70  static void
71  notebook_page_build (GtkNotebook *nb, GtkWidget *tv, char *filename) {
72    GtkWidget *scr = gtk_scrolled_window_new ();
73    GtkNotebookPage *nbp;
74    GtkWidget *lab;
75    int i;
76
77    gtk_text_view_set_wrap_mode (GTK_TEXT_VIEW (tv), GTK_WRAP_WORD_CHAR);
78    gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), tv);
79    lab = gtk_label_new (filename);
```

```
80    i = gtk_notebook_append_page (nb, scr, lab);
81    nbp = gtk_notebook_get_page (nb, scr);
82    g_object_set (nbp, "tab-expand", TRUE, NULL);
83    gtk_notebook_set_current_page (nb, i);
84    g_signal_connect (GTK_TEXT_VIEW (tv), "change-file", G_CALLBACK (file_changed_cb),
          nb);
85  }
86
87  static void
88  open_response (TfeTextView *tv, int response, GtkNotebook *nb) {
89    GFile *file;
90    char *filename;
91
92    if (response != TFE_OPEN_RESPONSE_SUCCESS || ! G_IS_FILE (file =
          tfe_text_view_get_file (tv))) {
93      g_object_ref_sink (tv);
94      g_object_unref (tv);
95    }else {
96      filename = g_file_get_basename (file);
97      g_object_unref (file);
98      notebook_page_build (nb, GTK_WIDGET (tv), filename);
99    }
100 }
101
102 void
103 notebook_page_open (GtkNotebook *nb) {
104   g_return_if_fail(GTK_IS_NOTEBOOK (nb));
105
106   GtkWidget *tv;
107
108   if ((tv = tfe_text_view_new ()) == NULL)
109     return;
110   g_signal_connect (TFE_TEXT_VIEW (tv), "open-response", G_CALLBACK (open_response),
          nb);
111   tfe_text_view_open (TFE_TEXT_VIEW (tv), GTK_WINDOW (gtk_widget_get_ancestor
          (GTK_WIDGET (nb), GTK_TYPE_WINDOW)));
112 }
113
114 void
115 notebook_page_new_with_file (GtkNotebook *nb, GFile *file) {
116   g_return_if_fail(GTK_IS_NOTEBOOK (nb));
117   g_return_if_fail(G_IS_FILE (file));
118
119   GtkWidget *tv;
120   char *filename;
121
122   if ((tv = tfe_text_view_new_with_file (file)) == NULL)
123     return; /* read error */
124   filename = g_file_get_basename (file);
125   notebook_page_build (nb, tv, filename);
126 }
127
128 void
129 notebook_page_new (GtkNotebook *nb) {
130   g_return_if_fail(GTK_IS_NOTEBOOK (nb));
131
132   GtkWidget *tv;
133   char *filename;
134
135   if ((tv = tfe_text_view_new ()) == NULL)
136     return;
137   filename = get_untitled ();
138   notebook_page_build (nb, tv, filename);
139 }
```

16.9 tfetextview.h

```
#ifndef __TFE_TEXT_VIEW_H__
#define __TFE_TEXT_VIEW_H__

#include <gtk/gtk.h>

#define TFE_TYPE_TEXT_VIEW tfe_text_view_get_type ()
G_DECLARE_FINAL_TYPE (TfeTextView, tfe_text_view, TFE, TEXT_VIEW, GtkTextView)

/* "open-response" signal response */
enum TfeTextViewOpenResponseType
{
  TFE_OPEN_RESPONSE_SUCCESS,
  TFE_OPEN_RESPONSE_CANCEL,
  TFE_OPEN_RESPONSE_ERROR
};

GFile *
tfe_text_view_get_file (TfeTextView *tv);

void
tfe_text_view_open (TfeTextView *tv, GtkWindow *win);

void
tfe_text_view_save (TfeTextView *tv);

void
tfe_text_view_saveas (TfeTextView *tv);

GtkWidget *
tfe_text_view_new_with_file (GFile *file);

GtkWidget *
tfe_text_view_new (void);

#endif /* __TFE_TEXT_VIEW_H__ */
```

16.10 tfetextview.c

```
#include <string.h>
#include "tfetextview.h"

struct _TfeTextView {
  GtkTextView parent;
  GFile *file;
};

G_DEFINE_TYPE (TfeTextView, tfe_text_view, GTK_TYPE_TEXT_VIEW);

enum {
  CHANGE_FILE,
  OPEN_RESPONSE,
  NUMBER_OF_SIGNALS
};

static guint tfe_text_view_signals[NUMBER_OF_SIGNALS];

static void
tfe_text_view_dispose (GObject *gobject) {
  TfeTextView *tv = TFE_TEXT_VIEW (gobject);

  if (G_IS_FILE (tv->file))
    g_clear_object (&tv->file);
```

```
25
26    G_OBJECT_CLASS (tfe_text_view_parent_class)->dispose (gobject);
27  }
28
29  static void
30  tfe_text_view_init (TfeTextView *tv) {
31    tv->file = NULL;
32  }
33
34  static void
35  tfe_text_view_class_init (TfeTextViewClass *class) {
36    GObjectClass *object_class = G_OBJECT_CLASS (class);
37
38    object_class->dispose = tfe_text_view_dispose;
39    tfe_text_view_signals[CHANGE_FILE] = g_signal_new ("change-file",
40                                    G_TYPE_FROM_CLASS (class),
41                                    G_SIGNAL_RUN_LAST | G_SIGNAL_NO_RECURSE |
                                        G_SIGNAL_NO_HOOKS,
42                                    0 /* class offset */,
43                                    NULL /* accumulator */,
44                                    NULL /* accumulator data */,
45                                    NULL /* C marshaller */,
46                                    G_TYPE_NONE /* return_type */,
47                                    0     /* n_params */
48                                    );
49    tfe_text_view_signals[OPEN_RESPONSE] = g_signal_new ("open-response",
50                                    G_TYPE_FROM_CLASS (class),
51                                    G_SIGNAL_RUN_LAST | G_SIGNAL_NO_RECURSE |
                                        G_SIGNAL_NO_HOOKS,
52                                    0 /* class offset */,
53                                    NULL /* accumulator */,
54                                    NULL /* accumulator data */,
55                                    NULL /* C marshaller */,
56                                    G_TYPE_NONE /* return_type */,
57                                    1     /* n_params */,
58                                    G_TYPE_INT
59                                    );
60  }
61
62  GFile *
63  tfe_text_view_get_file (TfeTextView *tv) {
64    g_return_val_if_fail (TFE_IS_TEXT_VIEW (tv), NULL);
65
66    if (G_IS_FILE (tv->file))
67      return g_file_dup (tv->file);
68    else
69      return NULL;
70  }
71
72  static gboolean
73  save_file (GFile *file, GtkTextBuffer *tb, GtkWindow *win) {
74    GtkTextIter start_iter;
75    GtkTextIter end_iter;
76    gchar *contents;
77    gboolean stat;
78    GtkWidget *message_dialog;
79    GError *err = NULL;
80
81    gtk_text_buffer_get_bounds (tb, &start_iter, &end_iter);
82    contents = gtk_text_buffer_get_text (tb, &start_iter, &end_iter, FALSE);
83    if (g_file_replace_contents (file, contents, strlen (contents), NULL, TRUE,
          G_FILE_CREATE_NONE, NULL, NULL, &err)) {
84      gtk_text_buffer_set_modified (tb, FALSE);
85      stat = TRUE;
```

```
86    } else {
87      message_dialog = gtk_message_dialog_new (win, GTK_DIALOG_MODAL,
88                                               GTK_MESSAGE_ERROR, GTK_BUTTONS_CLOSE,
89                                              "%s.\n", err->message);
90      g_signal_connect (message_dialog, "response", G_CALLBACK (gtk_window_destroy),
            NULL);
91      gtk_widget_show (message_dialog);
92      g_error_free (err);
93      stat = FALSE;
94    }
95    g_free (contents);
96    return stat;
97  }
98
99  static void
100 saveas_dialog_response (GtkWidget *dialog, gint response, TfeTextView *tv) {
101   GtkTextBuffer *tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
102   GFile *file;
103   GtkWidget *win = gtk_widget_get_ancestor (GTK_WIDGET (tv), GTK_TYPE_WINDOW);
104
105   if (response == GTK_RESPONSE_ACCEPT) {
106     file = gtk_file_chooser_get_file (GTK_FILE_CHOOSER (dialog));
107     if (! G_IS_FILE (file))
108       g_warning ("TfeTextView: gtk_file_chooser_get_file returns non GFile.\n");
109     else if (save_file(file, tb, GTK_WINDOW (win))) {
110       if (G_IS_FILE (tv->file))
111         g_object_unref (tv->file);
112       tv->file = file;
113       g_signal_emit (tv, tfe_text_view_signals[CHANGE_FILE], 0);
114     } else
115       g_object_unref (file);
116   }
117   gtk_window_destroy (GTK_WINDOW (dialog));
118 }
119
120 void
121 tfe_text_view_save (TfeTextView *tv) {
122   g_return_if_fail (TFE_IS_TEXT_VIEW (tv));
123
124   GtkTextBuffer *tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
125   GtkWidget *win = gtk_widget_get_ancestor (GTK_WIDGET (tv), GTK_TYPE_WINDOW);
126
127   if (! gtk_text_buffer_get_modified (tb))
128     return; /* no need to save it */
129   else if (tv->file == NULL)
130     tfe_text_view_saveas (tv);
131   else if (! G_IS_FILE (tv->file))
132     g_error ("TfeTextView: The pointer tv->file isn't NULL nor GFile.\n");
133   else
134     save_file (tv->file, tb, GTK_WINDOW (win));
135 }
136
137 void
138 tfe_text_view_saveas (TfeTextView *tv) {
139   g_return_if_fail (TFE_IS_TEXT_VIEW (tv));
140
141   GtkWidget *dialog;
142   GtkWidget *win = gtk_widget_get_ancestor (GTK_WIDGET (tv), GTK_TYPE_WINDOW);
143
144   dialog = gtk_file_chooser_dialog_new ("Save file", GTK_WINDOW (win),
          GTK_FILE_CHOOSER_ACTION_SAVE,
145                                         "Cancel", GTK_RESPONSE_CANCEL,
146                                         "Save", GTK_RESPONSE_ACCEPT,
147                                         NULL);
```

```
148   g_signal_connect (dialog, "response", G_CALLBACK (saveas_dialog_response), tv);
149   gtk_widget_show (dialog);
150 }
151
152 GtkWidget *
153 tfe_text_view_new_with_file (GFile *file) {
154   g_return_val_if_fail (G_IS_FILE (file), NULL);
155
156   GtkWidget *tv;
157   GtkTextBuffer *tb;
158   char *contents;
159   gsize length;
160
161   if (! g_file_load_contents (file, NULL, &contents, &length, NULL, NULL)) /* read
          error */
162     return NULL;
163
164   if ((tv = tfe_text_view_new()) != NULL) {
165     tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
166     gtk_text_buffer_set_text (tb, contents, length);
167     TFE_TEXT_VIEW (tv)->file = g_file_dup (file);
168     gtk_text_buffer_set_modified (tb, FALSE);
169   }
170   g_free (contents);
171   return tv;
172 }
173
174 static void
175 open_dialog_response(GtkWidget *dialog, gint response, TfeTextView *tv) {
176   GtkTextBuffer *tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
177   GFile *file;
178   char *contents;
179   gsize length;
180   GtkWidget *message_dialog;
181   GError *err = NULL;
182
183   if (response != GTK_RESPONSE_ACCEPT)
184     g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
          TFE_OPEN_RESPONSE_CANCEL);
185   else if (! G_IS_FILE (file = gtk_file_chooser_get_file (GTK_FILE_CHOOSER
          (dialog)))) {
186     g_warning ("TfeTextView: gtk_file_chooser_get_file returns non GFile.\n");
187     g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
          TFE_OPEN_RESPONSE_ERROR);
188   } else if (! g_file_load_contents (file, NULL, &contents, &length, NULL, &err)) {
        /* read error */
189     g_object_unref (file);
190     message_dialog = gtk_message_dialog_new (GTK_WINDOW (dialog), GTK_DIALOG_MODAL,
191                                              GTK_MESSAGE_ERROR, GTK_BUTTONS_CLOSE,
192                                             "%s.\n", err->message);
193     g_signal_connect (message_dialog, "response", G_CALLBACK (gtk_window_destroy),
          NULL);
194     gtk_widget_show (message_dialog);
195     g_error_free (err);
196     g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
          TFE_OPEN_RESPONSE_ERROR);
197   } else {
198     gtk_text_buffer_set_text (tb, contents, length);
199     g_free (contents);
200     if (G_IS_FILE (tv->file))
201       g_object_unref (tv->file);
202     tv->file = file;
203     gtk_text_buffer_set_modified (tb, FALSE);
204     g_signal_emit (tv, tfe_text_view_signals[OPEN_RESPONSE], 0,
```

```
        TFE_OPEN_RESPONSE_SUCCESS);
205     g_signal_emit (tv, tfe_text_view_signals[CHANGE_FILE], 0);
206   }
207   gtk_window_destroy (GTK_WINDOW (dialog));
208 }
209
210 void
211 tfe_text_view_open (TfeTextView *tv, GtkWindow *win) {
212   g_return_if_fail (TFE_IS_TEXT_VIEW (tv));
213   g_return_if_fail (GTK_IS_WINDOW (win));
214
215   GtkWidget *dialog;
216
217   dialog = gtk_file_chooser_dialog_new ("Open file", win,
        GTK_FILE_CHOOSER_ACTION_OPEN,
218                                         "Cancel", GTK_RESPONSE_CANCEL,
219                                         "Open", GTK_RESPONSE_ACCEPT,
220                                         NULL);
221   g_signal_connect (dialog, "response", G_CALLBACK (open_dialog_response), tv);
222   gtk_widget_show (dialog);
223 }
224
225 GtkWidget *
226 tfe_text_view_new (void) {
227   return GTK_WIDGET (g_object_new (TFE_TYPE_TEXT_VIEW, NULL));
228 }
```

16.11 Total number of lines, words and characters

```
$ LANG=C wc tfe5/meson.build tfe5/tfeapplication.c tfe5/tfe.gresource.xml tfe5/tfe.h
    tfe5/tfenotebook.c tfe5/tfenotebook.h tfetextview/tfetextview.c
    tfetextview/tfetextview.h tfe5/tfe.ui
   10    17   294 tfe5/meson.build
   99   304  3205 tfe5/tfeapplication.c
    6     9   153 tfe5/tfe.gresource.xml
    4     6    87 tfe5/tfe.h
  140   378  3601 tfe5/tfenotebook.c
   15    21   241 tfe5/tfenotebook.h
  229   671  8017 tfetextview/tfetextview.c
   35    60   701 tfetextview/tfetextview.h
   61   100  2073 tfe5/tfe.ui
  599  1566 18372 total
```

17 Menu and action

17.1 Menu

Users often use menus to tell a command to the computer. It is like this:

Now let's analyze the menu above. There are two types of object.

- "File", "Edit", "View", "Cut", "Copy", "Paste" and "Select All". They are called "menu item" or simply "item". When the user clicks one of these items, then something will happen.
- Menubar, submenu referenced by "Edit" item and two sections. They are called "menu". Menu is an ordered list of items. They are similar to arrays.

- Menubar is a menu which has three items, which are "File", "Edit" and "View".
- The menu item labeled "Edit" has a link to the submenu which has two items. These two items don't have labels. Each item refers to a section.
- The first section is a menu which has three items – "Cut", "Copy" and "Paste".
- The second section is a menu which has one item – "Select All".

Menus can build a complicated structure thanks to the links of menu items.

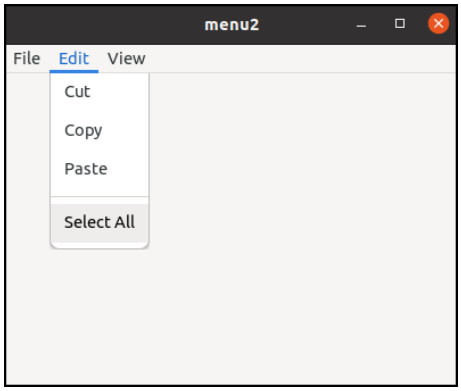

Figure 19: Menu

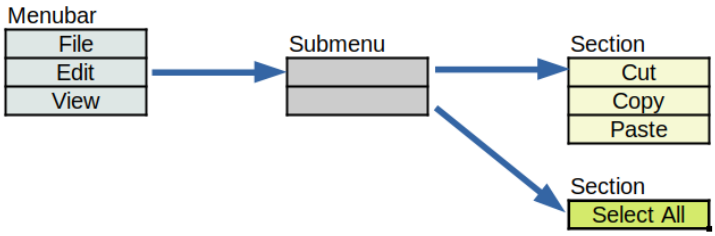

Figure 20: Menu structure

17.2 GMenuModel, GMenu and GMenuItem

GMenuModel is an abstract object which represents a menu. GMenu is a simple implementation of GMenuModel and a child object of GMenuModel.

```
GObject -- GMenuModel -- GMenu
```

Because GMenuModel is an abstract object, it isn't instantiatable. Therefore, it doesn't have any functions to create its instance. If you want to create a menu, use `g_menu_new` to create a GMenu instance. GMenu inherits all the functions of GMenuModel because of the child object.

GMenuItem is an object directly derived from GObject. GMenuItem and Gmenu (or GMenuModel) don't have a parent-child relationship.

```
GObject -- GMenuModel -- GMenu
GObject -- GMenuItem
```

GMenuItem has attributes. One of the attributes is label. For example, there is a menu item which has "Edit" label in the first diagram in this section. "Cut", "Copy", "Paste" and "Select All" are also the labels of the menu items. Other attributes will be explained later.

Some menu items have a link to another GMenu. There are two types of links, submenu and section.

GMenuItem can be inserted, appended or prepended to GMenu. When it is inserted, all of the attributes and link values of the item are copied and used to form a new item within the menu. The GMenuItem itself is not really inserted. Therefore, after the insertion, GMenuItem is useless and it should be freed. The same goes for appending or prepending.

The following code shows how to append GMenuItem to GMenu.

```
GMenu *menu = g_menu_new ();
GMenuItem *menu_item_quit = g_menu_item_new ("Quit", "app.quit");
g_menu_append_item (menu, menu_item_quit);
g_object_unref (menu_item_quit);
```

17.3 Menu and action

One of the attributes of menu items is an action. This attribute points an action object.

There are two action objects, GSimpleAction and GPropertyAction. GSimpleAction is often used. And it is used with a menu item. Only GSimpleAction is described in this section.

An action corresponds to a menu item will be activated when the menu item is clicked. Then the action emits an activate signal.

1. menu item is clicked.
2. The corresponding action is activated.
3. The action emits a signal.
4. The connected handler is invoked.

The following code is an example.

```
static void
quit_activated(GSimpleAction *action, GVariant *parameter, gpointer app) { ... ...
    ...}

GSimpleAction *act_quit = g_simple_action_new ("quit", NULL);
g_action_map_add_action (G_ACTION_MAP (app), G_ACTION (act_quit));
g_signal_connect (act_quit, "activate", G_CALLBACK (quit_activated), app);
GMenuItem *menu_item_quit = g_menu_item_new ("Quit", "app.quit");
```

1. `menu_item_quit` is a menu item. It has a label "Quit" and is connected to an action "app.quit". "app" is a prefix and "quit" is a name of the action. The prefix "app" means that the action belongs to a GtkApplication instance. If the menu is clicked, then the corresponding action "quit" which belongs to the GtkApplication will be activated.
2. `act_quit` is an action. It has a name "quit". The function `g_simple_action_new` creates a stateless action. So, `act_quit` is stateless. The meaning of stateless will be explained later. The argument `NULL` means that the action doesn't have an parameter. Most of the actions are stateless and have no parameter.
3. The action `act_quit` is added to the GtkApplication instance with `g_action_map_add_action`. When `act_quit` is activated, it will emit "activate" signal.
4. "activate" signal of the action is connected to the handler `quit_activated`. So, if the action is activated, the handler will be invoked.

17.4 Simple example

The following is a simple example of menus and actions.

```
1  #include <gtk/gtk.h>
2
3  static void
4  quit_activated(GSimpleAction *action, GVariant *parameter, gpointer user_data) {
5    GApplication *app = G_APPLICATION (user_data);
6
7    g_application_quit (app);
8  }
9
10 static void
11 app_activate (GApplication *app, gpointer user_data) {
12   GtkWidget *win = gtk_application_window_new (GTK_APPLICATION (app));
13   gtk_window_set_title (GTK_WINDOW (win), "menu1");
14   gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
15
16   GSimpleAction *act_quit = g_simple_action_new ("quit", NULL);
17   g_action_map_add_action (G_ACTION_MAP (app), G_ACTION (act_quit));
18   g_signal_connect (act_quit, "activate", G_CALLBACK (quit_activated), app);
19
20   GMenu *menubar = g_menu_new ();
21   GMenuItem *menu_item_menu = g_menu_item_new ("Menu", NULL);
22   GMenu *menu = g_menu_new ();
```

```
23    GMenuItem *menu_item_quit = g_menu_item_new ("Quit", "app.quit");
24    g_menu_append_item (menu, menu_item_quit);
25    g_object_unref (menu_item_quit);
26    g_menu_item_set_submenu (menu_item_menu, G_MENU_MODEL (menu));
27    g_menu_append_item (menubar, menu_item_menu);
28    g_object_unref (menu_item_menu);
29
30    gtk_application_set_menubar (GTK_APPLICATION (app), G_MENU_MODEL (menubar));
31    gtk_application_window_set_show_menubar (GTK_APPLICATION_WINDOW (win), TRUE);
32    gtk_window_present (GTK_WINDOW (win));
33  /*  gtk_widget_show (win); is also OKay instead of gtk_window_present. */
34  }
35
36  #define APPLICATION_ID "com.github.ToshioCP.menu1"
37
38  int
39  main (int argc, char **argv) {
40    GtkApplication *app;
41    int stat;
42
43    app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
44    g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
45
46    stat =g_application_run (G_APPLICATION (app), argc, argv);
47    g_object_unref (app);
48    return stat;
49  }
```

- 3-8: `quit_activated` is a handler of the "activate" signal on the action `act_quit`. Handlers of the "activate" signal have three parameters.
  1. The action instance on which the signal is emitted.
  2. Parameter. In this example it is `NULL` because the second argument of `g_simple_action_new` (line 15) is `NULL`. You don' t need to care about it.
  3. User data. It is the fourth parameter in the `g_signal_connect` (line 18) that connects the action and the handler.
- 7: A function `g_application_quit` immediately quits the application.
- 10-34: `app_activate` is a handler of "activate" signal on the GtkApplication instance.
- 12-14: Creates a GtkApplicationWindow `win`. And sets the title and the default size.
- 16: Creates GSimpleAction `act_quit`. It is stateless. The first argument of `g_simple_action_new` is a name of the action and the second argument is a parameter. If you don't need the parameter, pass `NULL`. Therefore, `act_quit` has a name "quit" and no parameter.
- 17: Adds the action to GtkApplication `app`. GtkApplication implements an interface GActionMap and GActionGroup. GtkApplication (GActionMap) can have a group of actions and the actions are added with the function `g_action_map_add_action`. This function is described in Gio API Reference, g_action_map_add_action.
- 18: Connects "activate" signal of the action and the handler `quit_activated`.
- 20-23: Creates GMenu and GMenuItem instances. `menubar` and `menu` are GMenu. `menu_item_menu` and `menu_item_quit` are GMenuItem. `menu_item_menu` has a label "Menu" and no action. `menu_item_quit` has a label "Quit" and an action "app.quit". The action "app.quit" is a combination of "app" and "quit". "app" is a prefix and it means that the action belongs to GtkApplication. "quit" is the name of the action. Therefore, "app.quit" points the action which belongs to the GtkApplication instance and is named "quit".
- 24-25: Appends `menu_item_quit` to `menu`. As I mentioned before, all the attributes and links are copied and used to form a new item in `menu`. Therefore after the appending, `menu_item_quit` is no longer needed. It is freed by `g_object_unref`.
- 26: Sets the submenu link in `menu_item_menu` to point `menu`.
- 27-28: Appends `menu_item_menu` to `menubar`. Then frees `menu_item_menu`. GMenu and GMenuItem are connected and finally a menu is made up. The structure of the menu is shown in the diagram below.
- 30: The menu is inserted to GtkApplication.
- 31: Sets GtkApplicationWindow to show the menubar.
- 32: Shows the window.

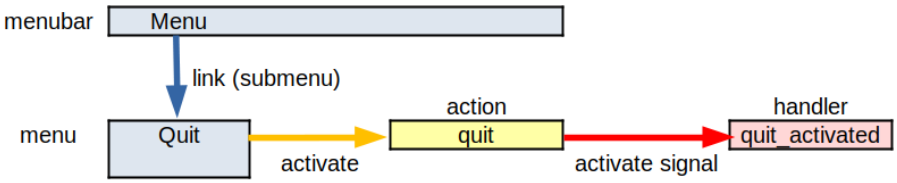

Figure 21: menu and action

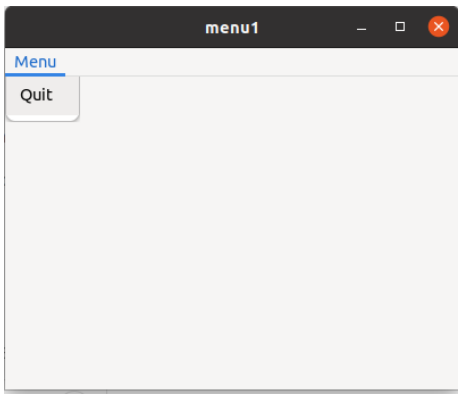

Figure 22: Screenshot of menu1

18 Stateful action

Some actions have states. The typical values of states is boolean or string. However, other types of states are possible if you want.

There's an example `menu2_int16.c` in the `src/men` directory. It behaves the same as `menu2.c`. But it uses gint16 type of states instead of string type.

Actions which have states are called stateful.

18.1 Stateful action without a parameter

Some menus are called toggle menu. For example, fullscreen menu has a state which has two values – fullscreen and non-fullscreen. The value of the state is changed every time the menu is clicked. An action corresponds to the fullscreen menu also have a state. Its value is TRUE or FALSE and it is called boolean value. TRUE corresponds to fullscreen and FALSE to non-fullscreen.

The following is an example code to implement a fullscreen menu except the signal handler. The signal handler will be described after the explanation of this code.

```
static void
app_activate (GApplication *app, gpointer user_data) {
  ... ... ...
  GSimpleAction *act_fullscreen = g_simple_action_new_stateful ("fullscreen",
                                  NULL, g_variant_new_boolean (FALSE));
  GMenuItem *menu_item_fullscreen = g_menu_item_new ("Full Screen",
      "win.fullscreen");
  g_signal_connect (act_fullscreen, "change-state", G_CALLBACK (fullscreen_changed),
      win);
  ... ... ...
}
```

- `act_fullscreen` is a GSimpleAction instance. It is created with `g_simple_action_new_stateful`. The function has three arguments. The first argument "fullscreen" is the name of the action. The second argument is a parameter type. `NULL` means the action doesn't have a parameter. The third argument is the initial state of the action. It is a GVariant value. GVariant will be explained in the next subsection.

The function `g_variant_new_boolean (FALSE)` returns a GVariant value which is the boolean value `FALSE`.

- `menu_item_fullscreen` is a GMenuItem instance. There are two arguments. The first argument "Full Screen" is a label of `menu_item_fullscreen`. The second argument is an action. The action "win.fullscreen" has a prefix "win" and an action name "fullscreen". The prefix says that the action belongs to the window.
- connects the action `act_fullscreen` and the "change-state" signal handler `fullscreen_changed`. If the fullscreen menu is clicked, then the corresponding action `act_fullscreen` is activated. But no handler is connected to the "activate" signal. Then, the default behavior for boolean-stated actions with a NULL parameter type like `act_fullscreen` is to toggle them via the "change-state" signal.

The following is the "change-state" signal handler.

```
static void
fullscreen_changed(GSimpleAction *action, GVariant *value, gpointer win) {
  if (g_variant_get_boolean (value))
    gtk_window_maximize (GTK_WINDOW (win));
  else
    gtk_window_unmaximize (GTK_WINDOW (win));
  g_simple_action_set_state (action, value);
}
```

- There are three parameters. The first parameter is the action which emits the "change-state" signal. The second parameter is the value of the new state of the action. The third parameter is a user data which is set in `g_signal_connect`.
- If the value is boolean type and `TRUE`, then it maximizes the window. Otherwise unmaximizes.
- Sets the state of the action with `value`. Note: the second argument was the toggled state value, but at this stage the state of the action has the original value. So, you need to set the state with the new value by `g_simple_action_set_state`.

You can use "activate" signal instead of "change-state" signal, or both signals. But the way above is the simplest and the best.

18.1.1 GVariant

GVarient can contain boolean, string or other type values. For example, the following program assigns TRUE to `value` whose type is GVariant.

```
GVariant *value = g_variant_new_boolean (TRUE);
```

Another example is:

```
GVariant *value2 = g_variant_new_string ("Hello");
```

`value2` is a GVariant and it has a string type value "Hello". GVariant can contain other types like int16, int32, int64, double and so on.

If you want to get the original value, use g_variant_get series functions. For example, you can get the boolean value by g_variant_get_boolean.

```
gboolean bool = g_variant_get_boolean (value);
```

Because `value` has been created as a boolean type GVariant and `TRUE` value, `bool` equals `TRUE`. In the same way, you can get a string from `value2`

```
const char *str = g_variant_get_string (value2, NULL);
```

The second parameter is a pointer to gsize type variable (gsize is defined as unsigned long). If it isn't NULL, then the length of the string will be set by the function. If it is NULL, nothing happens. The returned string `str` can't be changed.

18.2 Stateful action with a parameter

Another example of stateful actions is an action corresponds to color select menus. For example, there are three menus and each menu has red, green or blue color respectively. They determine the background color of a certain widget. One action is connected to the three menus. The action has a state which values are “red”, “green” and “blue”. The values are string. Those colors are given to the signal handler as a parameter.

```
static void
app_activate (GApplication *app, gpointer user_data) {
  ... ... ...
  GSimpleAction *act_color = g_simple_action_new_stateful ("color",
                    g_variant_type_new("s"), g_variant_new_string ("red"));
  GMenuItem *menu_item_red = g_menu_item_new ("Red", "win.color::red");
  GMenuItem *menu_item_green = g_menu_item_new ("Green", "win.color::green");
  GMenuItem *menu_item_blue = g_menu_item_new ("Blue", "win.color::blue");
  g_signal_connect (act_color, "activate", G_CALLBACK (color_activated), win);
  ... ... ...
}
```

- `act_color` is a GSimpleAction instance. It is created with `g_simple_action_new_stateful`. The function has three arguments. The first argument “color” is the name of the action. The second argument is a parameter type which is GVariantType. `g_variant_type_new("s")` creates GVariantType which is a string type (`G_VARIANT_TYPE_STRING`). The third argument is the initial state of the action. It is a GVariant. GVariantType will be explained in the next subsection. The function `g_variant_new_string ("red")` returns a GVariant value which has the string value “red”.
- `menu_item_red` is a GMenuItem instance. There are two arguments. The first argument “Red” is the label of `menu_item_red`. The second argument is a detailed action. Its prefix is “win”, action name is “color” and target is “red”. Target is sent to the action as a parameter. The same goes for `menu_item_green` and `menu_item_blue`.
- connects the action `act_color` and the “activate” signal handler `color_activated`. If one of the three menus is clicked, then the action `act_color` is activated with the target (parameter) which is given by the menu. No handler is connected to “change-state” signal. Then the default behavior is to call `g_simple_action_set_state()` to set the state to the requested value.

The following is the “activate” signal handler.

```
static void
color_activated(GSimpleAction *action, GVariant *parameter, gpointer win) {
  char *color = g_strdup_printf ("label#lb {background-color: %s;}",
                                 g_variant_get_string (parameter, NULL));
  gtk_css_provider_load_from_data (provider, color, -1);
  g_free (color);
  g_action_change_state (G_ACTION (action), parameter);
}
```

- There are three parameters. The first parameter is the action which emits the “activate” signal. The second parameter is the parameter given to the action. It is a color specified by the menu. The third parameter is a user data which is set in `g_signal_connect`.
- `color` is a CSS string created by `g_strdup_printf`. The parameter of `g_strdup_printf` is the same as printf C standard function. `g_variant_get_string` gets the string contained in `parameter`. You mustn’t change or free the string.
- Sets the color of the css provider.
- Frees the string `color`.
- Changes the state by `g_action_change_state`. The function just sets the state of the action to the parameter by `g_simple_action_set_state`. Therefore, you can use `g_simple_action_set_state` instead of `g_action_change_state`.

Note: If you have set a “change-state” signal handler, `g_action_change_state` will emit “change-state” signal instead of calling `g_simple_action_set_state`.

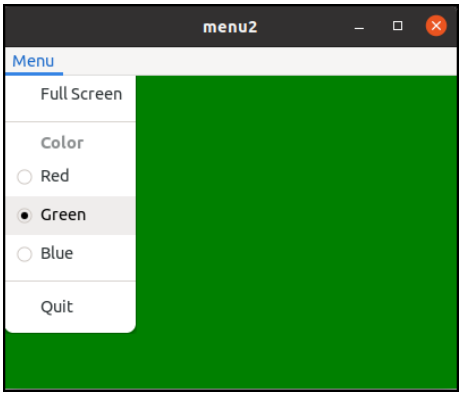

Figure 23: menu2

18.2.1 GVariantType

GVariantType gives a type of GVariant. GVariant can contain many kinds of types. And the type often needs to be recognized at runtime. GVariantType provides such functionality.

GVariantType is created with a string which expresses a type.

- “b” means boolean type.
- “s” means string type.

The following program is a simple example. It finally outputs the string “s”.

```
1  #include <glib.h>
2
3  int
4  main (int argc, char **argv) {
5    GVariantType *vtype = g_variant_type_new ("s");
6    const char *type_string = g_variant_type_peek_string (vtype);
7    g_print ("%s\n",type_string);
8  }
```

- `g_variant_type_new` creates GVariantType. It uses a type string “s” which means string.
- `g_variant_type_peek_string` takes a peek at `vtype`. It is the string “s” given to `vtype` when it was created.
- prints the string to the terminal.

18.3 Example code

The following code includes stateful actions above. This program has menus like this:

- Fullscreen menu toggles the size of the window between maximum and non-maximum. If the window is maximum size, which is called full screen, then a check mark is put before “fullscreen” label.
- Red, green and blue menu determines the back ground color of the label, which is the child widget of the window. The menus have radio buttons on the left of the menus. And the radio button of the selected menu turns on.
- Quit menu quits the application.

The code is as follows.

```
1  #include <gtk/gtk.h>
2
3  static GtkCssProvider *provider;
4
5  static void
6  fullscreen_changed(GSimpleAction *action, GVariant *value, gpointer win) {
7    if (g_variant_get_boolean (value))
8      gtk_window_maximize (GTK_WINDOW (win));
9    else
10     gtk_window_unmaximize (GTK_WINDOW (win));
```

```
11    g_simple_action_set_state (action, value);
12  }
13
14  static void
15  color_activated(GSimpleAction *action, GVariant *parameter, gpointer win) {
16    char *color = g_strdup_printf ("label#lb {background-color: %s;}",
          g_variant_get_string (parameter, NULL));
17    gtk_css_provider_load_from_data (provider, color, -1);
18    g_free (color);
19    g_action_change_state (G_ACTION (action), parameter);
20  }
21
22  static void
23  quit_activated(GSimpleAction *action, GVariant *parameter, gpointer app)
24  {
25    g_application_quit (G_APPLICATION(app));
26  }
27
28  static void
29  app_activate (GApplication *app, gpointer user_data) {
30    GtkWidget *win = gtk_application_window_new (GTK_APPLICATION (app));
31    gtk_window_set_title (GTK_WINDOW (win), "menu2");
32    gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
33
34    GtkWidget *lb = gtk_label_new (NULL);
35    gtk_widget_set_name (lb, "lb"); /* the name is used by CSS Selector */
36    gtk_window_set_child (GTK_WINDOW (win), lb);
37
38    GSimpleAction *act_fullscreen
39      = g_simple_action_new_stateful ("fullscreen", NULL, g_variant_new_boolean
            (FALSE));
40    GSimpleAction *act_color
41      = g_simple_action_new_stateful ("color", g_variant_type_new("s"),
            g_variant_new_string ("red"));
42    GSimpleAction *act_quit
43      = g_simple_action_new ("quit", NULL);
44
45    GMenu *menubar = g_menu_new ();
46    GMenu *menu = g_menu_new ();
47    GMenu *section1 = g_menu_new ();
48    GMenu *section2 = g_menu_new ();
49    GMenu *section3 = g_menu_new ();
50    GMenuItem *menu_item_fullscreen = g_menu_item_new ("Full Screen",
          "win.fullscreen");
51    GMenuItem *menu_item_red = g_menu_item_new ("Red", "win.color::red");
52    GMenuItem *menu_item_green = g_menu_item_new ("Green", "win.color::green");
53    GMenuItem *menu_item_blue = g_menu_item_new ("Blue", "win.color::blue");
54    GMenuItem *menu_item_quit = g_menu_item_new ("Quit", "app.quit");
55
56    g_signal_connect (act_fullscreen, "change-state", G_CALLBACK (fullscreen_changed),
          win);
57    g_signal_connect (act_color, "activate", G_CALLBACK (color_activated), win);
58    g_signal_connect (act_quit, "activate", G_CALLBACK (quit_activated), app);
59    g_action_map_add_action (G_ACTION_MAP (win), G_ACTION (act_fullscreen));
60    g_action_map_add_action (G_ACTION_MAP (win), G_ACTION (act_color));
61    g_action_map_add_action (G_ACTION_MAP (app), G_ACTION (act_quit));
62
63    g_menu_append_item (section1, menu_item_fullscreen);
64    g_menu_append_item (section2, menu_item_red);
65    g_menu_append_item (section2, menu_item_green);
66    g_menu_append_item (section2, menu_item_blue);
67    g_menu_append_item (section3, menu_item_quit);
68    g_object_unref (menu_item_red);
69    g_object_unref (menu_item_green);
```

```
70    g_object_unref (menu_item_blue);
71    g_object_unref (menu_item_fullscreen);
72    g_object_unref (menu_item_quit);
73
74    g_menu_append_section (menu, NULL, G_MENU_MODEL (section1));
75    g_menu_append_section (menu, "Color", G_MENU_MODEL (section2));
76    g_menu_append_section (menu, NULL, G_MENU_MODEL (section3));
77    g_menu_append_submenu (menubar, "Menu", G_MENU_MODEL (menu));
78
79    gtk_application_set_menubar (GTK_APPLICATION (app), G_MENU_MODEL (menubar));
80    gtk_application_window_set_show_menubar (GTK_APPLICATION_WINDOW (win), TRUE);
81
82  /*  GtkCssProvider *provider = gtk_css_provider_new ();*/
83    provider = gtk_css_provider_new ();
84    GdkDisplay *display = gtk_widget_get_display (GTK_WIDGET (win));
85    gtk_css_provider_load_from_data (provider, "label#lb {background-color: red;}",
          -1);
86    gtk_style_context_add_provider_for_display (display, GTK_STYLE_PROVIDER (provider),
87                                                GTK_STYLE_PROVIDER_PRIORITY_USER);
88
89  /*  gtk_widget_show (win);*/
90    gtk_window_present (GTK_WINDOW (win));
91  }
92
93  #define APPLICATION_ID "com.github.ToshioCP.menu2"
94
95  int
96  main (int argc, char **argv) {
97    GtkApplication *app;
98    int stat;
99
100   app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
101   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
102
103   stat =g_application_run (G_APPLICATION (app), argc, argv);
104   g_object_unref (app);
105   return stat;
106 }
```

- 5-26: Signal handlers. They have already been explained.
- 30-36: `win` and `lb` are GtkApplicationWindow and GtkLabel respectively. `win` has a title "menu2" and its default size is 400x300. `lb` is named as "lb". The name is used in CSS. `lb` is set to `win` as a child.
- 38-43: Three actions are defined. They are:
  - stateful and has no parameter. It has a toggle state.
  - stateful and has a parameter. Parameter is a string type.
  - stateless and has no parameter.
- 45-54: Creates GMenu and GMenuItem. There are three sections.
- 56-61: Signals are connected to handlers. And actions are added to GActionMap. Because `act_fullscreen` and `act_color` have "win" prefix and belong to GtkApplicationWindow, they are added to `win`. GtkApplicationWindow implements GActionModel interface like GtkApplication. `act_quit` has "app" prefix and belongs to GtkApplication. It is added to `app`.
- 63-77: Connects and builds the menus. Useless GMenuItem are freed.
- 79-80: GMenuModel `menubar` is inserted to `app`. Sets show menubar property of `win` to `TRUE`. Note: `gtk_application_window_set_show_menubar` creates GtkPopoverMenubar from GMenuModel. This is a different point between Gtk3 and Gtk4. And you can use GtkPopoverMenubar directly and set it as a descendant widget of the window. You may use GtkBox as a child widget of the window and insert GtkPopoverMenubar as the first child of the box.
- 82-87: Sets CSS. `provider` is GtkCssProvider which is defined in line three as a static variable. Its CSS data is: `label#lb {background-color: red;}`. "label#lb" is called selector. "label" is the node of GtkLabel. "#" precedes an ID which is an identifiable name of the widget. "lb" is the name of GtkLabel `lb`. (See line 35). The style is surrounded by open and close braces. The style is applied to GtkLabel which has a name "lb". Other GtkLabel have no effect from this. The provider is added to

GdkDisplay.
- 90: Shows the window.

19 Ui file for menu and action entries

19.1 Ui file for menu

You might have thought that building menus is really bothersome. Yes, the program was complicated and it needs lots of time to code it. The situation is similar to building widgets. When we built widgets, using ui file was a good way to avoid such complicated coding. The same goes for menus.

The ui file for menus has interface, menu tags. The file starts and ends with interface tag.

```
<interface>
  <menu id="menubar">
  </menu>
</interface>
```

`menu` tag corresponds to GMenu object. `id` attribute defines the name of the object. It will be referred by GtkBuilder.

```
<submenu>
  <attribute name="label">File</attribute>
    <item>
      <attribute name="label">New</attribute>
      <attribute name="action">win.new</attribute>
    </item>
</submenu>
```

`item` tag corresponds to item in GMenu which has the same structure as GMenuItem. The item above has a label attribute. Its value is "New". The item also has an action attribute and its value is "win.new". "win" is a prefix and "new" is an action name. `submenu` tag corresponds to both GMenuItem and GMenu. The GMenuItem has a link to GMenu.

The ui file above can be described as follows.

