GUI–Programmierung mit GTK

by Özcan Güngör
<ozcangungor(at)netscape.net>

About the author:

Ich benutze Linux seit 1997. Freiheit, Flexibilität, Open Source. Diese Eigenschaften mag ich.

Translated to English by:
Özcan Güngör
<ozcangungor(at)netscape.net>

Abstract:

In dieser Artikelserie werden wir lernen, wie man Programme mit graphischen Benutzerschnittstellen (Graphical User Interfaces, GUIs) unter Verwendung von GTK schreibt. Ich habe keine Ahnung, wie lange die Serie dauern wird. Um diese Artikel zu verstehen, solltest du folgendes über die Programmiersprache C wissen:

- Variablen
- Funktionen
- Zeiger

Was ist GTK?

GTK (GIMP Toolkit) ist eine Bibliothek, um graphische Benutzerschnittstellen zu erzeugen. Sie steht unter der GPL. Damit kannst du Open Source–, freie oder kommerzielle Programme schreiben.

Die Bibliothek heißt GIMP Toolkit, weil sie ursprünglich geschrieben wurde, um GIMP (GNU Image Manipulation Program, GNU Bildbearbeitungsprogramm) zu entwickeln. Die Autoren von GTK sind:

- Peter Mattis
- Spencer Kimball
- Josh MacDonald

GTK ist eine objekt–orientierte Benutzerschnittstelle für Applikationen. Obwohl es in C geschrieben ist, verwendet es die Idee von Klassen und Callback–Funktionen.

Kompilieren

Um GTK−Programme zu kompilieren, mußt Du gcc sagen, was die GTK−Bibliotheken sind und wo sie liegen. Das *gtk−config*−Kommando „weiß“ das.

```
# gtk-config --cflags --libs
```

Die Ausgabe dieses Befehls sieht so ähnlich aus (abhängig vom System):

```
    -I/opt/gnome/include/gtk-1.2 -I/opt/gnome/include/glib-1.2
    -I/opt/gnome/lib/glib /include -I/usr/X11R6/include
    -L/opt/gnome/lib -L/usr/X11R6/lib -lgtk -lgdk -rdynamic
    -lgmodule -lglib -ldl -l Xext -lX11 -lm
```

Die Erklärungen dieser Parameter sind:

`-l` Bibliothek: Sucht nach einer Bibliothek der Form lib*Bibliothek* in den definierten Pfaden.
`-L` Pfad: Fügt einen Pfad hinzu, in dem Bibliotheken gesucht werden.
`-I` Pfad: Fügt einen Pfad hinzu, in dem Header−Dateien gesucht werden, die im Programm benutzt werden.

Um ein GTK−Programm namens `hello.c` zu kompilieren, kann man folgenden Befehl benutzen:

```
gcc -o hello hello.c `gtk-config --cflags --libs`
```

Der String nach dem Parameter `-o` ist der Name des kompilierten Programms.

Ein erstes Programm

Es wird angenommen, daß GTK auf deinem System installiert ist. Die neuesten Versionen von GTK können auf ftp.gtk.org gefunden werden.

Auf zu unserem ersten Programm. Es erzeugt ein 200x200 Pixel großes, leeres Fenster.

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

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

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_widget_show  (window);

    gtk_main ();

    return(0);
}
```

`GtkWidget` ist ein Variablentyp, um verschiedene Komponenten wie Fenster, Buttons, Labels etc. zu definieren. In diesem Beispiel wird so ein Fenster erzeugt:

`GtkWidget *window;`

`void gtk_init(int *argc,char ***argv)` initiiert das Toolkit und bekommt die Parameter aus der Kommandozeile. Diese Funktion muß nach der Definition der Komponenten aufgerufen werden.

`GtkWidget *gtk_window_new(GtkWindowType windowtype)` erzeugt ein neues Fenster. *windowtype* kann sein:

- GTK_WINDOW_TOPLEVEL
- GTK_WINDOW_DIALOG
- GTK_WINDOW_POPUP

`void gtk_widget_show(GtkWidget *widget)` wird benutzt, um die Komponente in einem Fenster erscheinen zu lassen. Nachdem eine Komponente definiert ist und die Attribute geändert wurden, muß diese Funktion aufgerufen werden.

`void gtk_main(void)` bereitet die Fenster und alle Komponenten darauf vor, auf dem Bildschirm zu erscheinen. Diese Funktion muß am Ende von GTK−Programmen aufgerufen werden.

Wir wollen nun einige Fenstereigenschaften wie Titel, Größe und Position ändern...

`void gtk_window_set_title(GtkWindow *window,const gchar *title)` benutzt man zum Setzen oder Ändern des Titels von *window*. Der erste Parameter dieser Funktion ist vom Typ `GtkWindow`. Beim Kompilieren werden wir davor gewarnt. Obwohl das kompilierte Programm läuft, ist es besser, das zu korrigieren. `GTK_WINDOW(GtkWidget *widget)` wird stattdessen genommen. Der zweite Parameter *title* ist vom Typ `gchar`. `gchar` ist in der Bibliothek glib definiert und dasselbe wie der Typ `char`.

`void gtk_window_set_default_size(GtkWindow *window, gint width, gint height)` setzt die Größe von *window*. Wie `gchar` ist `gint` in der glib definiert und dasselbe wie `int`.

Die Funktion

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

legt die Position von *window* fest. *position* kann sein:

- GTK_WIN_POS_NONE
- GTK_WIN_POS_CENTER
- GTK_WIN_POS_MOUSE
- GTK_WIN_POS_CENTER_ALWAYS

Hier ist ein Beispiel:

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

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

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
```

```
    gtk_window_set_title(GTK_WINDOW(window),"Ýlk Program");
    gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);
    gtk_window_set_default_size(GTK_WINDOW(window),300,300);
    gtk_widget_show  (window);

    gtk_main ();

    return(0);
}
```