```
<item>
  <attribute name="label">File</attribute>
    <link name="submenu">
      <item>
        <attribute name="label">New</attribute>
        <attribute name="action">win.new</attribute>
      </item>
    </link>
</item>
```

`link` tag expresses the link to submenu. And at the same time it also expresses the submenu itself. This file illustrates the relationship between the menus and items better than the prior ui file. But `submenu` tag is simple and easy to understand. So, we usually prefer the former ui file style.

The following is a screenshot of the sample program in this section. Its name is `menu3`.

The following is the ui file of the menu in `menu3`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <interface>
3    <menu id="menubar">
4      <submenu>
5        <attribute name="label">File</attribute>
6        <section>
7          <item>
8            <attribute name="label">New</attribute>
9            <attribute name="action">win.new</attribute>
10         </item>
11         <item>
```

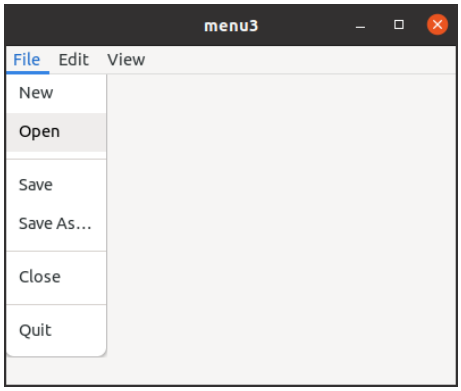

Figure 24: menu3

```
12            <attribute name="label">Open</attribute>
13            <attribute name="action">win.open</attribute>
14          </item>
15        </section>
16        <section>
17          <item>
18            <attribute name="label">Save</attribute>
19            <attribute name="action">win.save</attribute>
20          </item>
21          <item>
22            <attribute name="label">Save …As</attribute>
23            <attribute name="action">win.saveas</attribute>
24          </item>
25        </section>
26        <section>
27          <item>
28            <attribute name="label">Close</attribute>
29            <attribute name="action">win.close</attribute>
30          </item>
31        </section>
32        <section>
33          <item>
34            <attribute name="label">Quit</attribute>
35            <attribute name="action">app.quit</attribute>
36          </item>
37        </section>
38      </submenu>
39      <submenu>
40        <attribute name="label">Edit</attribute>
41        <section>
42          <item>
43            <attribute name="label">Cut</attribute>
44            <attribute name="action">win.cut</attribute>
45          </item>
46          <item>
47            <attribute name="label">Copy</attribute>
48            <attribute name="action">win.copy</attribute>
49          </item>
50          <item>
51            <attribute name="label">Paste</attribute>
52            <attribute name="action">win.paste</attribute>
53          </item>
54        </section>
55        <section>
56          <item>
57            <attribute name="label">Select All</attribute>
58            <attribute name="action">win.selectall</attribute>
```

```
        </item>
      </section>
    </submenu>
    <submenu>
      <attribute name="label">View</attribute>
      <section>
        <item>
          <attribute name="label">Full Screen</attribute>
          <attribute name="action">win.fullscreen</attribute>
        </item>
      </section>
    </submenu>
  </menu>
</interface>
```

The ui file is converted to the resource by the resource compiler `glib-compile-resouces` with xml file below.

```
<?xml version="1.0" encoding="UTF-8"?>
<gresources>
  <gresource prefix="/com/github/ToshioCP/menu3">
    <file>menu3.ui</file>
  </gresource>
</gresources>
```

GtkBuilder builds menus from the resource.

```
GtkBuilder *builder = gtk_builder_new_from_resource
    ("/com/github/ToshioCP/menu3/menu3.ui");
GMenuModel *menubar = G_MENU_MODEL (gtk_builder_get_object (builder, "menubar"));

gtk_application_set_menubar (GTK_APPLICATION (app), menubar);
g_object_unref (builder);
```

It is important that `builder` is unreferred after the GMenuModel `menubar` is inserted to the application. If you do it before setting, bad thing will happen – your computer might freeze.

19.2 Action entry

The coding for building actions and signal handlers is bothersome work as well. Therefore, it should be automated. You can implement them easily with GActionEntry structure and `g_action_map_add_action_entries` function.

GActionEntry contains action name, signal handlers, parameter and state.

```
typedef struct _GActionEntry GActionEntry;

struct _GActionEntry
{
  /* action name */
  const gchar *name;
  /* activate handler */
  void (* activate) (GSimpleAction *action, GVariant *parameter, gpointer user_data);
  /* the type of the parameter given as a single GVariant type string */
  const gchar *parameter_type;
  /* initial state given in GVariant text format */
  const gchar *state;
  /* change-state handler */
  void (* change_state) (GSimpleAction *action, GVariant *value, gpointer user_data);
  /*< private >*/
  gsize padding[3];
};
```

For example, the actions in the previous section are:

```
{ "fullscreen", NULL, NULL, "false", fullscreen_changed }
{ "color", color_activated, "s", "red", NULL }
{ "quit", quit_activated, NULL, NULL, NULL },
```

And `g_action_map_add_action_entries` does all the process instead of the functions you have needed.

```
const GActionEntry app_entries[] = {
  { "quit", quit_activated, NULL, NULL, NULL }
};
g_action_map_add_action_entries (G_ACTION_MAP (app), app_entries,
                                 G_N_ELEMENTS (app_entries), app);
```

The code above does:

- Builds the "quit" action
- Connects the action and the "activate" signal handler `quit_activated`
- Adds the action to the action map `app`.

The same goes for the other actions.

```
const GActionEntry win_entries[] = {
  { "fullscreen", NULL, NULL, "false", fullscreen_changed },
  { "color", color_activated, "s", "red", NULL }
};
g_action_map_add_action_entries (G_ACTION_MAP (win), win_entries,
                                 G_N_ELEMENTS (win_entries), win);
```

The code above does:

- Builds a "fullscreen" action and "color" action.
- Connects the "fullscreen" action and the "change-state" signal handler `fullscreen_changed`
- Its initial state is set to FALSE.
- Connects the "color" action and the "activate" signal handler `color_activated`
- Its parameter type is string and the initial value is "red".
- Adds the actions to the action map `win`.

19.3 Example code

The C source code of `menu3` and `meson.build` is as follows.

```
1  #include <gtk/gtk.h>
2
3  static void
4  new_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
5  }
6
7  static void
8  open_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
9  }
10
11 static void
12 save_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
13 }
14
15 static void
16 saveas_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
17 }
18
19 static void
20 close_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
21 }
22
23 static void
24 cut_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
25 }
```

```
26
27  static void
28  copy_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
29  }
30
31  static void
32  paste_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
33  }
34
35  static void
36  selectall_activated (GSimpleAction *action, GVariant *parameter, gpointer win) {
37  }
38
39  static void
40  fullscreen_changed (GSimpleAction *action, GVariant *state, gpointer win) {
41    if (g_variant_get_boolean (state))
42      gtk_window_maximize (GTK_WINDOW (win));
43    else
44      gtk_window_unmaximize (GTK_WINDOW (win));
45    g_simple_action_set_state (action, state);
46  }
47
48  static void
49  quit_activated (GSimpleAction *action, GVariant *parameter, gpointer app)
50  {
51    g_application_quit (G_APPLICATION(app));
52  }
53
54  static void
55  app_activate (GApplication *app, gpointer user_data) {
56    GtkWidget *win = gtk_application_window_new (GTK_APPLICATION (app));
57
58    const GActionEntry win_entries[] = {
59      { "new", new_activated, NULL, NULL, NULL },
60      { "open", open_activated, NULL, NULL, NULL },
61      { "save", save_activated, NULL, NULL, NULL },
62      { "saveas", saveas_activated, NULL, NULL, NULL },
63      { "close", close_activated, NULL, NULL, NULL },
64      { "cut", cut_activated, NULL, NULL, NULL },
65      { "copy", copy_activated, NULL, NULL, NULL },
66      { "paste", paste_activated, NULL, NULL, NULL },
67      { "selectall", selectall_activated, NULL, NULL, NULL },
68      { "fullscreen", NULL, NULL, "false", fullscreen_changed }
69    };
70    g_action_map_add_action_entries (G_ACTION_MAP (win), win_entries, G_N_ELEMENTS
          (win_entries), win);
71
72    gtk_application_window_set_show_menubar (GTK_APPLICATION_WINDOW (win), TRUE);
73
74    gtk_window_set_title (GTK_WINDOW (win), "menu3");
75    gtk_window_set_default_size (GTK_WINDOW (win), 400, 300);
76    gtk_widget_show (win);
77  }
78
79  static void
80  app_startup (GApplication *app, gpointer user_data) {
81    GtkBuilder *builder = gtk_builder_new_from_resource
          ("/com/github/ToshioCP/menu3/menu3.ui");
82    GMenuModel *menubar = G_MENU_MODEL (gtk_builder_get_object (builder, "menubar"));
83
84    gtk_application_set_menubar (GTK_APPLICATION (app), menubar);
85    g_object_unref (builder);
86
87    const GActionEntry app_entries[] = {
```

```
    { "quit", quit_activated, NULL, NULL, NULL }
  };
  g_action_map_add_action_entries (G_ACTION_MAP (app), app_entries, G_N_ELEMENTS
      (app_entries), app);
}

#define APPLICATION_ID "com.github.ToshioCP.menu3"

int
main (int argc, char **argv) {
  GtkApplication *app;
  int stat;

  app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
  g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
  g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);

  stat =g_application_run (G_APPLICATION (app), argc, argv);
  g_object_unref (app);
  return stat;
}
```

meson.build

```
project('menu3', 'c')

gtkdep = dependency('gtk4')

gnome=import('gnome')
resources = gnome.compile_resources('resources','menu3.gresource.xml')

sourcefiles=files('menu3.c')

executable('menu3', sourcefiles, resources, dependencies: gtkdep)
```

20 GtkMenuButton, accelerators, font, pango and gsettings

Traditional menu structure is fine. However, buttons or menu items we often use are not so many. Some mightn't be clicked at all. Therefore, it's a good idea to put some frequently used buttons on the toolbar and put the rest of the less frequently used operations into the menu. Such menu are often connected to GtkMenuButton.

We will restructure tfe text file editor in this section. It will be more practical. The buttons are changed to:

- Put open, save and close buttons to the toolbar. In addition, GtkMenuButton is added to the toolbar. This button shows a popup menu when clicked on. Here, popup means widely, including pull-down menu.
- Put new, save as, preference and quit items to the menu under the menu button.

20.1 Signal elements in ui files

The four buttons are included in the ui file `tfe.ui`. The difference from prior sections is signal tag. The following is extracted from `tfe.ui` and it describes the open button.

```
<object class="GtkButton" id="btno">
  <property name="label">Open</property>
  <signal name="clicked" handler="open_cb" swapped="TRUE" object="nb"></signal>
</object>
```

Signal tag specifies the name of the signal, handler and user_data object. They are the value of name, handler and object attributes. Swapped attribute has the same meaning as `g_signal_connect_swapped` function. So, the signal tag above works the same as the function below.

```
g_signal_connect_swapped (btno, "clicked", G_CALLBACK (open_cb), nb);
```

You need to compile the source file with "-WI, –export-dynamic" options. You can achieve this by adding "export_dynamic: true" argument to executable function in `meson.build`. And remove static class from the handler.

```
void
open_cb (GtkNotebook *nb) {
  notebook_page_open (nb);
}
```

If you add static, the function is in the scope of the file and it can't be seen from outside. Then the signal tag can't find the function.

20.2 Menu and GkMenuButton

Menus are described in `menu.ui` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <menu id="menu">
    <section>
      <item>
        <attribute name="label">New</attribute>
        <attribute name="action">win.new</attribute>
      </item>
      <item>
        <attribute name="label">Save …As</attribute>
        <attribute name="action">win.saveas</attribute>
      </item>
    </section>
    <section>
      <item>
        <attribute name="label">Preference</attribute>
        <attribute name="action">win.pref</attribute>
      </item>
    </section>
    <section>
      <item>
        <attribute name="label">Quit</attribute>
        <attribute name="action">win.close-all</attribute>
      </item>
    </section>
  </menu>
</interface>
```

There are four items, "New", "Saveas", "Preference" and "Quit".

- "New" menu creates a new empty page.
- "Saveas" menu saves the current page as a new filename.
- "Preference" menu sets preference items. This version of `tfe` has only font preference.
- "Quit" menu quits the application.

These four menus are not used so often. That's why they are put to the menu behind the menu button.

The menus and the menu button are connected with `gtk_menu_button_set_menu_model` function. The variable `btnm` below points a GtkMenuButton object.

```
build = gtk_builder_new_from_resource ("/com/github/ToshioCP/tfe/menu.ui");
menu = G_MENU_MODEL (gtk_builder_get_object (build, "menu"));
gtk_menu_button_set_menu_model (btnm, menu);
```

20.3 Actions and Accelerators

Menus are connected to actions. Actions are defined with an array and `g_action_map_add_action_entries` function.

```
const GActionEntry win_entries[] = {
  { "open", open_activated, NULL, NULL, NULL },
  { "save", save_activated, NULL, NULL, NULL },
  { "close", close_activated, NULL, NULL, NULL },
  { "new", new_activated, NULL, NULL, NULL },
  { "saveas", saveas_activated, NULL, NULL, NULL },
  { "pref", pref_activated, NULL, NULL, NULL },
  { "close-all", quit_activated, NULL, NULL, NULL }
};
g_action_map_add_action_entries (G_ACTION_MAP (win), win_entries, G_N_ELEMENTS
    (win_entries), nb);
```

There are seven actions, open, save, close, new, saveas, pref and close-all. But there were only four menus. New, saveas, pref and close-all actions correspond to new, saveas, preference and quit menu respectively. The three actions open, save and close doesn't have corresponding menus. Are thy necessary? These actions are defined because of accelerators.

Accelerators are a kind of short cut key function. They are defined with arrays and `gtk_application_set_accels_for_action` function.

```
struct {
  const char *action;
  const char *accels[2];
} action_accels[] = {
  { "win.open", { "<Control>o", NULL } },
  { "win.save", { "<Control>s", NULL } },
  { "win.close", { "<Control>w", NULL } },
  { "win.new", { "<Control>n", NULL } },
  { "win.saveas", { "<Shift><Control>s", NULL } },
  { "win.close-all", { "<Control>q", NULL } },
};

for (i = 0; i < G_N_ELEMENTS(action_accels); i++)
  gtk_application_set_accels_for_action(GTK_APPLICATION(app),
      action_accels[i].action, action_accels[i].accels);
```

This code is a bit complicated. The array `action-accels[]` is an array of structures. The structure is:

```
struct {
  const char *action;
  const char *accels[2];
}
```

The member `action` is a string. The member `accels` is an array of two strings. For example,

```
{ "win.open", { "<Control>o", NULL } },
```

This is the first element of the array `action_accels`.

- The member `action` is "win.open". This specifies the action "open" belongs to the window object.
- The member `accels` is an array of two strings, "<Control>o" and NULL. The first string specifies a key combination. Control key and 'o'. If you keep pressing the control key and push 'o' key, then it activates the action `win.open`. The second string NULL (or zero) means the end of the list (array). You can define more than one accelerator keys and the list must ends with NULL (zero). If you want to do so, the array length needs to be three or more. The parser recognizes "<control>o", "<Shift><Alt>F2", "<Ctrl>minus" and so on. If you want to use symbol key like "<Ctrl>-", use "<Ctrl>minus" instead. Such relation between lower case and symbol (its character code) is specified in `gdkkeysyms.h` in the Gtk4 source code.

20.4 Saveas handler

TfeTextView has already had a saveas function. So, only we need to write is the wrapper function in `tfenotebook.c`.

```
static TfeTextView *
get_current_textview (GtkNotebook *nb) {
  int i;
  GtkWidget *scr;
  GtkWidget *tv;

  i = gtk_notebook_get_current_page (nb);
  scr = gtk_notebook_get_nth_page (nb, i);
  tv = gtk_scrolled_window_get_child (GTK_SCROLLED_WINDOW (scr));
  return TFE_TEXT_VIEW (tv);
}

void
notebook_page_saveas (GtkNotebook *nb) {
  g_return_if_fail(GTK_IS_NOTEBOOK (nb));

  TfeTextView *tv;

  tv = get_current_textview (nb);
  tfe_text_view_saveas (TFE_TEXT_VIEW (tv));
}
```

The function `get_current_textview` is the same as before. The function `notebook_page_saveas` simply calls `tfe_text_view_saveas`.

In `tfeapplication.c`, saveas handler just call `notebook_page_saveas`.

```
static void
saveas_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
  GtkNotebook *nb = GTK_NOTEBOOK (user_data);
  notebook_page_saveas (nb);
}
```

20.5 Preference and alert dialog

20.5.1 Preference dialog

Preference dialog xml definition is added to `tfe.ui`.

```
<object class="GtkDialog" id="pref">
  <property name="title">Preferences</property>
  <property name="resizable">FALSE</property>
  <property name="modal">TRUE</property>
  <property name="transient-for">win</property>
  <child internal-child="content_area">
    <object class="GtkBox" id="content_area">
      <child>
        <object class="GtkBox" id="pref_boxh">
          <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
          <property name="spacing">12</property>
          <property name="margin-start">12</property>
          <property name="margin-end">12</property>
          <property name="margin-top">12</property>
          <property name="margin-bottom">12</property>
          <child>
            <object class="GtkLabel" id="fontlabel">
              <property name="label">Font:</property>
              <property name="xalign">1</property>
            </object>
          </child>
          <child>
            <object class="GtkFontButton" id="fontbtn">
            </object>
          </child>
        </object>
```

```
      </child>
    </object>
  </child>
</object>
```

- Preference dialog is an independent dialog. It is not a descendant widget of the top-level GtkApplicationwindow `win`. Therefore, There's no child tag that surrounds the dialog object.
- There are four properties of the dialog. GtkDialog is a child object (not child widget) of GtkWindow, so it inherits all the properties from GtkWindow. Title, resizable, modal and transient-for properties are inherited from GtkWindow. Transient-for specifies a temporary parent window, which the dialog's location is based on.
- internal-child attribute is used in the child tag above. GtkDialog has a GtkBox child widget. Its id is "content_area" in `gtkdialog.ui`, which is the ui file of GtkDialog. (It is in the Gtk4 source files.) This box is provided for users to add content widgets in it. The tag `<child internal-child="content_area">` is put at the top of the contents. Then you need to specify an object tag and define its class as GtkBox and its id as content_area. This object is defined in `gtkdialog.ui` but you need to define it again in the child tag.
- In the content area, defines GtkBox, GtkLabel and GtkFontButton.

I want the preference dialog to keep alive during the application lives. So, it is necessary to catch "close-request" signal from the dialog and stop the signal propagation. This is accomplished by returning TRUE by the signal handler.

```
pref_close_cb (GtkDialog *pref, gpointer user_data) {
  return TRUE;
}

g_signal_connect (GTK_DIALOG (pref), "close-request", G_CALLBACK (pref_close_cb),
    NULL);
```

Generally, signal emission consists of five stages.

1. Default handler is invoked if the signal's flag is `G_SIGNAL_RUN_FIRST`. Default handler is set when a signal is registered. It is different from user signal handler, simply called signal handler, connected by `g_signal_connect`series function. Default handler can be invoked in either stage 1, 3 or 5. Most of the default handlers are `G_SIGNAL_RUN_FIRST` or `G_SIGNAL_RUN_LAST`.
2. Signal handlers are invoked, unless it is connected by `g_signal_connect_after`.
3. Default handler is invoked if the signal's flag is `G_SIGNAL_RUN_LAST`.
4. Signal handlers are invoked, if it is connected by `g_signal_connect_after`.
5. Default handler is invoked if the signal's flag is `G_SIGNAL_RUN_CLEANUP`.

In the case of "close-request" signal, the default handler's flag is `G_SIGNAL_RUN_LAST`. The handler `pref_close_cb` is not connected by `g_signal_connect_after`. So the number of stages are two.

1. Signal handler `pref_close_cb` is invoked.
2. Default handler is invoked.

And If the user signal handler returns TRUE, then other handlers will be stopped being invoked. Therefore, the program above prevents the invocation of the default handler and stop the closing process of the dialog.

The following codes are extracted from `tfeapplication.c`.

```
static gulong pref_close_request_handler_id = 0;
static gulong alert_close_request_handler_id = 0;

... ...

static gboolean
dialog_close_cb (GtkDialog *dialog, gpointer user_data) {
  gtk_widget_hide (GTK_WIDGET (dialog));
  return TRUE;
}

... ...
```

```
static void
pref_activated (GSimpleAction *action, GVariant *parameter, gpointer nb) {
  gtk_widget_show (GTK_WIDGET (pref));
}

... ...

/* ----- quit application ----- */
void
tfe_application_quit (GtkWindow *win) {
  if (pref_close_request_handler_id > 0)
    g_signal_handler_disconnect (pref, pref_close_request_handler_id);
  if (alert_close_request_handler_id > 0)
    g_signal_handler_disconnect (alert, alert_close_request_handler_id);
  g_clear_object (&settings);
  gtk_window_destroy (GTK_WINDOW (alert));
  gtk_window_destroy (GTK_WINDOW (pref));
  gtk_window_destroy (win);
}

... ...

static void
tfe_startup (GApplication *application) {

  ... ...

  pref = GTK_DIALOG (gtk_builder_get_object (build, "pref"));
  pref_close_request_handler_id = g_signal_connect (GTK_DIALOG (pref),
      "close-request", G_CALLBACK (dialog_close_cb), NULL);

  ... ...
}
```

The function `tfe_application_quit` destroys top-level windows and quits the application. It first disconnects the handlers from the signal “close-request”.

20.5.2 Alert dialog

If a user closes a page which hasn’t been saved, it is advisable to show an alert to confirm it. Alert dialog is used in this application for such a situation.

```
<object class="GtkDialog" id="alert">
  <property name="title">Are you sure?</property>
  <property name="resizable">FALSE</property>
  <property name="modal">TRUE</property>
  <property name="transient-for">win</property>
  <child internal-child="content_area">
    <object class="GtkBox">
      <child>
        <object class="GtkBox">
          <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
          <property name="spacing">12</property>
          <property name="margin-start">12</property>
          <property name="margin-end">12</property>
          <property name="margin-top">12</property>
          <property name="margin-bottom">12</property>
          <child>
            <object class="GtkImage">
              <property name="icon-name">dialog-warning</property>
              <property name="icon-size">GTK_ICON_SIZE_LARGE</property>
            </object>
          </child>
          <child>
```

Figure 25: dialog-warning icon is like ...

```
            <object class="GtkLabel" id="lb_alert">
            </object>
          </child>
        </object>
      </child>
    </object>
  </child>
  <child type="action">
    <object class="GtkButton" id="btn_cancel">
      <property name="label">Cancel</property>
    </object>
  </child>
  <child type="action">
    <object class="GtkButton" id="btn_accept">
      <property name="label">Close</property>
    </object>
  </child>
  <action-widgets>
    <action-widget response="cancel" default="true">btn_cancel</action-widget>
    <action-widget response="accept">btn_accept</action-widget>
  </action-widgets>
  <signal name="response" handler="alert_response_cb" swapped="NO"
      object="nb"></signal>
</object>
```

This ui file describes the alert dialog. Some part are the same as preference dialog. There are two objects in the content area, GtkImage and GtkLabel.

GtkImage shows an image. The image can comes from files, resources, icon theme and so on. The image above displays an icon from the current icon theme. You can see icons in the theme by `gtk4-icon-browser`.

```
$ gtk4-icon-browser
```

The icon named “dialog-warning” is something like this.

These are made by my hand. The real image on the alert dialog is nicer.

The GtkLabel `lb_alert` has no text yet. An alert message will be inserted by the program later.

There are two child tags which have “action” type. They are button objects located in the action area. Action-widgets tag describes the actions of the buttons. `btn_cancel` button emits response signal with cancel response (`GTK_RESPONSE_CANCEL`) if it is clicked on. `btn_accept` button emits response signal with accept response (`GTK_RESPONSE_ACCEPT`) if it is clicked on. The response signal is connected to `alert_response_cb` handler.

The alert dialog keeps alive while the application lives. The “close-request” signal is stopped by the handler `dialog_close_cb` like the preference dialog.

20.6 Close and quit handlers

If a user closes a page or quits the application without saving the contents, the application alerts.

```
static gboolean is_quit;

... ...

void
close_cb (GtkNotebook *nb) {
```

```
  is_quit = false;
  if (has_saved (GTK_NOTEBOOK (nb)))
    notebook_page_close (GTK_NOTEBOOK (nb));
  else {
    gtk_label_set_text (lb_alert, "Contents aren't saved yet.\nAre you sure to
        close?");
    gtk_button_set_label (close_btn_close, "Close");
    gtk_widget_show (GTK_WIDGET (alert));
  }
}

... ...

static void
close_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
  GtkNotebook *nb = GTK_NOTEBOOK (user_data);
  close_cb (nb);
}

... ...

void
alert_response_cb (GtkDialog *alert, int response_id, gpointer user_data) {
  GtkNotebook *nb = GTK_NOTEBOOK (user_data);
  GtkWidget *win = gtk_widget_get_ancestor (GTK_WIDGET (nb), GTK_TYPE_WINDOW);

  gtk_widget_hide (GTK_WIDGET (alert));
  if (response_id == GTK_RESPONSE_ACCEPT) {
    if (is_quit)
      tfe_application_quit (GTK_WINDOW (win));
    else
      notebook_page_close (nb);
  }
}

static void
quit_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
  GtkNotebook *nb = GTK_NOTEBOOK (user_data);
  GtkWidget *win = gtk_widget_get_ancestor (GTK_WIDGET (nb), GTK_TYPE_WINDOW);

  is_quit = true;
  if (has_saved_all (nb))
    tfe_application_quit (GTK_WINDOW (win));
  else {
    gtk_label_set_text (lb_alert, "Contents aren't saved yet.\nAre you sure to
        quit?");
    gtk_button_set_label (btn_accept, "Quit");
    gtk_widget_show (GTK_WIDGET (alert));
  }
}

static void
tfe_startup (GApplication *application) {

  ... ...

  alert = GTK_DIALOG (gtk_builder_get_object (build, "alert"));
  alert_close_request_handler_id = g_signal_connect (GTK_DIALOG (alert),
      "close-request", G_CALLBACK (dialog_close_cb), NULL);
  lb_alert = GTK_LABEL (gtk_builder_get_object (build, "lb_alert"));
  btn_accept = GTK_BUTTON (gtk_builder_get_object (build, "btn_accept"));

  ... ...
```

```
}
```

The static variable `is_quit` is true when user tries to quit the application and false otherwise. When user presses "Ctrl-w", `close_activated` handler is invoked. It just calls `close_cb`. When user clicks on the close button, `close_cb` handler is invoked.

The handler sets `is_quit` to false. The function `has_saved` returns true if the current page has been saved. If it is true, it calls `notebook_page_close` to close the current page. Otherwise, it sets the message of the dialog and the label of the button, then shows the alert dialog.

The response signal of the dialog is connected to the handler `alert_response_cb`. It hides the dialog first. Then checks the `response_id`. If it is `GTK_RESPONSE_ACCEPT`, which means user clicked on the close button, then it closes the current page. Otherwise it does nothing.

When user press "Ctrl-q" or clicked on the quit menu, then `quit_activated` handler is invoked. The handler sets `is_quit` to true. The function `has_saved_all` returns true if all the pages have been saved. If it is true, it calls `tfe_application_quit` to quit the application. Otherwise, it sets the message of the dialog and the label of the button, then shows the alert dialog.

If the user clicked on the buttons on the alert dialog, `alert_resoponse_cb` is invoked. It hides the dialog and checks the `response_id`. If it is `GTK_RESPONSE_ACCEPT`, which means user clicked on the quit button, then it calls `tfe_application_quit` to quit the application. Otherwise it does nothing.

The static variables `alert`, `lb_alert` and `btn_accept` are set in the startup handler. And the signal "close-request" and `dialog_close_cb` handler are connected.

```
1  gboolean
2  has_saved (GtkNotebook *nb) {
3    g_return_val_if_fail (GTK_IS_NOTEBOOK (nb), false);
4
5    TfeTextView *tv;
6    GtkTextBuffer *tb;
7
8    tv = get_current_textview (nb);
9    tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
10   if (gtk_text_buffer_get_modified (tb))
11     return false;
12   else
13     return true;
14 }
15
16 gboolean
17 has_saved_all (GtkNotebook *nb) {
18   g_return_val_if_fail (GTK_IS_NOTEBOOK (nb), false);
19
20   int i, n;
21   GtkWidget *scr;
22   GtkWidget *tv;
23   GtkTextBuffer *tb;
24
25   n = gtk_notebook_get_n_pages (nb);
26   for (i = 0; i < n; ++i) {
27     scr = gtk_notebook_get_nth_page (nb, i);
28     tv = gtk_scrolled_window_get_child (GTK_SCROLLED_WINDOW (scr));
29     tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
30     if (gtk_text_buffer_get_modified (tb))
31       return false;
32   }
33   return true;
34 }
```

- 1-14: `has_saved` function.
- 10: The function `gtk_text_buffer_get_modified` returns true if the content of the buffer has been modified since the modified flag had set false. The flag is set to false when:
  - the buffer is created.

  - the contents of the buffer is replaced
  - the contents of the buffer is saved to a file.
- 11-13: This function returns true if the contents of the current page has been saved and no modification has been made. It returns false, if the current page has been modified and hasn't been saved.
- 16-33: `has_saved_all` function. This function is similar to `has_saved` function. It returns true if all the pages have been saved. It returns false if at least one page has been modified since it last had been saved.

20.7 Notebook page tab

If you have some pages and edit them together, you might be confused which file needs to be saved. Common file editors changes the tab when the contents are modified. GtkTextBuffer provides "modified-changed" signal to notify the modification.

```
static void
notebook_page_build (GtkNotebook *nb, GtkWidget *tv, char *filename) {
  ... ...
  g_signal_connect (GTK_TEXT_VIEW (tv), "change-file", G_CALLBACK (file_changed_cb),
      NULL);
  g_signal_connect (tb, "modified-changed", G_CALLBACK (modified_changed_cb), tv);
}
```

When a page is built, connect "change-file" and "modified-changed" signals to `file_changed_cb` and `modified_changed_cb` handlers respectively.

```
1  static void
2  file_changed_cb (TfeTextView *tv) {
3    GtkWidget *nb =  gtk_widget_get_ancestor (GTK_WIDGET (tv), GTK_TYPE_NOTEBOOK);
4    GtkWidget *scr;
5    GtkWidget *label;
6    GFile *file;
7    char *filename;
8
9    if (! GTK_IS_NOTEBOOK (nb)) /* tv not connected to nb yet */
10     return;
11   file = tfe_text_view_get_file (tv);
12   scr = gtk_widget_get_parent (GTK_WIDGET (tv));
13   if (G_IS_FILE (file)) {
14     filename = g_file_get_basename (file);
15     g_object_unref (file);
16   } else
17     filename = get_untitled ();
18   label = gtk_label_new (filename);
19   gtk_notebook_set_tab_label (GTK_NOTEBOOK (nb), scr, label);
20 }
21
22 static void
23 modified_changed_cb (GtkTextBuffer *tb, gpointer user_data) {
24   TfeTextView *tv = TFE_TEXT_VIEW (user_data);
25   GtkWidget *scr = gtk_widget_get_parent (GTK_WIDGET (tv));
26   GtkWidget *nb =  gtk_widget_get_ancestor (GTK_WIDGET (tv), GTK_TYPE_NOTEBOOK);
27   GtkWidget *label;
28   const char *filename;
29   char *text;
30
31   if (! GTK_IS_NOTEBOOK (nb)) /* tv not connected to nb yet */
32     return;
33   else if (gtk_text_buffer_get_modified (tb)) {
34     filename = gtk_notebook_get_tab_label_text (GTK_NOTEBOOK (nb), scr);
35     text = g_strdup_printf ("*%s", filename);
36     label = gtk_label_new (text);
37     g_free (text);
38     gtk_notebook_set_tab_label (GTK_NOTEBOOK (nb), scr, label);
39   } else
```

```
40      file_changed_cb (tv);
41  }
```

- 1-20: `file_changed_cb` handler.
- 9-10: If the signal emits during the page is being built, it is possible that `tv` isn't a descendant of `nb`. That is, there's no page corresponds to `tv`. Then, it isn't necessary to change the name of the tab because no tab exists.
- 13-15: If `file` is GFile, then it gets the filename and unrefs `file`.
- 16-17: Otherwise, `file` is probably NULL and it assigns "Untitled" related name to `filename`
- 18-19: Creates GtkLabel with `filename` and sets the tab of the page with the GtkLabel.
- 22-41: `modified_changed_cb` handler.
- 31-32: If `tv` isn't a descendant of `nb`, then nothing needs to be done.
- 33-35: If the content is modified, then it gets the text of the tab and adds asterisk at the beginning of the text.
- 36-38: Sets the tab with the asterisk prepended text.
- 39-40: Otherwise the modified bit is off. It is because content is saved. It calls `file_changed_cb` and resets the filename, that means it leaves out the asterisk.

20.8 Font

20.8.1 GtkFontButton and GtkFontChooser

The GtkFontButton is a button which displays the current font. It opens a font chooser dialog if a user clicked on the button. A user can change the font (family, style, weight and size) with the dialog. Then the button keeps the new font and displays it.

The button and its signal "font-set" is initialized in the application startup process.

```
static void
font_set_cb (GtkFontButton *fontbtn, gpointer user_data) {
  GtkWindow *win = GTK_WINDOW (user_data);
  PangoFontDescription *pango_font_desc;

  pango_font_desc = gtk_font_chooser_get_font_desc (GTK_FONT_CHOOSER (fontbtn));
  set_font_for_display_with_pango_font_desc (win, pango_font_desc);
}

static void
tfe_startup (GApplication *application) {

  ... ...

  fontbtn = GTK_FONT_BUTTON (gtk_builder_get_object (build, "fontbtn"));
  g_signal_connect (fontbtn, "font-set", G_CALLBACK (font_set_cb), win);

  ... ...

}
```

In the startup handler, set the variable `fontbtn` to point the GtkFontButton object. Then connect the "font-set" signal to `font_set_cb` handler. The signal "font-set" is emitted when the user selects a font.

GtkFontChooser is an interface implemented by GtkFontButton. The function `gtk_font_chooser_get_font_desc` gets the PangoFontDescription of the currently selected font.

Another function `gtk_font_chooser_get_font` returns a font name which includes family, style, weight and size. I thought it might be able to be applied to tfe editor. The font name can be used to the `font` property of GtkTextTag as it is. But it can't be used to the CSS without converting the string to fit. CSS is appropriate to change the font of entire text in all the buffers. I think GtkTextTag is less appropriate. If you know a good solution, please post it to issue and let me know.

It takes many codes to set the CSS from the PangoFontDescription so the task is left to the function `set_font_for_display_with_pango_font_desc`.

20.8.2 CSS and Pango

A new file `css.c` is made for functions related to CSS.

```
#include "tfe.h"

void
set_css_for_display (GtkWindow *win, const char *css) {
  GdkDisplay *display;

  display = gtk_widget_get_display (GTK_WIDGET (win));
  GtkCssProvider *provider = gtk_css_provider_new ();
  gtk_css_provider_load_from_data (provider, css, -1);
  gtk_style_context_add_provider_for_display (display, GTK_STYLE_PROVIDER
      (provider), GTK_STYLE_PROVIDER_PRIORITY_USER);
}

void
set_font_for_display (GtkWindow *win, const char *fontfamily, const char *fontstyle,
    const char *fontweight, int fontsize) {
  char *textview_css;

  textview_css = g_strdup_printf ("textview {padding: 10px; font-family: \"%s\";
      font-style: %s; font-weight: %s; font-size: %dpt;}",
                                  fontfamily, fontstyle, fontweight, fontsize);
  set_css_for_display (win, textview_css);
  g_free (textview_css);
}

void
set_font_for_display_with_pango_font_desc (GtkWindow *win, PangoFontDescription
    *pango_font_desc) {
  PangoStyle pango_style;
  PangoWeight pango_weight;
  const char *family;
  const char *style;
  const char *weight;
  int fontsize;

  family = pango_font_description_get_family (pango_font_desc);
  pango_style = pango_font_description_get_style (pango_font_desc);
  switch (pango_style) {
  case PANGO_STYLE_NORMAL:
    style = "normal";
    break;
  case PANGO_STYLE_ITALIC:
    style = "italic";
    break;
  case PANGO_STYLE_OBLIQUE:
    style = "oblique";
    break;
  default:
    style = "normal";
    break;
  }
  pango_weight = pango_font_description_get_weight (pango_font_desc);
  switch (pango_weight) {
  case PANGO_WEIGHT_THIN:
    weight = "100";
    break;
  case PANGO_WEIGHT_ULTRALIGHT:
    weight = "200";
    break;
  case PANGO_WEIGHT_LIGHT:
    weight = "300";
```

```
58        break;
59      case PANGO_WEIGHT_SEMILIGHT:
60        weight = "350";
61        break;
62      case PANGO_WEIGHT_BOOK:
63        weight = "380";
64        break;
65      case PANGO_WEIGHT_NORMAL:
66        weight = "400"; /* or "normal" */
67        break;
68      case PANGO_WEIGHT_MEDIUM:
69        weight = "500";
70        break;
71      case PANGO_WEIGHT_SEMIBOLD:
72        weight = "600";
73        break;
74      case PANGO_WEIGHT_BOLD:
75        weight = "700"; /* or "bold" */
76        break;
77      case PANGO_WEIGHT_ULTRABOLD:
78        weight = "800";
79        break;
80      case PANGO_WEIGHT_HEAVY:
81        weight = "900";
82        break;
83      case PANGO_WEIGHT_ULTRAHEAVY:
84        weight = "900"; /* In PangoWeight definition, the weight is 1000. But CSS allows
              the weight below 900. */
85        break;
86      default:
87        weight = "normal";
88        break;
89      }
90      fontsize = pango_font_description_get_size (pango_font_desc) / PANGO_SCALE;
91      set_font_for_display (win, family, style, weight, fontsize);
92  }
```

- 3-11: `set_css_for_display`. This function sets CSS for GdkDisplay. The content of the function is the same as the part of startup handler in the previous version of `tfeapplication.c`.
- 13-20: `set_font_for_display`. This function sets CSS with font-family, font-style, font-weight and font-size.
  - font-family is a name of a font. For example, sans-serif, monospace, Helvetica and “Noto Sans” are font-family. It is recommended to quote font family names that contains white space, digits, or punctuation characters other than hyphens.
  - font-style is one of normal, italic and oblique.
  - font-weight specifies the thickness of a font. It is normal or bold. It can be specified with a number between 100 and 900. Normal is the same as 400. Bold is 700.
  - font-size specifies the size of a font. Small, medium, large and 12pt are font-size.
- 17: Makes CSS text. The function `g_strdup_printf` creates a new string with printf-like formatting.
- 23-92: `set_font_for_display_with_pango_font_desc`. This function takes out font-family, font-style, font-weight and font-size from the PangoFontDescription object and calls `set_font`for_display‘.
- 32: Gets the font-family of `pango_font_desc`.
- 33-47: Gets the font-style of `pango_font_desc`. The functions `pango_font_description_get_style` returns an enumerated value.
- 48-89: Gets the font-weight of `pango_font_desc`. The function `pango_font_description_get_weight` returns an enumerated value. They corresponds to the numbers from 100 to 900.
- 90: Gets the font-size of `pango_font_desc`. The function `pango_font_description_get_size` returns the size of a font. The unit of this size is (1/PANGO_SCALE)pt. If the font size is 10pt, the function returns 10*PANGO_SCALE. PANGO_SCALE is defined as 1024. Therefore, 10*PANGO_SCALE is 10240.
- 91: calls `set_font_for_display` to set CSS for the GdkDisplay.

For further information, see Pango API Reference.

20.9 GSettings

We want to maintain the font data after the application quits. There are some ways to implement it.

- Make a configuration file. For example, a text file "~/.config/tfe/font.cfg" keeps font information.
- Use GSettings object. The basic idea of GSettings are similar to configuration file. Configuration information data is put into a database file.

The coding with GSettings object is simple and easy. However, it is a bit hard to understand the concept. This subsection describes the concept first and then how to program it.

20.9.1 GSettings schema

GSettings schema describes a set of keys, value types and some other information. GSettings object uses this schema and it writes/reads the value of a key to/from the right place in the database.

- A schema has an id. The id must be unique. We often use the same string as application id, but schema id and application id are different. You can use different name from application id. Schema id is a string delimited by periods. For example, "com.github.ToshioCP.tfe" is a correct schema id.
- A schema usually has a path. The path is a location in the database. Each key is stored under the path. For example, if a key `font` is defined with a path `/com/github/ToshioCP/tfe/`, the key's location in the database is `/com/github/ToshioCP/tfe/font`. Path is a string begins with and ends with a slash (`/`). And it is delimited by slashes.
- GSettings save information as key-value style. Key is a string begins with lower case characters followed by lower case, digit or dash (`-`) and ends with lower case or digit. No consecutive dashes are allowed. Values can be any type. GSettings stores values as GVariant type, which may contain, for example, integer, double, boolean, string or complex types like an array. The type of values needs to be defined in the schema.
- A default value needs to be set for each key.
- A summery and description can be set for each key optionally.

Schemas are described in an XML format. For example,

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <schemalist>
3    <schema path="/com/github/ToshioCP/tfe/" id="com.github.ToshioCP.tfe">
4      <key name="font" type="s">
5        <default>'Monospace 12'</default>
6        <summary>Font</summary>
7        <description>The font to be used for textview.</description>
8      </key>
9    </schema>
10 </schemalist>
```

- 4: The type attribute is "s". It is GLib API Reference, GVariant Type Strings. Other common types are:
  - "b": gboolean
  - "i": gint32.
  - "d": double.

Further information is in Glib API Reference, VarientType.

20.9.2 gsettings

First, let's try `gsettings` application. It is a configuration tool for GSettings.

```
$ gsettings help
Usage:
  gsettings --version
  gsettings [--schemadir SCHEMADIR] COMMAND [ARGS?]

Commands:
  help                        Show this information
```

```
  list-schemas              List installed schemas
  list-relocatable-schemas  List relocatable schemas
  list-keys                 List keys in a schema
  list-children             List children of a schema
  list-recursively          List keys and values, recursively
  range                     Queries the range of a key
  describe                  Queries the description of a key
  get                       Get the value of a key
  set                       Set the value of a key
  reset                     Reset the value of a key
  reset-recursively         Reset all values in a given schema
  writable                  Check if a key is writable
  monitor                   Watch for changes

Use "gsettings help COMMAND" to get detailed help.
```

List schemas.

```
$ gsettings list-schemas
org.gnome.rhythmbox.podcast
ca.desrt.dconf-editor.Demo.Empty
org.gnome.gedit.preferences.ui
org.gnome.evolution-data-server.calendar
org.gnome.rhythmbox.plugins.generic-player

... ...
```

Each line is an id of a schema. Each schema has a key-value configuration data. You can see them with list-recursively command. Let's look at the keys and values of `org.gnome.calculator` schema.

```
$ gsettings list-recursively org.gnome.calculator
org.gnome.calculator source-currency ''
org.gnome.calculator source-units 'degree'
org.gnome.calculator button-mode 'basic'
org.gnome.calculator target-currency ''
org.gnome.calculator base 10
org.gnome.calculator angle-units 'degrees'
org.gnome.calculator word-size 64
org.gnome.calculator accuracy 9
org.gnome.calculator show-thousands false
org.gnome.calculator window-position (122, 77)
org.gnome.calculator refresh-interval 604800
org.gnome.calculator target-units 'radian'
org.gnome.calculator precision 2000
org.gnome.calculator number-format 'automatic'
org.gnome.calculator show-zeroes false
```

This schema is used by Gnome Calculator. Run the calculator and change the mode, then check the schema again.

```
$ gnome-calculator
```

Then, change the mode to advanced and quit.

Run gsettings and check whether the value of `button-mode` changes.

```
$ gsettings list-recursively org.gnome.calculator

... ...

org.gnome.calculator button-mode 'advanced'

... ...
```

Now we know that Gnome Calculator used gsettings and it has set `button-mode` key to "advanced". The value remains even the calculator quits. So when the calculator is run again, it will appear as an advanced mode calculator.

Figure 26: gnome-calculator basic mode

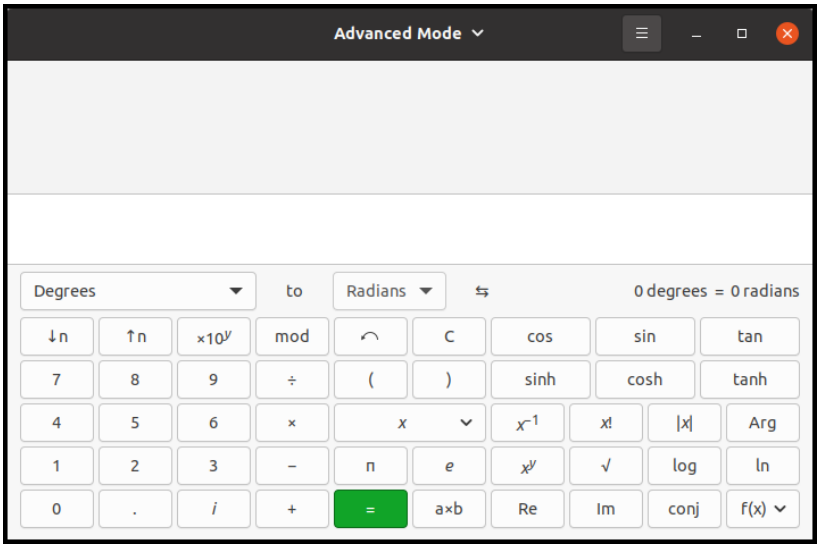

Figure 27: gnome-calculator advanced mode

20.9.3 glib-compile-schemas

GSettings schemas are specified with an XML format. The XML schema files must have the filename extension `.gschema.xml`. The following is the XML schema file for the application `tfe`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <schemalist>
3    <schema path="/com/github/ToshioCP/tfe/" id="com.github.ToshioCP.tfe">
4      <key name="font" type="s">
5        <default>'Monospace 12'</default>
6        <summary>Font</summary>
7        <description>The font to be used for textview.</description>
8      </key>
9    </schema>
10 </schemalist>
```

The filename is "com.github.ToshioCP.tfe.gschema.xml". Schema XML filenames are usually the schema id followed by ".gschema.xml" suffix. You can use different name from schema id, but it is not recommended.

- 2: The top level element is `<schemalist>`.
- 3: schema tag has `path` and `id` attributes. A path determines where the settings are stored in the conceptual global tree of settings. An id identifies the schema.
- 4: Key tag has two attributes. Name is the name of the key. Type is the type of the value of the key and specified with GLib API Reference, VariantType.
- 5: default value of the key `font` is `Monospace 12`.
- 6: Summery and description elements describes the key. They are optional, but it is recommended to add them in the XML file.

The XML file is compiled by glib-compile-schemas. When compiling, `glib-compile-schemas` compiles all the XML files which have ".gschema.xml" file extension in the directory given as an argument. It converts the XML file into a binary file `gschemas.compiled`. Suppose the XML file above is under `tfe6` directory.

```
$ glib-compile-schemas tfe6
```

Then, `gschemas.compiled` is generated under `tfe6`. When you test your application, set `GSETTINGS_SCHEMA_DIR` so that GSettings objet can find `gschemas.compiled`.

```
$ GSETTINGS_SCHEMA_DIR=(the directory gschemas.compiled is
    located):$GSETTINGS_SCHEMA_DIR (your application name)
```

This is because GSettings object searches `GSETTINGS_SCHEMA_DIR` for `gschemas.compiled`.

GSettings object looks for this file by the following process.

- It searches `glib-2.0/schemas` subdirectories of all the directories specified in the environment variable `XDG_DATA_DIRS`. Most common directory is `/usr/share/glib-2.0/schemas`.
- If `GSETTINGS_SCHEMA_DIR` environment variable is defined, it searches all the directories specified in the variable. `GSETTINGS_SCHEMA_DIR` can specify multiple directories delimited by colon (:).

In the directories above, all the `.gschema.xml` files are stored. Therefore, when you install your application, follow the instruction below to install your schemas.

1. Make `.gschema.xml` file.
2. Copy it to one of the directories above. For example, `/usr/local/share/glib-2.0/schemas`.
3. Run `glib-compile-schemas` on the directory above.

20.9.4 Meson.build

Meson provides `gnome.compile_schemas` method to compile XML file in the build directory. This is used to test the application. Write the following to the `meson.build` file.

```
gnome.compile_schemas(build_by_default: true, depend_files:
    'com.github.ToshioCP.tfe.gschema.xml')
```

- `build_by_default`: If it is true, the target will be build by default.
- `depend_files`: XML files to be compiled.

In the example above, this method runs `glib-compile-schemas` to generate `gschemas.compiled` from the XML file `com.github.ToshioCP.tfe.gschema.xml`. The file `gschemas.compiled` is located under the build directory. If you run meson as `meson _build` and ninja as `ninja -C _build`, then it is under `_build` directory.

After compilation, you can test your application like this:

```
$ GSETTINGS_SCHEMA_DIR=_build:$GSETTINGS_SCHEMA_DIR _build/tfe
```

20.9.5 GSettings object and g_settings_bind

Write gsettings related codes to 'tfeapplication.c'.

```
... ...
static GSettings *settings;
... ...

void
tfe_application_quit (GtkWindow *win) {
  ... ...
  g_clear_object (&settings);
  ... ...
}

static void
tfe_startup (GApplication *application) {
  ... ...
  settings = g_settings_new ("com.github.ToshioCP.tfe");
  g_settings_bind (settings, "font", fontbtn, "font", G_SETTINGS_BIND_DEFAULT);
  ... ...
}
```

Static variable `settings` keeps a pointer to GSettings instance. Before application quits, the application releases the GSettings instance. The function `g_clear_object` is used.

Startup handler creates GSettings instance with the schema id "com.github.ToshioCP.tfe" and assigns the pointer to `settings`. The function `g_settings_bind` connects the settings keys (key and value) and the "font" property of `fontbtn`. Then the two values will be always the same. If one value changes then the other will automatically change.

You need to make an effort to understand GSettings concept, but coding is very simple. Just create a GSettings object and bind it to a property of an object.

20.10 Installation

It is a good idea to install your application in `$HOME/local/bin` directory if you have installed Gtk4 from the source (See Section 2). Then you need to put `--prefix=$HOME/local` option to meson like this.

```
$ meson --prefix=$HOME/local _build
```

If you've installed Gtk4 from the distribution package, `--prefix` option isn't necessary. You just install `tfe` to the default bin directory like `/usr/local/bin`.

Modify `meson.build` and add install option and set it true in executable function.

```
executable('tfe', sourcefiles, resources, dependencies: gtkdep, export_dynamic:
    true, install: true)
```

You can install your application by:

```
$ ninja -C _build install
```

However, you need to do one more thing. Copy your XML file to `$HOME/local/share/glib-2.0/schemas/`, which is specified in `GSETTINGS_SCHEMA_DIR` environment variable, and run `glib-compile-schemas` on that directory.

```
schema_dir = get_option('prefix') / get_option('datadir') / 'glib-2.0/schemas/'
install_data('com.github.ToshioCP.tfe.gschema.xml', install_dir: schema_dir)
```

- get_option: This function returns the value of build options. The default value of the option 'prefix' is "/usr/local", but it is "$HOME/local" because we have run meson with prefix option. The default value of the option 'datadir' is "share". The operator '/' connects the strings with '/' separator. So, `$HOME/local/share/glib-2.0/schemas` is assigned to the variable `schema_dir`.
- install_data: This function installs the data to the install directory.

Meson can run a post compile script.

```
meson.add_install_script('glib-compile-schemas', schema_dir)
```

This method runs 'glib-compile-schemas' with an argument `schema_dir`. The following is `meson.build`.

```
1  project('tfe', 'c')
2
3  gtkdep = dependency('gtk4')
4
5  gnome=import('gnome')
6  resources = gnome.compile_resources('resources','tfe.gresource.xml')
7  gnome.compile_schemas(build_by_default: true, depend_files:
       'com.github.ToshioCP.tfe.gschema.xml')
8
9  sourcefiles=files('tfeapplication.c', 'tfenotebook.c', 'css.c',
       '../tfetextview/tfetextview.c')
10
11 executable('tfe', sourcefiles, resources, dependencies: gtkdep, export_dynamic:
       true, install: true)
12
13 schema_dir = get_option('prefix') / get_option('datadir') / 'glib-2.0/schemas/'
14 install_data('com.github.ToshioCP.tfe.gschema.xml', install_dir: schema_dir)
15 meson.add_install_script('glib-compile-schemas', schema_dir)
```

Source files of `tfe` is under src/tfe6 directory. Copy them to your temporary directory and try to compile and install.

```
$ meson --prefix=$HOME/local _build
$ ninja -C _build
$ GSETTINGS_SCHEMA_DIR=_build:$GSETTINGS_SCHEMA_DIR _build/tfe
$ ninja -C _build install
$ tfe
$ ls $HOME/local/bin
... ...
... tfe
... ...
$ ls $HOME/local/share/glib-2.0/schemas
com.github.ToshioCP.tfe.gschema.xml
gschema.dtd
gschemas.compiled
... ...
```

The screenshot is as follows.

21 Template XML and composite widget

The tfe program in the previous section is not so good because many things are crammed into `tfepplication.c`. Many static variables in `tfepplication.c` shows that.

```
static GtkDialog *pref;
static GtkFontButton *fontbtn;
static GSettings *settings;
static GtkDialog *alert;
static GtkLabel *lb_alert;
static GtkButton *btn_accept;

static gulong pref_close_request_handler_id = 0;
```

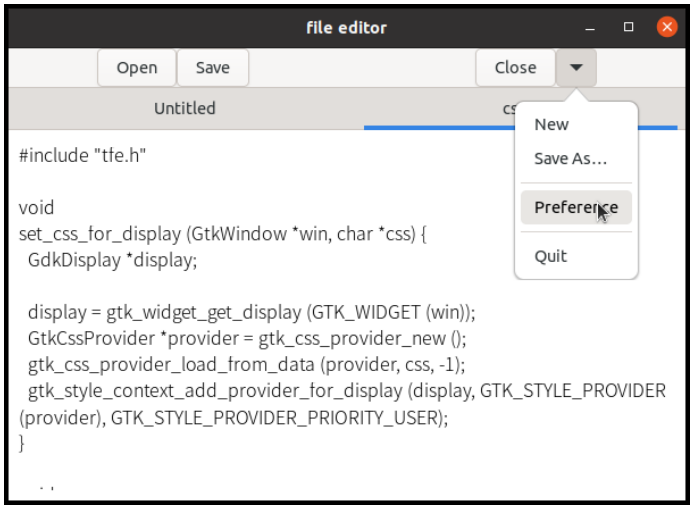

Figure 28: tfe6

```
static gulong alert_close_request_handler_id = 0;
static gboolean is_quit;
```

Generally, if there are many global or static variables in the program, it is not a good program. Such programs are difficult to maintain.

The file `tfeapplication.c` should be divided into several files.

- `tfeapplication.c` only has codes related to GtkApplication.
- A file for GtkApplicationWindow
- A file for a preference dialog
- A file for an alert dialog

The preference dialog is defined by a ui file. And it has GtkBox, GtkLabel and GtkFontButton in it. Such widget is called composite widget. Composite widget is a child object (not child widget) of a widget. For example, the preference composite widget is a child object of GtkDialog. Composite widget can be built from template XML. Next subsection shows how to build a preference dialog.

21.1 Preference dialog

First, write a template XML file.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <interface>
3    <template class="TfePref" parent="GtkDialog">
4      <property name="title">Preferences</property>
5      <property name="resizable">FALSE</property>
6      <property name="modal">TRUE</property>
7      <child internal-child="content_area">
8        <object class="GtkBox" id="content_area">
9          <child>
10           <object class="GtkBox" id="pref_boxh">
11             <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
12             <property name="spacing">12</property>
13             <property name="margin-start">12</property>
14             <property name="margin-end">12</property>
15             <property name="margin-top">12</property>
16             <property name="margin-bottom">12</property>
17             <child>
18               <object class="GtkLabel" id="fontlabel">
19                 <property name="label">Font:</property>
20                 <property name="xalign">1</property>
21               </object>
22             </child>
```

```
23              <child>
24                <object class="GtkFontButton" id="fontbtn">
25                </object>
26              </child>
27            </object>
28          </child>
29        </object>
30      </child>
31    </template>
32  </interface>
```

- 3: Template tag specifies a composite widget. The value of a class attribute is the object name of the composite widget. This XML file names the object "TfePref". It is defined in a C source file and it will be shown later. A parent attribute specifies the direct parent object of the composite widget. `TfePref` is a child object of `GtkDialog`. Therefore the value of the attribute is "GtkDialog". A parent attribute is optional but it is recommended to specify.

Other lines are the same as before. The object `TfePref` is defined in `tfepref.h` and `tfepref.c`.

```
1  #ifndef __TFE_PREF_H__
2  #define __TFE_PREF_H__
3
4  #include <gtk/gtk.h>
5
6  #define TFE_TYPE_PREF tfe_pref_get_type ()
7  G_DECLARE_FINAL_TYPE (TfePref, tfe_pref, TFE, PREF, GtkDialog)
8
9  GtkWidget *
10 tfe_pref_new (GtkWindow *win);
11
12 #endif /* __TFE_PREF_H__ */
```

- 6-7: When you define a new object, you need to write these two lines. Refer to Section 8.
- 9-10: `tfe_pref_new` creates a new TfePref object. It has a parameter `win` which is used as a transient parent window to show the dialog.

```
1  #include "tfepref.h"
2
3  struct _TfePref
4  {
5    GtkDialog parent;
6    GSettings *settings;
7    GtkFontButton *fontbtn;
8  };
9
10 G_DEFINE_TYPE (TfePref, tfe_pref, GTK_TYPE_DIALOG);
11
12 static void
13 tfe_pref_dispose (GObject *gobject) {
14   TfePref *pref = TFE_PREF (gobject);
15
16   g_clear_object (&pref->settings);
17   G_OBJECT_CLASS (tfe_pref_parent_class)->dispose (gobject);
18 }
19
20 static void
21 tfe_pref_init (TfePref *pref) {
22   gtk_widget_init_template (GTK_WIDGET (pref));
23   pref->settings = g_settings_new ("com.github.ToshioCP.tfe");
24   g_settings_bind (pref->settings, "font", pref->fontbtn, "font",
       G_SETTINGS_BIND_DEFAULT);
25 }
26
27 static void
28 tfe_pref_class_init (TfePrefClass *class) {
```

```
29    GObjectClass *object_class = G_OBJECT_CLASS (class);
30
31    object_class->dispose = tfe_pref_dispose;
32    gtk_widget_class_set_template_from_resource (GTK_WIDGET_CLASS (class),
          "/com/github/ToshioCP/tfe/tfepref.ui");
33    gtk_widget_class_bind_template_child (GTK_WIDGET_CLASS (class), TfePref, fontbtn);
34  }
35
36  GtkWidget *
37  tfe_pref_new (GtkWindow *win) {
38    return GTK_WIDGET (g_object_new (TFE_TYPE_PREF, "transient-for", win, NULL));
39  }
```

- 3-8: The structure of an instance of this object. It has two variables, settings and fontbtn.
- 10: `G_DEFINE_TYPE` macro. This macro registers the TfePref type.
- 12-18: Dispose handler. This handler is called when the instance is destroyed. The destruction process has two stages, disposing and finalizing. When disposing, the instance releases all the references (to the other instances). TfePref object holds a reference to the GSettings instance. It is released in line 16. After that parents dispose handler is called in line 17. For further information about destruction process, refer to Section 11.
- 27-34: Class initialization function.
- 31: Set the dispose handler.
- 32: `gtk_widget_class_set_template_from_resource` function associates the description in the XML file (`tfepref.ui`) with the widget. At this moment no instance is created. It just make the class to know the structure of the object. That's why the top level tag is not `<object>` but `<template>` in the XML file.
- 33: `gtk_widget_class_bind_template_child` function binds a private variable of the object with a child object in the template. This function is a macro. The name of the private variable (`fontbtn` in line 7) and the id `fontbtn` in the XML file (line 24) must be the same. The pointer to the instance will be assigned to the variable `fontbtn` when the instance is created.
- 20-25: Instance initialization function.
- 22: Creates the instance based on the template in the class. The template has been made during the class initialization process.
- 23: Create GSettings instance with the id "com.github.ToshioCP.tfe".
- 24: Bind the font key in the GSettings object to the font property in the GtkFontButton.
- 36-39: The function `tfe_pref_new` creates an instance of TfePref. The parameter `win` is a transient parent.

Now, It is very simple to use this dialog. A caller just creates this object and shows it.

```
TfePref *pref;
pref = tfe_pref_new (win) /* win is the top-level window */
gtk_widget_show (GTK_WINDOW (win));
```

This instance is automatically destroyed when a user clicks on the close button. That's all. If you want to show the dialog again, just create and show it.

21.2 Alert dialog

It is almost same as preference dialog.

Its XML file is:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <interface>
3    <template class="TfeAlert" parent="GtkDialog">
4      <property name="title">Are you sure?</property>
5      <property name="resizable">FALSE</property>
6      <property name="modal">TRUE</property>
7      <child internal-child="content_area">
8        <object class="GtkBox">
9          <child>
10           <object class="GtkBox">
```

```
            <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
            <property name="spacing">12</property>
            <property name="margin-start">12</property>
            <property name="margin-end">12</property>
            <property name="margin-top">12</property>
            <property name="margin-bottom">12</property>
            <child>
              <object class="GtkImage">
                <property name="icon-name">dialog-warning</property>
                <property name="icon-size">GTK_ICON_SIZE_LARGE</property>
              </object>
            </child>
            <child>
              <object class="GtkLabel" id="lb_alert">
              </object>
            </child>
          </object>
        </child>
      </object>
    </child>
    <child type="action">
      <object class="GtkButton" id="btn_cancel">
        <property name="label">Cancel</property>
      </object>
    </child>
    <child type="action">
      <object class="GtkButton" id="btn_accept">
        <property name="label">Close</property>
      </object>
    </child>
    <action-widgets>
      <action-widget response="cancel" default="true">btn_cancel</action-widget>
      <action-widget response="accept">btn_accept</action-widget>
    </action-widgets>
  </template>
</interface>
```

The header file is:

```
#ifndef __TFE_ALERT_H__
#define __TFE_ALERT_H__

#include <gtk/gtk.h>

#define TFE_TYPE_ALERT tfe_alert_get_type ()
G_DECLARE_FINAL_TYPE (TfeAlert, tfe_alert, TFE, ALERT, GtkDialog)

void
tfe_alert_set_message (TfeAlert *alert, const char *message);

void
tfe_alert_set_button_label (TfeAlert *alert, const char *label);

GtkWidget *
tfe_alert_new (GtkWindow *win);

#endif /* __TFE_ALERT_H__ */
```

There are three public functions. The functions `tfe_alert_set_message` and `tfe_alert_set_button_label` sets the label and button name of the alert dialog. For example, if you want to show an alert that the user tries to close without saving the content, set them like:

```
tfe_alert_set_message (alert, "Are you really close without saving?"); /* alert
    points to a TfeAlert instance */
tfe_alert_set_button_label (alert, "Close");
```

The function `tfe_alert_new` creates a TfeAlert dialog.

The C source file is:

```
#include "tfealert.h"

struct _TfeAlert
{
  GtkDialog parent;
  GtkLabel *lb_alert;
  GtkButton *btn_accept;
};

G_DEFINE_TYPE (TfeAlert, tfe_alert, GTK_TYPE_DIALOG);

void
tfe_alert_set_message (TfeAlert *alert, const char *message) {
  gtk_label_set_text (alert->lb_alert, message);
}

void
tfe_alert_set_button_label (TfeAlert *alert, const char *label) {
  gtk_button_set_label (alert->btn_accept, label);
}

static void
tfe_alert_init (TfeAlert *alert) {
  gtk_widget_init_template (GTK_WIDGET (alert));
}

static void
tfe_alert_class_init (TfeAlertClass *class) {
  gtk_widget_class_set_template_from_resource (GTK_WIDGET_CLASS (class),
      "/com/github/ToshioCP/tfe/tfealert.ui");
  gtk_widget_class_bind_template_child (GTK_WIDGET_CLASS (class), TfeAlert,
      lb_alert);
  gtk_widget_class_bind_template_child (GTK_WIDGET_CLASS (class), TfeAlert,
      btn_accept);
}

GtkWidget *
tfe_alert_new (GtkWindow *win) {
  return GTK_WIDGET (g_object_new (TFE_TYPE_ALERT, "transient-for", win, NULL));
}
```

The program is almost same as `tfepref.c`.

The instruction how to use this object is as follows.

1. Write a “response” signal handler.
2. Create a TfeAlert object.
3. Connect “response” signal to a handler
4. Show the dialog
5. In the signal handler, do something with regard to the response-id. Then destroy the dialog.

21.3 Top-level window

In the same way, create a child object of GtkApplicationWindow. The object name is “TfeWindow”.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <template class="TfeWindow" parent="GtkApplicationWindow">
    <property name="title">file editor</property>
    <property name="default-width">600</property>
    <property name="default-height">400</property>
```

```
7       <child>
8         <object class="GtkBox" id="boxv">
9           <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
10          <child>
11            <object class="GtkBox" id="boxh">
12              <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
13              <child>
14                <object class="GtkLabel" id="dmy1">
15                  <property name="width-chars">10</property>
16                </object>
17              </child>
18              <child>
19                <object class="GtkButton" id="btno">
20                  <property name="label">Open</property>
21                  <property name="action-name">win.open</property>
22                </object>
23              </child>
24              <child>
25                <object class="GtkButton" id="btns">
26                  <property name="label">Save</property>
27                  <property name="action-name">win.save</property>
28                </object>
29              </child>
30              <child>
31                <object class="GtkLabel" id="dmy2">
32                  <property name="hexpand">TRUE</property>
33                </object>
34              </child>
35              <child>
36                <object class="GtkButton" id="btnc">
37                  <property name="label">Close</property>
38                  <property name="action-name">win.close</property>
39                </object>
40              </child>
41              <child>
42                <object class="GtkMenuButton" id="btnm">
43                  <property name="direction">down</property>
44                  <property name="halign">start</property>
45                  <property name="icon-name">open-menu-symbolic</property>
46                </object>
47              </child>
48              <child>
49                <object class="GtkLabel" id="dmy3">
50                  <property name="width-chars">10</property>
51                </object>
52              </child>
53            </object>
54          </child>
55          <child>
56            <object class="GtkNotebook" id="nb">
57              <property name="scrollable">TRUE</property>
58              <property name="hexpand">TRUE</property>
59              <property name="vexpand">TRUE</property>
60            </object>
61          </child>
62        </object>
63      </child>
64    </template>
65  </interface>
```

This XML file is almost same as before except template tag and “action-name” property in buttons.

GtkButton implements GtkActionable interface, which has “action-name” property. If this property is set, GtkButton activates the action when it is clicked. For example, if an open button is clicked, “win.open”

action will be activated and `open_activated` handler will be invoked.

This action is also used by "<Control>o" accelerator (See the source code of `tfewindow.c` below). If you use "clicked" signal for the button, you need its signal handler. Then, there are two handlers:

- a handler for the "clicked" signal on the button
- a handler for the "activate" signal on the "win.open" action, to which "<Control>o" accelerator is connected

These two handlers do almost same thing. It is inefficient. Connecting buttons to actions is a good way to reduce unnecessary codes.

```
#ifndef __TFE_WINDOW_H__
#define __TFE_WINDOW_H__

#include <gtk/gtk.h>

#define TFE_TYPE_WINDOW tfe_window_get_type ()
G_DECLARE_FINAL_TYPE (TfeWindow, tfe_window, TFE, WINDOW, GtkApplicationWindow)

void
tfe_window_notebook_page_new (TfeWindow *win);

void
tfe_window_notebook_page_new_with_files (TfeWindow *win, GFile **files, int n_files);

GtkWidget *
tfe_window_new (GtkApplication *app);

#endif /* __TFE_WINDOW_H__ */
```

There are three public functions. The function `tfe_window_notebook_page_new` creates a new notebook page. This is a wrapper function for `notebook_page_new`. It is called by GtkApplication object. The function `tfe_window_notebook_page_new_with_files` creates notebook pages with a contents read from the given files. The function `tfe_window_new` creates a TfeWindow instance.

```
#include "tfewindow.h"
#include "tfenotebook.h"
#include "tfepref.h"
#include "tfealert.h"
#include "css.h"

struct _TfeWindow {
  GtkApplicationWindow parent;
  GtkMenuButton *btnm;
  GtkNotebook *nb;
  GSettings *settings;
  gboolean is_quit;
};

G_DEFINE_TYPE (TfeWindow, tfe_window, GTK_TYPE_APPLICATION_WINDOW);

/* alert response signal handler */
static void
alert_response_cb (GtkDialog *alert, int response_id, gpointer user_data) {
  TfeWindow *win = TFE_WINDOW (user_data);

  if (response_id == GTK_RESPONSE_ACCEPT) {
    if (win->is_quit)
      gtk_window_destroy(GTK_WINDOW (win));
    else
      notebook_page_close (win->nb);
  }
  gtk_window_destroy (GTK_WINDOW (alert));
}
```

```
30
31  /* ----- action activated handlers ----- */
32  static void
33  open_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
34    TfeWindow *win = TFE_WINDOW (user_data);
35
36    notebook_page_open (GTK_NOTEBOOK (win->nb));
37  }
38
39  static void
40  save_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
41    TfeWindow *win = TFE_WINDOW (user_data);
42
43    notebook_page_save (GTK_NOTEBOOK (win->nb));
44  }
45
46  static void
47  close_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
48    TfeWindow *win = TFE_WINDOW (user_data);
49    TfeAlert *alert;
50
51    if (has_saved (win->nb))
52      notebook_page_close (win->nb);
53    else {
54      win->is_quit = false;
55      alert = TFE_ALERT (tfe_alert_new (GTK_WINDOW (win)));
56      tfe_alert_set_message (alert, "Contents aren't saved yet.\nAre you sure to
          close?");
57      tfe_alert_set_button_label (alert, "Close");
58      g_signal_connect (GTK_DIALOG (alert), "response", G_CALLBACK
          (alert_response_cb), win);
59      gtk_widget_show (GTK_WIDGET (alert));
60    }
61  }
62
63  static void
64  new_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
65    TfeWindow *win = TFE_WINDOW (user_data);
66
67    notebook_page_new (GTK_NOTEBOOK (win->nb));
68  }
69
70  static void
71  saveas_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
72    TfeWindow *win = TFE_WINDOW (user_data);
73
74    notebook_page_saveas (GTK_NOTEBOOK (win->nb));
75  }
76
77  static void
78  pref_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
79    TfeWindow *win = TFE_WINDOW (user_data);
80    GtkWidget *pref;
81
82    pref = tfe_pref_new (GTK_WINDOW (win));
83    gtk_widget_show (pref);
84  }
85
86  static void
87  quit_activated (GSimpleAction *action, GVariant *parameter, gpointer user_data) {
88    TfeWindow *win = TFE_WINDOW (user_data);
89
90    TfeAlert *alert;
91
```

```
92    if (has_saved_all (GTK_NOTEBOOK (win->nb)))
93      gtk_window_destroy (GTK_WINDOW (win));
94    else {
95      win->is_quit = true;
96      alert = TFE_ALERT (tfe_alert_new (GTK_WINDOW (win)));
97      tfe_alert_set_message (alert, "Contents aren't saved yet.\nAre you sure to
            quit?");
98      tfe_alert_set_button_label (alert, "Quit");
99      g_signal_connect (GTK_DIALOG (alert), "response", G_CALLBACK
            (alert_response_cb), win);
100     gtk_widget_show (GTK_WIDGET (alert));
101   }
102 }
103
104 /* gsettings changed::font signal handler */
105 static void
106 changed_font_cb (GSettings *settings, char *key, gpointer user_data) {
107   GtkWindow *win = GTK_WINDOW (user_data);
108   char *font;
109   PangoFontDescription *pango_font_desc;
110
111   font = g_settings_get_string (settings, "font");
112   pango_font_desc = pango_font_description_from_string (font);
113   g_free (font);
114   set_font_for_display_with_pango_font_desc (win, pango_font_desc);
115 }
116
117 /* --- public functions --- */
118
119 void
120 tfe_window_notebook_page_new (TfeWindow *win) {
121   notebook_page_new (win->nb);
122 }
123
124 void
125 tfe_window_notebook_page_new_with_files (TfeWindow *win, GFile **files, int n_files)
        {
126   int i;
127
128   for (i = 0; i < n_files; i++)
129     notebook_page_new_with_file (win->nb, files[i]);
130   if (gtk_notebook_get_n_pages (win->nb) == 0)
131     notebook_page_new (win->nb);
132 }
133
134 /* --- TfeWindow object construction/destruction --- */
135 static void
136 tfe_window_dispose (GObject *gobject) {
137   TfeWindow *window = TFE_WINDOW (gobject);
138
139   g_clear_object (&window->settings);
140   G_OBJECT_CLASS (tfe_window_parent_class)->dispose (gobject);
141 }
142
143 static void
144 tfe_window_init (TfeWindow *win) {
145   GtkBuilder *build;
146   GMenuModel *menu;
147
148   gtk_widget_init_template (GTK_WIDGET (win));
149
150   build = gtk_builder_new_from_resource ("/com/github/ToshioCP/tfe/menu.ui");
151   menu = G_MENU_MODEL (gtk_builder_get_object (build, "menu"));
152   gtk_menu_button_set_menu_model (win->btnm, menu);
```

```
153   g_object_unref(build);
154
155   win->settings = g_settings_new ("com.github.ToshioCP.tfe");
156   g_signal_connect (win->settings, "changed::font", G_CALLBACK (changed_font_cb),
          win);
157
158 /* ----- action ----- */
159   const GActionEntry win_entries[] = {
160     { "open", open_activated, NULL, NULL, NULL },
161     { "save", save_activated, NULL, NULL, NULL },
162     { "close", close_activated, NULL, NULL, NULL },
163     { "new", new_activated, NULL, NULL, NULL },
164     { "saveas", saveas_activated, NULL, NULL, NULL },
165     { "pref", pref_activated, NULL, NULL, NULL },
166     { "close-all", quit_activated, NULL, NULL, NULL }
167   };
168   g_action_map_add_action_entries (G_ACTION_MAP (win), win_entries, G_N_ELEMENTS
          (win_entries), win);
169
170   changed_font_cb(win->settings, "font", win);
171 }
172
173 static void
174 tfe_window_class_init (TfeWindowClass *class) {
175   GObjectClass *object_class = G_OBJECT_CLASS (class);
176
177   object_class->dispose = tfe_window_dispose;
178   gtk_widget_class_set_template_from_resource (GTK_WIDGET_CLASS (class),
          "/com/github/ToshioCP/tfe/tfewindow.ui");
179   gtk_widget_class_bind_template_child (GTK_WIDGET_CLASS (class), TfeWindow, btnm);
180   gtk_widget_class_bind_template_child (GTK_WIDGET_CLASS (class), TfeWindow, nb);
181 }
182
183 GtkWidget *
184 tfe_window_new (GtkApplication *app) {
185   return GTK_WIDGET (g_object_new (TFE_TYPE_WINDOW, "application", app, NULL));
186 }
```

- 17-29: `alert_response_cb` is a call back function of the "response" signal of TfeAlert dialog. This is the same as before except `gtk_window_destroy(GTK_WINDOW (win))` is used instead of `tfe_application_quit`.
- 31-102: Handlers of action activated signal. The `user_data` is a pointer to TfeWindow instance.
- 104-115: A handler of "changed::font" signal of GSettings object.
- 111: Gets the font from GSettings data.
- 112: Gets a PangoFontDescription from the font. In the previous version, the program gets the font description from the GtkFontButton. The button data and GSettings data are the same. Therefore, the data got here is the same as the data in the GtkFontButton. In addition, we don't need to worry about the preference dialog is alive or not thanks to the GSettings.
- 114: Sets CSS on the display with the font description.
- 117-132: Public functions.
- 134-141: Dispose handler. The GSettings object needs to be released.
- 143-171: Instance initialization function.
- 148: Creates a composite widget instance with the template.
- 151-153: Insert `menu` to the menu button.
- 155-156: Creates a GSettings instance with the id "com.github.ToshioCP.tfe". Connects "changed::font" signal to the handler `changed_font_cb`. This signal emits when the GSettings data is changed. The second part "font" of the signal name "changed::font" is called details. Signals can have details. If a GSettings instance has more than one key, "changed" signal emits only if the key, which has the same name as the detail, changes its value. For example, Suppose a GSettings object has three keys "a", "b" and "c".
  - "changed::a" is emitted when the value of the key "a" is changed. It isn't emitted when the value of "b" or "c" is changed.
  - "changed::b" is emitted when the value of the key "b" is changed. It isn't emitted when the

value of “a” or “c” is changed.
  - “changed::c” is emitted when the value of the key “c” is changed. It isn’t emitted when the value of “a” or “b” is changed. In this version of tfe, there is only one key (“font”). So, even if the signal doesn’t have a detail, the result is the same. But in the future version, it will probably need details.
- 158-168: Creates actions.
- 170: Sets CSS font.
- 173-181: Class initialization function.
- 177: Sets the dispose handler.
- 178: Sets the composite widget template
- 179-180: Binds private variable with child objects in the template.
- 183-186: `tfe_window_new`. This function creates TfeWindow instance.

21.4 TfeApplication

The file `tfeapplication.c` is now very simple.

```
1  #include "tfewindow.h"
2
3  /* ----- activate, open, startup handlers ----- */
4  static void
5  app_activate (GApplication *application) {
6    GtkApplication *app = GTK_APPLICATION (application);
7    GtkWidget *win = GTK_WIDGET (gtk_application_get_active_window (app));
8
9    tfe_window_notebook_page_new (TFE_WINDOW (win));
10   gtk_widget_show (GTK_WIDGET (win));
11 }
12
13 static void
14 app_open (GApplication *application, GFile ** files, gint n_files, const gchar
       *hint) {
15   GtkApplication *app = GTK_APPLICATION (application);
16   GtkWidget *win = GTK_WIDGET (gtk_application_get_active_window (app));
17
18   tfe_window_notebook_page_new_with_files (TFE_WINDOW (win), files, n_files);
19   gtk_widget_show (win);
20 }
21
22 static void
23 app_startup (GApplication *application) {
24   GtkApplication *app = GTK_APPLICATION (application);
25   int i;
26
27   tfe_window_new (app);
28
29 /* ----- accelerator ----- */
30   struct {
31     const char *action;
32     const char *accels[2];
33   } action_accels[] = {
34     { "win.open", { "<Control>o", NULL } },
35     { "win.save", { "<Control>s", NULL } },
36     { "win.close", { "<Control>w", NULL } },
37     { "win.new", { "<Control>n", NULL } },
38     { "win.saveas", { "<Shift><Control>s", NULL } },
39     { "win.close-all", { "<Control>q", NULL } },
40   };
41
42   for (i = 0; i < G_N_ELEMENTS(action_accels); i++)
43     gtk_application_set_accels_for_action(GTK_APPLICATION(app),
           action_accels[i].action, action_accels[i].accels);
44 }
```

```
45
46 /* ----- main ----- */
47
48 #define APPLICATION_ID "com.github.ToshioCP.tfe"
49
50 int
51 main (int argc, char **argv) {
52   GtkApplication *app;
53   int stat;
54
55   app = gtk_application_new (APPLICATION_ID, G_APPLICATION_HANDLES_OPEN);
56
57   g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
58   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
59   g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);
60
61   stat =g_application_run (G_APPLICATION (app), argc, argv);
62   g_object_unref (app);
63   return stat;
64 }
```

- 4-11: Activate signal handler. It uses `tfe_window_notebook_page_new` instead of `notebook_page_new`.
- 13-20: Open signal handler. Thanks to `tfe_window_notebook_page_new_with_files`, this handler becomes very simple.
- 22-44: Startup signal handler. Most of the tasks are moved to TfeWindow, the remaining tasks are creating a window and setting accelerations.
- 48-64: A function `main`.

21.5 Other files

Resource XML file.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <gresources>
3   <gresource prefix="/com/github/ToshioCP/tfe">
4     <file>tfewindow.ui</file>
5     <file>tfepref.ui</file>
6     <file>tfealert.ui</file>
7     <file>menu.ui</file>
8   </gresource>
9 </gresources>
```

GSchema XML file

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <schemalist>
3   <schema path="/com/github/ToshioCP/tfe/" id="com.github.ToshioCP.tfe">
4     <key name="font" type="s">
5       <default>'Monospace 12'</default>
6       <summary>Font</summary>
7       <description>The font to be used for textview.</description>
8     </key>
9   </schema>
10 </schemalist>
```

Meson.build

```
1 project('tfe', 'c')
2
3 gtkdep = dependency('gtk4')
4
5 gnome=import('gnome')
6 resources = gnome.compile_resources('resources','tfe.gresource.xml')
7 gnome.compile_schemas(build_by_default: true, depend_files:
      'com.github.ToshioCP.tfe.gschema.xml')
```

```
8
9  sourcefiles=files('tfeapplication.c', 'tfewindow.c', 'tfenotebook.c', 'tfepref.c',
       'tfealert.c', 'css.c', '../tfetextview/tfetextview.c')
10
11 executable('tfe', sourcefiles, resources, dependencies: gtkdep, export_dynamic:
       true, install: true)
12
13 schema_dir = get_option('prefix') / get_option('datadir') / 'glib-2.0/schemas/'
14 install_data('com.github.ToshioCP.tfe.gschema.xml', install_dir: schema_dir)
15 meson.add_install_script('glib-compile-schemas', schema_dir)
```

21.6 Compilation and installation.

If you build Gtk4 from the source, use `--prefix` option.

```
$ meson --prefix=$HOME/local _build
$ ninja -C _build
$ ninja -C _build install
```

If you install Gtk4 from the distribution packages, you don't need the prefix option. Maybe you need root privilege to install it.

```
$ meson _build
$ ninja -C _build
$ ninja -C _build install  # or 'sudo ninja -C _build install'
```

Source files are in src/tfe7 directory.

We made a very small text editor. You can add features to this editor. When you add a new feature, care about the structure of the program. Maybe you need to divide a file into several files like this section. It isn't good to put many things into one file. And it is important to think about the relationship between source files and widget structures. It is appropriate that they correspond to each other in many cases.

22 GtkDrawingArea and Cairo

If you want to draw dynamically on the screen, like an image window of gimp graphics editor, the GtkDrawingArea widget is the most suitable widget. You can freely draw or redraw an image in this widget. This is called custom drawing.

GtkDrawingArea provides a cairo drawing context so users can draw images by using cairo functions. In this section, I will explain:

1. Cairo, but only briefly; and
2. GtkDrawingArea, with a very simple example.

22.1 Cairo

Cairo is a set of two dimensional graphical drawing functions (or graphics library). There is a lot of documentation on Cairo's website. If you aren't familiar with Cairo, it is worth reading their tutorial.

The following is a gentle introduction to the Cairo library and how to use it. Firstly, in order to use Cairo you need to know about surfaces, sources, masks, destinations, cairo context and transformations.

- A surface represents an image. It is like a canvas. We can draw shapes and images with different colors on surfaces.
- The source pattern, or simply source, is like paint, which will be transferred to destination surface by cairo functions.
- The mask describes the area to be used in the copy;
- The destination is a target surface;
- The cairo context manages the transfer from source to destination, through mask with its functions; For example, `cairo_stroke` is a function to draw a path to the destination by the transfer.

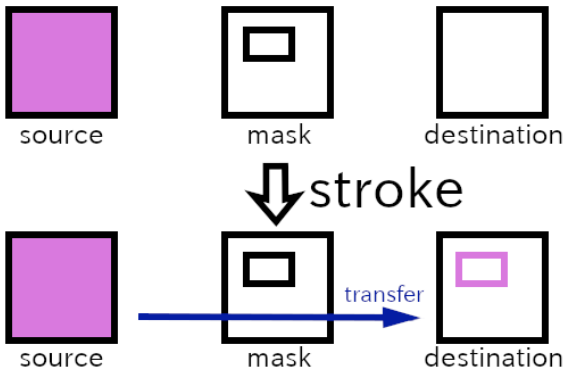

Figure 29: Stroke a rectangle

- A transformation can be applied before the transfer completes. The transformation which is applied is called affine, which is a mathematical term meaning transofrmations that preserve straight lines. Scaling, rotating, reflecting, shearing and translating are all examples of affine transformations. They are mathematically represented by matrix multiplication and vector addition. In this section we don't use it, instead we will only use the identity transformation. This means that the coordinates in the source and mask are the same as the coordinates in destination.

The instruction is as follows:

1. Create a surface. This will be the destination.
2. Create a cairo context with the surface, the surface will be the destination of the context.
3. Create a source pattern within the context.
4. Create paths, which are lines, rectangles, arcs, texts or more complicated shapes in the mask.
5. Use a drawing operator such as `cairo_stroke` to transfer the paint in the source to the destination.
6. Save the destination surface to a file if necessary.

Here's a simple example program that draws a small square and saves it as a png file.

```
1  #include <cairo.h>
2
3  int
4  main (int argc, char **argv)
5  {
6    cairo_surface_t *surface;
7    cairo_t *cr;
8    int width = 100;
9    int height = 100;
10   int square_size = 40.0;
11
12   /* Create surface and cairo */
13   surface = cairo_image_surface_create (CAIRO_FORMAT_RGB24, width, height);
14   cr = cairo_create (surface);
15
16   /* Drawing starts here. */
17   /* Paint the background white */
18   cairo_set_source_rgb (cr, 1.0, 1.0, 1.0);
19   cairo_paint (cr);
20   /* Draw a black rectangle */
21   cairo_set_source_rgb (cr, 0.0, 0.0, 0.0);
22   cairo_set_line_width (cr, 2.0);
23   cairo_rectangle (cr,
24                    width/2.0 - square_size/2,
25                    height/2.0 - square_size/2,
26                    square_size,
27                    square_size);
```

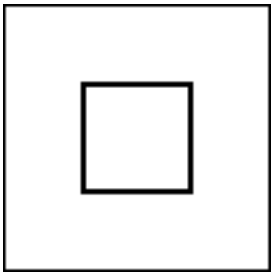

Figure 30: rectangle.png

```
28    cairo_stroke (cr);
29
30    /* Write the surface to a png file and clean up cairo and surface. */
31    cairo_surface_write_to_png (surface, "rectangle.png");
32    cairo_destroy (cr);
33    cairo_surface_destroy (surface);
34
35    return 0;
36  }
```

- 1: Includes the header file of Cairo.
- 6: `cairo_surface_t` is the type of a surface.
- 7: `cairo_t` is the type of a cairo context.
- 8-10: `width` and `height` are the size of `surface`. `square_size` is the size of a square to be drawn on the surface.
- 13: `cairo_image_surface_create` creates an image surface. `CAIRO_FORMAT_RGB24` is a constant which means that each pixel has red, green and blue data, and each data point is an 8 bits number (for 24 bits in total). Modern displays have this type of color depth. Width and height are in pixels and given as integers.
- 14: Creates cairo context. The surface given as an argument will be the destination of the context.
- 18: `cairo_set_source_rgb` creates a source pattern, which in this case is a solid white paint. The second to fourth argument are red, green and blue color values respectively, and they are of type float. The values are between zero (0.0) and one (1.0), with black being given by (0.0,0.0,0.0) and white by (1.0,1.0,1.0).
- 19: `cairo_paint` copies everywhere in the source to destination. The destination is filled with white pixels with this command.
- 21: Sets the source color to black.
- 22: `cairo_set_line_width` set the width of lines. In this case, the line width is set to be two pixels and will end up that same size. (It is because the transformation is identity. If the transformation isn't identity, for example scaling with the factor three, the actual width in destination will be six (2x3=6) pixels.)
- 23: Draws a rectangle (square) on the mask. The square is located at the center.
- 24: `cairo_stroke` transfer the source to destination through the rectangle in the mask.
- 27: Outputs the image to a png file `rectangle.png`.
- 28: Destroys the context. At the same time the source is destroyed.
- 29: Destroys the surface.

To compile this, type the following.

```
$ gcc `pkg-config --cflags cairo` cairo.c `pkg-config --libs cairo`
```

See the Cairo's website for more details.

22.2 GtkDrawingArea

The following is a very simple example.

```
1  #include <gtk/gtk.h>
2
3  static void
```

```
4  draw_function (GtkDrawingArea *area, cairo_t *cr, int width, int height, gpointer
       user_data) {
5    int square_size = 40.0;
6
7    cairo_set_source_rgb (cr, 1.0, 1.0, 1.0); /* white */
8    cairo_paint (cr);
9    cairo_set_line_width (cr, 2.0);
10   cairo_set_source_rgb (cr, 0.0, 0.0, 0.0); /* black */
11   cairo_rectangle (cr,
12                    width/2.0 - square_size/2,
13                    height/2.0 - square_size/2,
14                    square_size,
15                    square_size);
16   cairo_stroke (cr);
17 }
18
19 static void
20 app_activate (GApplication *app, gpointer user_data) {
21   GtkWidget *win = gtk_application_window_new (GTK_APPLICATION (app));
22   GtkWidget *area = gtk_drawing_area_new ();
23
24   gtk_window_set_title (GTK_WINDOW (win), "da1");
25   gtk_drawing_area_set_draw_func (GTK_DRAWING_AREA (area), draw_function, NULL,
         NULL);
26   gtk_window_set_child (GTK_WINDOW (win), area);
27
28   gtk_widget_show (win);
29 }
30
31 #define APPLICATION_ID "com.github.ToshioCP.da1"
32
33 int
34 main (int argc, char **argv) {
35   GtkApplication *app;
36   int stat;
37
38   app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
39   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
40   stat =g_application_run (G_APPLICATION (app), argc, argv);
41   g_object_unref (app);
42   return stat;
43 }
```

The function `main` is almost same as before. The two functions `app_activate` and `draw_function` are important in this example.