Signale und Ereignisse

In GUIs mußt du Maus und Tastatur benutzen können, etwa um auf einen Button zu klicken. Dafür verwendet man folgende GTK−Funktion:

```
guint gtk_signal_connect_object(GtkObject
    *object,const gchar *name,GtkSignalFu nc func,GtkObject
    *slot_object);
```

object ist die Komponente, die Signale aussendet. Wenn Du z.B. wissen willst, ob ein Button gedrückt wurde, wird *object* dieser Button sein. *name* ist der Name des Ereignisses und kann sein:

- event
- button_press_event
- button_release_event
- motion_notify_event
- delete_event
- destroy_event
- expose_event
- key_press_event
- key_release_event
- enter_notify_event
- leave_notify_event
- configure_event
- focus_in_event
- focus_out_event
- map_event
- unmap_event
- property_notify_event
- selection_clear_event
- selection_request_event
- selection_notify_event
- proximity_in_event
- proximity_out_event
- drag_begin_event
- drag_request_event
- drag_end_event
- drop_enter_event
- drop_leave_event
- drop_data_available_event
- other_event

func ist der Name der Funktion, die aufgerufen wird, sobald das Ereignis eintritt. Hier ist ein Beispiel:

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

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

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

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (close), NULL);
    gtk_widget_show  (window);

    gtk_main ();

    return(0);
}
```

Die Funktion

```
gtk_signal_connect (GTK_OBJECT (window),
    "destroy",GTK_SIGNAL_FUNC (close), NULL)
```

achtet auf das „Fenster schließen“−Event. Wenn versucht wird, das Fenster zu schließen, wird die Funktion *close* gerufen. Diese ruft wiederum `gtk_main_quit()` auf, und das Programm endet.

Mit den Details über Signale und Ereignisse setzen wir uns später auseinander...

Ein gewöhnlicher Button

Normale Buttons lösen für gewöhnlich bestimmte Dinge aus, wenn auf sie geklickt wird. In der GTK−Bibliothek gibt es zwei Arten, Buttons zu erzeugen:

1. `GtkWidget* gtk_button_new (void);`
2. `GtkWidget* gtk_button_new_with_label (const gchar *label);`

Die erste Funktion erzeugt einen Button ohne eine Beschriftung, die zweite hingegen erzeugt einen Button mit der Beschriftung *label*.

Nun benutzen wir eine neue Funktion:

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

Damit können wir einen Button in einem Fenster erscheinen lassen; oder allgemeiner gesagt, können wir eine Komponente in einen Container legen. Im nächsten Beispiel handelt es sich bei dem Container um ein Fenster und bei der hinzuzufügenden Komponente um einen Button. Einige andere Container werden wir zu einem

späteren Zeitpunkt kennenlernen.

Das Wichtigste an einem Button ist es, zu wissen, ob er gedrückt wird oder nicht. Wieder wird zu diesem Zweck die Funktion `gtk_signal_connect` benutzt. Dadurch wird eine andere Funktion gerufen, welche die Funktionalität „hinter“ dem Button realisiert. Hier das Beispiel:

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

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

void clicked(GtkWidget *widget,gpointer *data)
{
        g_print("Button Clicked\n");
}
int main( int   argc,char *argv[] )
{
    GtkWidget *window,*button;

    gtk_init (&argc, &argv);

    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (close), NULL);

    button=gtk_button_new_with_label("Button");
    gtk_container_add(GTK_CONTAINER(window),button);
    gtk_signal_connect(GTK_OBJECT(button),"clicked",
                       GTK_SIGNAL_FUNC(clicked),NULL);
    gtk_widget_show(button);

    gtk_widget_show(window);

    gtk_main ();

    return(0);
}
```