- 18: Creates a GtkDrawingArea instance; and
- 21: Sets a drawing function of the widget. GtkDrawingArea widget uses the function to draw the contents of itself whenever its necessary. For example, when a user drag a mouse pointer and resize a top-level window, GtkDrawingArea also changes the size. Then, the whole window needs to be redrawn. For the information of `gtk_drawing_area_set_draw_func`, see Gtk API Reference, gtk_drawing_area_set_draw_func.

The drawing function has five parameters.

```
void drawing_function (GtkDrawingArea *drawing_area, cairo_t *cr, int width, int
    height,
                       gpointer user_data);
```

The first parameter is the GtkDrawingArea widget. You can't change any properties, for example `content-width` or `content-height`, in this function. The second parameter is a cairo context given by the widget. The destination surface of the context is connected to the contents of the widget. What you draw to this surface will appear in the widget on the screen. The third and fourth parameters are the size of the destination surface. Now, look at the program example again.

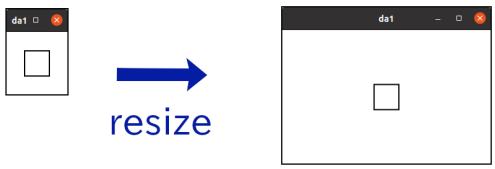

Figure 31: Square in the window

- 3-13: The drawing function.
- 7-8: Sets the source to be white and paint the destination white.
- 9: Sets the line width to be 2.
- 10: Sets the source to be black.
- 11: Adds a rectangle to the mask.
- 12: Draws the rectangle with black color to the destination.

Compile and run it, then a window with a black rectangle (square) appears. Try resizing the window. The square always appears at the center of the window because the drawing function is invoked each time the window is resized.

23 Periodic Events

This chapter was written by Paul Schulz paul@mawsonlakes.org.

23.1 How do we create an animation?

In this section we will continue to build on our previous work. We will create an analog clock application. By adding a function which periodically redraws GtkDrawingArea, the clock will be able to continuously display the time.

The application uses a compiled in 'resource' file, so if the GTK4 libraries and their dependencies are installed and available, the application will run from anywhere.

The program also makes use of some standard mathematical and time handling functions.

The clocks mechanics were taken from a Cairo drawing example, using gtkmm4, which can be found here.

The complete code is at the end.

23.2 Drawing the clock face, hour, minute and second hands

The `draw_clock()` function does all the work. See the in-file comments for an explanation of how the Cairo drawing works.

For a detailed reference of what each of the Cairo functions does see the cairo_t reference.

```
1  static void
2  draw_clock (GtkDrawingArea *area, cairo_t *cr, int width, int height, gpointer
       user_data) {
3
4      // Scale to unit square and translate (0, 0) to be (0.5, 0.5), i.e.
5      // the center of the window
6      cairo_scale(cr, width, height);
7      cairo_translate(cr, 0.5, 0.5);
8
9      // Set the line width and save the cairo drawing state.
10     cairo_set_line_width(cr, m_line_width);
11     cairo_save(cr);
12
13     // Set the background to a slightly transparent green.
14     cairo_set_source_rgba(cr, 0.337, 0.612, 0.117, 0.9);   // green
```

```
15      cairo_paint(cr);
16
17      // Resore back to precious drawing state and draw the circular path
18      // representing the clockface. Save this state (including the path) so we
19      // can reuse it.
20      cairo_restore(cr);
21      cairo_arc(cr, 0.0, 0.0, m_radius, 0.0, 2.0 * M_PI);
22      cairo_save(cr);
23
24      // Fill the clockface with white
25      cairo_set_source_rgba(cr, 1.0, 1.0, 1.0, 0.8);
26      cairo_fill_preserve(cr);
27      // Restore the path, paint the outside of the clock face.
28      cairo_restore(cr);
29      cairo_stroke_preserve(cr);
30      // Set the 'clip region' to the inside of the path (fill region).
31      cairo_clip(cr);
32
33      // Clock ticks
34      for (int i = 0; i < 12; i++)
35      {
36          // Major tick size
37          double inset = 0.05;
38
39          // Save the graphics state, restore after drawing tick to maintain pen
40          // size
41          cairo_save(cr);
42          cairo_set_line_cap(cr, CAIRO_LINE_CAP_ROUND);
43
44          // Minor ticks are shorter, and narrower.
45          if(i % 3 != 0)
46          {
47              inset *= 0.8;
48              cairo_set_line_width(cr, 0.03);
49          }
50
51          // Draw tick mark
52          cairo_move_to(
53              cr,
54              (m_radius - inset) * cos (i * M_PI / 6.0),
55              (m_radius - inset) * sin (i * M_PI / 6.0));
56          cairo_line_to(
57              cr,
58              m_radius * cos (i * M_PI / 6.0),
59              m_radius * sin (i * M_PI / 6.0));
60          cairo_stroke(cr);
61          cairo_restore(cr); /* stack-pen-size */
62      }
63
64      // Draw the analog hands
65
66      // Get the current Unix time, convert to the local time and break into time
67      // structure to read various time parts.
68      time_t rawtime;
69      time(&rawtime);
70      struct tm * timeinfo = localtime (&rawtime);
71
72      // Calculate the angles of the hands of our clock
73      double hours   = timeinfo->tm_hour * M_PI / 6.0;
74      double minutes = timeinfo->tm_min * M_PI / 30.0;
75      double seconds = timeinfo->tm_sec * M_PI / 30.0;
76
77      // Save the graphics state
78      cairo_save(cr);
```

```
79      cairo_set_line_cap(cr, CAIRO_LINE_CAP_ROUND);
80
81      cairo_save(cr);
82
83      // Draw the seconds hand
84      cairo_set_line_width(cr, m_line_width / 3.0);
85      cairo_set_source_rgba(cr, 0.7, 0.7, 0.7, 0.8);   // gray
86      cairo_move_to(cr, 0.0, 0.0);
87      cairo_line_to(cr,
88                    sin(seconds) * (m_radius * 0.9),
89                    -cos(seconds) * (m_radius * 0.9));
90      cairo_stroke(cr);
91      cairo_restore(cr);
92
93      // Draw the minutes hand
94      cairo_set_source_rgba(cr, 0.117, 0.337, 0.612, 0.9);   // blue
95      cairo_move_to(cr, 0, 0);
96      cairo_line_to(cr,
97                    sin(minutes + seconds / 60) * (m_radius * 0.8),
98                    -cos(minutes + seconds / 60) * (m_radius * 0.8));
99      cairo_stroke(cr);
100
101     // draw the hours hand
102     cairo_set_source_rgba(cr, 0.337, 0.612, 0.117, 0.9);   // green
103     cairo_move_to(cr, 0.0, 0.0);
104     cairo_line_to(cr,
105                   sin(hours + minutes / 12.0) * (m_radius * 0.5),
106                   -cos(hours + minutes / 12.0) * (m_radius * 0.5));
107     cairo_stroke(cr);
108     cairo_restore(cr);
109
110     // Draw a little dot in the middle
111     cairo_arc(cr, 0.0, 0.0, m_line_width / 3.0, 0.0, 2.0 * M_PI);
112     cairo_fill(cr);
113 }
```

In order for the clock to be drawn, the drawing function `draw_clock()` needs to be registered with GTK4. This is done in the `app_activate()` function (on line 24).

Whenever the application needs to redraw the GtkDrawingArea, it will now call `draw_clock()`.

There is still a problem though. In order to animate the clock we need to also tell the application that the clock needs to be redrawn every second. This process starts by registering (on the next line, line 15) a timeout function with `g_timeout_add()` that will wakeup and run another function `time_handler`, every second (or 1000ms).

```
1   static void
2   app_activate (GApplication *app, gpointer user_data) {
3       GtkWidget *win;
4       GtkWidget *clock;
5       GtkBuilder *build;
6
7       build = gtk_builder_new_from_resource ("/com/github/ToshioCP/tfc/tfc.ui");
8       win = GTK_WIDGET (gtk_builder_get_object (build, "win"));
9       gtk_window_set_application (GTK_WINDOW (win), GTK_APPLICATION (app));
10
11      clock = GTK_WIDGET (gtk_builder_get_object (build, "clock"));
12      g_object_unref(build);
13
14      gtk_drawing_area_set_draw_func(GTK_DRAWING_AREA (clock), draw_clock, NULL, NULL);
15      g_timeout_add(1000, (GSourceFunc) time_handler, (gpointer) clock);
16      gtk_widget_show(win);
17
18  }
```

Our `time_handler()` function is very simple, as it just calls `gtk_widget_queue_draw()` which schedules a redraw of the widget.

```
gboolean
time_handler(GtkWidget* widget) {
    gtk_widget_queue_draw(widget);

    return TRUE;
}
```

.. and that is all there is to it. If you compile and run the example you will get a ticking analog clock.

If you get this working, you can try modifying some of the code in `draw_clock()` to tweak the application (such as change the color or size and length of the hands) or even add text, or create a digital clock.

23.3 The Complete code

You can find the source files in the `tfc` directory. it can be compiled with `./comp tfc`.

`tfc.c`

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

float m_radius     = 0.42;
float m_line_width = 0.05;

static void
draw_clock (GtkDrawingArea *area, cairo_t *cr, int width, int height, gpointer
    user_data) {

    // Scale to unit square and translate (0, 0) to be (0.5, 0.5), i.e.
    // the center of the window
    cairo_scale(cr, width, height);
    cairo_translate(cr, 0.5, 0.5);

    // Set the line width and save the cairo drawing state.
    cairo_set_line_width(cr, m_line_width);
    cairo_save(cr);

    // Set the background to a slightly transparent green.
    cairo_set_source_rgba(cr, 0.337, 0.612, 0.117, 0.9);   // green
    cairo_paint(cr);

    // Resore back to precious drawing state and draw the circular path
    // representing the clockface. Save this state (including the path) so we
    // can reuse it.
    cairo_restore(cr);
    cairo_arc(cr, 0.0, 0.0, m_radius, 0.0, 2.0 * M_PI);
    cairo_save(cr);

    // Fill the clockface with white
    cairo_set_source_rgba(cr, 1.0, 1.0, 1.0, 0.8);
    cairo_fill_preserve(cr);
    // Restore the path, paint the outside of the clock face.
    cairo_restore(cr);
    cairo_stroke_preserve(cr);
    // Set the 'clip region' to the inside of the path (fill region).
    cairo_clip(cr);

    // Clock ticks
    for (int i = 0; i < 12; i++)
    {
        // Major tick size
```

```
44          double inset = 0.05;
45
46          // Save the graphics state, restore after drawing tick to maintain pen
47          // size
48          cairo_save(cr);
49          cairo_set_line_cap(cr, CAIRO_LINE_CAP_ROUND);
50
51          // Minor ticks are shorter, and narrower.
52          if(i % 3 != 0)
53          {
54              inset *= 0.8;
55              cairo_set_line_width(cr, 0.03);
56          }
57
58          // Draw tick mark
59          cairo_move_to(
60              cr,
61              (m_radius - inset) * cos (i * M_PI / 6.0),
62              (m_radius - inset) * sin (i * M_PI / 6.0));
63          cairo_line_to(
64              cr,
65              m_radius * cos (i * M_PI / 6.0),
66              m_radius * sin (i * M_PI / 6.0));
67          cairo_stroke(cr);
68          cairo_restore(cr); /* stack-pen-size */
69      }
70
71      // Draw the analog hands
72
73      // Get the current Unix time, convert to the local time and break into time
74      // structure to read various time parts.
75      time_t rawtime;
76      time(&rawtime);
77      struct tm * timeinfo = localtime (&rawtime);
78
79      // Calculate the angles of the hands of our clock
80      double hours   = timeinfo->tm_hour * M_PI / 6.0;
81      double minutes = timeinfo->tm_min * M_PI / 30.0;
82      double seconds = timeinfo->tm_sec * M_PI / 30.0;
83
84      // Save the graphics state
85      cairo_save(cr);
86      cairo_set_line_cap(cr, CAIRO_LINE_CAP_ROUND);
87
88      cairo_save(cr);
89
90      // Draw the seconds hand
91      cairo_set_line_width(cr, m_line_width / 3.0);
92      cairo_set_source_rgba(cr, 0.7, 0.7, 0.7, 0.8);   // gray
93      cairo_move_to(cr, 0.0, 0.0);
94      cairo_line_to(cr,
95                    sin(seconds) * (m_radius * 0.9),
96                    -cos(seconds) * (m_radius * 0.9));
97      cairo_stroke(cr);
98      cairo_restore(cr);
99
100     // Draw the minutes hand
101     cairo_set_source_rgba(cr, 0.117, 0.337, 0.612, 0.9);   // blue
102     cairo_move_to(cr, 0, 0);
103     cairo_line_to(cr,
104                   sin(minutes + seconds / 60) * (m_radius * 0.8),
105                   -cos(minutes + seconds / 60) * (m_radius * 0.8));
106     cairo_stroke(cr);
107
```

```
108     // draw the hours hand
109     cairo_set_source_rgba(cr, 0.337, 0.612, 0.117, 0.9);   // green
110     cairo_move_to(cr, 0.0, 0.0);
111     cairo_line_to(cr,
112                   sin(hours + minutes / 12.0) * (m_radius * 0.5),
113                   -cos(hours + minutes / 12.0) * (m_radius * 0.5));
114     cairo_stroke(cr);
115     cairo_restore(cr);
116
117     // Draw a little dot in the middle
118     cairo_arc(cr, 0.0, 0.0, m_line_width / 3.0, 0.0, 2.0 * M_PI);
119     cairo_fill(cr);
120 }
121
122
123 gboolean
124 time_handler(GtkWidget* widget) {
125     gtk_widget_queue_draw(widget);
126
127     return TRUE;
128 }
129
130
131 static void
132 app_activate (GApplication *app, gpointer user_data) {
133     GtkWidget *win;
134     GtkWidget *clock;
135     GtkBuilder *build;
136
137     build = gtk_builder_new_from_resource ("/com/github/ToshioCP/tfc/tfc.ui");
138     win = GTK_WIDGET (gtk_builder_get_object (build, "win"));
139     gtk_window_set_application (GTK_WINDOW (win), GTK_APPLICATION (app));
140
141     clock = GTK_WIDGET (gtk_builder_get_object (build, "clock"));
142     g_object_unref(build);
143
144     gtk_drawing_area_set_draw_func(GTK_DRAWING_AREA (clock), draw_clock, NULL, NULL);
145     g_timeout_add(1000, (GSourceFunc) time_handler, (gpointer) clock);
146     gtk_widget_show(win);
147
148 }
149
150 static void
151 app_open (GApplication *app, GFile **files, gint n_files, gchar *hint, gpointer
        user_data) {
152     app_activate(app,user_data);
153 }
154
155 int
156 main (int argc, char **argv) {
157     GtkApplication *app;
158     int stat;
159
160     app = gtk_application_new ("com.github.ToshioCP.tfc",
            G_APPLICATION_HANDLES_OPEN);
161     g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
162     g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);
163     stat = g_application_run (G_APPLICATION (app), argc, argv);
164     g_object_unref (app);
165     return stat;
166 }
```

tfc.ui

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

Figure 32: color

```
<interface>
  <object class="GtkApplicationWindow" id="win">
    <property name="title">Clock</property>
    <property name="default-width">200</property>
    <property name="default-height">200</property>
    <child>
      <object class="GtkDrawingArea" id="clock">
        <property name="hexpand">TRUE</property>
        <property name="vexpand">TRUE</property>
      </object>
    </child>
  </object>
</interface>
```

tfc.gresource.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<gresources>
  <gresource prefix="/com/github/ToshioCP/tfc">
    <file>tfc.ui</file>
  </gresource>
</gresources>
```

comp

```
glib-compile-resources $1.gresource.xml --target=$1.gresource.c --generate-source
gcc `pkg-config --cflags gtk4` $1.gresource.c $1.c `pkg-config --libs gtk4` -lm
```

24 Combine GtkDrawingArea and TfeTextView

Now, we will make a new application which has GtkDrawingArea and TfeTextView in it. Its name is "color". If you write a name of a color in TfeTextView and click on the `run` button, then the color of GtkDrawingArea changes to the color given by you.

The following colors are available.

- white
- black
- red
- green
- blue

In addition the following two options are also available.

- light: Make the color of the drawing area lighter.
- dark: Make the color of the drawing area darker.

This application can only do very simple things. However, it tells us that if we add powerful parser to it, we will be able to make it more efficient. I want to show it to you in the later section by making a turtle graphics language like Logo program language.

In this section, we focus on how to bind the two objects.

24.1 Color.ui and color.gresource.xml

First, We need to make the ui file of the widgets. The image in the previous subsection gives us the structure of the widgets. Title bar, four buttons in the tool bar and two widgets textview and drawing area. The ui file is as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <object class="GtkApplicationWindow" id="win">
    <property name="title">color changer</property>
    <property name="default-width">600</property>
    <property name="default-height">400</property>
    <child>
      <object class="GtkBox" id="boxv">
        <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
        <child>
          <object class="GtkBox" id="boxh1">
            <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
            <child>
              <object class="GtkLabel" id="dmy1">
                <property name="width-chars">10</property>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btnr">
                <property name="label">Run</property>
                <signal name="clicked" handler="run_cb"></signal>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btno">
                <property name="label">Open</property>
                <signal name="clicked" handler="open_cb"></signal>
              </object>
            </child>
            <child>
              <object class="GtkLabel" id="dmy2">
                <property name="hexpand">TRUE</property>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btns">
                <property name="label">Save</property>
                <signal name="clicked" handler="save_cb"></signal>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btnc">
                <property name="label">Close</property>
                <signal name="clicked" handler="close_cb"></signal>
              </object>
            </child>
            <child>
              <object class="GtkLabel" id="dmy3">
                <property name="width-chars">10</property>
              </object>
            </child>
          </object>
```

```
53          </child>
54          <child>
55            <object class="GtkBox" id="boxh2">
56              <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
57              <property name="homogeneous">TRUE</property>
58              <child>
59                <object class="GtkScrolledWindow" id="scr">
60                  <property name="hexpand">TRUE</property>
61                  <property name="vexpand">TRUE</property>
62                  <child>
63                    <object class="TfeTextView" id="tv">
64                      <property name="wrap-mode">GTK_WRAP_WORD_CHAR</property>
65                    </object>
66                  </child>
67                </object>
68              </child>
69              <child>
70                <object class="GtkDrawingArea" id="da">
71                  <property name="hexpand">TRUE</property>
72                  <property name="vexpand">TRUE</property>
73                </object>
74              </child>
75            </object>
76          </child>
77        </object>
78      </child>
79    </object>
80  </interface>
```

- 10-53: This part is the tool bar which has four buttons, `Run`, `Open`, `Save` and `Close`. This is similar to the toolbar of tfe text editor in Section 9. There are two differences. `Run` button replaces `New` button. A signal element is added to each button object. It has "name" attribute which is a signal name and "handler" attribute which is the name of its signal handler function. Options "-WI, –export-dynamic" CFLAG is necessary when you compile the application. You can achieve this by adding "export_dynamic: true" argument to executable function in `meson.build`. And be careful that the handler must be defined without 'static' class.
- 54-76: Puts GtkScrolledWindow and GtkDrawingArea into GtkBox. GtkBox has "homogeneous property" with TRUE value, so the two children have the same width in the box. TfeTextView is a child of GtkScrolledWindow.

The xml file for the resource compiler is almost same as before. Just substitute "color" for "tfe".

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <gresources>
3    <gresource prefix="/com/github/ToshioCP/color">
4      <file>color.ui</file>
5    </gresource>
6  </gresources>
```

24.2 Tfetextview.h, tfetextview.c and color.h

First two files are the same as before. Color.h just includes tfetextview.h.

```
1  #include <gtk/gtk.h>
2
3  #include "../tfetextview/tfetextview.h"
```

24.3 Colorapplication.c

This is the main file. It deals with:

- Building widgets by GtkBuilder.
- Setting a drawing function of GtkDrawingArea. And connecting a handler to "resize" signal on GtkDrawingArea.

- Implementing each call back functions. Particularly, `Run` signal handler is the point in this program.

The following is `colorapplication.c`.

```
#include "color.h"

static GtkWidget *win;
static GtkWidget *tv;
static GtkWidget *da;

static cairo_surface_t *surface = NULL;

static void
run (void) {
  GtkTextBuffer *tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
  GtkTextIter start_iter;
  GtkTextIter end_iter;
  char *contents;
  cairo_t *cr;

  gtk_text_buffer_get_bounds (tb, &start_iter, &end_iter);
  contents = gtk_text_buffer_get_text (tb, &start_iter, &end_iter, FALSE);
  if (surface) {
    cr = cairo_create (surface);
    if (g_strcmp0 ("red", contents) == 0)
      cairo_set_source_rgb (cr, 1, 0, 0);
    else if (g_strcmp0 ("green", contents) == 0)
      cairo_set_source_rgb (cr, 0, 1, 0);
    else if (g_strcmp0 ("blue", contents) == 0)
      cairo_set_source_rgb (cr, 0, 0, 1);
    else if (g_strcmp0 ("white", contents) == 0)
      cairo_set_source_rgb (cr, 1, 1, 1);
    else if (g_strcmp0 ("black", contents) == 0)
      cairo_set_source_rgb (cr, 0, 0, 0);
    else if (g_strcmp0 ("light", contents) == 0)
      cairo_set_source_rgba (cr, 1, 1, 1, 0.5);
    else if (g_strcmp0 ("dark", contents) == 0)
      cairo_set_source_rgba (cr, 0, 0, 0, 0.5);
    else
      cairo_set_source_surface (cr, surface, 0, 0);
    cairo_paint (cr);
    cairo_destroy (cr);
  }
  g_free (contents);
}

void
run_cb (GtkWidget *btnr) {
  run ();
  gtk_widget_queue_draw (GTK_WIDGET (da));
}

void
open_cb (GtkWidget *btno) {
  tfe_text_view_open (TFE_TEXT_VIEW (tv), GTK_WINDOW (win));
}

void
save_cb (GtkWidget *btns) {
  tfe_text_view_save (TFE_TEXT_VIEW (tv));
}

void
close_cb (GtkWidget *btnc) {
  if (surface)
```

```
62      cairo_surface_destroy (surface);
63    gtk_window_destroy (GTK_WINDOW (win));
64  }
65
66  static void
67  resize_cb (GtkDrawingArea *drawing_area, int width, int height, gpointer user_data) {
68    if (surface)
69      cairo_surface_destroy (surface);
70    surface = cairo_image_surface_create (CAIRO_FORMAT_ARGB32, width, height);
71    run ();
72  }
73
74  static void
75  draw_func (GtkDrawingArea *drawing_area, cairo_t *cr, int width, int height,
        gpointer user_data) {
76    if (surface) {
77      cairo_set_source_surface (cr, surface, 0, 0);
78      cairo_paint (cr);
79    }
80  }
81
82  static void
83  app_activate (GApplication *application) {
84    gtk_widget_show (win);
85  }
86
87  static void
88  app_startup (GApplication *application) {
89    GtkApplication *app = GTK_APPLICATION (application);
90    GtkBuilder *build;
91
92    build = gtk_builder_new_from_resource ("/com/github/ToshioCP/color/color.ui");
93    win = GTK_WIDGET (gtk_builder_get_object (build, "win"));
94    gtk_window_set_application (GTK_WINDOW (win), app);
95    tv = GTK_WIDGET (gtk_builder_get_object (build, "tv"));
96    da = GTK_WIDGET (gtk_builder_get_object (build, "da"));
97    g_object_unref(build);
98    g_signal_connect (GTK_DRAWING_AREA (da), "resize", G_CALLBACK (resize_cb), NULL);
99    gtk_drawing_area_set_draw_func (GTK_DRAWING_AREA (da), draw_func, NULL, NULL);
100
101 GdkDisplay *display;
102
103   display = gtk_widget_get_display (GTK_WIDGET (win));
104   GtkCssProvider *provider = gtk_css_provider_new ();
105   gtk_css_provider_load_from_data (provider, "textview {padding: 10px; font-family:
          monospace; font-size: 12pt;}", -1);
106   gtk_style_context_add_provider_for_display (display, GTK_STYLE_PROVIDER
          (provider), GTK_STYLE_PROVIDER_PRIORITY_USER);
107 }
108
109 #define APPLICATION_ID "com.github.ToshioCP.color"
110
111 int
112 main (int argc, char **argv) {
113   GtkApplication *app;
114   int stat;
115
116   app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
117
118   g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
119   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
120
121   stat =g_application_run (G_APPLICATION (app), argc, argv);
122   g_object_unref (app);
```

```
123   return stat;
124 }
```

- 109-124: The function `main` is almost same as before but there are some differences. The application ID is "com.github.ToshioCP.color". `G_APPLICATION_FLAGS_NONE` is specified so no open signal handler is necessary.
- 87-107: Startup handler.
- 92-97: Builds widgets. The pointers of the top window, TfeTextView and GtkDrawingArea objects are stored to static variables `win`, `tv` and `da` respectively. This is because these objects are often used in handlers. They never be rewritten so they're thread safe.
- 98: connects "resize" signal and the handler.
- 99: sets the drawing function.
- 82-85: Activate handler, which just shows the widgets.
- 74-80: The drawing function. It just copies `surface` to destination.
- 66-72: Resize handler. Re-creates the surface to fit its width and height for the drawing area and paints by calling the function `run`.
- 59-64: Close handler. It destroys `surface` if it exists. Then it destroys the top-level window and quits the application.
- 49-57: Open and save handler. They just call the corresponding functions of TfeTextView.
- 43-47: Run handler. It calls run function to paint the surface. After that `gtk_widget_queue_draw` is called. This function adds the widget (GtkDrawingArea) to the queue to be redrawn. It is important to know that the window is redrawn whenever it is necessary. For example, when another window is moved and uncovers part of the widget, or when the window containing it is resized. But repainting `surface` is not automatically notified to gtk. Therefore, you need to call `gtk_widget_queue_draw` to redraw the widget.
- 9-41: Run function paints the surface. First, it gets the contents of GtkTextBuffer. Then it compares it to "red", "green" and so on. If it matches the color, then the surface is painted the color. If it matches "light" or "dark", then the color of the surface is lightened or darkened respectively. Alpha channel is used.

24.4 Meson.build

This file is almost same as before. An argument "export_dynamic: true" is added to executable function.

```
1  project('color', 'c')
2
3  gtkdep = dependency('gtk4')
4
5  gnome=import('gnome')
6  resources = gnome.compile_resources('resources','color.gresource.xml')
7
8  sourcefiles=files('colorapplication.c', '../tfetextview/tfetextview.c')
9
10 executable('color', sourcefiles, resources, dependencies: gtkdep, export_dynamic:
       true)
```

24.5 Compile and execute it

First you need to export some variables (refer to Section 2) if you've installed Gtk4 from the source. If you've installed Gtk4 from the distribution packages, you don't need to do this.

```
$ . env.sh
```

Then type the following to compile it.

```
$ meson _build
$ ninja -C _build
```

The application is made in `_build` directory. Type the following to execute it.

```
$ _build/color
```

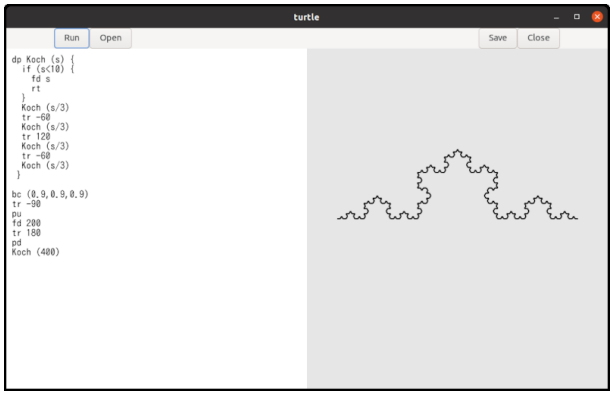

Figure 33: Koch curve

Type “red”, “green”, “blue”, “white”, black“,”light” or “dark” in the TfeTextView. Then, click on `Run` button. Make sure the color of GtkDrawingArea changes.

In this program TfeTextView is used to change the color. You can use buttons or menus instead of textview. Probably it is more appropriate. Using textview is unnatural. It is a good practice to make such application by yourself.

25 Tiny turtle graphics interpreter

A program `turtle` is an example with the combination of TfeTextView and GtkDrawingArea objects. It is a very small interpreter but it provides a tool to draw fractal curves. The following diagram is a Koch curve, which is a famous example of fractal curves.

This program uses flex and bison. Flex is a lexical analyzer. Bison is a parser generator. These two programs are similar to lex and yacc which are proprietary software developed in Bell Laboratory. However, flex and bison are open source software. I will write about how to use those software, but they are not topics about gtk. So, readers can skip that part of this sections.

25.1 How to use turtle

The documentation of turtle is in the appendix. I'll show you a simple example.

```
fc (1,0,0) # Foreground color is red, rgb = (1,0,0).
pd         # Pen down.
fd 100     # Go forward by 100 pixels.
tr 90      # Turn right by 90 degrees.
fd 100
tr 90
fd 100
tr 90
fd 100
tr 90
```

1. Compile and install `turtle` (See the documentation above). Then, run `turtle`.
2. Type the program above in the editor (left part of the window).
3. Click on the `Run` button, then a red square appears on the right part of the window. The side of the square is 100 pixels long.

In the same way, you can draw other curves. The documentation above shows some fractal curves such as tree, snow and square-koch. The source code in turtle language is located at src/turtle/example directory. You can read these files into `turtle` editor by clicking on the `Open` button.

25.2 Combination of TfeTextView and GtkDrawingArea objects

Turtle uses TfeTextView and GtkDrawingArea. It is similar to `color` program in the previous section.

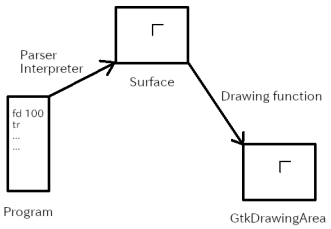

Figure 34: Parser, interpreter and drawing function

1. A user inputs/reads a turtle program into the buffer in the TfeTextView instance.
2. The user clicks on the “Run” button.
3. The parser reads the program and generates tree-structured data.
4. The interpriter reads the data and executes it step by step. And it draws shapes on a surface. The surface is different from the surface of the GtkDrawingArea widget.
5. The widget is added to the queue. It will be redrawn with the drawing function. The function just copies the surface, which is drawn by the interpreter, into the surface of the GtkDrawingArea.

The body of the interpreter is written with flex and bison. The codes are not thread safe. So the handler of “clicked” signal of the `Run` button prevents from reentering.

```
1  void
2  run_cb (GtkWidget *btnr) {
3    GtkTextBuffer *tb = gtk_text_view_get_buffer (GTK_TEXT_VIEW (tv));
4    GtkTextIter start_iter;
5    GtkTextIter end_iter;
6    char *contents;
7    int stat;
8    static gboolean busy = FALSE;
9
10   /* yyparse() and run() are NOT thread safe. */
11   /* The variable busy avoids reentry. */
12   if (busy)
13     return;
14   busy = TRUE;
15   gtk_text_buffer_get_bounds (tb, &start_iter, &end_iter);
16   contents = gtk_text_buffer_get_text (tb, &start_iter, &end_iter, FALSE);
17   if (surface) {
18     init_flex (contents);
19     stat = yyparse ();
20     if (stat == 0) /* No error */ {
21       run ();
22     }
23     finalize_flex ();
24   }
25   g_free (contents);
26   gtk_widget_queue_draw (GTK_WIDGET (da));
27   busy = FALSE;
28 }
29
30 static void
31 resize_cb (GtkDrawingArea *drawing_area, int width, int height, gpointer user_data) {
32   if (surface)
33     cairo_surface_destroy (surface);
34   surface = cairo_image_surface_create (CAIRO_FORMAT_ARGB32, width, height);
35 }
```

- 8-13: The static value `busy` holds a status of the interpreter. If it is `TRUE`, the interpreter is running and it is not possible to call the interpreter again because it’s not a re-entrant program. If it is `FALSE`, it is safe to call the interpreter.
- 14: Now it is about to call the interpreter so it changes `busy` to TRUE.

- 15-16: Gets the contents of `tb`.
- 17: The variable `surface` is a static variable. It points to a `cairo_surface_t` instance. It is created when the GtkDrawingArea instance is realized and whenever it is resized. Therefore, `surface` isn't NULL usually. But if it is NULL, the interpreter won't be called.
- 18: Initializes lexical analyzer.
- 19: Calls parser. Parser analyzes the program codes syntactically and generate a tree structured data.
- 20-22: If the parser successfully parsed, it calls `run` (runtime routine).
- 23: finalizes the lexical analyzer.
- 25: frees `contents`.
- 26: Adds the drawing area widget to the queue to draw.
- 27: The interpreter program has finished so `busy` is now changed to FALSE.
- 29-34: A handler of "resized" signal. If `surface` isn't NULL, it destroys the old surface. Then it creates a new surface. Its size is the same as the surface of the GtkDrawingArea instance.

Other part of `turtleapplication.c` is almost same as the codes of `colorapplication.c` in the previous section. The codes of `turtleapplication.c` is in the turtle directory.

25.3 What does the interpreter do?

Suppose that the turtle runs with the following program.

```
distance = 100
fd distance*2
```

The turtle recognizes the program above and works as follows.

- Generally, a program consists of tokens. Tokens are "distance", "=", "100", "fd", "*" and "2" in the above example..
- The parser calls a function `yylex` to read a token in the source file. `yylex` returns a code which is called "token kind" and sets a global variable `yylval` with a value, which is called a semantic value. The type of `yylval` is union and `yylval.ID` is string and `yylval.NUM` is double. There are seven tokens in the program so `yylex` is called seven times.

	token kind	yylval.ID	yylval.NUM
1	ID	distance	
2	=		
3	NUM		100
4	FD		
5	ID	distance	
6	*		
7	NUM		2

- `yylex` returns a token kind every time, but it doesn't set `yylval.ID` or `yylval.NUM` every time. It is because keywords (`FD`) and symbols (`=` and `*`) don't have any semantic values. The function `yylex` is called lexical analyzer or scanner.
- `turtle` makes a tree structured data. This part of `turtle` is called parser.
- `turtle` analyzes the tree and executes it. This part of `turtle` is called runtime routine or interpreter. The tree consists of rectangles and line segments between the rectangles. The rectangles are called nodes. For example, N_PROGRAM, N_ASSIGN, N_FD and N_MUL are nodes.
  1. Goes down from N_PROGRAM to N_ASSIGN.
  2. N_ASSIGN node has two children, ID and NUM. This node comes from "distance = 100" which is "ID = NUM" syntactically. First, `turtle` checks if the first child is ID. If it's ID, then `turtle` looks for the variable in the variable table. If it doesn't exist, it registers the ID (`distance`) to the table. Then go back to the N_ASSIGN node.
  3. `turtle` calculates the second child. In this case its a number 100. Saves 100 to the variable table at the `distance` record.
  4. `turtle` goes back to N_PROGRAM then go to the next node N_FD. It has only one child. Goes down to the child N_MUL.

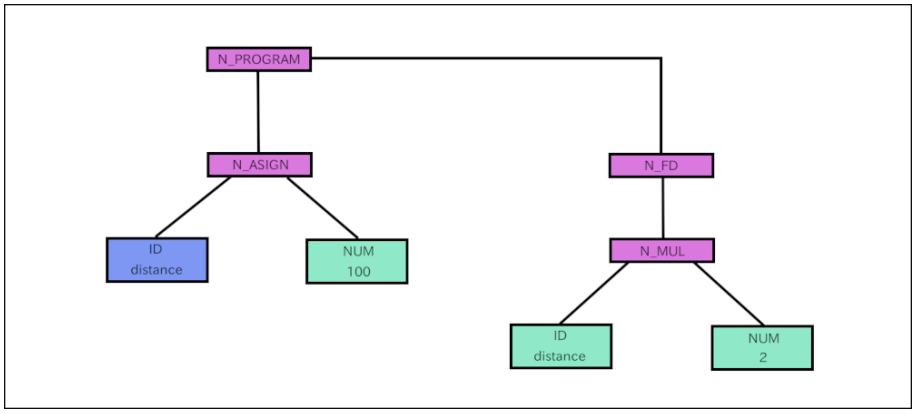

Figure 35: turtle parser tree

5. The first child is ID (distance). Searches the variable table for the variable `distance` and gets the value 100. The second child is a number 2. Multiplies 100 by 2 and gets 200. Then `turtle` goes back to N_FD.
6. Now `turtle` knows the distance is 200. It moves the cursor forward by 200 pixels. The segment is drawn on the surface (`surface`).
7. There are no node follows. Runtime routine returns to the function `run_cb`.

- `run_cb` calls `gtk_widget_queue_draw` and put the GtkDrawingArea widget to the queue.
- The system redraws the widget. At that time drawing function `draw_func` is called. The function copies the surface (`surface`) to the surface in the GtkDrawingArea.

Actual turtle program is more complicated than the example above. However, what turtle does is basically the same. Interpretation consists of three parts.

- Lexical analysis
- Syntax Parsing and tree generation
- Interpretation and execution of the tree.

25.4 Compilation flow

The source files are:

- flex source file => `turtle.lex`
- bison source file => `turtle.y`
- C header file => `turtle.h`, `turtle_lex.h`
- C source file => `turtleapplication.c`
- other files => `turtle.ui`, `turtle.gresources.xml` and `meson.build`

The compilation process is a bit complicated.

1. glib-compile-resources compiles `turtle.ui` to `resources.c` according to `turtle.gresource.xml`. It also generates `resources.h`.
2. bison compiles `turtle.y` to `turtle_parser.c` and generates `turtle_parser.h`
3. flex compiles `turtle.lex` to `turtle_lex.c`.
4. gcc compiles `application.c`, `resources.c`, `turtle_parser.c` and `turtle_lex.c` with `turtle.h`, `turtle_lex.h`, `resources.h` and `turtle_parser.h`. It generates an executable file `turtle`.

Meson controls the process and the instruction is described in `meson.build`.

```
1 project('turtle', 'c')
2
3 compiler = meson.get_compiler('c')
4 mathdep = compiler.find_library('m', required : true)
5
6 gtkdep = dependency('gtk4')
7
8 gnome=import('gnome')
9 resources = gnome.compile_resources('resources','turtle.gresource.xml')
```

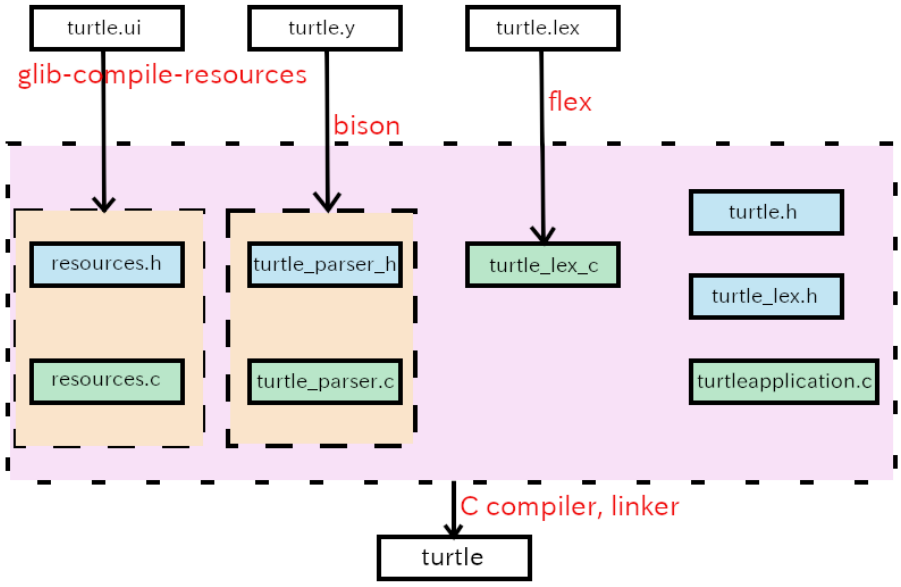

Figure 36: compile process

```
10
11 flex = find_program('flex')
12 bison = find_program('bison')
13 turtleparser = custom_target('turtleparser', input: 'turtle.y', output:
       ['turtle_parser.c', 'turtle_parser.h'], command: [bison, '-d', '-o',
       'turtle_parser.c', '@INPUT@'])
14 turtlelexer = custom_target('turtlelexer', input: 'turtle.lex', output:
       'turtle_lex.c', command: [flex, '-o', '@OUTPUT@', '@INPUT@'])
15
16 sourcefiles=files('turtleapplication.c', '../tfetextview/tfetextview.c')
17
18 executable('turtle', sourcefiles, resources, turtleparser, turtlelexer,
       turtleparser[1], dependencies: [mathdep, gtkdep], export_dynamic: true, install:
       true)
```

- 3: Gets C compiler. It is usually `gcc` in linux.
- 4: Gets math library. This program uses trigonometric functions. They are defined in the math library, but the library is optional. So, it is necessary to include it by `#include <math.h>` and also link the library with the linker.
- 6: Gets gtk4 library.
- 8: Gets gnome module. Module is a system provided by meson. See Meson build system website, GNUME module for further information.
- 9: Compiles ui file to C source file according to the XML file `turtle.gresource.xml`.
- 11: Gets flex.
- 12: Gets bison.
- 13: Compiles `turtle.y` to `turtle_parser.c` and `turtle_parser.h` by bison. The function `custom_target` creates a custom top level target. See Meson build system website, custom target for further information.
- 14: Compiles `turtle.lex` to `turtle_lex.c` by flex.
- 16: Specifies C source files.
- 18: Compiles C source files including generated files by glib-compile-resources, bison and flex. The argument `turtleparser[1]` refers to `tirtle_parser.h` which is the second output in the line 13.

25.5 Turtle.lex

25.5.1 What does flex do?

Flex creates lexical analyzer from flex source file. Flex source file is a text file. Its syntactic rule will be explained later. Generated lexical analyzer is a C source file. It is also called scanner. It reads a text file, which is a source file of a program language, and gets variable names, numbers and symbols. Suppose here is a turtle source file.

```
fc (1,0,0) # Foreground color is red, rgb = (1,0,0).
pd         # Pen down.
distance = 100
angle = 90
fd distance    # Go forward by distance (100) pixels.
tr angle     # Turn right by angle (90) degrees.
```

The content of the text file is separated into `fc`, `(`, `1` and so on. The words `fc`, `pd`, `distance`, `angle`, `tr`, `1`, `0`, `100` and `90` are called tokens. The characters '`(`' (left parenthesis), '`,`' (comma), '`)`' (right parenthesis) and '`=`' (equal sign) are called symbols. (Sometimes those symbols called tokens, too.)

Flex reads `turtle.lex` and generates the C source file of a scanner. The file `turtle.lex` specifies tokens, symbols and the behavior which corresponds to each token or symbol. Turtle.lex isn't a big program.

```
1  %top{
2  #include <string.h>
3  #include <stdlib.h>
4  #include "turtle.h"
5
6    static int nline = 1;
7    static int ncolumn = 1;
8    static void get_location (char *text);
9
10   /* Dinamically allocated memories are added to the single list. They will be freed
         in the finalize function. */
11   extern GSList *list;
12 }
13
14 %option noyywrap
15
16 REAL_NUMBER (0|[1-9][0-9]*)(\.[0-9]+)?
17 IDENTIFIER [a-zA-Z][a-zA-Z0-9]*
18 %%
19   /* rules */
20 #.*               ; /* comment. Be careful. Dot symbol (.) matches any character but
       new line. */
21 [ ]               ncolumn++;
22 \t                ncolumn += 8; /* assume that tab is 8 spaces. */
23 \n                nline++; ncolumn = 1;
24   /* reserved keywords */
25 pu                get_location (yytext); return PU; /* pen up */
26 pd                get_location (yytext); return PD; /* pen down */
27 pw                get_location (yytext); return PW; /* pen width = line width */
28 fd                get_location (yytext); return FD; /* forward */
29 tr                get_location (yytext); return TR; /* turn right */
30 bc                get_location (yytext); return BC; /* background color */
31 fc                get_location (yytext); return FC; /* foreground color */
32 dp                get_location (yytext); return DP; /* define procedure */
33 if                get_location (yytext); return IF; /* if statement */
34 rt                get_location (yytext); return RT; /* return statement */
35 rs                get_location (yytext); return RS; /* reset the status */
36   /* constant */
37 {REAL_NUMBER}     get_location (yytext); yylval.NUM = atof (yytext); return NUM;
38   /* identifier */
39 {IDENTIFIER}      { get_location (yytext); yylval.ID = g_strdup(yytext);
40                     list = g_slist_prepend (list, yylval.ID);
41                     return ID;
```

```
42                      }
43  "="                 get_location (yytext); return '=';
44  ">"                 get_location (yytext); return '>';
45  "<"                 get_location (yytext); return '<';
46  "+"                 get_location (yytext); return '+';
47  "-"                 get_location (yytext); return '-';
48  "*"                 get_location (yytext); return '*';
49  "/"                 get_location (yytext); return '/';
50  "("                 get_location (yytext); return '(';
51  ")"                 get_location (yytext); return ')';
52  "{"                 get_location (yytext); return '{';
53  "}"                 get_location (yytext); return '}';
54  ","                 get_location (yytext); return ',';
55  .                   ncolumn++;             return YYUNDEF;
56  %%
57
58  static void
59  get_location (char *text) {
60    yylloc.first_line = yylloc.last_line = nline;
61    yylloc.first_column = ncolumn;
62    yylloc.last_column = (ncolumn += strlen(text)) - 1;
63  }
64
65  static YY_BUFFER_STATE state;
66
67  void
68  init_flex (const char *text) {
69    state = yy_scan_string (text);
70  }
71
72  void
73  finalize_flex (void) {
74    yy_delete_buffer (state);
75  }
```

The file consists of three sections which are separated by "%%" (line 18 and 56). They are definitions, rules and user code sections.

25.5.2 Definitions section

- 1-12: Lines between "%top{" and "}" are C source codes. They will be copied to the top of the generated C source file.
- 2-3: The function `strlen`, in line 62, is defined in `string.h` The function `atof`, in line 37, is defined in `stdlib.h`.
- 6-8: The current input position is pointed by `nline` and `ncolumn`. The function `get_location` (line 58-63) sets `yyllocto` point the start and end point of `yytext` in the buffer. This function is declared here so that it can be called before the function is defined.
- 11: GSlist is used to keep allocated memories.
- 14: This option (`%option noyywrap`) must be specified when you have only single source file to the scanner. Refer to "9 The Generated Scanner" in the flex documentation in your distribution for further information. (The documentation is not on the internet.)
- 16-17: `REAL_NUMBER` and `IDENTIFIER` are names. A name begins with a letter or an underscore followed by zero or more letters, digits, underscores (_) or dashes (-). They are followed by regular expressions which are their definition. They will be used in rules section and will expand to the definition. You can leave out such definitions here and use regular expressions in rules section directly.

25.5.3 Rules section

This section is the most important part. Rules consist of patterns and actions. The patterns are regular expressions or names surrounded by braces. The names must be defined in the definitions section. The definition of the regular expression is written in the flex documentation.

For example, line 37 is a rule.

- `{REAL_NUMBER}` is a pattern
- `get_location (yytext); yylval.NUM = atof (yytext); return NUM;` is an action.

`{REAL_NUMBER}` is defined in the 16th line, so it expands to `(0|[1-9][0-9]*)(\.[0-9]+)?`. This regular expression matches numbers like `0`, `12` and `1.5`. If the input is a number, it matches the pattern in line 37. Then the matched text is assigned to `yytext` and corresponding action is executed. A function `get_location` changes the location variables. It assigns `atof (yytext)`, which is double sized number converted from `yytext`, to `yylval.NUM` and return `NUM`. `NUM` is an integer defined by `turtle.y`.

The scanner generated by flex and C compiler has `yylex` function. If `yylex` is called and the input is "123.4", then it works as follows.

1. A string "123.4" matches `{REAL_NUMBER}`.
2. Update the location variable `ncolumn` and `yyllocwith` `get_location`.
3. `atof` converts the string "123.4" to double type number 123.4.
4. It is assigned to `yylval.NUM`.
5. `yylex` returns `NUM` to the caller.

Then the caller knows the input is `NUM` (number), and its value is 123.4.

- 19-55: Rules section.
- 20: The symbol . (dot) matches any character except newline. Therefore, a comment begins `#` followed by any characters except newline. No action happens.
- 21: White space just increases a variable `ncolumn` by one.
- 22: Tab is assumed to be equal to eight spaces.
- 23: New line increases a variable `nline` by one and resets `ncolumn`.
- 25-35: Keywords just updates the location variables `ncolumn` and `yylloc`, and return the codes of the keywords.
- 37: Real number constant.
- 38: `IDENTIFIER` is defined in line 17. The location variables are updated and the name of the identifier is assigned to `yylval.ID`. The memory of the name is allocated by the function `g_strdup`. The memory is registered to the list (GSlist type list). The memory will be freed after the runtime routine finishes. Returns `ID`.
- 43-54: Symbols just update the location variable and return the codes. The code is the same as the symbol itself.
- 55: If the input doesn't match above patterns, then it is error. Returns `YYUNDEF`.

25.5.4 User code section

This section is just copied to C source file.

- 58-63: A function `get_location`. The location of the input is recorded to `nline` and `ncolumn`. A variable `yylloc` is referred by the parser. It is a C structure and has four members, `first_line`, `first_column`, `last_line` and `last_column`. They point the start and end of the current input text.
- 65: `YY_BUFFER_STATE` is a pointer points the input buffer.
- 67-70: `init_flex` is called by `run_cb` signal handler, which is called when `Run` button is clicked on. `run_cb` calls `init_flex` with one argument which is the copy of the content of GtkTextBuffer. `yy_scan_string` sets the input buffer to read from the text.
- 72-75: `finalize_flex` is called after runtime routine finishes. It deletes the input buffer.

25.6 Turtle.y

Turtle.y has more than 800 lines so it is difficult to explain all the source code. So I will explain the key points and leave out other less important parts.

25.6.1 What does bison do?

Bison creates C source file from bison source file. Bison source file is a text file. A parser analyzes a program source code according to its grammar. Suppose here is a turtle source file.

```
fc (1,0,0) # Foreground color is red, rgb = (1,0,0).
pd         # Pen down.
distance = 100
```

```
angle = 90
fd distance    # Go forward by distance (100) pixels.
tr angle     # Turn right by angle (90) degrees.
```

The parser calls `yylex` to get a token. The token consists of its type (token kind) and value (semantic value). So, the parser gets items in the following table whenever it calls `yylex`.

	token kind	yylval.ID	yylval.NUM
1	FC		
2	(		
3	NUM		1.0
4	,		
5	NUM		0.0
6	,		
7	NUM		0.0
8	)		
9	PD		
10	ID	distance	
11	=		
12	NUM		100.0
13	ID	angle	
14	=		
15	NUM		90.0
16	FD		
17	ID	distance	
18	TR		
19	ID	angle	

Bison source code specifies the grammar rules of turtle language. For example, `fc (1,0,0)` is called primary procedure. A procedure is like a void type function in C source code. It doesn't return any values. Programmers can define their own procedures. On the other hand, `fc` is a built-in procedure. Such procedures are called primary procedures. It is described in bison source code like:

```
primary_procedure: FC '(' expression ',' expression ',' expression ')';
expression: ID | NUM;
```

This means:

- Primary procedure is FC followed by '(', expression, ',', expression, ',', expression and ')'.
- expression is ID or NUM.

The description above is called BNF (Backus-Naur form). More precisely, it is similar to BNF.

The first line is:

```
FC '(' NUM ',' NUM ',' NUM ')';
```

The parser analyzes the turtle source code and if the input matches the definition above, the parser recognizes it as a primary procedure.

The grammar of turtle is described in the document. The following is an extract from the document.

```
program:
  statement
| program statement
;

statement:
  primary_procedure
| procedure_definition
;
```

```
primary_procedure:
  PU
| PD
| PW expression
| FD expression
| TR expression
| BC '(' expression ',' expression ',' expression ')'
| FC '(' expression ',' expression ',' expression ')'
| ID '=' expression
| IF '(' expression ')' '{' primary_procedure_list '}'
| RT
| RS
| ID '(' ')'
| ID '(' argument_list ')'
;

procedure_definition:
  DP ID '('  ')' '{' primary_procedure_list '}'
| DP ID '(' parameter_list ')' '{' primary_procedure_list '}'
;

parameter_list:
  ID
| parameter_list ',' ID
;

argument_list:
  expression
| argument_list ',' expression
;

primary_procedure_list:
  primary_procedure
| primary_procedure_list primary_procedure
;

expression:
  expression '=' expression
| expression '>' expression
| expression '<' expression
| expression '+' expression
| expression '-' expression
| expression '*' expression
| expression '/' expression
| '-' expression %prec UMINUS
| '(' expression ')'
| ID
| NUM
;
```

The grammar rule defines `program` first.

- program is a statement or a program followed by a statement.

The definition is recursive.

- `statement` is program.
- `statement statement` is `program statemet`. Therefore, it is program.
- `statement statement statement` is `program statemet`. Therefore, it is program.

You can find that a list of statements is program like this.

`program` and `statement` aren't tokens. They don't appear in the input. They are called non terminal symbols. On the other hand, tokens are called terminal symbols. The word "token" used here has wide meaning, it includes tokens and symbols which appear in the input. Non terminal symbols are often shortened to

nterm.

Let's analyze the program above as bison does.

	token kind	yylval.ID	yylval.NUM	parse	S/R
1	FC			FC	S
2	(			FC(	S
3	NUM		1.0	FC(NUM	S
				FC(expression	R
4	,			FC(expression,	S
5	NUM		0.0	FC(expression,NUM	S
				FC(expression,expression	R
6	,			FC(expression,expression,	S
7	NUM		0.0	FC(expression,expression,NUM	S
				FC(expression,expression,expression	R
8	)			FC(expression,expression,expression)	S
				primary_procedure	R
				statement	R
				program	R
9	PD			program PD	S
				program primary_procedure	R
				program statement	R
				program	R
10	ID	distance		program ID	S
11	=			program ID=	S
12	NUM		100.0	program ID=NUM	S
				program ID=expression	R
				program primary_procedure	R
				program statement	R
				program	R
13	ID	angle		program ID	S
14	=			program ID=	S
15	NUM		90.0	program ID=NUM	S
				program ID=expression	R
				program primary_procedure	R
				program statement	R
				program	R
16	FD			program FD	S
17	ID	distance		program FD ID	S
				program FD expression	R
				program primary_procedure	R
				program statement	R
				program	R
18	TR			program TR	S
19	ID	angle		program TR ID	S
				program TR expression	R
				program primary_procedure	R
				program statement	R
				program	R

The right most column shows shift/reduce. Shift is appending an input to the buffer. Reduce is substituting a higher nterm for the pattern in the buffer. For example, NUM is replaced by expression in the forth row. This substitution is "reduce".

Bison repeats shift and reduction until the end of the input. If the result is reduced to `program`, the input is syntactically valid. Bison executes an action whenever reduction occurs. Actions build a tree. The tree is analyzed and executed by runtime routine later.

Bison source files are called bison grammar files. A bison grammar file consists of four sections, prologue, declarations, rules and epilogue. The format is as follows.

```
%{
prologue
%}
declarations
%%
rules
%%
epilogue
```

25.6.2 Prologue

Prologue section consists of C codes and the codes are copied to the parser implementation file. You can use `%code` directives to qualify the prologue and identifies the purpose explicitly. The following is an extract from `turtle.y`.

```
%code top{
  #include <stdarg.h>
  #include <setjmp.h>
  #include <math.h>
  #include "turtle.h"

  /* error reporting */
  static void yyerror (char const *s) { /* for syntax error */
    g_print ("%s from line %d, column %d to line %d, column %d\n",s,
        yylloc.first_line, yylloc.first_column, yylloc.last_line,
        yylloc.last_column);
  }
  /* Node type */
  enum {
    N_PU,
    N_PD,
    N_PW,
 ... ... ...
  };
}
```

The directive `%code top` copies its contents to the top of the parser implementation file. It usually includes `#include` directives, declarations of functions and definitions of constants. A function `yyerror` reports a syntax error and is called by the parser. Node type identifies a node in the tree.

Another directive `%code requires` copies its contents to both the parser implementation file and header file. The header file is read by the scanner C source file and other files.

```
%code requires {
  int yylex (void);
  int yyparse (void);
  void run (void);

  /* semantic value type */
  typedef struct _node_t node_t;
  struct _node_t {
    int type;
    union {
      struct {
        node_t *child1, *child2, *child3;
      } child;
      char *name;
      double value;
    } content;
  };
}
```

- `yylex` is shared by parser implementation file and scanner file.
- `yyparse` and `run` is called by `run_cb` in `turtleapplication.c`.
- `node_t` is the type of the semantic value of nterms. The header file defines `YYSTYPE`, which is the semantic value type, with all the token and nterm value types. The following is extracted from the header file.

```
/* Value type.  */
#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
union YYSTYPE
{
  char * ID;                                /* ID  */
  double NUM;                               /* NUM  */
  node_t * program;                         /* program  */
  node_t * statement;                       /* statement  */
  node_t * primary_procedure;               /* primary_procedure  */
  node_t * primary_procedure_list;          /* primary_procedure_list  */
  node_t * procedure_definition;            /* procedure_definition  */
  node_t * parameter_list;                  /* parameter_list  */
  node_t * argument_list;                   /* argument_list  */
  node_t * expression;                      /* expression  */
};
```

Other useful macros and declarations are put into the `%code` directive.

```
%code {
/* The following macro is convenient to get the member of the node. */
  #define child1(n) (n)->content.child.child1
  #define child2(n) (n)->content.child.child2
  #define child3(n) (n)->content.child.child3
  #define name(n) (n)->content.name
  #define value(n) (n)->content.value

  /* start of nodes */
  static node_t *node_top = NULL;
  /* functions to generate trees */
  static node_t *tree1 (int type, node_t *child1, node_t *child2, node_t *child3);
  static node_t *tree2 (int type, double value);
  static node_t *tree3 (int type, char *name);
}
```

25.6.3 Bison declarations

Bison declarations defines terminal and non-terminal symbols. It also specifies some directives.

```
%locations
%define api.value.type union /* YYSTYPE, the type of semantic values, is union of
    following types */
 /* key words */
%token PU
%token PD
%token PW
%token FD
%token TR
%token BC
%token FC
%token DP
%token IF
%token RT
%token RS
 /* constant */
%token <double> NUM
 /* identirier */
%token <char *> ID
 /* non terminal symbol */
%nterm <node_t *> program
```

```
%nterm <node_t *> statement
%nterm <node_t *> primary_procedure
%nterm <node_t *> primary_procedure_list
%nterm <node_t *> procedure_definition
%nterm <node_t *> parameter_list
%nterm <node_t *> argument_list
%nterm <node_t *> expression
 /* logical relation symbol */
%left '=' '<' '>'
 /* arithmetic symbol */
%left '+' '-'
%left '*' '/'
%precedence UMINUS /* unary minus */
```

`%locations` directive inserts the location structure into the header file. It is like this.

```
typedef struct YYLTYPE YYLTYPE;
struct YYLTYPE
{
  int first_line;
  int first_column;
  int last_line;
  int last_column;
};
```

This type is shared by the scanner file and the parser implementation file. The error report function `yyerror` uses it so that it can inform the location that error occurs.

`%define api.value.type union` generates semantic value type with tokens and nterms and inserts it to the header file. The inserted part is shown in the previous subsection as the extracts that shows the value type (YYSTYPE).

`%token` and `%nterm` directives define tokens and non terminal symbols respectively.

```
%token PU
... ...
%token <double> NUM
```

These directives define a token `PU` and `NUM`. The values of token kinds `PU` and `NUM` are defined as an enumeration constant in the header file.

```
  enum yytokentype
  {
  ... ... ...
    PU = 258,                      /* PU  */
  ... ... ...
    NUM = 269,                     /* NUM  */
  ... ... ...
  };
  typedef enum yytokentype yytoken_kind_t;
```

In addition, the type of the semantic value of `NUM` is defined as double in the header file because of `<double>` tag.

```
union YYSTYPE
{
  char * ID;                                /* ID  */
  double NUM;                               /* NUM  */
  ... ...
}
```

All the nterm symbols have the same type `* node_t` of the semantic value.

`%left` and `%precedence` directives define the precedence of operation symbols.

```
 /* logical relation symbol */
%left '=' '<' '>'
 /* arithmetic symbol */
%left '+' '-'
%left '*' '/'
%precedence UMINUS /* unary minus */
```

`%left` directive defines the following symbols as left-associated operators. If an operator `+` is left-associated, then

```
A + B + C = (A + B) + C
```

That is, the calculation is carried out the left operator first, then the right operator. If an operator `*` is right-associated, then:

```
A * B * C = A * (B * C)
```

The definition above decides the behavior of the parser. Addition and multiplication hold associative law so the result of `(A+B)+C` and `A+(B+C)` are equal in terms of mathematics. However, the parser will be confused if left (or right) associativity is not specified.

`%left` and `%precedence` directives show the precedence of operators. Later declared operators have higher precedence than former declared ones. The declaration above says, for example,

```
v=w+z*5+7 is the same as v=((w+(z*5))+7)
```

Be careful. The operator `=` above is an assignment. Assignment is not expression in turtle language. It is primary_procedure. But if `=` appears in an expression, it is a logical operater, not an assignment. The logical equal '`=`' usually used in the conditional expression, for example, in `if` statement.

25.6.4 Grammar rules

Grammar rules section defines the syntactic grammar of the language. It is similar to BNF form.

```
result: components { action };
```

- result is a nterm.
- components are list of tokens or nterms.
- action is C codes. It is executed whenever the components are reduced to the result. Action can be left out.

The following is a part of the grammar rule in `turtle.y`.

```
program:
  statement { node_top = $$ = $1; }
;
statement:
  primary_procedure
;
primary_procedure:
  FD expression    { $$ = tree1 (N_FD, $2, NULL, NULL); }
;
expression:
  NUM   { $$ = tree2 (N_NUM, $1); }
;
```

- `program` is `statement`.
- Whenever `statement` is reduced to `program`, an action `node_top=$$=$1;` is executed.
- `node_top` is a static variable. It points the top node of the tree.
- `$$` is a semantic value of the result, which is `program` in the second line of the example above. The semantic value of `program` is a pointer to `node_t` type structure. It was defined in the declaration section.
- `$1` is a semantic value of the first component, which is `statement`. The semantic value of `statement` is also a pointer to `node_t`.

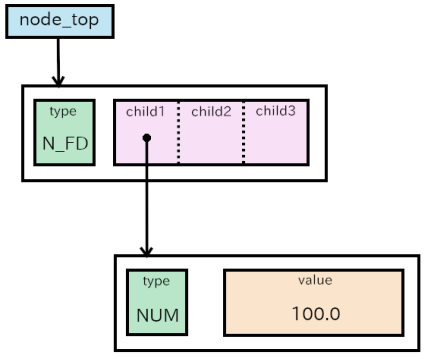

Figure 37: tree

- `statement` is `primary_procedure`. There's no action specified. Then, the default action is executed. It is `$$ = $1`.
- `primary_procedure` is `FD` followed by expression. The action calls `tree1` and assigns its return value to `$$`. `tree1` makes a tree node. The tree node has type and union of three pointers to children nodes, string or double.

```
node --+-- type
       +-- union contents
                    +---struct {node_t *child1, *child2, *child3;};
                    +---char *name
                    +---double value
```

- `tree1` assigns the four arguments to type, child1, child2 and child3 members.
- `expression` is `NUM`.
- `tree2` makes a tree node. The paremeters of `tree2` are a type and a semantic value.

Suppose the parser reads the following program.

```
fd 100
```

What does the parser do?

1. The parser recognizes the input is `FD`. Maybe it is the start of `primary_procedure`, but parser needs to read the next token.
2. `yylex` returns the token kind `NUM` and sets `yylval.NUM` to 100.0 (the type is double). The parser reduces `NUM` to `expression`. At the same time, it sets the semantic value of the `expression` to point a new node. The node has an type `N_NUM` and a semantic value 100.0.
3. After the reduction, the buffer has `FD` and `expression`. The parser reduces it to `primary_procedure`. And it sets the semantic value of the `primary_procedure` to point a new node. The node has an type `N_FD` and its member child1 points the node of `expression`, whose type is `N_NUM`.
4. The parser reduces `primary_procedure` to `statement`. The semantic value of `statement` is the same as the one of `primary_procedure`, which points to the node `N_FD`.
5. The parser reduces `statement` to `program`. The semantic value of `statement` is assigned to the one of `program` and the static variable `node_top`.
6. Finally `node_top` points the node `N_FD` and the node `N_FD` points the node `N_NUM`.

The following is the grammar rule extracted from `turtle.y`. The rules there are based on the same idea above. I don't want to explain the whole rules below. Please look into each line carefully so that you will understand all the rules and actions.

```
program:
  statement { node_top = $$ = $1; }
| program statement {
        node_top = $$ = tree1 (N_program, $1, $2, NULL);
        }
;
```

```
statement:
  primary_procedure
| procedure_definition
;

primary_procedure:
  PU    { $$ = tree1 (N_PU, NULL, NULL, NULL); }
| PD    { $$ = tree1 (N_PD, NULL, NULL, NULL); }
| PW expression    { $$ = tree1 (N_PW, $2, NULL, NULL); }
| FD expression    { $$ = tree1 (N_FD, $2, NULL, NULL); }
| TR expression    { $$ = tree1 (N_TR, $2, NULL, NULL); }
| BC '(' expression ',' expression ',' expression ')' { $$ = tree1 (N_BC, $3, $5,
    $7); }
| FC '(' expression ',' expression ',' expression ')' { $$ = tree1 (N_FC, $3, $5,
    $7); }
 /* assignment */
| ID '=' expression   { $$ = tree1 (N_ASSIGN, tree3 (N_ID, $1), $3, NULL); }
 /* control flow */
| IF '(' expression ')' '{' primary_procedure_list '}' { $$ = tree1 (N_IF, $3, $6,
    NULL); }
| RT    { $$ = tree1 (N_RT, NULL, NULL, NULL); }
| RS    { $$ = tree1 (N_RS, NULL, NULL, NULL); }
 /* user defined procedure call */
| ID '(' ')'  { $$ = tree1 (N_procedure_call, tree3 (N_ID, $1), NULL, NULL); }
| ID '(' argument_list ')'  { $$ = tree1 (N_procedure_call, tree3 (N_ID, $1), $3,
    NULL); }
;

procedure_definition:
  DP ID '('  ')' '{' primary_procedure_list '}'  {
         $$ = tree1 (N_procedure_definition, tree3 (N_ID, $2), NULL, $6);
        }
| DP ID '(' parameter_list ')' '{' primary_procedure_list '}'  {
         $$ = tree1 (N_procedure_definition, tree3 (N_ID, $2), $4, $7);
        }
;

parameter_list:
  ID { $$ = tree3 (N_ID, $1); }
| parameter_list ',' ID  { $$ = tree1 (N_parameter_list, $1, tree3 (N_ID, $3),
    NULL); }
;

argument_list:
  expression
| argument_list ',' expression { $$ = tree1 (N_argument_list, $1, $3, NULL); }
;

primary_procedure_list:
  primary_procedure
| primary_procedure_list primary_procedure {
         $$ = tree1 (N_primary_procedure_list, $1, $2, NULL);
        }
;

expression:
  expression '=' expression { $$ = tree1 (N_EQ, $1, $3, NULL); }
| expression '>' expression { $$ = tree1 (N_GT, $1, $3, NULL); }
| expression '<' expression { $$ = tree1 (N_LT, $1, $3, NULL); }
| expression '+' expression { $$ = tree1 (N_ADD, $1, $3, NULL); }
| expression '-' expression { $$ = tree1 (N_SUB, $1, $3, NULL); }
| expression '*' expression { $$ = tree1 (N_MUL, $1, $3, NULL); }
| expression '/' expression { $$ = tree1 (N_DIV, $1, $3, NULL); }
```

```
| '-' expression %prec UMINUS { $$ = tree1 (N_UMINUS, $2, NULL, NULL); }
| '(' expression ')' { $$ = $2; }
| ID    { $$ = tree3 (N_ID, $1); }
| NUM   { $$ = tree2 (N_NUM, $1); }
;
```

25.6.5 Epilogue

The epilogue is written in C language and copied to the parser implementation file. Generally, you can put anything into the epilogue. In the case of turtle interpreter, the runtime routine and some other functions are in the epilogue.

Functions to create tree nodes There are three functions, `tree1`, `tree2` and `tree3`.

- `tree1` creates a node and sets the node type and pointers to its three children (NULL is possible).
- `tree2` creates a node and sets the node type and a value (double).
- `tree3` creates a node and sets the node type and a pointer to a string.

Each function gets memories first and build a node on them. The memories are inserted to the list. They will be freed when runtime routine finishes.

The three functions are called in the actions in the rules section.

```
/* Dynamically allocated memories are added to the single list. They will be freed
    in the finalize function. */
GSList *list = NULL;

node_t *
tree1 (int type, node_t *child1, node_t *child2, node_t *child3) {
  node_t *new_node;

  list = g_slist_prepend (list, g_malloc (sizeof (node_t)));
  new_node = (node_t *) list->data;
  new_node->type = type;
  child1(new_node) = child1;
  child2(new_node) = child2;
  child3(new_node) = child3;
  return new_node;
}

node_t *
tree2 (int type, double value) {
  node_t *new_node;

  list = g_slist_prepend (list, g_malloc (sizeof (node_t)));
  new_node = (node_t *) list->data;
  new_node->type = type;
  value(new_node) = value;
  return new_node;
}

node_t *
tree3 (int type, char *name) {
  node_t *new_node;

  list = g_slist_prepend (list, g_malloc (sizeof (node_t)));
  new_node = (node_t *) list->data;
  new_node->type = type;
  name(new_node) = name;
  return new_node;
}
```

Symbol table Variables and user defined procedures are registered in a symbol table. This table is a C array. It should be replaced by more appropriate data structure with memory allocation in the future version

- Variables are registered with its name and value.
- Procedures are registered with its name and a pointer to the node of the procedure.

Therefore the table has the following fields.

- type to identify variable or procedure
- name
- value or pointer to a node

```
#define MAX_TABLE_SIZE 100
enum {
  PROC,
  VAR
};

typedef union _object_t object_t;
union _object_t {
  node_t *node;
  double value;
};

struct {
  int type;
  char *name;
  object_t object;
} table[MAX_TABLE_SIZE];
int tp;

void
init_table (void) {
  tp = 0;
}
```

`init_table` initializes the table. This must be called before any registrations.

There are five functions to access the table,

- `proc_install` installs a procedure.
- `var_install` installs a variable.
- `proc_lookup` looks up a procedure. If the procedure is found, it returns a pointer to the node. Otherwise it returns NULL.
- `var_lookup` looks up a variable. If the variable is found, it returns TRUE and sets the pointer (argument) to point the value. Otherwise it returns FALSE.
- `var_replace` replaces the value of a variable. If the variable hasn't registered yet, it installs the variable.

```
int
tbl_lookup (int type, char *name) {
  int i;

  if (tp == 0)
    return -1;
  for (i=0; i<tp; ++i)
    if (type == table[i].type && strcmp(name, table[i].name) == 0)
      return i;
  return -1;
}

void
tbl_install (int type, char *name, object_t object) {
  if (tp >= MAX_TABLE_SIZE)
    runtime_error ("Symbol table overflow.\n");
```

```
  else if (tbl_lookup (type, name) >= 0)
    runtime_error ("Name %s is already registered.\n", name);
  else {
    table[tp].type = type;
    table[tp].name = name;
    if (type == PROC)
      table[tp++].object.node = object.node;
    else
      table[tp++].object.value = object.value;
  }
}

void
proc_install (char *name, node_t *node) {
  object_t object;
  object.node = node;
  tbl_install (PROC, name, object);
}

void
var_install (char *name, double value) {
  object_t object;
  object.value = value;
  tbl_install (VAR, name, object);
}

void
var_replace (char *name, double value) {
  int i;
  if ((i = tbl_lookup (VAR, name)) >= 0)
    table[i].object.value = value;
  else
    var_install (name, value);
}

node_t *
proc_lookup (char *name) {
  int i;
  if ((i = tbl_lookup (PROC, name)) < 0)
    return NULL;
  else
    return table[i].object.node;
}

gboolean
var_lookup (char *name, double *value) {
  int i;
  if ((i = tbl_lookup (VAR, name)) < 0)
    return FALSE;
  else {
    *value = table[i].object.value;
    return TRUE;
  }
}
```

Stack for parameters and arguments Stack is a last-in first-out data structure. It is shortened to LIFO. Turtle uses a stack to keep parameters and arguments. They are like `auto` class variables in C language. They are pushed to the stack whenever the procedure is called. LIFO structure is useful for recursive calls.

Each element of the stack has name and value.

```
#define MAX_STACK_SIZE 500
struct {
```

```
  char *name;
  double value;
} stack[MAX_STACK_SIZE];
int sp, sp_biggest;

void
init_stack (void) {
  sp = sp_biggest = 0;
}
```

`sp` is a stack pointer. It is an index of the array `stack` and it always points an element of the array to store the next data. `sp_biggest` is the biggest number assigned to `sp`. We can know the amount of elements used in the array during the runtime. The purpose of the variable is to find appropriate `MAX_STACK_SIZE`. It will be unnecessary in the future version if the stack is implemented with better data structure and memory allocation.

The runtime routine push data to the stack when it executes a node of a procedure call. (The type of the node is `N_procedure_call`.)

```
dp drawline (angle, distance) { ... ... ... }
drawline (90, 100)
```

- The first line defines a procedure `drawline`. The runtime routine stores the name `drawline` and the node of the procedure to the symbol table.
- The second line calls the procedure. First, it looks for the procedure in the symbol table and gets its node. Then it searches the node for the parameters and gets `angle` and `distance`.
- It pushes ("distance", 100.0) to the stack.
- It pushes ("angle", 90.0) to the stack.
- It pushes (NULL, 2.0) to the stack. The number 2.0 is the number of parameters (or arguments). It is used when the procedure returns.

The following diagram shows the structure of the stack. First, `procedure 1` is called. The procedure has two parameters. In the `procedure 1`, another procedure `procedure 2`, which has one parameter, is called. And in the `procedure 2`, `procedure 3`, which has three parameters, is called.

Programs push data to a stack from a low address memory to a high address memory. In the following diagram, the lowest address is at the top and the highest address is at the bottom. That is the order of the address. However, "the top of the stack" is the last pushed data and "the bottom of the stack" is the first pushed data. Therefore, "the top of the stack" is the bottom of the rectangle in the diagram and "the bottom of the stack" is the top of the rectangle.

There are four functions to access the stack.

- `stack_push` pushes data to the stack.
- `stack_lookup` searches the stack for the variable given its name as an argument. It searches only the parameters of the latest procedure. It returns TRUE and sets the argument `value` to point the value, if the variable has been found. Otherwise it returns FALSE.
- `stack_replace` replaces the value of the variable in the stack. If it succeeds, it returns TRUE. Otherwise returns FALSE.
- `stack_return` throws away the latest parameters. The stack pointer goes back to the point before the latest procedure call so that it points to parameters of the previous called procedure.

```
void
stack_push (char *name, double value) {
  if (sp >= MAX_STACK_SIZE)
    runtime_error ("Stack overflow.\n");
  else {
    stack[sp].name = name;
    stack[sp++].value = value;
    sp_biggest = sp > sp_biggest ? sp : sp_biggest;
  }
}

int
```

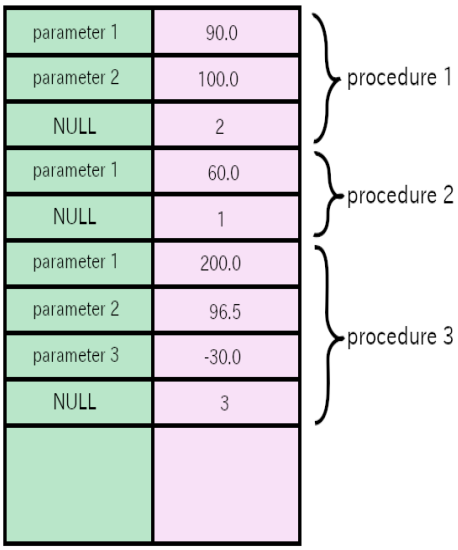

Figure 38: Stack

```
stack_search (char *name) {
  int depth, i;

  if (sp == 0)
    return -1;
  depth = (int) stack[sp-1].value;
  if (depth + 1 > sp) /* something strange */
    runtime_error ("Stack error.\n");
  for (i=0; i<depth; ++i)
    if (strcmp(name, stack[sp-(i+2)].name) == 0) {
      return sp-(i+2);
    }
  return -1;
}

gboolean
stack_lookup (char *name, double *value) {
  int i;

  if ((i = stack_search (name)) < 0)
    return FALSE;
  else {
    *value = stack[i].value;
    return TRUE;
  }
}

gboolean
stack_replace (char *name, double value) {
  int i;

  if ((i = stack_search (name)) < 0)
    return FALSE;
  else {
    stack[i].value = value;
    return TRUE;
  }
}
```

$$\begin{bmatrix} z \\ w \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} p \\ q \end{bmatrix}$$

Figure 39: transformation

```
void
stack_return(void) {
  int depth;

  if (sp <= 0)
    return;
  depth = (int) stack[sp-1].value;
  if (depth + 1 > sp) /* something strange */
    runtime_error ("Stack error.\n");
  sp -= depth + 1;
}
```

Surface and cairo A global variable `surface` is shared by `turtleapplication.c` and `turtle.y`. It is initialized in `turtleapplication.c`.

The runtime routine has its own cairo context. This is different from the cairo of GtkDrawingArea. Runtime routine draws a shape on the `surface` with the cairo context. After runtime routine returns to `run_cb`, `run_cb` adds the GtkDrawingArea widget to the queue to redraw. When the widget is redraw,the drawing function `draw_func` is called. It copies the `surface` to the surface in the GtkDrawingArea object.

`turtle.y` has two functions `init_cairo` and `destroy_cairo`.

- `init_cairo` initializes static variables and cairo context. The variables keep pen status (up or down), direction, initial location, line width and color. The size of the `surface` changes according to the size of the window. Whenever a user drags and resizes the window, the `surface` is also resized. `init_cairo` gets the size first and sets the initial location of the turtle (center of the surface) and the transformation matrix.
- `destroy_cairo` just destroys the cairo context.

Turtle has its own coordinate. The origin is at the center of the surface, and positive direction of x and y axes are right and up respectively. But surfaces have its own coordinate. Its origin is at the top-left corner of the surface and positive direction of x and y are right and down respectively. A plane with the turtle's coordinate is called user space, which is the same as cairo's user space. A plane with the surface's coordinate is called device space.

Cairo provides a transformation which is an affine transformation. It transforms a user-space coordinate (x, y) into a device-space coordinate (z, w).

`init_cairo` gets the width and height of the `surface` (See the program below).

- The center of the surface is (0,0) with regard to the user-space coordinate and (width/2, height/2) with regard to the device-space coordinate.
- The positive direction of x axis in the two spaces are the same. So, (1,0) is transformed into (1+width/2,height/2).
- The positive direction of y axis in the two spaces are opposite. So, (0,1) is transformed into (width/2,-1+height/2).

You can determine a, b, c, d, p and q by substituting the numbers above for x, y, z and w in the equation above. The solution of the simultaneous equations is:

```
a = 1, b = 0, c = 0, d = -1, p = width/2, q = height/2
```

Cairo provides a structure `cairo_matrix_t`. `init_cairo` uses it and sets the cairo transformation (See the program below). Once the matrix is set, the transformation always performs whenever `cairo_stroke` function is invoked.

```
/* status of the surface */
static gboolean pen = TRUE;
static double angle = 90.0; /* angle starts from x axis and measured
    counterclockwise */
                  /* Initially facing to the north */
static double cur_x = 0.0;
static double cur_y = 0.0;
static double line_width = 2.0;

struct color {
  double red;
  double green;
  double blue;
};
static struct color bc = {0.95, 0.95, 0.95}; /* white */
static struct color fc = {0.0, 0.0, 0.0}; /* black */

/* cairo */
static cairo_t *cr;
gboolean
init_cairo (void) {
  int width, height;
  cairo_matrix_t matrix;

  pen = TRUE;
  angle = 90.0;
  cur_x = 0.0;
  cur_y = 0.0;
  line_width = 2.0;
  bc.red = 0.95; bc.green = 0.95; bc.blue = 0.95;
  fc.red = 0.0; fc.green = 0.0; fc.blue = 0.0;

  if (surface) {
    width = cairo_image_surface_get_width (surface);
    height = cairo_image_surface_get_height (surface);
    matrix.xx = 1.0; matrix.xy = 0.0; matrix.x0 = (double) width / 2.0;
    matrix.yx = 0.0; matrix.yy = -1.0; matrix.y0 = (double) height / 2.0;

    cr = cairo_create (surface);
    cairo_transform (cr, &matrix);
    cairo_set_source_rgb (cr, bc.red, bc.green, bc.blue);
    cairo_paint (cr);
    cairo_set_source_rgb (cr, fc.red, fc.green, fc.blue);
    cairo_move_to (cr, cur_x, cur_y);
    return TRUE;
  } else
    return FALSE;
}

void
destroy_cairo () {
  cairo_destroy (cr);
}
```

Eval function A function `eval` evaluates an expression and returns the value of the expression. It calls itself recursively. For example, if the node is `N_ADD`, then:

1. Calls eval(child1(node)) and gets the value1.
2. Calls eval(child2(node)) and gets the value2.
3. Returns value1+value2.

This is performed by a macro `calc` defined in the sixth line in the following program.

```
double
```

```
eval (node_t *node) {
double value = 0.0;
  if (node == NULL)
    runtime_error ("No expression to evaluate.\n");
#define calc(op) eval (child1(node)) op eval (child2(node))
  switch (node->type) {
    case N_EQ:
      value = (double) calc(==);
      break;
    case N_GT:
      value = (double) calc(>);
      break;
    case N_LT:
      value = (double) calc(<);
      break;
    case N_ADD:
      value = calc(+);
      break;
    case N_SUB:
      value = calc(-);
      break;
    case N_MUL:
      value = calc(*);
      break;
    case N_DIV:
      if (eval (child2(node)) == 0.0)
        runtime_error ("Division by zerp.\n");
      else
        value = calc(/);
      break;
    case N_UMINUS:
      value = -(eval (child1(node)));
      break;
    case N_ID:
      if (! (stack_lookup (name(node), &value)) && ! var_lookup (name(node), &value)
          )
        runtime_error ("Variable %s not defined.\n", name(node));
      break;
    case N_NUM:
      value = value(node);
      break;
    default:
      runtime_error ("Illegal expression.\n");
  }
  return value;
}
```

Execute function Primary procedures and procedure definitions are analyzed and executed by the function `execute`. It doesn't return any values. It calls itself recursively. The process of `N_RT` and `N_procedure_call` is complicated. It will explained after the following program. Other parts are not so difficult. Read the program below carefully so that you will understand the process.

```
/* procedure - return status */
static int proc_level = 0;
static int ret_level = 0;

void
execute (node_t *node) {
  double d, x, y;
  char *name;
  int n, i;

  if (node == NULL)
    runtime_error ("Node is NULL.\n");
```

```
  if (proc_level > ret_level)
    return;
  switch (node->type) {
    case N_program:
      execute (child1(node));
      execute (child2(node));
      break;
    case N_PU:
      pen = FALSE;
      break;
    case N_PD:
      pen = TRUE;
      break;
    case N_PW:
      line_width = eval (child1(node)); /* line width */
      break;
    case N_FD:
      d = eval (child1(node)); /* distance */
      x = d * cos (angle*M_PI/180);
      y = d * sin (angle*M_PI/180);
      /* initialize the current point = start point of the line */
      cairo_move_to (cr, cur_x, cur_y);
      cur_x += x;
      cur_y += y;
      cairo_set_line_width (cr, line_width);
      cairo_set_source_rgb (cr, fc.red, fc.green, fc.blue);
      if (pen)
        cairo_line_to (cr, cur_x, cur_y);
      else
        cairo_move_to (cr, cur_x, cur_y);
      cairo_stroke (cr);
      break;
    case N_TR:
      angle -= eval (child1(node));
      for (; angle < 0; angle += 360.0);
      for (; angle>360; angle -= 360.0);
      break;
    case N_BC:
      bc.red = eval (child1(node));
      bc.green = eval (child2(node));
      bc.blue = eval (child3(node));
#define fixcolor(c)  c = c < 0 ? 0 : (c > 1 ? 1 : c)
      fixcolor (bc.red);
      fixcolor (bc.green);
      fixcolor (bc.blue);
      /* clear the shapes and set the background color */
      cairo_set_source_rgb (cr, bc.red, bc.green, bc.blue);
      cairo_paint (cr);
      break;
    case N_FC:
      fc.red = eval (child1(node));
      fc.green = eval (child2(node));
      fc.blue = eval (child3(node));
      fixcolor (fc.red);
      fixcolor (fc.green);
      fixcolor (fc.blue);
      break;
    case N_ASSIGN:
      name = name(child1(node));
      d = eval (child2(node));
      if (! stack_replace (name, d)) /* First, tries to replace the value in the
          stack (parameter).*/
        var_replace (name, d); /* If the above fails, tries to replace the value in
            the table. If the variable isn't in the table, installs it, */
```

```
      break;
    case N_IF:
      if (eval (child1(node)))
        execute (child2(node));
      break;
    case N_RT:
      ret_level--;
      break;
    case N_RS:
      pen = TRUE;
      angle = 90.0;
      cur_x = 0.0;
      cur_y = 0.0;
      line_width = 2.0;
      fc.red = 0.0; fc.green = 0.0; fc.blue = 0.0;
      /* To change background color, use bc. */
      break;
    case N_procedure_call:
      name = name(child1(node));
node_t *proc = proc_lookup (name);
      if (! proc)
        runtime_error ("Procedure %s not defined.\n", name);
      if (strcmp (name, name(child1(proc))) != 0)
        runtime_error ("Unexpected error. Procedure %s is called, but invoked
            procedure is %s.\n", name, name(child1(proc)));
/* make tuples (parameter (name), argument (value)) and push them to the stack */
node_t *param_list;
node_t *arg_list;
      param_list = child2(proc);
      arg_list = child2(node);
      if (param_list == NULL) {
        if (arg_list == NULL) {
          stack_push (NULL, 0.0); /* number of argument == 0 */
        } else
          runtime_error ("Procedure %s has different number of argument and
              parameter.\n", name);
      }else {
/* Don't change the stack until finish evaluating the arguments. */
#define TEMP_STACK_SIZE 20
        char *temp_param[TEMP_STACK_SIZE];
        double temp_arg[TEMP_STACK_SIZE];
        n = 0;
        for (; param_list->type == N_parameter_list; param_list =
            child1(param_list)) {
          if (arg_list->type != N_argument_list)
            runtime_error ("Procedure %s has different number of argument and
                parameter.\n", name);
          if (n >= TEMP_STACK_SIZE)
            runtime_error ("Too many parameters. the number must be %d or less.\n",
                TEMP_STACK_SIZE);
          temp_param[n] = name(child2(param_list));
          temp_arg[n] = eval (child2(arg_list));
          arg_list = child1(arg_list);
          ++n;
        }
        if (param_list->type == N_ID && arg_list -> type != N_argument_list) {
          temp_param[n] = name(param_list);
          temp_arg[n] = eval (arg_list);
          if (++n >= TEMP_STACK_SIZE)
            runtime_error ("Too many parameters. the number must be %d or less.\n",
                TEMP_STACK_SIZE);
          temp_param[n] = NULL;
          temp_arg[n] = (double) n;
          ++n;
```

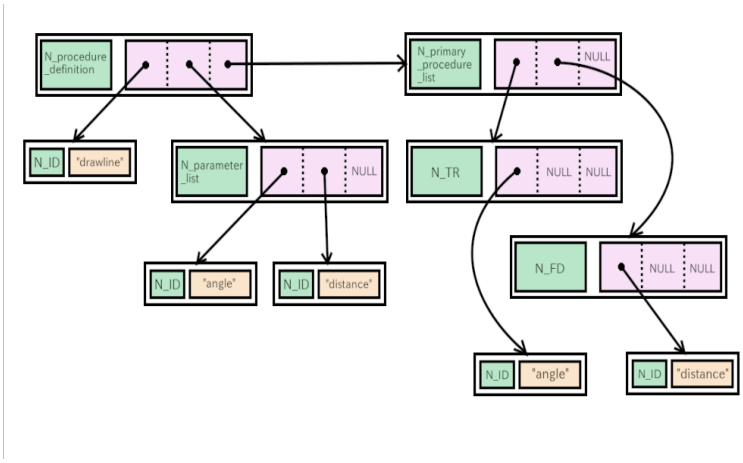

Figure 40: Nodes of drawline

```
        } else
          runtime_error ("Unexpected error.\n");
        for (i = 0; i < n; ++i)
          stack_push (temp_param[i], temp_arg[i]);
      }
      ret_level = ++proc_level;
      execute (child3(proc));
      ret_level = --proc_level;
      stack_return ();
      break;
    case N_procedure_definition:
      name = name(child1(node));
      proc_install (name, node);
      break;
    case N_primary_procedure_list:
      execute (child1(node));
      execute (child2(node));
      break;
    default:
      runtime_error ("Unknown statement.\n");
  }
}
```

A node `N_procedure_call` is created by the parser when it has found a user defined procedure call. The procedure has been defined in the prior statement. Suppose the parser reads the following example code.

```
dp drawline (angle, distance) {
  tr angle
  fd distance
}
drawline (90, 100)
drawline (90, 100)
drawline (90, 100)
drawline (90, 100)
```

This example draws a square.

When The parser reads the lines from one to four, it creates nodes like this:

Runtime routine just stores the procedure to the symbol table with its name and node.

When the parser reads the fifth line in the example, it creates nodes like this:

When the runtime routine meets `N_procedure_call` node, it behaves like this:

1. Searches the symbol table for the procedure with the name.

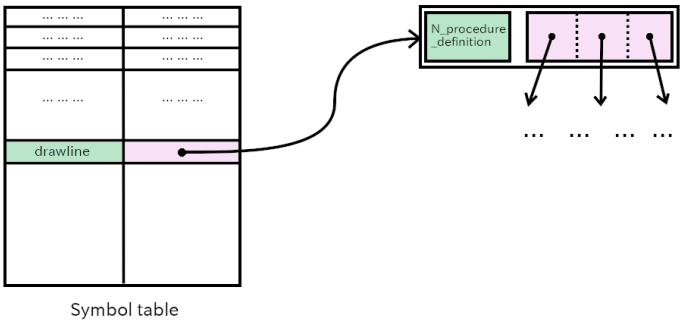

Figure 41: Symbol table

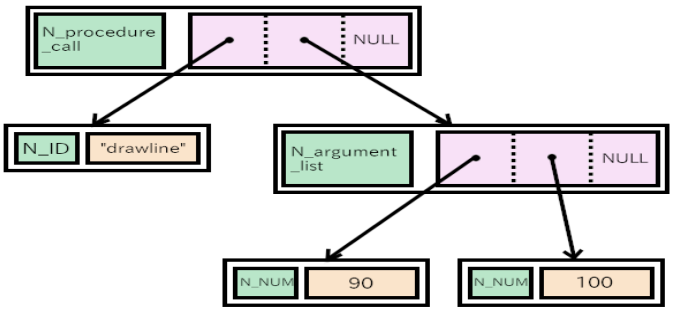

Figure 42: Nodes of procedure call

2. Gets pointers to the node to parameters and the node to the body.
3. Creates a temporary stack. Makes a tuple of each parameter name and argument value. Pushes the tuples into the stack, and (NULL, number of parameters) finally. If no error occurs, copies them from the temporary stack to the parameter stack.
4. Increases `prc_level` by one. Sets `ret_level` to the same value as `proc_level`. `proc_level` is zero when runtime routine runs on the main routine. If it goes into a procedure, `proc_level` increases by one. Therefore, `proc_level` is the depth of the procedure call. `ret_level` is the level to return. If it is the same as `proc_level`, runtime routine executes commands in order of the commands in the procedure. If it is smaller than `proc_level`, runtime routine doesn't execute commands until it becomes the same level as `proc_level`. `ret_level` is used to return the procedure.
5. Executes the node of the body of the procedure.
6. Decreases `proc_level` by one. Sets `ret_level` to the same value as `proc_level`. Calls `stack_return`.

When the runtime routine meets `N_RT` node, it decreases `ret_level` by one so that the following commands in the procedure are ignored by the runtime routine.

Runtime entry and error functions A function `run` is the entry of the runtime routine. A function `runtime_error` reports an error occurred during the runtime routine runs. (Errors which occur during the parsing are called syntax error and reported by `yyerror`.) After `runtime_error` reports an error, it stops the command execution and goes back to `run` to exit.

Setjmp and longjmp functions are used. They are declared in `<setjmp.h>`. `setjmp (buf)` saves state information in `buf` and returns zero. `longjmp(buf, 1)` restores the state information from `buf` and returns `1` (the second argument). Because the information is the status at the time `setjmp` is called, so longjmp resumes the execution at the next of setjmp function call. In the following program, longjmp resumes at the assignment to the variable `i`. When setjmp is called, 0 is assigned to `i` and `execute(node_top)` is called. On the other hand, when longjmp is called, 1 is assigned to `i` and `execute(node_top)` is not called..

`g_slist_free_full` frees all the allocated memories.

```
static jmp_buf buf;

void
run (void) {
  int i;

  if (! init_cairo()) {
    g_print ("Cairo not initialized.\n");
    return;
  }
  init_table();
  init_stack();
  ret_level = proc_level = 1;
  i = setjmp (buf);
  if (i == 0)
    execute(node_top);
  /* else ... get here by calling longjmp */
  destroy_cairo ();
  g_slist_free_full (g_steal_pointer (&list), g_free);
}

/* format supports only %s, %f and %d */
static void
runtime_error (char *format, ...) {
  va_list args;
  char *f;
  char b[3];
  char *s;
  double v;
  int i;

  va_start (args, format);
  for (f = format; *f; f++) {
    if (*f != '%') {
      b[0] = *f;
      b[1] = '\0';
      g_print ("%s", b);
      continue;
    }
    switch (*++f) {
      case 's':
        s = va_arg(args, char *);
        g_print ("%s", s);
        break;
      case 'f':
        v = va_arg(args, double);
        g_print("%f", v);
        break;
      case 'd':
        i = va_arg(args, int);
        g_print("%d", i);
        break;
      default:
        b[0] = '%';
        b[1] = *f;
        b[2] = '\0';
        g_print ("%s", b);
        break;
    }
  }
  va_end (args);

  longjmp (buf, 1);
}
```

A function `runtime_error` has a variable-length argument list.

```
void runtime_error (char *format, ...)
```

This is implemented with `<stdarg.h>` header file. The `va_list` type variable `args` will refer to each argument in turn. A function `va_start` initializes `args`. A function `va_arg` returns an argument and moves the reference of `args` to the next. A function `va_end` cleans up everything necessary at the end.

The function `runtime_error` has a similar format of printf standard function. But its format has only `%s`, `%f` and `%d`.

The functions declared in `<setjmp.h>` and `<stdarg.h>` are explained in the very famous book "The C programming language" written by Brian Kernighan and Dennis Ritchie. I referred to the book to write the program above.

The program `turtle` is unsophisticated and unpolished. If you want to make your own language, you need to know more and more. I don't know any good textbook about compilers and interpreters. If you know a good book, please let me know.

However, the following information is very useful (but old).

- Bison documentation
- Flex documentation
- Software tools written by Brian W. Kernighan & P. J. Plauger (1976)
- Unix programming environment written by Brian W. Kernighan and Rob Pike (1984)
- Source code of a language, for example, ruby.

Lately, lots of source codes are in the internet. Maybe reading source codes are the most useful for programmers.

26 GtkListView

Gtk4 has added new list objects GtkListView, GtkGridView and GtkColumnView. The new feature is described in Gtk API Reference, List Widget Overview.

Gtk4 has other means to implement lists. They are GtkListBox and GtkTreeView which are took over from Gtk3. There's an article in Gtk Development blog about list widgets by Matthias Clasen. He described why GtkListView are developed to replace GtkListBox and GtkTreeView.

I want to explain GtkListView and its related objects in this tutorial.

26.1 Outline

A list is a sequential data structure. For example, an ordered string sequence "one", "two", "three", "four" is a list. Each element of the list is called item. A list is like an array, but in many cases it is implemented with pointers which point to the next item of the list. And it has a start point. So, each item can be referred by the index of the item (first item, second item, ..., nth item, ...). There are two cases. One is the index starts from one (one-based) and the other is it starts from zero (zero-based).

Gio provides GListModel interface. It is a zero-based list of the same type of GObject objects, or objects that implement the same interface. An object implements GListModel is usually not a widget. So, the list is not displayed on the screen directly. There's another object GtkListView which is a widget to display the list. The items in the list need to be connected to the items in GtkListView. GtkListItemFactory object maps items in the list to GListView.

The instruction to build the whole list related objects is:

1. Implement the list object which implements GListModel.
2. Build widgets and put GtkListView as a child of GtkScrolledWindow.
3. Set GtkListItemFactory.

26.2 GListModel

If you want to make a list of strings with GListModel, for example, "one", "two", "three", "four", note that strings can't be items of the list. Because GListModel is a list of GObject objects and strings aren't

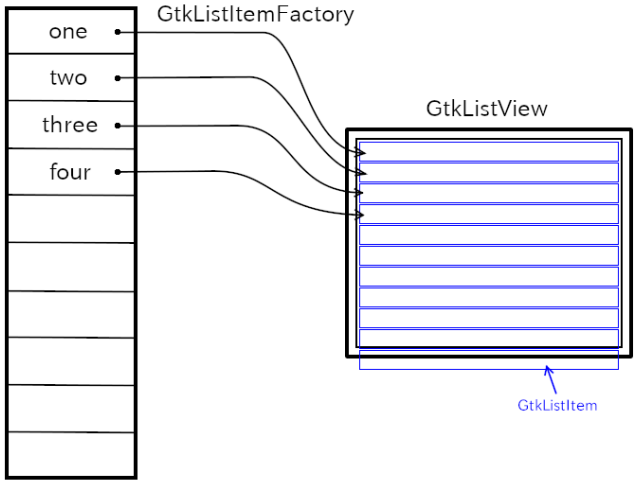

Figure 43: List

GObject objects. So, you need a wrapper which is a GObject and contains a string. GtkStringObject is the wrapper object and GStringList, implements GListModel, is a list of GtkStringObject.

```
char *array[] = {"one", "two", "three", "four", NULL};
GtkStringList *stringlist = gtk_string_list_new ((const char * const *) array);
```

The function `gtk_string_list_new` creates GtkStringList object. Its items are GtkStringObject objects which contain the strings “one”, “two”, “three” and “four”. There are functions to add items to the list or remove items from the list.

- `gtk_string_list_append` appends an item to the list
- `gtk_string_list_remove` removes an item from the list
- `gtk_string_list_get_string` gets a string in the list

See Gtk4 API Reference, GtkStringList for further information.

I’ll explain the other list objects later.

26.3 GtkSelectionModel

GtkSelectionModel is an interface to support for selections. Thanks to this model, user can select items by clicking on them. It is implemented by GtkMultiSelection, GtkNoSelection and GtkSingleSelection objects. These three objects are usually enough to build an application. They are created with GListModel. You can also create them alone and add GListModel later.

- GtkMultiSelection supports multiple selection.
- GtkNoSelection supports no selection. This is a wrapper to GListModel when GtkSelectionModel is needed.
- GtkSingleSelection supports single selection.

26.4 GtkListView

GtkListView is a widget to show GListModel items. GtkListItem is used by GtkListView to represent items of a list model. But, GtkListItem itself is not a widget, so a user needs to set a widget, for example GtkLabel, as a child of GtkListItem to display an item of the list model. “item” property of GtkListItem points an object that belongs to the list model.

In case the number of items is very big, for example more than a thousand, GtkListItem is recycled and connected to another item which is newly displayed. This recycle makes the number of GtkListItem objects fairly small, less than 200. This is very effective to restrain the growth of memory consumption so that GListModel can contain lots of items, for example, more than a million items.

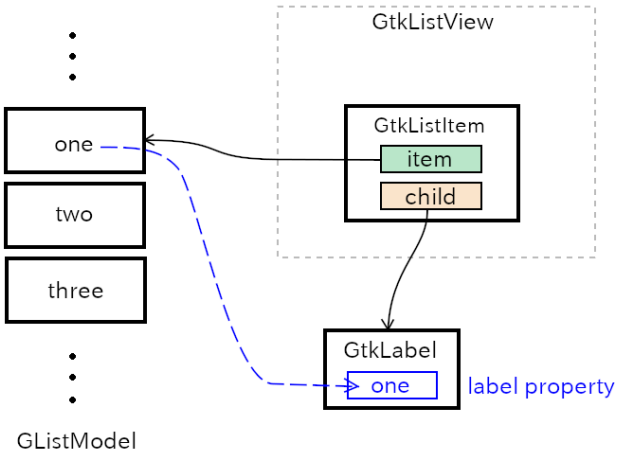

Figure 44: GtkListItem

26.5 GtkListItemFactory

GtkListItemFactory creates or recycles GtkListItem and connects it with an item of the list model. There are two child objects of this factory, GtkSignalListItemFactory and GtkBuilderListItemFactory.

26.5.1 GtkSignalListItemFactory

GtkSignalListItemFactory provides signals for users to configure a GtkListItem object. There are four signals.

1. "setup" is emitted to set up GtkListItem object. A user sets its child widget in the handler. For example, creates a GtkLabel widget and sets the child property of GtkListItem to it. This setting is kept even the GtkListItem instance is recycled (to bind to another item of GListModel).
2. "bind" is emitted to bind an item in the list model to the widget. For example, a user gets the item from "item" property of the GtkListItem instance. Then gets the string of the item and sets the label property of the GtkLabel instance with the string. This signal is emitted when the GtkListItem is newly created, recycled or some changes has happened to the item of the list.
3. "unbind" is emitted to unbind an item. A user undoes everything done in step 2 in the signal handler. If some object are created in step 2, they must be destroyed.
4. "teardown" is emitted to undo everything done in step 1. So, the widget created in step 1 must be destroyed. After this signal, the list item will be destroyed.

The following program `list1.c` shows the list of strings "one", "two", "three" and "four". GtkNoSelection is used, so user can't select any item.

```
1  #include <gtk/gtk.h>
2
3  static void
4  setup_cb (GtkListItemFactory *factory, GtkListItem *listitem, gpointer user_data) {
5    GtkWidget *lb = gtk_label_new (NULL);
6    gtk_list_item_set_child (listitem, lb);
7  }
8
9  static void
10 bind_cb (GtkSignalListItemFactory *self, GtkListItem *listitem, gpointer user_data) {
11   GtkWidget *lb = gtk_list_item_get_child (listitem);
12   GtkStringObject *strobj = gtk_list_item_get_item (listitem);
13   const char *text = gtk_string_object_get_string (strobj);
14
15   gtk_label_set_text (GTK_LABEL (lb), text);
16 }
```

```
17
18  static void
19  unbind_cb (GtkSignalListItemFactory *self, GtkListItem *listitem, gpointer
        user_data) {
20    /* There's nothing to do here. */
21    /* If you does something like setting a signal in bind_cb, */
22    /* then disconnecting the signal is necessary in unbind_cb. */
23  }
24
25  static void
26  teardown_cb (GtkListItemFactory *factory, GtkListItem *listitem, gpointer user_data)
        {
27    gtk_list_item_set_child (listitem, NULL);
28  /*  When the child of listitem is set to NULL, the reference to GtkLabel will be
        released and lb will be destroyed. */
29  /* Therefore, g_object_unref () for the GtkLabel object doesn't need in the user
        code. */
30  }
31
32  /* ----- activate, open, startup handlers ----- */
33  static void
34  app_activate (GApplication *application) {
35    GtkApplication *app = GTK_APPLICATION (application);
36    GtkWidget *win = gtk_application_window_new (app);
37    gtk_window_set_default_size (GTK_WINDOW (win), 600, 400);
38    GtkWidget *scr = gtk_scrolled_window_new ();
39    gtk_window_set_child (GTK_WINDOW (win), scr);
40
41    char *array[] = {
42      "one", "two", "three", "four", NULL
43    };
44    GtkStringList *sl = gtk_string_list_new ((const char * const *) array);
45    GtkNoSelection *ns = gtk_no_selection_new (G_LIST_MODEL (sl));
46
47    GtkListItemFactory *factory = gtk_signal_list_item_factory_new ();
48    g_signal_connect (factory, "setup", G_CALLBACK (setup_cb), NULL);
49    g_signal_connect (factory, "bind", G_CALLBACK (bind_cb), NULL);
50    g_signal_connect (factory, "unbind", G_CALLBACK (unbind_cb), NULL);
51    g_signal_connect (factory, "teardown", G_CALLBACK (teardown_cb), NULL);
52
53    GtkWidget *lv = gtk_list_view_new (GTK_SELECTION_MODEL (ns), factory);
54    gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), lv);
55    gtk_widget_show (win);
56  }
57
58  static void
59  app_startup (GApplication *application) {
60  }
61
62  /* ----- main ----- */
63  #define APPLICATION_ID "com.github.ToshioCP.list1"
64
65  int
66  main (int argc, char **argv) {
67    GtkApplication *app;
68    int stat;
69
70    app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
71
72    g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
73    g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
74
75    stat =g_application_run (G_APPLICATION (app), argc, argv);
76    g_object_unref (app);
```

Figure 45: list1

```
77    return stat;
78  }
```

The file `list1.c` is located under the directory src/misc. Make a shell script below and save it to your bin directory. (If you've installed Gtk4 from the source to $HOME/local, then your bin directory is $Home/local/bin. Otherwise, $Home/bin is your private bin directory.)

```
gcc `pkg-config --cflags gtk4` $1.c `pkg-config --libs gtk4`
```

Change the current directory to the directory includes `list1.c` and type as follows.

```
$ chmod 755 $HOME/local/bin/comp # or chmod 755 $Home/bin/comp
$ comp list1
$ ./a.out
```

Then, `list1.c` has been compiled and executed.

I think the program is not so difficult. If you feel some difficulty, read this section again, especially GtkSignalListItemFactory subsubsection.

26.5.2 GtkBuilderListItemFactory

GtkBuilderListItemFactory is another GtkListItemFactory. Its behavior is defined with ui file.

```
<interface>
  <template class="GtkListItem">
    <property name="child">
      <object class="GtkLabel">
        <binding name="label">
          <lookup name="string" type="GtkStringObject">
            <lookup name="item">GtkListItem</lookup>
          </lookup>
        </binding>
      </object>
    </property>
  </template>
</interface>
```

Template tag is used to define GtkListItem. And its child property is GtkLabel object. The factory sees this template and creates GtkLabel and sets the child property of GtkListItem. This is the same as what setup handler of GtkSignalListItemFactory did.

Then, bind the label property of GtkLabel to string property of GtkStringObject. The string object is referred to by item property of GtkListItem. So, the lookup tag is like this:

```
string <- GtkStringObject <- item <- GtkListItem
```

The last lookup tag has a content `GtkListItem`. Usually, C type like `GtkListItem` doesn't appear in the content of tags. This is a special case. There is an explanation about it in the GTK Development Blog by Matthias Clasen.

Remember that the classname (GtkListItem) in a ui template is used as the “this” pointer referring to the object that is being instantiated.

Therefore, GtkListItem instance is used as the `this` object of the lookup tag when it is evaluated. `this` object will be explained in section 28.

The C source code is as follows. Its name is `list2.c` and located under src/misc directory.

```
1  #include <gtk/gtk.h>
2
3  /* ----- activate, open, startup handlers ----- */
4  static void
5  app_activate (GApplication *application) {
6    GtkApplication *app = GTK_APPLICATION (application);
7    GtkWidget *win = gtk_application_window_new (app);
8    gtk_window_set_default_size (GTK_WINDOW (win), 600, 400);
9    GtkWidget *scr = gtk_scrolled_window_new ();
10   gtk_window_set_child (GTK_WINDOW (win), scr);
11
12   char *array[] = {
13     "one", "two", "three", "four", NULL
14   };
15   GtkStringList *sl = gtk_string_list_new ((const char * const *) array);
16   GtkSingleSelection *ss = gtk_single_selection_new (G_LIST_MODEL (sl));
17
18   const char *ui_string =
19 "<interface>"
20   "<template class=\"GtkListItem\">"
21     "<property name=\"child\">"
22       "<object class=\"GtkLabel\">"
23         "<binding name=\"label\">"
24           "<lookup name=\"string\" type=\"GtkStringObject\">"
25             "<lookup name=\"item\">GtkListItem</lookup>"
26           "</lookup>"
27         "</binding>"
28       "</object>"
29     "</property>"
30   "</template>"
31 "</interface>"
32 ;
33   GBytes *gbytes = g_bytes_new_static (ui_string, strlen (ui_string));
34   GtkListItemFactory *factory = gtk_builder_list_item_factory_new_from_bytes (NULL,
       gbytes);
35
36   GtkWidget *lv = gtk_list_view_new (GTK_SELECTION_MODEL (ss), factory);
37   gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), lv);
38   gtk_widget_show (win);
39 }
40
41 static void
42 app_startup (GApplication *application) {
43 }
44
45 /* ----- main ----- */
46 #define APPLICATION_ID "com.github.ToshioCP.list2"
47
48 int
49 main (int argc, char **argv) {
50   GtkApplication *app;
51   int stat;
52
53   app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
54
55   g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
56   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
```

```
57
58    stat =g_application_run (G_APPLICATION (app), argc, argv);
59    g_object_unref (app);
60    return stat;
61  }
```

No signal handler is needed for GtkBulderListItemFactory. GtkSingleSelection is used, so user can select one item at a time.

Because this is a small program, the ui data is given as a string.

26.6 GtkDirectoryList

GtkDirectoryList is a list model containing GFileInfo objects which are information of files under a certain directory. It uses `g_file_enumerate_children_async()` to get the GFileInfo objects. The list model is created by `gtk_directory_list_new` function.

```
GtkDirectoryList *gtk_directory_list_new (const char *attributes, GFile *file);
```

`attributes` is a comma separated list of file attributes. File attributes are key-value pairs. A key consists of a namespace and a name. For example, "standard::name" key is the name of a file. "standard" means general file information. "name" means filename. The following table shows some example.

key	meaning
standard::type	file type. for example, regular file, directory, symbolic link, etc.
standard::name	filename
standard::size	file size in bytes
access::can-read	read privilege if the user is able to read the file
time::modified	the time the file was last modified in seconds since the UNIX epoch

The current directory is ".". The following program makes GtkDirectoryList `dl` and its contents are GFileInfo objects under the current directory.

```
GFile *file = g_file_new_for_path (".");
GtkDirectoryList *dl = gtk_directory_list_new ("standard::name", file);
g_object_unref (file);
```

It is not so difficult to make file listing program by changing `list2.c` in the previous subsection. One problem is that GInfoFile doesn't have properties. Lookup tag look for a property, so it is useless for looking for a filename from a GFileInfo object. Instead, closure tag is appropriate in this case. Closure tag specifies a function and the type of the return value of the function.

```
char *
get_file_name (GtkListItem *item, GFileInfo *info) {
  if (! G_IS_FILE_INFO (info))
    return NULL;
  else
    return g_strdup (g_file_info_get_name (info));
}

... ...
... ...

"<interface>"
  "<template class=\"GtkListItem\">"
    "<property name=\"child\">"
      "<object class=\"GtkLabel\">"
        "<binding name=\"label\">"
          "<closure type=\"gchararray\" function=\"get_file_name\">"
            "<lookup name=\"item\">GtkListItem</lookup>"
          "</closure>"
        "</binding>"
```

```
      "</object>"
    "</property>"
  "</template>"
"</interface>"
```

- "gchararray" is the type name of strings. "gchar" is the same as "char" type. Therefore, "gchararray" is "an array of char type", which is the same as string type. It is used to get the type of GValue object. GValue is a generic value and it can contain various type of values. For example, the type name can be gboolean, gchar (char), gint (int), gfloat (float), gdouble (double), gchararray (char *) and so on. These type names are the names of the fundamental types that are registered to the type system. See GObject tutorial.
- closure tag has type attribute and function attribute. Function attribute specifies the function name and type attribute specifies the type of the return value of the function. The contents of closure tag (it is between <closure…> and</closure>) is parameters of the function. `<lookup name="item">GtkListItem</lookup>` gives the value of the item property of the GtkListItem. This will be the second argument of the function. The first parameter is always the GListItem instance.
- `gtk_file_name` function first check the `info` parameter. Because it can be NULL when GListItem `item` is unbound. If its GFileInfo, then return the filename (copy of the filename).

The whole program (`list3.c`) is as follows. The program is located in src/misc directory.

```
1  #include <gtk/gtk.h>
2
3  char *
4  get_file_name (GtkListItem *item, GFileInfo *info) {
5    if (! G_IS_FILE_INFO (info))
6      return NULL;
7    else
8      return g_strdup (g_file_info_get_name (info));
9  }
10
11 /* ----- activate, open, startup handlers ----- */
12 static void
13 app_activate (GApplication *application) {
14   GtkApplication *app = GTK_APPLICATION (application);
15   GtkWidget *win = gtk_application_window_new (app);
16   gtk_window_set_default_size (GTK_WINDOW (win), 600, 400);
17   GtkWidget *scr = gtk_scrolled_window_new ();
18   gtk_window_set_child (GTK_WINDOW (win), scr);
19
20   GFile *file = g_file_new_for_path (".");
21   GtkDirectoryList *dl = gtk_directory_list_new ("standard::name", file);
22   g_object_unref (file);
23   GtkNoSelection *ns = gtk_no_selection_new (G_LIST_MODEL (dl));
24
25   const char *ui_string =
26 "<interface>"
27   "<template class=\"GtkListItem\">"
28     "<property name=\"child\">"
29       "<object class=\"GtkLabel\">"
30         "<binding name=\"label\">"
31           "<closure type=\"gchararray\" function=\"get_file_name\">"
32             "<lookup name=\"item\">GtkListItem</lookup>"
33           "</closure>"
34         "</binding>"
35       "</object>"
36     "</property>"
37   "</template>"
38 "</interface>"
39 ;
40   GBytes *gbytes = g_bytes_new_static (ui_string, strlen (ui_string));
41   GtkListItemFactory *factory = gtk_builder_list_item_factory_new_from_bytes (NULL,
       gbytes);
42
```

```
43    GtkWidget *lv = gtk_list_view_new (GTK_SELECTION_MODEL (ns), factory);
44    gtk_scrolled_window_set_child (GTK_SCROLLED_WINDOW (scr), lv);
45    gtk_widget_show (win);
46  }
47
48  static void
49  app_startup (GApplication *application) {
50  }
51
52  /* ----- main ----- */
53  #define APPLICATION_ID "com.github.ToshioCP.list3"
54
55  int
56  main (int argc, char **argv) {
57    GtkApplication *app;
58    int stat;
59
60    app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
61
62    g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
63    g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
64
65    stat =g_application_run (G_APPLICATION (app), argc, argv);
66    g_object_unref (app);
67    return stat;
68  }
```

The ui data (xml data above) is used to build the GListItem template at runtime. GtkBuilder refers to the symbol table to find the function `get_file_name`.

Generally, a symbol table is used by a linker to link objects to an executable file. It includes function names and their location. A linker usually doesn't put a symbol table into the created executable file. But if `--export-dynamic` option is given, the linker adds the symbol table to the executable file.

To accomplish it, an option `-Wl,--export-dynamic` is given to the C compiler.

- `-Wl` is a C compiler option that passes the following option to the linker.
- `--export-dynamic` is a linker option. The following is cited from the linker document. "When creating a dynamically linked executable, add all symbols to the dynamic symbol table. The dynamic symbol table is the set of symbols which are visible from dynamic objects at run time."

Compile and execute it.

```
$ gcc -Wl,--export-dynamic `pkg-config --cflags gtk4` list3.c `pkg-config --libs
    gtk4`
```

You can also make a shell script to compile `list3.c`

```
gcc -Wl,--export-dynamic `pkg-config --cflags gtk4` $1.c `pkg-config --libs gtk4`
```

Save this one liner to a file `comp`. Then, copy it to `$HOME/bin` and give it executable permission.

```
$ cp comp $HOME/bin/comp
$ chmod +x $HOME/bin/comp
```

You can compile `list3.c` and execute it, like this:

```
$ comp list3
$ ./a.out
```

27 GtkGridView and activate signal

GtkGridView is similar to GtkListView. It displays a GListModel as a grid, which is like a square tessellation.

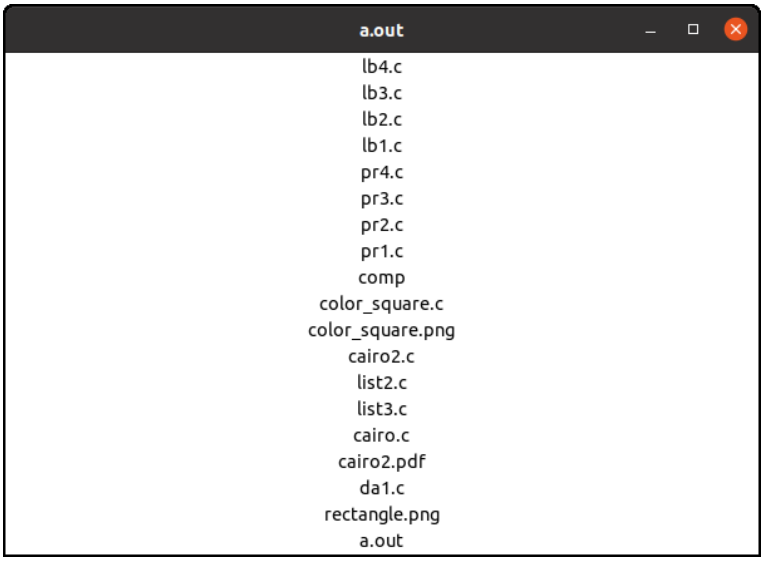

Figure 46: screenshot list3

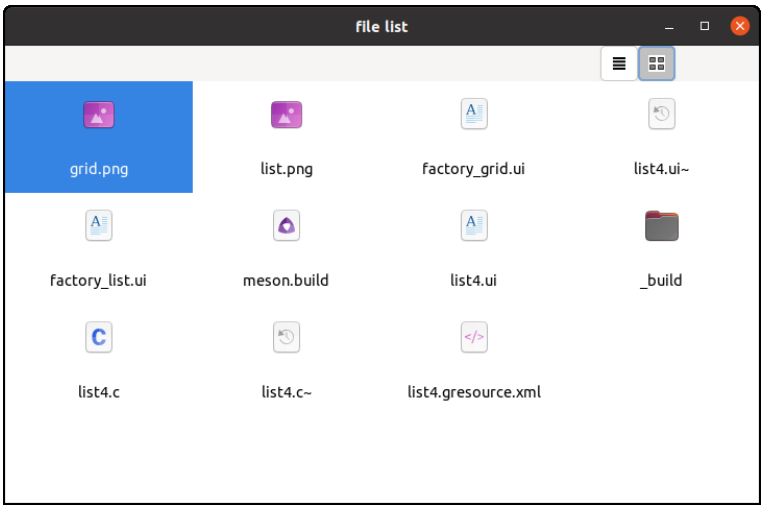

Figure 47: Grid

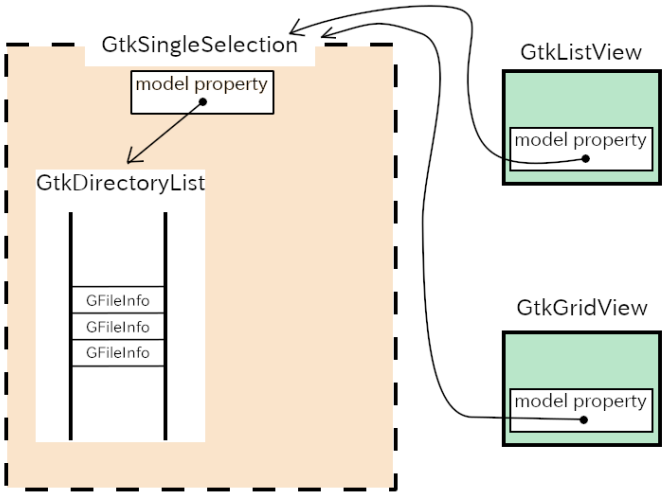

Figure 48: DirectoryList

This is often seen when you use a file browser like nautilus.

In this section, let's make a very simple file browser `list4`. It just shows the files in the current directory. And a user can choose list or grid by clicking on buttons in the tool bar. Each item in the list or grid has an icon and a filename. In addition, `list4` provides the way to open the `tfe` text editor to show a text file. A user can do that by double clicking on an item or pressing enter key when an item is selected.

27.1 GtkDirectoryList

GtkDirectoryList implements GListModel and it contains information of files in a certain directory. The items of the list are GFileInfo objects.

In the `list4` source files, GtkDirectoryList is described in a ui file and built by GtkBuilder. The GtkDirectoryList instance is assigned to the "model" property of a GtkSingleSelection instance. And the GtkSingleSelection instance is assigned to the "model" property of a GListView or GGridView instance.

```
GtkListView (model property) => GtkSingleSelection (model property) =>
    GtkDirectoryList
GtkGridView (model property) => GtkSingleSelection (model property) =>
    GtkDirectoryList
```

The following is the part of the ui file `list4.ui`. It defines GtkListView, GtkSingleSelection and GtkDirectoryList. It also defines GtkGridView and GtkSingleSelection.

```
<object class="GtkListView" id="list">
  <property name="model">
    <object class="GtkSingleSelection" id="singleselection">
      <property name="model">
        <object class="GtkDirectoryList" id="directorylist">
          <property
              name="attributes">standard::name,standard::icon,standard::content-type</property>
        </object>
      </property>
    </object>
  </property>
</object>
<object class="GtkGridView" id="grid">
  <property name="model">singleselection</property>
</object>
```

GtkDirectoryList has an "attributes" property. It is attributes of GFileInfo such as "standard::name", "standard::icon" and "standard::content-type".

- standard::name is a filename.
- standard::icon is an icon of the file. It is a GIcon object.
- standard::content-type is a content-type. Content-type is the same as mime type for the internet technology. For example, “text/plain” is a text file, “text/x-csrc” is a C source code and so on. (“text/x-csrc”is not registered to IANA media types. Such “x-” subtype is not a standard mime type.) Content type is also used by the desktop system.

GtkGridView has the same structure as GtkListView. But it is enough to specify its model property to `singleselection` which is the identification of the GtkSingleSelection. Therefore the description for GtkGridView is very short.

27.2 Ui file of the window

Look at the screenshot of `list4` at the top of this section. The widgets are built with the following ui file.

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <object class="GtkApplicationWindow" id="win">
    <property name="title">file list</property>
    <property name="default-width">600</property>
    <property name="default-height">400</property>
    <child>
      <object class="GtkBox" id="boxv">
        <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
        <child>
          <object class="GtkBox" id="boxh">
            <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
            <child>
              <object class="GtkLabel" id="dmy1">
                <property name="hexpand">TRUE</property>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btnlist">
                <property name="name">btnlist</property>
                <property name="action-name">win.view</property>
                <property name="action-target">&apos;list&apos;</property>
                <child>
                  <object class="GtkImage">
                    <property
                        name="resource">/com/github/ToshioCP/list4/list.png</property>
                  </object>
                </child>
              </object>
            </child>
            <child>
              <object class="GtkButton" id="btngrid">
                <property name="name">btngrid</property>
                <property name="action-name">win.view</property>
                <property name="action-target">&apos;grid&apos;</property>
                <child>
                  <object class="GtkImage">
                    <property
                        name="resource">/com/github/ToshioCP/list4/grid.png</property>
                  </object>
                </child>
              </object>
            </child>
            <child>
              <object class="GtkLabel" id="dmy2">
                <property name="width-chars">10</property>
              </object>
            </child>
          </object>
```

```
48          </child>
49          <child>
50            <object class="GtkScrolledWindow" id="scr">
51              <property name="hexpand">TRUE</property>
52              <property name="vexpand">TRUE</property>
53            </object>
54          </child>
55        </object>
56      </child>
57    </object>
58    <object class="GtkListView" id="list">
59      <property name="model">
60        <object class="GtkSingleSelection" id="singleselection">
61          <property name="model">
62            <object class="GtkDirectoryList" id="directorylist">
63              <property
                    name="attributes">standard::name,standard::icon,standard::content-type</property>
64            </object>
65          </property>
66        </object>
67      </property>
68    </object>
69    <object class="GtkGridView" id="grid">
70      <property name="model">singleselection</property>
71    </object>
72  </interface>
```

The file consists of two parts. The first part begins at the third line and ends at the 57th line. This part is the widgets from the top level window to the scrolled window. It also includes two buttons. The second part begins at the 58th line and ends at the 71st line. This is the part of GtkListView and GtkGridView. They are described in the previous section.

- 13-17, 42-46: Two labels are dummy labels. They just work as a space to put the two buttons at the appropriate position.
- 19-41: GtkButton `btnlist` and `btngrid`. These two buttons work as selection buttons to switch from list to grid and vice versa. These two buttons are connected to a stateful action `win.view`. This action is stateful and has a parameter. Such action consists of prefix, action name and parameter. The prefix of the action is `win`, which means the action belongs to the top level window. So, a prefix gives the scope of the action. The action name is `view`. The parameters are `list` or `grid`, which show the state of the action. A parameter is also called a target, because it is a target to which the buttons are clicked on to change the action state. We often write the detailed action like "win.view::list" or "win.view::grid".
- 21-22: The properties "action-name" and "action-target" belong to GtkActionable interface. GtkButton implements GtkActionable. The action name is "win.view" and the target is "list". Generally, a target is GVariant, which can be string, integer, float and so on. You need to use GVariant text format to write GVariant value in ui files. If the type of the GVariant value is string, then the value with GVariant text format is bounded by single quotes or double quotes. Because ui file is xml format text, single quote cannot be written without escape. Its escape sequence is '. Therefore, the target 'list' is written as 'list'. Because the button is connected to the action, "clicked" signal handler isn't needed.
- 23-27: The child widget of the button is GtkImage. GtkImage has a "resource" property. It is a GResource and GtkImage reads an image data from the resource and sets the image. This resource is built from 24x24-sized png image data, which is an original icon.
- 50-53: GtkScrolledWindow. Its child widget will be GtkListView or GtkGridView.

The action `view` is created, connected to the "activate" signal handler and inserted to the window (action map) as follows.

```
act_view = g_simple_action_new_stateful ("view", g_variant_type_new("s"),
    g_variant_new_string ("list"));
g_signal_connect (act_view, "activate", G_CALLBACK (view_activated), scr); /* scr
    is the GtkScrolledWindow object */
g_action_map_add_action (G_ACTION_MAP (win), G_ACTION (act_view));
```

The signal handler `view_activated` will be explained later.

27.3 Factories

Each view (GtkListView and GtkGridView) has its own factory because its items have different structure of widgets. The factories are GtkBuilderListItemFactory objects. Their ui files are as follows.

factory_list.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <template class="GtkListItem">
    <property name="child">
      <object class="GtkBox">
        <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
        <property name="spacing">20</property>
        <child>
          <object class="GtkImage">
            <binding name="gicon">
              <closure type="GIcon" function="get_icon">
                <lookup name="item">GtkListItem</lookup>
              </closure>
            </binding>
          </object>
        </child>
        <child>
          <object class="GtkLabel">
            <property name="hexpand">TRUE</property>
            <property name="xalign">0</property>
            <binding name="label">
              <closure type="gchararray" function="get_file_name">
                <lookup name="item">GtkListItem</lookup>
              </closure>
            </binding>
          </object>
        </child>
      </object>
    </property>
  </template>
</interface>
```

factory_grid.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <template class="GtkListItem">
    <property name="child">
      <object class="GtkBox">
        <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
        <property name="spacing">20</property>
        <child>
          <object class="GtkImage">
            <property name="icon-size">GTK_ICON_SIZE_LARGE</property>
            <binding name="gicon">
              <closure type="GIcon" function="get_icon">
                <lookup name="item">GtkListItem</lookup>
              </closure>
            </binding>
          </object>
        </child>
        <child>
          <object class="GtkLabel">
            <property name="hexpand">TRUE</property>
            <property name="xalign">0.5</property>
            <binding name="label">
```

```
23                <closure type="gchararray" function="get_file_name">
24                  <lookup name="item">GtkListItem</lookup>
25                </closure>
26              </binding>
27            </object>
28          </child>
29        </object>
30      </property>
31    </template>
32  </interface>
```

The two files above are almost same. The difference is:

- The orientation of the box
- The icon size
- The position of the text of the label

```
$ cd list4; diff factory_list.ui factory_grid.ui
6c6
<         <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
---
>         <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
9a10
>             <property name="icon-size">GTK_ICON_SIZE_LARGE</property>
20c21
<             <property name="xalign">0</property>
---
>             <property name="xalign">0.5</property>
```

Each view item has two properties, “gicon” property of GtkImage and “label” property of GtkLabel. Because GFileInfo doesn’t have properties correspond to icon or filename, the factory uses closure tag to bind “gicon” and “label” properties to GFileInfo information. A function `get_icon` gets GIcon the GFileInfo object has. And a function `get_file_name` gets a filename the GFileInfo object has.

```
1  GIcon *
2  get_icon (GtkListItem *item, GFileInfo *info) {
3    GIcon *icon;
4
5    if (! G_IS_FILE_INFO (info))
6      return NULL;
7    else {
8      icon = g_file_info_get_icon (info);
9      g_object_ref (icon);
10     return icon;
11   }
12 }
13
14 char *
15 get_file_name (GtkListItem *item, GFileInfo *info) {
16   if (! G_IS_FILE_INFO (info))
17     return NULL;
18   else
19     return g_strdup (g_file_info_get_name (info));
20 }
```

One important thing is view items own the instance or string. It is achieved by `g_object_ref` to increase the reference count by one, or `strdup` to create a copy of the string. The object or string will be automatically freed in unbinding process when the view item is recycled.

27.4 An activate signal handler of the action

An activate signal handler `view_activate` switches the view. It does two things.

- Changes the child widget of GtkScrolledWindow.

- Changes the CSS of buttons to show the current state.

```
1  static void
2  view_activated(GSimpleAction *action, GVariant *parameter, gpointer user_data) {
3    GtkScrolledWindow *scr = GTK_SCROLLED_WINDOW (user_data);
4    const char *view = g_variant_get_string (parameter, NULL);
5    const char *other;
6    char *css;
7
8    if (strcmp (view, "list") == 0) {
9      other = "grid";
10     gtk_scrolled_window_set_child (scr, list);
11   }else {
12     other = "list";
13     gtk_scrolled_window_set_child (scr, grid);
14   }
15   css = g_strdup_printf ("button#btn%s {background: silver;} button#btn%s
         {background: white;}", view, other);
16   gtk_css_provider_load_from_data (provider, css, -1);
17   g_free (css);
18   g_action_change_state (G_ACTION (action), parameter);
19 }
```

The second parameter of this handler is the target of the clicked button. Its type is GVariant.

- If `btnlist` has been clicked, then `parameter` is a GVariant of the string "list".
- If `btngrid` has been clicked, then `parameter` is a GVariant of the string "grid".

The third parameter `user_data` points GtkScrolledWindow, which is set in the `g_signal_connect` function.

- 4: `g_variant_get_string` gets the string from the GVariant variable.
- 8-14: Sets the child of `scr`. The function `gtk_scrolled_window_set_child` decreases the reference count of the old child by one. And it increases the reference count of the new child by one.
- 15-17: Sets the CSS of the buttons. The background of the clicked button will be silver color and the other button will be white.
- 18: Changes the state of the action.

27.5 Activate signal of GtkListView and GtkGridView

Views (GtkListView and GtkGridView) have an "activate" signal. It is emitted when an item in the view is double clicked or the enter key is pressed. You can do anything you like by connecting the "activate" signal to the handler.

The example `list4` launches `tfe` text file editor if the item of the list is a text file.

```
static void
list_activate (GtkListView *list, int position, gpointer user_data) {
  GFileInfo *info = G_FILE_INFO (g_list_model_get_item (G_LIST_MODEL
      (gtk_list_view_get_model (list)), position));
  launch_tfe_with_file (info);
}

static void
grid_activate (GtkGridView *grid, int position, gpointer user_data) {
  GFileInfo *info = G_FILE_INFO (g_list_model_get_item (G_LIST_MODEL
      (gtk_grid_view_get_model (grid)), position));
  launch_tfe_with_file (info);
}

... ...
... ...

  g_signal_connect (GTK_LIST_VIEW (list), "activate", G_CALLBACK (list_activate),
      NULL);
  g_signal_connect (GTK_GRID_VIEW (grid), "activate", G_CALLBACK (grid_activate),
      NULL);
```

The second parameter of the handlers is the position of the item (GFileInfo) of the GListModel. So you can get the item with `g_list_model_get_item` function.

27.6 Content type and launching an application

The function `launch_tfe_with_file` gets a file from the GFileInfo instance. If the file is a text file, it launches `tfe` with the file.

GFileInfo has information about file type. The file type is like "text/plain", "text/x-csrc" and so on. It is called content type. Content type can be got with `g_file_info_get_content_type` function.

```
1  static void
2  launch_tfe_with_file (GFileInfo *info) {
3    GError *err = NULL;
4    GFile *file;
5    GList *files = NULL;
6    const char *content_type;
7    const char *text_type = "text/";
8    GAppInfo *appinfo;
9    int i;
10
11   if (! info)
12     return;
13   content_type = g_file_info_get_content_type (info);
14 g_print ("%s\n", content_type);  /* This line can be commented out if unnecessary */
15   if (! content_type)
16     return;
17   for (i=0;i<5;++i) {
18     if (content_type[i] != text_type[i])
19       return;
20   }
21   appinfo = g_app_info_create_from_commandline ("tfe", "tfe",
         G_APP_INFO_CREATE_NONE, &err);
22   if (err) {
23     g_printerr ("%s\n", err->message);
24     g_error_free (err);
25     return;
26   }
27   err = NULL;
28   file = g_file_new_for_path (g_file_info_get_name (info));
29   files = g_list_append (files, file);
30   if (! (g_app_info_launch (appinfo, files, NULL, &err))) {
31     g_printerr ("%s\n", err->message);
32     g_error_free (err);
33   }
34   g_list_free_full (files, g_object_unref);
35   g_object_unref (appinfo);
36 }
```

- 13: Gets the content type of the file from GFileInfo.
- 14: Prints the content type. This is only useful to know a content type of a file. You can delete it if unnecessary.
- 17-20: If the content type doesn't begin with "text/", then it returns.
- 21: Creates GAppInfo object of `tfe` application. GAppInfo is an interface and the variable `appinfo` points a GDesktopAppInfo instance. GAppInfo is a collection of information of an application.
- 30: Launches the application (`tfe`) with an argument `file`. `g_app_info_launch` has four parameters. The first parameter is GAppInfo object. The second parameter is a list of GFile objects. In this function, only one GFile instance is given to `tfe`, but you can give more arguments. The third parameter is GAppLaunchContext, but this program gives NULL instead. The last parameter is the pointer to the pointer to a GError.
- 34: `g_list_free_full` frees the memories used by the list and items.

If your distribution supports Gtk4, using `g_app_info_launch_default_for_uri` is convenient. The function automatically determines the default application from the file and launches it. For example, if the file is

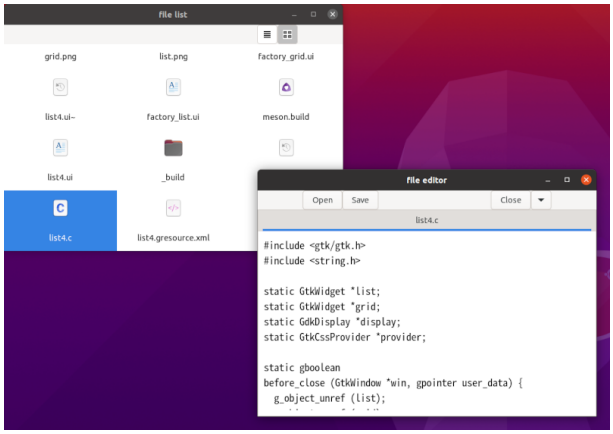

Figure 49: Screenshot

text, then it launches gedit with the file. Such functionality comes from desktop.

27.7 Compilation and execution

The source files are located in src/list4 directory. To compile and execute list4, type as follows.

```
$ cd list4 # or cd src/list4. It depends your current directory.
$ meson _build
$ ninja -C _build
$ _build/list4
```

Then a file list appears as a list style. Click on a button on the tool bar so that you can change the style to grid or back to list. Double click “list4.c” item, then `tfe` text editor runs with the argument “list4.c”. The following is the screenshot.

27.8 “gbytes” property of GtkBuilderListItemFactory

GtkBuilderListItemFactory has “gbytes” property. The property contains a byte sequence of ui data. If you use this property, you can put the contents of `factory_list.ui` and `factory_grid.ui`into `list4.ui`. The following shows a part of the new ui file (`list5.ui`).

```
  <object class="GtkListView" id="list">
    <property name="model">
      <object class="GtkSingleSelection" id="singleselection">
        <property name="model">
          <object class="GtkDirectoryList" id="directorylist">
            <property
                name="attributes">standard::name,standard::icon,standard::content-type</property>
          </object>
        </property>
      </object>
    </property>
    <property name="factory">
      <object class="GtkBuilderListItemFactory">
        <property name="bytes"><![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <template class="GtkListItem">
    <property name="child">
      <object class="GtkBox">
        <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
        <property name="spacing">20</property>
        <child>
          <object class="GtkImage">
            <binding name="gicon">
              <closure type="GIcon" function="get_icon">
```

```
                <lookup name="item">GtkListItem</lookup>
              </closure>
            </binding>
          </object>
        </child>
        <child>
          <object class="GtkLabel">
            <property name="hexpand">TRUE</property>
            <property name="xalign">0</property>
            <binding name="label">
              <closure type="gchararray" function="get_file_name">
                <lookup name="item">GtkListItem</lookup>
              </closure>
            </binding>
          </object>
        </child>
      </object>
    </property>
  </template>
</interface>
        ]]></property>
      </object>
    </property>
  </object>
```

CDATA section begins with "<[CDATA[" and ends with "]]>". The contents of CDATA section is recognized as a string. Any character, even if it is a key syntax marker such as '<' or '>', is recognized literally. Therefore, the text between "<[CDATA[" and "]]>" is inserted to "bytes" property as it is.

This method decreases the number of ui files. But, the new ui file is a bit complicated especially for the beginners. If you feel some difficulty, it is better for you to separate the ui file.

A directory src/list5 includes the ui file above.

28 GtkExpression

GtkExpression is a fundamental type. It is not a descendant of GObject. GtkExpression provides a way to describe references to values. GtkExpression needs to be evaluated to obtain a value.

It is similar to arithmetic calculation.

```
1 + 2 = 3
```

`1+2` is an expression. It shows the way how to calculate. `3` is the value comes from the expression. Evaluation is to calculate the expression and get the value.

GtkExpression is a way to get a value. Evaluation is like a calculation. A value is got by evaluating the expression.

First, I want to show you the C file of the example for GtkExpression. Its name is `exp.c` and located under src/expression directory. You don't need to understand the details now, just look at it. It will be explained in the next subsection.

```
1  #include <gtk/gtk.h>
2
3  GtkWidget *win1;
4  int width, height;
5  GtkExpressionWatch *watch_width;
6  GtkExpressionWatch *watch_height;
7
8  /* Notify is called when "default-width" or "default-height" property is changed. */
9  static void
10 notify (gpointer user_data) {
11   GValue value = G_VALUE_INIT;
12   char *title;
```

```
13
14    if (watch_width && gtk_expression_watch_evaluate (watch_width, &value))
15      width = g_value_get_int (&value);
16    g_value_unset (&value);
17    if (watch_height && gtk_expression_watch_evaluate (watch_height, &value))
18      height = g_value_get_int (&value);
19    g_value_unset (&value);
20    title = g_strdup_printf ("%d x %d", width, height);
21    gtk_window_set_title (GTK_WINDOW (win1), title);
22    g_free (title);
23  }
24
25  /* This function is used by closure tag in exp.ui. */
26  char *
27  set_title (GtkWidget *win, int width, int height) {
28    return g_strdup_printf ("%d x %d", width, height);
29  }
30
31  /* ----- activate, open, startup handlers ----- */
32  static void
33  app_activate (GApplication *application) {
34    GtkApplication *app = GTK_APPLICATION (application);
35    GtkWidget *box;
36    GtkWidget *label1, *label2, *label3;
37    GtkWidget *entry;
38    GtkEntryBuffer *buffer;
39    GtkBuilder *build;
40    GtkExpression *expression, *expression1, *expression2;
41    GValue value = G_VALUE_INIT;
42    char *s;
43
44    /* Creates GtkApplicationWindow instance. */
45    /* The codes below are complecated. It does the same as "win1 =
          gtk_application_window_new (app);". */
46    /* The codes are written just to show how to use GtkExpression. */
47    expression = gtk_cclosure_expression_new (GTK_TYPE_APPLICATION_WINDOW, NULL, 0,
          NULL,
48                  G_CALLBACK (gtk_application_window_new), NULL, NULL);
49    if (gtk_expression_evaluate (expression, app, &value)) {
50      win1 = GTK_WIDGET (g_value_get_object (&value)); /* GtkApplicationWindow */
51      g_object_ref (win1);
52      g_print ("Got GtkApplicationWindow instance.\n");
53    }else
54      g_print ("The cclosure expression wasn't evaluated correctly.\n");
55    gtk_expression_unref (expression);
56    g_value_unset (&value);    /* At the same time, the reference count of win1 is
          decreased by one. */
57
58    /* Builds a window with components */
59    box = gtk_box_new (GTK_ORIENTATION_VERTICAL, 10);
60    label1 = gtk_label_new (NULL);
61    label2 = gtk_label_new (NULL);
62    label3 = gtk_label_new (NULL);
63    buffer = gtk_entry_buffer_new (NULL, 0);
64    entry = gtk_entry_new_with_buffer (buffer);
65    gtk_box_append (GTK_BOX (box), label1);
66    gtk_box_append (GTK_BOX (box), label2);
67    gtk_box_append (GTK_BOX (box), label3);
68    gtk_box_append (GTK_BOX (box), entry);
69    gtk_window_set_child (GTK_WINDOW (win1), box);
70
71    /* Constant expression */
72    expression = gtk_constant_expression_new (G_TYPE_INT,100);
73    if (gtk_expression_evaluate (expression, NULL, &value)) {
```

```
74      s = g_strdup_printf ("%d", g_value_get_int (&value));
75      gtk_label_set_text (GTK_LABEL (label1), s);
76      g_free (s);
77    } else
78      g_print ("The constant expression wasn't evaluated correctly.\n");
79    gtk_expression_unref (expression);
80    g_value_unset (&value);
81
82    /* Property expression and binding*/
83    expression1 = gtk_property_expression_new (GTK_TYPE_ENTRY, NULL, "buffer");
84    expression2 = gtk_property_expression_new (GTK_TYPE_ENTRY_BUFFER, expression1,
          "text");
85    gtk_expression_bind (expression2, label2, "label", entry);
86
87    /* Constant expression instead of "this" instance */
88    expression1 = gtk_constant_expression_new (GTK_TYPE_APPLICATION, app);
89    expression2 = gtk_property_expression_new (GTK_TYPE_APPLICATION, expression1,
          "application-id");
90    if (gtk_expression_evaluate (expression2, NULL, &value))
91      gtk_label_set_text (GTK_LABEL (label3), g_value_get_string (&value));
92    else
93      g_print ("The property expression wasn't evaluated correctly.\n");
94    gtk_expression_unref (expression1); /* expression 2 is also freed. */
95    g_value_unset (&value);
96
97    width = 800;
98    height = 600;
99    gtk_window_set_default_size (GTK_WINDOW (win1), width, height);
100   notify(NULL);
101
102   /* GtkExpressionWatch */
103   expression1 = gtk_property_expression_new (GTK_TYPE_WINDOW, NULL, "default-width");
104   watch_width = gtk_expression_watch (expression1, win1, notify, NULL, NULL);
105   expression2 = gtk_property_expression_new (GTK_TYPE_WINDOW, NULL,
          "default-height");
106   watch_height = gtk_expression_watch (expression2, win1, notify, NULL, NULL);
107
108   gtk_widget_show (win1);
109
110   /* Builds a window with exp.ui resource */
111   build = gtk_builder_new_from_resource ("/com/github/ToshioCP/exp/exp.ui");
112   GtkWidget *win2 = GTK_WIDGET (gtk_builder_get_object (build, "win2"));
113   gtk_window_set_application (GTK_WINDOW (win2), app);
114   g_object_unref (build);
115
116   gtk_widget_show (win2);
117 }
118
119 static void
120 app_startup (GApplication *application) {
121 }
122
123 #define APPLICATION_ID "com.github.ToshioCP.exp"
124
125 int
126 main (int argc, char **argv) {
127   GtkApplication *app;
128   int stat;
129
130   app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
131
132   g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
133   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
134 /*  g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);*/
```

```
135
136   stat =g_application_run (G_APPLICATION (app), argc, argv);
137   g_object_unref (app);
138   return stat;
139 }
```

`exp.c` consists of five functions.

- `notify`
- `set_title`
- `app_activate`. This is a handler of "activate" signal on GtkApplication instance.
- `app_startup`. This is a handler of "startup"signal. But nothing is done in this function.
- `main`.

The function `app_activate` is an actual main body in `exp.c`.

28.1 Constant expression

Constant expression provides constant value or instance when it is evaluated.

- 72-80: A constant expression. It is extracted and put into here.

```
expression = gtk_constant_expression_new (G_TYPE_INT,100);
if (gtk_expression_evaluate (expression, NULL, &value)) {
  s = g_strdup_printf ("%d", g_value_get_int (&value));
  gtk_label_set_text (GTK_LABEL (label1), s);
  g_free (s);
} else
  g_print ("The constant expression wasn't evaluated correctly.\n");
gtk_expression_unref (expression);
g_value_unset (&value);
```

- Constant expression is created with `gtk_constant_expression_new` function. The parameter of the function is a type (GType) and a value (or instance).
- `gtk_expression_evaluate` evaluates the expression. It has three parameters, the expression to evaluate, `this` instance and GValue for being set with the value. `this` instance isn't necessary for constant expressions. Therefore the second argument is NULL. `gtk_expression_evaluate` returns TRUE if it successfully evaluates the expression. Otherwise it returns FALSE.
- If it returns TRUE, the GValue `value` is set with the value of the expression. The type of the value is int. `g_strdup_printf` converts the value to a string `s`.
- GtkLabel `label1` is set with `s`. The string `s` needs to be freed.
- If the evaluation fails a message is outputted to stderr.
- The expression and GValue are freed.

Constant expression is usually used to give a constant value or instance to another expression.

28.2 Property expression

Property expression looks up a property in a GObject object. For example, a property expression that refers "label" property in GtkLabel object is created like this.

```
expression = gtk_property_expression_new (GTK_TYPE_LABEL, another_expression,
    "label");
```

`another_expression` is expected to give a GtkLabel instance when it is evaluated. For example,

```
label = gtk_label_new ("Hello");
another_expression = gtk_constant_expression_new (GTK_TYPE_LABEL, label);
expression = gtk_property_expression_new (GTK_TYPE_LABEL, another_expression,
    "label");
```

If `expression` is evaluated, the second parameter `another_expression` is evaluated in advance. The value of `another_expression` is `label` (GtkLabel instance). Then, `expression` looks up "label" property of `label` and the evaluation result is "Hello".

In the example above, the second argument of `gtk_property_expression_new` is another expression. But the second argument can be NULL. If it is NULL, `this` instance is used instead. `this` is given by `gtk_expression_evaluate` function at the evaluation.

Now look at `exp.c`. The lines from 83 to 85 is extracted here.

```
expression1 = gtk_property_expression_new (GTK_TYPE_ENTRY, NULL, "buffer");
expression2 = gtk_property_expression_new (GTK_TYPE_ENTRY_BUFFER, expression1,
    "text");
gtk_expression_bind (expression2, label2, "label", entry);
```

- `expression1` looks up "buffer" property of `this` object, which is `GTK_TYPE_ENTRY` type.
- `expression2` looks up "text" property of GtkEntryBuffer object given by `epression1`.
- `gtk_expression_bind` binds a property to a value given by the expression. In this program, it binds a "label" property in `label2` to the value evaluated with `expresion2` with `entry` as `this` object. The evaluation process is as follows.
  1. `expression2` is evaluated. But it includes `expression1` so `expression1` is evaluated in advance.
  2. Because the second argument of `expression1` is NULL, `this` object is used. `this` is given by `gtk_expression_bind`. It is `entry` (GtkEntry instance). `expression1` looks up "buffer" property in `entry`. It is a GtkEntryBuffer instance `buffer`. (See line 64 in `exp.c`.)
  3. Then, `expression2` looks up "text" property in `buffer`. It is a text held in `entry`.
  4. The text is assigned to "label" property in `label2`.
- `gtk_expression_bind` creates a GtkExpressionWatch. (But it isn't assigned to a variable in the program above. If you want to keep the GtkExpressionWatch instance, assign it to a variable.)

```
GtkExpressionWatch *watch;
watch = gtk_expression_bind (expression2, label2, "label", entry);
```

- Whenever the value from `expression2` changes, it evaluates `expression2` and set "label" property in `label2`. So, the change of the text in `entry` makes the "label" property reflect it immediately.

28.3 Closure expression

Closure expression calls closure when it is evaluated. A closure is a generic representation of a callback (a pointer to a function). For information about closure, see GObject API Reference, The GObject messaging system. A closure expression is created with `gtk_cclosure_expression_new` function.

```
GtkExpression *
gtk_cclosure_expression_new (GType value_type,
                             GClosureMarshal marshal,
                             guint n_params,
                             GtkExpression **params,
                             GCallback callback_func,
                             gpointer user_data,
                             GClosureNotify user_destroy);
```

- `value_type` is the type of the value when it is evaluated.
- `marshal` is a marshaller. You can assign NULL. If it is NULL, then `g_cclosure_marshal_generic ()` is used as a marshaller. It is a generic marshaller function implemented via libffi.
- `n_params` is the number of parameters.
- `params` points expressions for each parameter of the call back function.
- `callback_func` is a callback function.
- `user_data` is user data. You can add it for the closure. It is like `user_data` in `g_signal_connect`. If it is not necessary, assign NULL.
- `user_destroy` is a destroy notify for `user_data`. It is called to destroy `user_data` when it is no longer needed. If NULL is assigned to `user_data`, assign NULL to `user_destroy`, too.

The following is extracted from `exp.c`. It is from line 47 to line 56.

```
expression = gtk_cclosure_expression_new (GTK_TYPE_APPLICATION_WINDOW, NULL, 0, NULL,
                G_CALLBACK (gtk_application_window_new), NULL, NULL);
if (gtk_expression_evaluate (expression, app, &value)) {
  win1 = GTK_WIDGET (g_value_get_object (&value)); /* GtkApplicationWindow */
  g_object_ref (win1);
```

```
  g_print ("Got GtkApplicationWindow object.\n");
}else
  g_print ("The cclosure expression wasn't evaluated correctly.\n");
gtk_expression_unref (expression);
g_value_unset (&value);    /* At the same time, the reference count of win1 is
    decreased by one. */
```

The callback function is `gtk_application_window_new`. This function has one parameter which is an instance of GtkApplication. And it returns newly created GtkApplicationWindow instance. So, the first argument is `GTK_TYPE_APPLICATION_WINDOW` which is the type of the return value. The second argument is NULL so general marshaller `g_cclosure_marshal_generic ()` will be used. I think assigning NULL works in most cases when you program in C language.

The arguments given to the call back function are `this` object and parameters which are the fourth argument of `gtk_cclosure_expression_new`. So, the number of arguments is `n_params + 1`. Because `gtk_application_window_new` has one parameter, so `n_params` is zero and `**params` is NULL. No user data is necessary, so `user_data` and `user_destroy` are NULL.

`gtk_expression_evaluate` evaluates the expression. `this` instance will be the first argument for `gtk_application_window_new`, so it is `app`.

If the evaluation succeeds, the GValue `value` holds a newly created GtkApplicationWindow instance. It is assigned to `win1`. The GValue will be unset when it is no longer used. And when it is unset, the GtkApplicationWindow instance will be released and its reference count will be decreased by one. It is necessary to increase the reference count by one in advance to keep the instance. `gtk_expression_unref` frees `expression` and `value` is unset.

As a result, we got a GtkApplicationWindow instance `win1`. We can do the same by:

```
win1 = gtk_application_window_new (app);
```

The example is more complicated and not practical than this one line code. The aim of the example is just to show how closure expression works.

Closure expression is flexible than other type of expression because you can specify your own callback function.

28.4 GtkExpressionWatch

GtkExpressionWatch watches an expression and if the value of the expression changes it calls its notify handler.

The example uses GtkExpressionWatch in the line 103 to 106.

```
expression1 = gtk_property_expression_new (GTK_TYPE_WINDOW, NULL, "default-width");
watch_width = gtk_expression_watch (expression1, win1, notify, NULL, NULL);
expression2 = gtk_property_expression_new (GTK_TYPE_WINDOW, NULL, "default-height");
watch_height = gtk_expression_watch (expression2, win1, notify, NULL, NULL);
```

The expressions above refer to “default-width” and “default-height” properties of GtkWindow. The variable `watch_width` watches `expression1`. The second argument `win1` is `this` instance for `expression1`. So, `watch_width` watches the value of “default-width” property of `win1`. If the value changes, it calls `notify` handler. The fourth and fifth arguments are NULL because no user data is necessary.

The variable `watch_height` connects `notify` handler to `expression2`. So, `notiry` is also called when “default-height” changes.

The handler `norify` is as follows.

```
1 static void
2 notify (gpointer user_data) {
3   GValue value = G_VALUE_INIT;
4   char *title;
5
6   if (watch_width && gtk_expression_watch_evaluate (watch_width, &value))
7     width = g_value_get_int (&value);
```

```
8     g_value_unset (&value);
9     if (watch_height && gtk_expression_watch_evaluate (watch_height, &value))
10      height = g_value_get_int (&value);
11    g_value_unset (&value);
12    title = g_strdup_printf ("%d x %d", width, height);
13    gtk_window_set_title (GTK_WINDOW (win1), title);
14    g_free (title);
15  }
```

- 6-11: Evaluates `expression1` and `expression2` with `expression_watch_evaluate` function.
- 12: Creates a string `title`. It contains the width and height, for example, “800 x 600”.
- 13: Sets the title of `win1` with the string `title`.

The title of the window reflects the size of the window.

28.5 exp.ui

`exp.c` builds a GtkWindow instance `win2` with `exp.ui`. The ui file `exp.ui` includes tags to create GtkExpressions. The tags are:

- constant tag to create constant expression
- lookup tag to create property expression
- closure tag to create closure expression
- binding tag to bind a property to an expression

The window `win2` behaves like `win1`. Because similar expressions are built with the ui file.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <interface>
3    <object class="GtkWindow" id="win2">
4      <binding name="title">
5        <closure type="gchararray" function="set_title">
6          <lookup name="default-width" type="GtkWindow"></lookup>
7          <lookup name="default-height" type="GtkWindow"></lookup>
8        </closure>
9      </binding>
10     <property name="default-width">600</property>
11     <property name="default-height">400</property>
12     <child>
13       <object class="GtkBox">
14         <property name="orientation">GTK_ORIENTATION_VERTICAL</property>
15         <child>
16           <object class="GtkLabel">
17             <binding name="label">
18               <constant type="gint">100</constant>
19             </binding>
20           </object>
21         </child>
22         <child>
23           <object class="GtkLabel">
24             <binding name="label">
25               <lookup name="text">
26                 <lookup name="buffer">
27                   entry
28                 </lookup>
29               </lookup>
30             </binding>
31           </object>
32         </child>
33         <child>
34           <object class="GtkLabel">
35             <binding name="label">
36               <lookup name="application-id">
37                 <lookup name="application">win2</lookup>
38               </lookup>
```

```
39              </binding>
40            </object>
41          </child>
42          <child>
43            <object class="GtkEntry" id="entry">
44              <property name="buffer">
45                <object class="GtkEntryBuffer"></object>
46              </property>
47            </object>
48          </child>
49        </object>
50      </child>
51    </object>
52  </interface>
```

28.5.1 Constant tag

A constant tag corresponds to a constant expression.

- 18: Constant tag. The constant expression is created with the tag. It returns 100, the type is "gint", when it is evaluated. The type "gint" is a name of `G_TYPE_INT` type. Similarly, the types which is registered to the type system has type and name. For example, "gchararray" is a name of `G_TYPE_STRING` type. You need to use the name of types for the `type`attribute. See GObject tutorial.
- 17-19: Binding tag corresponds to `gtk_expression_bind` function. `name` attribute specifies the "label" property of the GtkLabel object just before the binding tag. The expression returns a int type GValue. On the other hand "label" property holds a string type GValue. When a GValue is copied to another GValue, the type is automatically converted if possible. In this case, an int `100` is converted to a string `"100"`.

These binding and constant tag works. But they are not good. A property tag is more straightforward.

```
<object class="GtkLabel">
  <property name="label">100</property>
</object>
```

This example just shows the way how to use constant tag. Constant tag is mainly used to give a constant argument to a closure.

28.5.2 Lookup tag

A lookup tag corresponds to a property expression. Line 23 to 31 is copied here.

```
<object class="GtkLabel">
  <binding name="label">
    <lookup name="text">
      <lookup name="buffer">
        entry
      </lookup>
    </lookup>
  </binding>
</object>
```

- binding tag binds a "label" property in GtkLabel to an expression. The expression is defined with a lookup tag.
- The lookup tag defines a property expression looks up a "text" property in the instance which is defined in the next expression. The next expression is created with the lookup tag. The expression looks up the `buffer` property of the `entry` instance. The `entry` instance is defined in the line 43. It is a GtkEntry `entry`. A lookup tag takes an instance in some ways to look up for a property.
  - If it has no contents, it takes `this` instance when it is evaluated.
  - If it has a content of a tag for an expression, which is constant, lookup or closure tag, the value of the expression will be the instance to look up when it is evaluated.
  - If it has a content of an id of an object, then the instance of the object will be taken as the instance to lookup.

As a result, the label of the GtkLabel instance are bound to the text in the field of GtkEntry. If a user input a text in the field in the GtkEntry, GtkLabel displays the same text.

Another lookup tag is in the lines from 34 to 40.

```
<object class="GtkLabel">
  <binding name="label">
    <lookup name="application-id">
      <lookup name="application">win2</lookup>
    </lookup>
  </binding>
</object>
```

- Two expressions are nested.
- A lookup tag looks up "application-id" property of the next expression.
- The next lookup tag looks up "application" property of `win2` instance.

As a result, the "label" property in the GtkLabel instance is bound to the "application-id" property. The nested tag makes a chain like:

```
"label" <= "application-id" <= "application" <= `win2`
```

By the way, the application of `win2` is set after the objects in ui file are built. Look at `exp.c`. `gtk_window_set_application` is called after `gtk_build_new_from_resource`.

```
build = gtk_builder_new_from_resource ("/com/github/ToshioCP/exp/exp.ui");
GtkWidget *win2 = GTK_WIDGET (gtk_builder_get_object (build, "win2"));
gtk_window_set_application (GTK_WINDOW (win2), app);
```

Therefore, before the call for `gtk_window_set_application`, the "application" property of `win2` is *not* set. So, the evaluation of `<lookup name="application">win2</lookup>` fails. And the evaluation of `<lookup name="application-id">` also fails. A function `gtk_expression_bind ()`, which corresponds to `binding` tag, doesn't update the target property if the expression fails. So, the "label" property isn't updated at the first evaluation.

Note that an evaluation can fail. The care is especially necessary when you write a callback for a closure tag which has contents of expressions like lookup tags. The expressions are given to the callback as an argument. If an expression fails the argument will be NULL. You need to check if the argument exactly points the instance that is expected by the callback.

28.5.3 Closure tag

The lines from 3 to 9 include a closure tag.

```
<object class="GtkWindow" id="win2">
  <binding name="title">
    <closure type="gchararray" function="set_title">
      <lookup name="default-width" type="GtkWindow"></lookup>
      <lookup name="default-height" type="GtkWindow"></lookup>
    </closure>
  </binding>
```

- A binding tag corresponds to a `gtk_expression_bind` function. `name` attribute specifies the "title" property of `win2`. Binding tag gives `win2` as the `this` instance to the expressions, which are the contents of the binding tag. So, closure tag and lookup tags use `win2` as the `this` object when they are evaluated.
- A closure tag corresponds to a closure expression. Its callback function is `set_title` and it returns "gchararray" type, which is "an array of characters" i.e. a string. The contents of the closure tag are assigned to parameters of the function. So, `set_title` has three parameters, `win2` (`this` instance), default width and default height.
- Lookup tags correspond to property expressions. They lookup "default-width" and "default-height" properties of `win2` (`this` instance).
- Binding tab creates GtkExpressionWatch automatically, so "title" property reflects the changes of "default-width" and "default-height" properties.

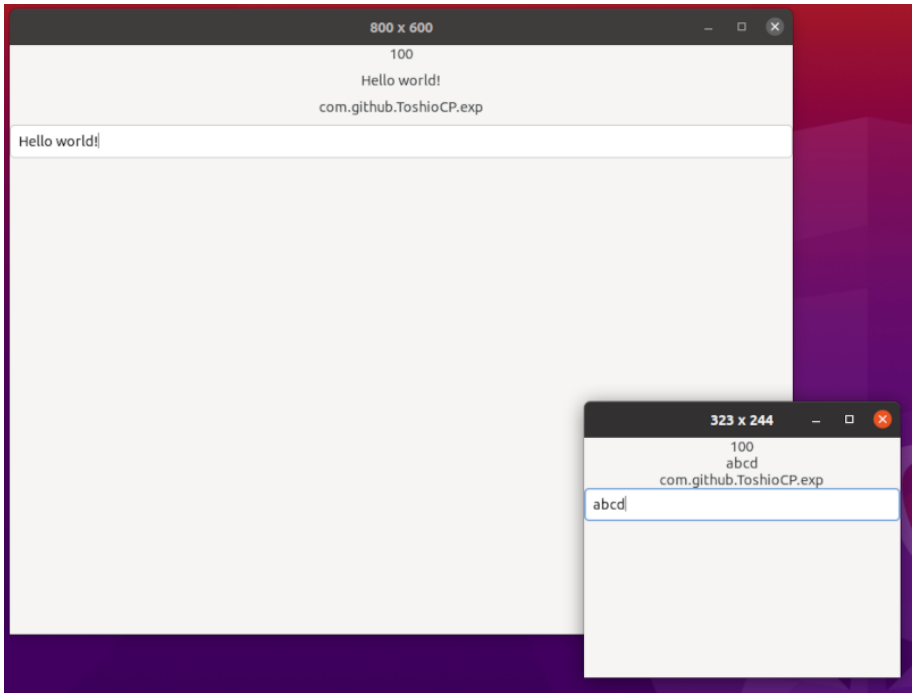

Figure 50: Expression

`set_title` function in `exp.c` is as follows.

```
1  char *
2  set_title (GtkWidget *win, int width, int height) {
3    return g_strdup_printf ("%d x %d", width, height);
4  }
```

It just creates a string, for example, “800 x 600”, and returns it.

You’ve probably been noticed that ui file is easier and clearer than the corresponding C program. One of the most useful case of GtkExpression is building GtkListItem instance with GtkBuilderListItemFatory. Such case has already been described in the prior two sections.

It will be used in the next section to build GtkListItem in GtkColumnView, which is the most useful view object for GListModel.

28.6 Compilation and execution

All the sources are in src/expression directory. Change your current directory to the directory and run meson and ninja. Then, execute the application.

```
$ meson _build
$ ninja -C _build
$ build/exp
```

Then, two windows appear.

If you put some text in the field of the entry, then the same text appears in the second GtkLabel. Because the “label” property of the second GtkLabel instance is bound to the text in the GtkEntryBuffer.

If you resize the window, then the size appears in the title bar because the “title” property is bound to “default-width” and “default-height” properties.

29 GtkColumnView

29.1 GtkColumnView

GtkColumnView is like GtkListView, but it has multiple columns. Each column is GtkColumnViewColumn.

Figure 51: Column View

- GtkColumnView has “model” property. The property points a GtkSelectionModel object.
- Each GtkColumnViewColumn has “factory” property. The property points a GtkListItemFactory (GtkSignalListItemFactory or GtkBuilderListItemFactory).
- The factory connects GtkListItem, which belongs to GtkColumnViewColumn, and items of GtkSelectionModel. And the factory builds the descendants widgets of GtkColumnView to display the item on the display. This process is the same as the one in GtkListView.

The following diagram shows the image how it works.

The example in this section is a window that displays information of files in a current directory. The information is the name, size and last modified datetime of files. So, there are three columns.

In addition, the example uses GtkSortListModel and GtkSorter to sort the information.

29.2 column.ui

Ui file specifies whole widgets and their structure.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <interface>
3    <object class="GtkApplicationWindow" id="win">
4      <property name="title">file list</property>
5      <property name="default-width">800</property>
6      <property name="default-height">600</property>
7      <child>
8        <object class="GtkScrolledWindow" id="scr">
9          <property name="hexpand">TRUE</property>
10         <property name="vexpand">TRUE</property>
11         <child>
12           <object class="GtkColumnView" id="columnview">
13             <property name="model">
14               <object class="GtkSingleSelection" id="singleselection">
15                 <property name="model">
16                   <object class="GtkSortListModel" id="sortlist">
17                     <property name="model">
18                       <object class="GtkDirectoryList" id="directorylist">
19                         <property
                               name="attributes">standard::name,standard::icon,standard::size,time::m
```

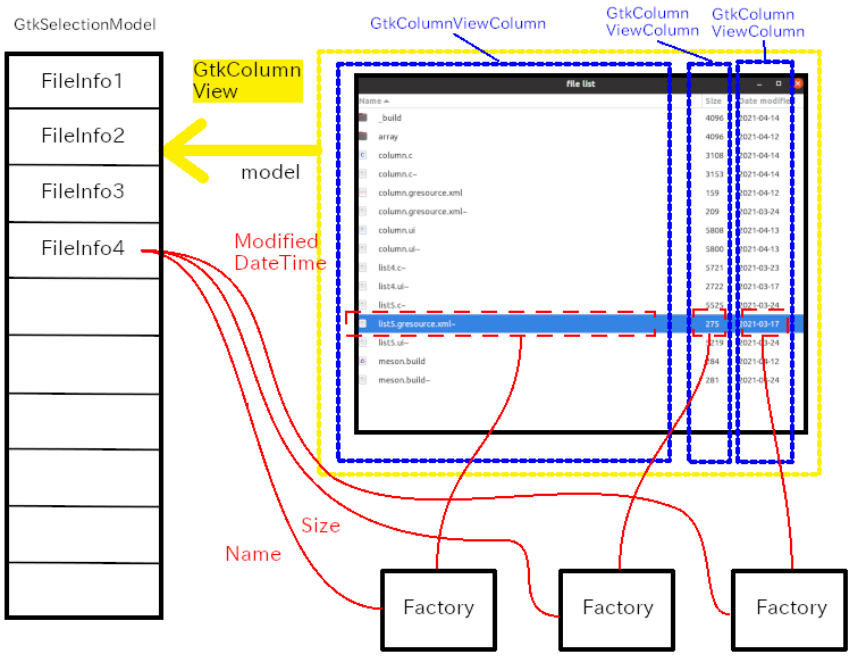

Figure 52: ColumnView

```
20                      </object>
21                    </property>
22                    <binding name="sorter">
23                      <lookup name="sorter">columnview</lookup>
24                    </binding>
25                  </object>
26                </property>
27              </object>
28            </property>
29            <child>
30              <object class="GtkColumnViewColumn" id="column1">
31                <property name="title">Name</property>
32                <property name="expand">TRUE</property>
33                <property name="factory">
34                  <object class="GtkBuilderListItemFactory">
35                    <property name="bytes"><![CDATA[
36  <?xml version="1.0" encoding="UTF-8"?>
37  <interface>
38    <template class="GtkListItem">
39      <property name="child">
40        <object class="GtkBox">
41          <property name="orientation">GTK_ORIENTATION_HORIZONTAL</property>
42          <property name="spacing">20</property>
43          <child>
44            <object class="GtkImage">
45              <binding name="gicon">
46                <closure type="GIcon" function="get_icon_factory">
47                  <lookup name="item">GtkListItem</lookup>
48                </closure>
49              </binding>
50            </object>
51          </child>
52          <child>
53            <object class="GtkLabel">
54              <property name="hexpand">TRUE</property>
55              <property name="xalign">0</property>
56              <binding name="label">
```

```
              <closure type="gchararray" function="get_file_name_factory">
                <lookup name="item">GtkListItem</lookup>
              </closure>
            </binding>
          </object>
        </child>
      </object>
    </property>
  </template>
</interface>
                    ]]></property>
                  </object>
                </property>
                <property name="sorter">
                  <object class="GtkStringSorter" id="sorter_name">
                    <property name="expression">
                      <closure type="gchararray" function="get_file_name">
                      </closure>
                    </property>
                  </object>
                </property>
              </object>
            </child>
            <child>
              <object class="GtkColumnViewColumn" id="column2">
                <property name="title">Size</property>
                <property name="factory">
                  <object class="GtkBuilderListItemFactory">
                    <property name="bytes"><![CDATA[
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <template class="GtkListItem">
    <property name="child">
      <object class="GtkLabel">
        <property name="hexpand">TRUE</property>
        <property name="xalign">0</property>
        <binding name="label">
          <closure type="gchararray" function="get_file_size_factory">
            <lookup name="item">GtkListItem</lookup>
          </closure>
        </binding>
      </object>
    </property>
  </template>
</interface>
                    ]]></property>
                  </object>
                </property>
                <property name="sorter">
                  <object class="GtkNumericSorter" id="sorter_size">
                    <property name="expression">
                      <closure type="gint64" function="get_file_size">
                      </closure>
                    </property>
                    <property name="sort-order">GTK_SORT_ASCENDING</property>
                  </object>
                </property>
              </object>
            </child>
            <child>
              <object class="GtkColumnViewColumn" id="column3">
                <property name="title">Date modified</property>
                <property name="factory">
                  <object class="GtkBuilderListItemFactory">
```

```
121                     <property name="bytes"><![CDATA[
122 <?xml version="1.0" encoding="UTF-8"?>
123 <interface>
124   <template class="GtkListItem">
125     <property name="child">
126       <object class="GtkLabel">
127         <property name="hexpand">TRUE</property>
128         <property name="xalign">0</property>
129         <binding name="label">
130           <closure type="gchararray" function="get_file_time_modified_factory">
131             <lookup name="item">GtkListItem</lookup>
132           </closure>
133         </binding>
134       </object>
135     </property>
136   </template>
137 </interface>
138                     ]]></property>
139                   </object>
140                 </property>
141                 <property name="sorter">
142                   <object class="GtkNumericSorter" id="sorter_datetime_modified">
143                     <property name="expression">
144                       <closure type="gint64" function="get_file_unixtime_modified">
145                       </closure>
146                     </property>
147                     <property name="sort-order">GTK_SORT_ASCENDING</property>
148                   </object>
149                 </property>
150               </object>
151             </child>
152           </object>
153         </child>
154       </object>
155     </child>
156   </object>
157 </interface>
```

- 3-12: Widget parent-child relationship is GtkApplicationWindow => GtkScrolledWindow => GtkColumnView.
- 12-18: GtkColumnView has “model” property. It points GtkSelectionModel interface. In this ui file, GtkSingleSelection is used as GtkSelectionModel. GtkSingleSelection is an object that implements GtkSelectionModel. And again, it has “model” property. It points GtkSortListModel. This list model supports sorting the list. It will be explained in the later subsection. And it also has “model” property. It points GtkDirectoryList. Therefore, the chain is: GtkColumnView => GtkSingleSelection => GtkSortListModel => GtkDirectoryList.
- 18-20: GtkDirectoryList. It is a list of GFileInfo, which holds information of files under a directory. It has “attributes” property. It specifies what attributes is kept in each GFileInfo.
  - “standard::name” is a name of the file.
  - “standard::icon” is a GIcon object of the file
  - “standard::size” is the file size.
  - “time::modified” is the date and time the file was last modified.
- 29-79: The first GtkColumnViewColumn object. There are four properties, “title”, “expand”, factory” and “sorter”.
- 31: Sets the “title” property with “Name”. This is the title on the header of the column.
- 32: Sets the “expand” property to TRUE to allow the column to expand as much as possible. (See the image above).
- 33- 69: Sets the “factory” property with GtkBuilderListItemFactory. The factory has “bytes” property which holds a ui string to define a template to build GtkListItem composite widget. The CDATA section (line 36-66) is the ui string to put into the “bytes” property. The contents are the same as the ui file `factory_list.ui` in the section 27.
- 70-77: Sets the “sorter” property with GtkStringSorter object. This object provides a sorter that

compares strings. It has “expression” property which is set with GtkExpression. A closure tag with a string type function `get_file_name` is used here. The function will be explained later.

- 80-115: The second GtkColumnViewColumn object. Its “title”, “factory” and “sorter” properties are set. GtkNumericSorter is used.
- 116-151: The third GtkColumnViewColumn object. Its “title”, “factory” and “sorter” properties are set. GtkNumericSorter is used.

29.3 GtkSortListModel and GtkSorter

GtkSortListModel is a list model that sorts its elements according to a GtkSorter. It has “sorter” property that is set with GtkSorter. The property is bound to “sorter” property of GtkColumnView in line 22 to 24.

```
<object class="GtkSortListModel" id="sortlist">
... ... ...
  <binding name="sorter">
    <lookup name="sorter">columnview</lookup>
  </binding>
```

Therefore, `columnview` determines the way how to sort the list model. The “sorter” property of GtkColumnView is read-only property and it is a special sorter. It reflects the user’s sorting choice. If a user clicks the header of a column, then the sorter (“sorter” property) of the column is referenced by “sorter” property of the GtkColumnView. If the user clicks the header of another column, then the “sorter” property of the GtkColumnView refers to the newly clicked column’s “sorter” property.

The binding above makes a indirect connection between the “sorter” property of GtkSortListModel and the “sorter” property of each column.

GtkSorter has several child objects.

- GtkStringSorter compares strings.
- GtkNumericSorter compares numbers.
- GtkCustomSorter uses a callback to compare.
- GtkMultiSorter combines multiple sorters.

The example uses GtkStringSorter and GtkNumericSorter.

GtkStringSorter uses GtkExpression to get the strings from the objects. The GtkExpression is stored in the “expression” property of GtkStringSorter. For example, in the ui file above, the GtkExpression is in the line 71 to 76.

```
<object class="GtkStringSorter" id="sorter_name">
  <property name="expression">
    <closure type="gchararray" function="get_file_name">
    </closure>
  </property>
</object>
```

The GtkExpression calls `get_file_name` function when it is evaluated.

```
1 char *
2 get_file_name (GFileInfo *info) {
3   g_return_val_if_fail (G_IS_FILE_INFO (info), NULL);
4
5   return g_strdup(g_file_info_get_name (info));
6 }
```

The function is given the item (GFileInfo) of the GtkSortListModel as an argument (`this` object). The function retrieves a filename from `info`. The string is owned by `info` so it is necessary to duplicate it. And it returns the copied string. The string will be owned by the expression.

GtkNumericSorter compares numbers. It is used in the line 106 to 112 and line 142 to 148. The lines from 106 to 112 is:

```
<object class="GtkNumericSorter" id="sorter_size">
  <property name="expression">
    <closure type="gint64" function="get_file_size">
```

```
    </closure>
  </property>
  <property name="sort-order">GTK_SORT_ASCENDING</property>
</object>
```

The closure tag specifies a callback function `get_file_size`.

```
goffset
get_file_size (GFileInfo *info) {
  g_return_val_if_fail (G_IS_FILE_INFO (info), -1);

  return g_file_info_get_size (info);
}
```

It just returns the size of `info`. The type of the size is `goffset`. The type `goffset` is the same as `gint64`.

The lines from 142 to 148 is:

```
<object class="GtkNumericSorter" id="sorter_datetime_modified">
  <property name="expression">
    <closure type="gint64" function="get_file_unixtime_modified">
    </closure>
  </property>
  <property name="sort-order">GTK_SORT_ASCENDING</property>
</object>
```

The closure tag specifies a callback function `get_file_unixtime_modified`.

```
gint64
get_file_unixtime_modified (GFileInfo *info) {
  g_return_val_if_fail (G_IS_FILE_INFO (info), -1);

  GDateTime *dt;

  dt = g_file_info_get_modification_date_time (info);
  return g_date_time_to_unix (dt);
}
```

It gets the modification date and time (GDateTime type) of `info`. Then it gets a unix time from `dt`. Unix time, sometimes called unix epoch, is the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. It returns the unix time (gint64 type).

29.4 column.c

`column.c` is as follows.

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

/* functions (closures) for GtkBuilderListItemFactory */
GIcon *
get_icon_factory (GtkListItem *item, GFileInfo *info) {
  GIcon *icon;
  if (! G_IS_FILE_INFO (info))
    return NULL;
  else {
    icon = g_file_info_get_icon (info);
    g_object_ref (icon);
    return icon;
  }
}

char *
get_file_name_factory (GtkListItem *item, GFileInfo *info) {
  if (! G_IS_FILE_INFO (info))
    return NULL;
```

```
20    else
21      return g_strdup (g_file_info_get_name (info));
22  }
23
24  char *
25  get_file_size_factory (GtkListItem *item, GFileInfo *info) {
26    /* goffset is gint64 */
27    goffset size;
28
29    if (! G_IS_FILE_INFO (info))
30      return NULL;
31    else {
32      size = g_file_info_get_size (info);
33      return g_strdup_printf ("%ld", (long int) size);
34    }
35  }
36
37  char *
38  get_file_time_modified_factory (GtkListItem *item, GFileInfo *info) {
39    GDateTime *dt;
40
41    if (! G_IS_FILE_INFO (info))
42      return NULL;
43    else {
44      dt = g_file_info_get_modification_date_time (info);
45      return g_date_time_format (dt, "%F");
46    }
47  }
48
49  /* Functions (closures) for GtkSorter */
50  char *
51  get_file_name (GFileInfo *info) {
52    g_return_val_if_fail (G_IS_FILE_INFO (info), NULL);
53
54    return g_strdup(g_file_info_get_name (info));
55  }
56
57  goffset
58  get_file_size (GFileInfo *info) {
59    g_return_val_if_fail (G_IS_FILE_INFO (info), -1);
60
61    return g_file_info_get_size (info);
62  }
63
64  gint64
65  get_file_unixtime_modified (GFileInfo *info) {
66    g_return_val_if_fail (G_IS_FILE_INFO (info), -1);
67
68    GDateTime *dt;
69
70    dt = g_file_info_get_modification_date_time (info);
71    return g_date_time_to_unix (dt);
72  }
73
74  /* ----- activate, open, startup handlers ----- */
75  static void
76  app_activate (GApplication *application) {
77    GtkApplication *app = GTK_APPLICATION (application);
78    GFile *file;
79
80    GtkBuilder *build = gtk_builder_new_from_resource
          ("/com/github/ToshioCP/column/column.ui");
81    GtkWidget *win = GTK_WIDGET (gtk_builder_get_object (build, "win"));
```

```
82    GtkDirectoryList *directorylist = GTK_DIRECTORY_LIST (gtk_builder_get_object
          (build, "directorylist"));
83    g_object_unref (build);
84
85    gtk_window_set_application (GTK_WINDOW (win), app);
86
87    file = g_file_new_for_path (".");
88    gtk_directory_list_set_file (directorylist, file);
89    g_object_unref (file);
90
91    gtk_widget_show (win);
92  }
93
94  static void
95  app_startup (GApplication *application) {
96  }
97
98  #define APPLICATION_ID "com.github.ToshioCP.columnview"
99
100 int
101 main (int argc, char **argv) {
102   GtkApplication *app;
103   int stat;
104
105   app = gtk_application_new (APPLICATION_ID, G_APPLICATION_FLAGS_NONE);
106
107   g_signal_connect (app, "startup", G_CALLBACK (app_startup), NULL);
108   g_signal_connect (app, "activate", G_CALLBACK (app_activate), NULL);
109 /*  g_signal_connect (app, "open", G_CALLBACK (app_open), NULL);*/
110
111   stat =g_application_run (G_APPLICATION (app), argc, argv);
112   g_object_unref (app);
113   return stat;
114 }
```

- 4-47: Functions for the closure tag in the "bytes" property of GtkBuilderListItemFactory. These are almost same as the functions in section 26 and 26.
- 50-72: Functions for the closure in the expression property of GtkStringSorter or GtkNumericSorter.
- 75-92: `app_activate` is an "activate" handler of GApplication.
- 80-83: Builds objects with ui resource and gets `win` and `directorylist`.
- 85: Sets the application of the top level window with `app`.
- 87-89: Sets the file of `directorylist` with "." (current directory).
- 94-96: Startup handler.
- 98-114: `main` function.

`exp.c` is simple and short thanks to `exp.ui`.

29.5 Compilation and execution.

All the source files are in src/column directory. Change your current directory to the directory and type the following.

```
$ meson _build
$ ninja -C _build
$ _build/column
```

Then, a window appears.

If you click the header of a column, then the whole lists are sorted by the column. If you click the header of another column, then the whole lists are sorted by the newly selected column.

GtkColumnView is very useful and it can manage very big GListModel. It is possible to use it for file list, application list, database frontend and so on.

Figure 53: Column View

A Turtle's manual

Turtle is a simple interpreter for turtle graphics.

A.1 Prerequisite and compiling

Turtle is written in C language. You need:

- Linux. Turtle is tested on ubuntu 20.10
- gcc, meson and ninja
- gtk4

It is easy to compile the source file of turtle. If you have installed gtk4 with an option `--prefix=$HOME/local`, put the same option to meson so that you can install `turtle` under the directory `$HOME/local/bin`. The instruction is:

```
$ meson --prefix=$HOME/local _build
$ ninja -C _build
$ ninja -C _build install
```

Type the following command then turtle shows the following window.

```
$ turtle
```

The left half is a text editor and the right half is a surface. Surface is like a canvas to draw shapes.

Write turtle language in the text editor and click on `run` button, then the program will be executed and it draws shapes on the surface.

If you add the following line in `turtle.h`, then codes to inform the status will also be compiled. However, the speed will be quite slow because of the output messages.

```
# define debug 1
```

A.2 Example

Imagine a turtle. The turtle has a pen and initially he is at the center of the screen, facing to the north (to the north means up on the screen). You can let the turtle down the pen or up the pen. You can order the turtle to move forward.

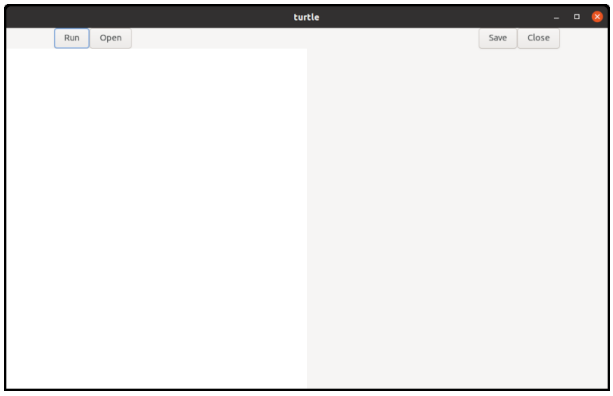

Figure 54: Screenshot just after it's executed

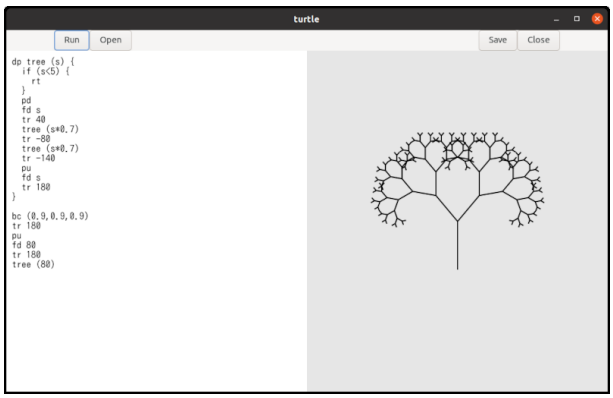

Figure 55: Tree

```
pd
fd 100
```

- pd: Pen Down. The turtle put the pen down so that the turtle will draw a line if he/she moves.
- fd 100: move ForwarD 100. The turtle goes forward 100 pixels.

If you click on `run` button, then a line segment appears on the screen. One of the endpoints of the line segment is at the center of the surface and the other is at 100 pixels up from the center. The point at the center is the start point of the turtle and the other endpoint is the end point of the movement.

If the turtle picks the pen up, then no line segment appears.

```
pu
fd 100
```

The command `pu` means "Pen Up".

The turtle can change the direction.

```
pd
fd 100
tr 90
fd 100
```

The command `tr` is "Turn Right". The argument is angle with degrees. Therefore, `tr 90` means "Turn right by 90 degrees". If you click on the `run`button, then two line segments appears. One is vertical and the other is horizontal.

A.3 Background and foreground color

Colors are specified with RGB. A vector (r, g, b) denotes RGB color. Each of the elements is a real number between 0 and 1.

Figure 56: Two line segments on the surface

Figure 57: Change the foreground color

- Red is (1.0, 0.0, 0.0). You can write (1, 0, 0) instead.
- Green is (0.0, 1.0, 0.0)
- Blue is (0.0, 0.0, 1.0)
- Black is (0.0, 0.0, 0.0)
- White is (1.0, 1.0, 1.0)

You can express a variety of colors by changing each element.

There are two commands to change colors.

- bc: Background Color. `bc (1,0,0)` changes the background color to red. This command clear the surface and change the background color. So, the shapes on the surface disappears.
- fc: Foreground Color. `fc (0,1,0)` changes the foreground color to green. This command changes the pen color. The prior shapes on the surface aren't affected. After this command, the turtle draws lines with the new color.

A.4 Other simple commands

- pw: Pen Width. This is the same as pen size or line width. For example, `pw 5` makes lines thick and `pw 1` makes it thin.
- rs: ReSet. The turtle moves back to the initial position and direction. In addition, The command initialize the pen, line width (pen size), and foreground color. The pen is down, the line width is 2 and the foreground color is black.

An order such as `fd 100`, `pd` and so on is a statement. Statements are executed in the order from the top to the end

A.5 Comment and spaces

Characters between `#` (hash mark) and `\n` (new line) inclusive are comment. If the comment is at the end of the file, the trailing new line can be left out. Comments are ignored.

```
# draw a triangle
fd 100 # forward 100 pixels<NEW LINE>
tr 120 # turn right by 90 degrees<NEW LINE>
fd 100<NEW LINE>
tr 120<NEW LINE>
fd 100 # Now a triangle appears.<EOF>
```

<NEW LINE> and <EOF> indicate newline code and end of file respectively. The comments in the line 1, 2, 3 and 6 are correct syntactically.

Spaces (white space, tab and new line) are ignored. They are used only as delimiters. Tabs are recognized as eight spaces to calculate the column number.

A.6 Variables and expressions

Variable begins alphabet followed by alphabet or digit. Key words like `fd`, `tr` can't be variables. `Distance` and `angle5` are variables, but `1step` isn't a variable because the first character isn't alphabet. Variable names are case sensitive. Variables keep real numbers. Their type is the same as `double` in C language. Integers are casted to real numbers automatically. So 1 and 1.0 are the same value. Numbers begin digits, not signs (`+` or `-`).

- 100, 20.34 and 0.01 are numbers
- +100 isn't a number. It causes syntax error. Use 100 instead.
- -100 isn't a number. But turtle recognizes it unary minus and a number 100. So turtle calculate it and the result is -100.
- 100 + -20: This is recognized 100 + (- 20). However, using bracket, 100 + (-20), is better for easy reading.

```
distance = 100
fd distance
```

A value 100 is assigned to the variable `distance` in the first line. Assignment is a statement. Most of statements begin with commands like `fd`. Assignment is the only exception.

The example above draws a line segment of 100 pixels long.

You can use variables in expressions. There are 8 kinds of calculations available.

- addition: x + y
- subtraction: x - y
- multiplication: x * y
- division: x / y
- unary minus: - x
- logical equal: x = y. This symbol `=` works as `==` in C language.
- greater than: x > y
- less than: x < y

The last three symbols are mainly used in the condition of if statement.

Variables are registered to a symbol table when it is assigned a value for the first time. Evaluating a variable before the registration isn't allowed and occurs an error.

A.7 If statement

Turtle language has very simple if statement.

```
if (x > 50) {
  fd x
}
```

There is no else part.

A.8 Procedures

Procedures are similar to functions in C language. The difference is that procedures don't have return values.

```
dp triangle (side) {
  fd side
  tr 120
  fd side
  tr 120
  fd side
}

triangle (100)
```

`dp` (Define Procedure) is a key word followed by procedure name, parameters, and body. Procedure names start alphabet followed by alphabet or digit. Parameters are a list of variables. For example

```
dp abc (a) { ... ... }
dp abc (a, b) { ... ... }
dp abc (a, b, c) { ... ... }
```

Body is a sequence of statements. The statements aren't executed when the procedure is defined. They will be executed when the procedure is called later.

Procedures are called by the name followed by arguments.

```
dp proc (a, b, c) { ... ... }

proc (100, 0, -20*3)
```

The number of parameters and arguments must be the same. Arguments can be any expressions. When you call a procedure, brackets following the procedure name must exist even if the procedure has no argument.

Procedure names and variable names don't conflict.

```
dp a () {fd a}
a=100
a ()
```

This is a correct program.

- 1: Defines a procedure `a`. A variable `a` is in its body.
- 2: Assigns 100 to a variable `a`.
- 3: Procedure `a` is called.

However, using the same name to a procedure and variable makes confusing. You should avoid that.

A.9 Recursive call

Procedures can be called recursively.

```
dp repeat (n) {
  n = n - 1
  if (n < 0) {
    rt
  }
  fd 100
  tr 90
  repeat (n)
}

repeat (4)
```

Repeat is called in the body of repeat. The call to itself is a recursive call. Parameters are created and set each time the procedure is called. So, parameter `n` is 4 at the first call but it is 3 at the second call. Each

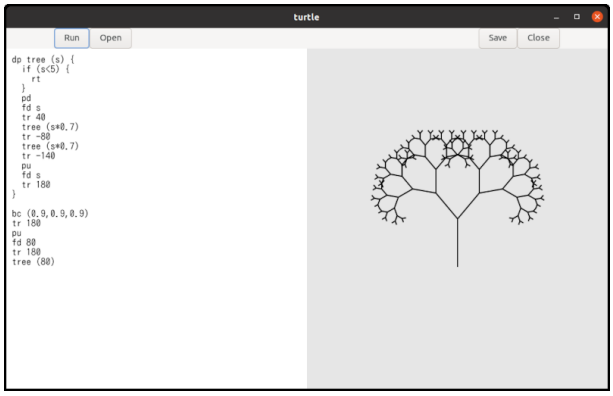

Figure 58: Tree

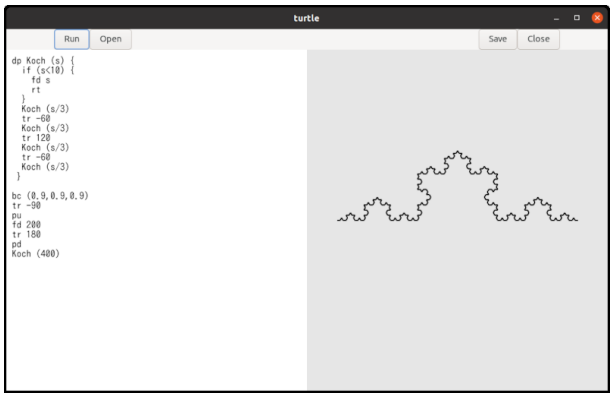

Figure 59: Koch curve

time the procedure is called, the parameter `n` decreases by one. Finally, it becomes less than zero, then the procedures return.

The program above draws a square.

Turtle doesn't have any primary loop statements. It should probably be added to the future version. However, the program above shows that we can program loop with a recursive call.

A.10 Fractal curves

Recursive call can be applied to draw fractal curves. Fractal curves appear when a procedure is applied to it repeatedly. The procedure replaces a part of the curve with the contracted curve.

This shape is called tree. The basic pattern of this shape is a line segment. It is the first stage. The second stage adds two shorter line segments at the endpoint of the original segment. The new segment has 70 percent length to the original segment and the orientation is +30 or -30 degrees different. The third stage adds two shorter line segments to the second stage line segments. And repeats it several times.

This repeating is programmed by recursive call. Two more examples are shown here. They are Koch curve and Square Koch curve.

A.11 Tokens and punctuations

The following is the list of tokens.

Keywords:

- pu: pen up
- pd: pen down
- pw: pen width = line width
- fd: forward

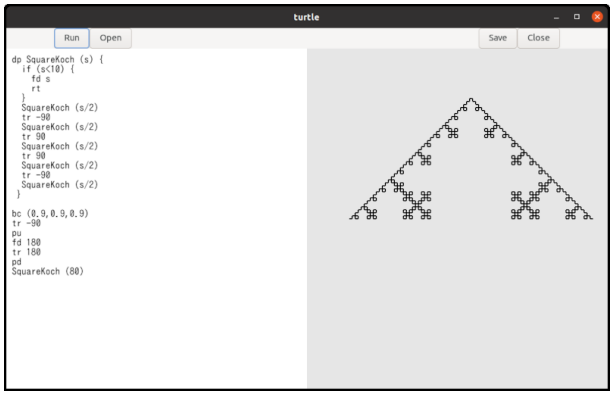

Figure 60: Square Koch curve

- tr: turn right
- bc: background color
- fc: foreground color
- if: if statement
- rt: return
- rs: reset
- dp: define procedure

identifiers and numbers:

- identifier: This is used for the name of variables, parameters and procedures. It is expressed `[a-zA-Z][a-zA-Z0-9]*` in regular expression.
- number: This is expressed `(0|[1-9][0-9]*)(\.[0-9]+)?` in regular expression. It doesn't have `+` or `-` sign because they bring some syntactic confusion. However negative number such as `-10` can be recognized as unary minus and a number.

Symbols for expression

- `=`
- `>`
- `<`
- `+`
- `-`
- `*`
- `/`
- `(`
- `)`

Delimiters

- `(`
- `)`
- `{`
- `}`
- `,`

Comments and spaces:

- comment: This is characters between `#` and new line inclusive. If a comment is at the end of the file, the trailing new line can be left out.
- white space:
- horizontal tab: tab is recognized as eight spaces.
- new line: This is the end of a line.

These characters are used to separate tokens explicitly. They doesn't have any syntactic meaning and are ignored by the parser.

A.12 Grammar

```
program:
  statement
| program statement
;

statement:
  primary_procedure
| procedure_definition
;

primary_procedure:
  PU
| PD
| PW expression
| FD expression
| TR expression
| BC '(' expression ',' expression ',' expression ')'
| FC '(' expression ',' expression ',' expression ')'
| ID '=' expression
| IF '(' expression ')' '{' primary_procedure_list '}'
| RT
| RS
| ID '(' ')'
| ID '(' argument_list ')'
;

procedure_definition:
  DP ID '('  ')' '{' primary_procedure_list '}'
| DP ID '(' parameter_list ')' '{' primary_procedure_list '}'
;

parameter_list:
  ID
| parameter_list ',' ID
;

argument_list:
  expression
| argument_list ',' expression
;

primary_procedure_list:
  primary_procedure
| primary_procedure_list primary_procedure
;

expression:
  expression '=' expression
| expression '>' expression
| expression '<' expression
| expression '+' expression
| expression '-' expression
| expression '*' expression
| expression '/' expression
| '-' expression %prec UMINUS
| '(' expression ')'
| ID
| NUM
;
```

B TfeTextView API reference

TfeTextView – Child object of GtkTextView. It holds GFile which the contents of GtkTextBuffer correponds to.

B.1 Functions

- GFile *tfe_text_view_get_file ()
- void tfe_text_view_open ()
- void tfe_text_view_save ()
- void tfe_text_view_saveas ()
- GtkWidget *tfe_text_view_new_with_file ()
- GtkWidget *tfe_text_view_new ()

B.2 Signals

- void change-file
- void open-response

B.3 Types and Values

- TfeTextView
- TfeTextViewClass
- TfeTextViewOpenResponseType

B.4 Object Hierarchy

```
GObject
+--GInitiallyUnowned
   +--GtkWidget
      +--GtkTextView
         +--TfeTextView
```

B.5 Includes

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

B.6 Description

TfeTextView holds GFile which the contents of GtkTextBuffer corresponds to. File manipulation functions are added to this object.

B.7 Functions

B.7.1 tfe_text_view_get_file()

```
GFile *
tfe_text_view_get_file (TfeTextView *tv);
```

Returns the copy of the GFile in the TfeTextView.

Parameters

- tv: a TfeTextView

B.7.2 tfe_text_view_open()

```
void
tfe_text_view_open (TfeTextView *tv, GtkWidget *win);
```

Just shows a GtkFileChooserDialog so that a user can choose a file to read. This function doesn't do any I/O operations. They are done by the signal handler connected to the `response` signal emitted by GtkFileChooserDialog. Therefore the caller can't know the I/O status directly from the function. Instead, the status is informed by `open-response` signal. The caller needs to set a handler to this signal in advance.

parameters

- tv: a TfeTextView
- win: the top level window

B.7.3 tfe_text_view_save()

```
void
tfe_text_view_save (TfeTextView *tv);
```

Saves the contents of a TfeTextView to a file. If `tv` holds a GFile, it is used. Otherwise, this function shows GtkFileChosserDialog so that a user can choose a file to save.

Parameters

- tv: a TfeTextView

B.7.4 tfe_text_view_saveas()

```
void
tfe_text_view_saveas (TfeTextView *tv);
```

Saves the content of a TfeTextView to a file. This function shows GtkFileChosserDialog so that a user can choose a file to save.

Parameters

- tv: a TfeTextView

B.7.5 tfe_text_view_new_with_file()

```
GtkWidget *
tfe_text_view_new_with_file (GFile *file);
```

Creates a new TfeTextView and reads the contents of the `file` and set it to the GtkTextBuffer corresponds to the newly created TfeTextView. Then returns the TfeTextView as GtkWidget. If an error happens, it returns `NULL`.

Parameters

- file: a GFile

Returns

- a new TfeTextView.

B.7.6 tfe_text_view_new()

```
GtkWidget *
tfe_text_view_new (void);
```

Creates a new TfeTextView and returns the TfeTextView as GtkWidget. If an error happens, it returns `NULL`.

Returns

- a new TfeTextView.

B.8 Types and Values

B.8.1 TfeTextView

```
typedef struct _TfeTextView TfeTextView
struct _TfeTextView
{
  GtkTextView parent;
  GFile *file;
};
```

The members of this structure are not allowed to be accessed by any outer objects. If you want to obtain a copy of the GFile, use `tfe_text_view_get_file`.

B.8.2 TfeTextViewClass

```
typedef struct {
  GtkTextViewClass parent_class;
} TfeTextViewClass;
```

No member is added because TfeTextView is a final type object.

B.8.3 enum TfeTextViewOpenResponseType

Predefined values for the response id given by `open-response` signal.

Members:

- TFE_OPEN_RESPONSE_SUCCESS: The file is successfully opened.
- TFE_OPEN_RESPONSE_CANCEL: Reading file is canceled by the user.
- TFE_OPEN_RESPONSE_ERROR: An error happened during the opening or reading process.

B.9 Signals

B.9.1 change-file

```
void
user_function (TfeTextView *tv,
               gpointer user_data)
```

Emitted when the GFile in the TfeTextView object is changed. The signal is emitted when:

- a new file is opened and read
- a user choose a file with GtkFileChooserDialog and save the contents.
- an error occured during I/O operation, and GFile is removed as a result.

B.9.2 open-response

```
void
user_function (TfeTextView *tv,
               TfeTextViewOpenResponseType response-id,
               gpointer user_data)
```

Emitted after the user calls `tfe_text_view_open`. This signal informs the status of file opening and reading.

C Construir el Tutorial de Gtk4

C.1 Guía rápida

1. Necesitas el sistema operativo linux, ruby, make, pandoc y latex instalados.
2. Descarga este repositorio y descomprime el archivo.
3. Cambia al directorio superior de los archivos fuente.
4. Escribe `rake html` para crear archivos html. Los archivos se crearán dentro de la carpeta `docs`.
5. Escribe `rake pdf`para crear el pdf. El archivo se creará dentro de la carpeta `latex`.

C.2 Prerequisitos

- Sistema operativo Linux Los programas de este repositorio han sido probados en Ubuntu 21.04
- Descarga los archivos en el repositorio. Hay 2 maneras de hacer la descarga.

1. Usar git. Escribe `git clone https://github.com/cjdg/Gtk4-tutorial-spanish.git` en la línea de comandos.
2. Descargar un archivo zip. Click en el botón `Code` en la página del repositorio. Después, click en "Download ZIP".

- Ruby y rake.
- Pandoc. Es usado para convertir archivos markdown a html y/o latex.
- Latex. Textlive2020 o posterior. Se usa para generar el PDF.

C.3 Markdown Github

Cuando ves el repositorio del Tutorial Gtk4 en español, verás el contenido del archivo `Readme.md`. Este archivo está escrito en el lenguake Markdown Los archivos Markdown tienen el sufijo `.md`.

Existen muchas versiones de Markdown. `Readme.md` usa la versión Github de Markdown (GFM). Los archivos Markdown en la carpeta `gfm` están escritos en GFM. Si no estás familiarizado con la GFM, puedes consultar la documentación en github flavor markdown spec.

C.4 Markdown pandoc

Este tutorial tambien usa otro tipo de markdown, 'pandoc'. Pandoc es un convertidor entre markdown, latex, doc, docx, etc. Este tipo de markdown se usa para convertir markdown a html y/o latex.

C.5 Archivo .Src.md

Los archivos .Src.md tienen el sufijo ".src.md". La sintaxis de los archivos .src.md es similar a markdown pero tienen un comando especial que no esta incluido en la sintaxis markdown. Es el comando @@@. Este comando inicia una línea con "@@@" y termina con una linea "@@@". Por ejemplo,

```
@@@include
tfeapplication.c
@@@
```

Existen 4 tipos de comando @@@

C.5.1 @@@include

Este tipo inicia con el comando @@@ con una línea "@@@include"

```
@@@include
tfeapplication.c
@@@
```

Este comando reemplaza el texto con el contenido del archivo C entre los comandos `@@@include` y `@@@`. Si la función precede al nombre de archivo, sólo serán importadas las funciones listadas.

```
@@@include
tfeapplication.c main startup
@@@
```

El comando aqui arriba será reemplazado por las funciones `main` y `startup` del archivo `tfeapplication.c`.

Otros lenguajes pueden ser importados también El siguiente ejemplo importa el archivo 'lib_src2md.rb'

```
@@@include
lib_src2md.rb
@@@
```

No se puede insertar funciones que no sean de C.

El texto insertado es convertido para delimitar el bloque de código. El delimitador del bloque de código comienza con ~~~ y termina con ~~~ Los contenidos son mostrados tal cual. ~~~ parece una cerca, por lo que el bloque se llama "cerca de bloque de código"

Si el objetivo markdown es GFM, entonces una cadena de información puede seguir al delimitador inicial. El siguiente ejemplo muestra como el comando @@@ incluye un archivo fuente C llamado `sample.c`

```
$ cat src/sample.c
int
main (int argc, char **argv) {
  ... ...
}
$cat src/sample.src.md
  ... ...
@@@include -N
sample.c
@@@
  ... ...
$ ruby src2md.rb src/sample.src.md
$ cat gfm/sample.md
  ... ...
~~~C
int
main (int argc, char **argv) {
  ... ...
}
~~~
  ... ...
```

Las cadenas de información son usualmente languajes como C, ruby, xml, etc. Esta cadena se procesa con la extensión del archivo.

- `.c` => C
- `.rb` => ruby
- `.xml` => xml

Los lenguajes permitidos estan escritos en el método `lang` en `lib/lib_src2md.rb`.

Los números de línea serán insertados arriba de cada línea en el bloque de código. Si no deseas que se inserten, incluye la opción "-N" al comando @@@include

Opciones

- `-n`: Inserta un número de línea arriba de cada línea (predeterminado).
- `-N`: No se insertan números de línea.

El siguiente ejemplo muestra los números de linea y como son insertados al inicio de cada línea.

```
$cat src/sample.src.md
  ... ...
@@@include
sample.c
@@@
  ... ...
$ ruby src2md.rb src/sample.src.md
$ cat gfm/sample.md
  ... ...
~~~C
 1 int
 2 main (int argc, char **argv) {
  ... ...
14 }
~~~
  ... ...
```

Si un archivo markdown es un intermediario a uno html, otro tipo de información seguirá el delimitador Si el comando @@@include no tiene una opción -N, entonces el markdown generado será:

```
~~~{.C .numberLines}
int
main (int argc, char **argv) {
  ... ...
}
~~~
```

La cadena de información `.C` específica lenguaje C. La cadena de información `.numberLines` es una clase del markdown pandoc. Si la clase se especifíca, pandoc genera el CSS para insertar los números de líneas al código fuente html. Esa es la razón de usar el delimitador de bloque de código, por que markdown no tiene líneas numeradas, que es diferente del markdown GFM. Si se usa la opción `-N`, sólo aparecerá la cadena informativa `{.C`.

Si un archivo markdown es un intermediario a un archivo latex, la misma cadena sigue al delimitador.

```
~~~{.C .numberLines}
int
main (int argc, char **argv) {
  ... ...
}
~~~
```

Rake usa pandoc con la opción `--listings`para convertir el markdown a un archivo latex. El archivo generado usa `listings package` para listar los archivos fuente en vez del entorno inmediato. El markdown generado arriba se convierte al archivo latex siguiente:

```
\begin{lstlisting}[language=C, numbers=left]
int
main (int argc, char **argv) {
  ... ...
}
\end{lstlisting}
```

El listado de paquetes puede colorear o enfátizar palabras clave, cadenas, comentarios y directivas. Pero no analiza la sintaxis del lenguaje, sólo las palabras clave.

El comando @@@include posee 2 ventajas:

1. Escribir menos.
2. No necesitas modificar el archivo .Src.md, aun si cambias el archivo fuente C.

C.5.2 @@@shell

Este tipo de comando @@@ comienza con una línea "@@@shell":

```
@@@shell
shell command
 ... ...
@@@
```

Este comando se reemplaza a si mismo con:

- el comando shell
- la salida estándar del comando

Por ejemplo:

```
@@@shell
wc Rakefile
@@@
```

Se convierte a:

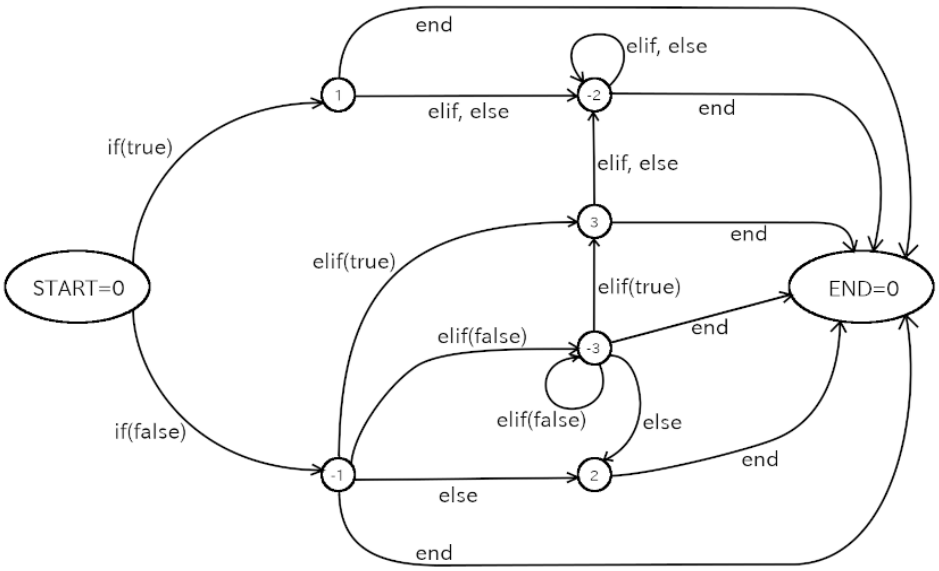

Figure 61: diagrama de estado

```
~~~
$ wc Rakefile
164  475 4971 Rakefile
~~~
```

C.5.3 series @@@if

Este tipo de comando @@@ comienza con una linea "@@@if"m, y seguida por "@@@elif", "@@@else@" o "@@@end". Es similar a "#if", "#elif", "#else" y "#endif" en el preprocesador C. Por ejemplo,

```
@@@if gfm
Refer to  [tfetextview API reference](tfetextview_doc.md)
@@@elif html
Refer to  [tfetextview API reference](tfetextview_doc.html)
@@@elif latex
Refer to tfetextview API reference in appendix.
@@@end
```

`@@@if` y `@@@elif` poseen condiciones Son `gfm`, `html` o `latex` hasta ahora.

- gfm: si el objetivo es GFM
- html: si el objetivo es html
- latex: si el objetivo es pdf.

Otros condicionales pueden estar disponibles en versiones futuras.

El analizador de las series @@@if es un poco complicado. Esta basado en el diagrama de estados de abajo.

C.5.4 @@@table

Este comando @@@ comienza con una línea "@@@table". El contenido es una tabla en formato GFM o pandoc. El comando crea una table fácil de leer. Por ejemplo, un archivo `sample.md` posee una tabla de la siguiente manera:

```
Price list

@@@table
|item|price|
|:---:|:---:|
```

```
|mouse|$10|
|PC|$500|
@@@
```

El comando se transforma en lo siguiente:

```
Price list

|item |price|
|:---:|:---:|
|mouse| $10 |
| PC  |$500 |
```

Este comando cambia la apariencia de la tabla. No influye en los archivos html/latex que son convertidos desde markdown. El comando sólo soporta el formato arriba mencionado.

El script `mktbl.rb` soporta este comando. Si ejecutas el script:

```
$ ruby mktbl.rb sample.md
```

Las tablas en 'sample.md' serán organizadas El script tambien hace un respaldo `sample.md.bak`

La tarea del script parece fácil, pero el programa no es tan simple. El script `mktbl.rb` usa la líbreria `lib/lib_src2md.rb`

Los comandos @@@ son efectivos en todo el texto. Esto significa que no puedes detenerlos. Pero algunas veces necesitas mostrar los comandos como en este documento. Una solución es agregar 4 espacios al inicio de la línea. Entonces los comandos @@@ no son efectivos por que deben estar al inicio de la línea.

C.6 Conversiones

Los comandos @@@ son usados por el método `src2md` el cual se encuentra en el archivo `lib/lib_src2md.rb` Este método convierte los archivos `.src.md` en archivos `.md`. Adicionalmente, otras conversiones son realizadas por el método `src2md`.

- Los vínculos relativos se sustituyen de acuerdo al cambio en el directorio base.
- La opción de tamaño del link de imagen se elimina cuando el destino es GFM o html.
- Los vínculos relativos se eliminan menos los relativos a .src.md cuando el destino es GFM o html.
- Los vínculos relativos se eliminan cuando el destino es latex.

El orden de las conversiones es:

1. @@@if## Src directory and the top directory
2. @@@table
3. @@@include
4. @@@shell
5. otros

Hay un archivo `src2md.rb` en el directorio principal. Invoca el método `src2md`. De la misma forma, el método se ejecuta en la accion del archivo `Rakefile`

C.7 Estructura del directorio

Existen siete directorios dentro del tutorial. Son `gfm`, `docs`, `latex`, `src`, `image`, `test` y `lib`. Tres directorios: `gfm`, `docs` y `latex` son los destinos para GFM, html y latex. Es posible que estos directorios no existan antes de la conversión.

- src: Este directorio contiene los archivos .src.md y los archivos C relacionados.
- image: Este directorio contiene imágenes.
- gfm `rake` convierte los archivos .src.md a archivos GFM y los guarda en este directorio.
- docs: `rake html` convertirá los archivos .src.md a html y los guarda en este directorio.
- latex: `rake pdf` convertirá los archivos .src.md a latex y los guarda en este directorio y crea un pdf en el directorio.
- lib: contiene las líbrerias de ruby.
- test: contiene los archivos de prueba, estos se realizan escribiendo `rake test` en la terminal.

C.8 Directorios Src y superior.

El directorio Src contiene los archivos .src.md y los archivo fuente C. El directorio principal, que es gt_tutotial_spanish, contiene los archivos `Rakefile`, `src2md.rb` y otros. Cuando el archivo `Readme.md` es generado, se ubicará en el directorio principal. `Readme.md` tiene título, resumen, tabla de contenido con enlaces a los archivos GFM.

Src directory contains .src.md files and C-related source files. The top directory, which is gtk_tutorial directory, contains `Rakefile`, `src2md.rb` and some other files. When `Readme.md` is generated, it will be located at the top directory. `Readme.md` has title, abstract, table of contents with links to GFM files.

Rakefile describes how to convert .src.md files into GFM, html and/or pdf files. Rake carries out the conversion according to the `Rakefile`.

C.9 The name of files in src directory

Files in `src` directory are an abstract, sections of the document and other .src.md files. An `abstract.src.md` contains the abstract of this tutorial. Each section filename is "sec", number of the section and ".src.md" suffix. For example, "sec1.src.md", "sec5.src.md" or "sec12.src.md". They are the files correspond to the section 1, section 5 and section 12 respectively.

C.10 C source file directory

Most of .src.md files have `@@@include` commands and they include C source files. Such C source files are located in the subdirectories of `src` directory.

Those C files have been compiled and tested. When you compile source files, some auxiliary files and a target file like `a.out` are created. Or `_build` directory is made when `meson` and `ninja` is used when compiling. Those files are not tracked by `git` because they are specified in `.gitignore`.

The name of the subdirectories should be independent of section names. It is because of renumbering, which will be explained in the next subsection.

C.11 Renumbering

Sometimes you might want to insert a new section. For example, you want to insert it between section 4 and section 5. You can make a temporary section 4.5, that is a rational number between 4 and 5. However, section numbers are usually integer so section 4.5 must be changed to section 5. And the numbers of the following sections must be increased by one.

This renumbering is done by the `renumber` method in the `lib/lib_renumber.rb` file.

- It changes file names.
- If there are references (links) to sections in .src.md files, the section numbers will be automatically renumbered.

C.12 Rakefile

Rakefile is similar to Makefile but controlled by rake, which is a ruby script. Rakefile in this tutorial has the following tasks.

- md: generate GFM markdown files. This is the default.
- html: generate html files.
- pdf: generate latex files and a pdf file, which is compiled by lualatex.
- all: generate md, html and pdf files.
- clean: delete latex intermediate files.
- clobber: delete all the generated files.

Rake does renumbering before the tasks above.

C.13 Generate GFM markdown files

Markdown files (GFM) are generated by rake.

```
$ rake
```

This command generates `Readme.md` with `src/abstract.src.md` and titles of each `.src.md` file. At the same time, it converts each .src.md file into a GFM file under the `gfm` directory. Navigation lines are added at the top and bottom of each markdown section file.

You can describe width and height of images in .src.md files. For example,

```
![sample image](../image/sample_image.png){width=10cm height=6cm}
```

The size between left brace and right brace is used in latex file and it is not fit to GFM syntax. So the size will be removed in the conversion.

If a .src.md file has relative URL links, they will be changed by conversion. Because .src.md files are located under the `src` directory and GFM files are located under the `gfm` directory. That means the base directory of the relative link are different. For example, `[src/sample.c](sample.c)` is translated to `[src/sample.c](../src/sample.c)`.

If a link points another .src.md file, then the target filename will be changed to .md file. For example, `[Section 5](sec5.src.md)` is translated to `[Section 5](sec5.md)`.

If you want to clean the directory, that means remove all the generated markdown files, type `rake clobber`.

```
$ rake clobber
```

Sometimes this is necessary before generating GFM files.

```
$ rake clobber
$ rake
```

For example, if you append a new section and other files are still the same as before, `rake clobber` is necessary. Because the navigation of the previous section of the newly added section needs to be updated. If you don't do `rake clobber`, then it won't be updated because the the timestamp of .md file in gfm is newer than the one of .src.md file. In this case, using `touch` to the previous section .src.md also works to update the file.

If you see the github repository (ToshioCP/Gtk4-tutorial), `Readme.md` is shown below the code. And `Readme.md` includes links to each markdown files. The repository not only stores source files but also shows the whole tutorial.

C.14 Generate html files

Src.md files can be translated to html files. You need pandoc to do this. Most linux distribution has pandoc package. Refer to your distribution document to install it.

Type `rake html` to generate html files.

```
$ rake html
```

First, it generates pandoc's markdown files under `docs` directory. Then, pandoc converts them to html files. The width and height of image files are removed. Links to .src.md files will be converted like this.

```
[Section 5](sec5.src.md) => [Section 5](sec5.html)
```

Image files are copied to `docs/image` direcotiry and links to them will be converted like this:

```
[sample.png](../image/sample.png) => [sample.png](image/sample.png)
```

Other relative links will be removed.

`index.html` is the top html file. If you want to clean html files, type `rake clobber` or `cleanhtml`.

```
$ rake clobber
```

Every html file has a header (`<head> -- </head>`). It is created by pandoc with '-s' option. You can customize the output with your own template file for pandoc. Rake uses `lib/lib_mk_html_template.rb` to create its own template. The template inserts bootstrap CSS and Javascript through `jsDelivr`.

The `docs` directory contains all the necessary html files. They are used in the github pages of this repository.

So if you want to publish this tutorial on your own web site, just upload the files in the `docs` directory to your site.

C.15 Generate a pdf file

You need pandoc to convert markdown files into latex source files.

Type `rake pdf` to generate latex files and finally make a pdf file.

```
$ rake pdf
```

First, it generates pandoc's markdown files under `latex` directory. Then, pandoc converts them into latex files. Links to files or directories are removed because latex doesn't support them. However, links to full URL and image files are kept. Image size is set with the size between the left brace and right brace.

```
![sample image](../image/sample_image.png){width=10cm height=6cm}
```

You need to specify appropriate width and height. It is almost `0.015 x pixels` cm. For example, if the width of an image is 400 pixels, the width in a latex file will be almost 6cm.

A file `main.tex` is the root file of all the generated latex files. It has `\input` commands, which inserts each section file, between `\begin{document}` and `\end{document}`. It also has `\input`, which inserts `helper.tex`, in the preamble. Two files `main.tex` and `helper.tex` are created by `lib/lib_gen_main_tex.rb`. It has a sample markdown code and converts it witn `pandoc -s`. Then, it extracts the preamble in the generated file and puts it into `helper.tex`. You can customize `helper.tex` by modifying `lib/lib_gen_main_tex.rb`.

Finally, lualatex compiles the `main.tex` into a pdf file.

If you want to clean `latex` directory, type `rake clobber` or `rake cleanlatex`

```
$ rake clobber
```

This removes all the latex source files and a pdf file.