

Grafische Benutzerschnittstellen in C++ mit GTKmm Betriebssystemunabhängig gestalten

1. WILLKOMMEN IN DER WELT VON GTK+

Bevor wir uns mit der betriebssystemunabhängigen GTKmm-Programmierung befassen, möchte ich erst einmal auf die Herkunft und die Entstehung von GTKmm und GTK+ eingehen. GTKmm ist eine portable Bibliothek, mit der man vor allem unter Linux, MacOS und Windows portabel Programme entwerfen kann.

Hierfür müssen die Programme zwar für jedes System neu kompiliert werden, aber dies ist ohne große Änderungen und manchmal auch ganz ohne Änderungen zu bewerkstelligen.

Das Woher, Warum, Wie und Was soll in diesem Tutorial geklärt werden.

Sollte es noch Fragen und Verbesserungsvorschläge geben, schreibt sie mir doch bitte an evilissimo@c-plusplus.de.

Ich werde dieses Tutorial kapitelweise veröffentlichen und verspreche keine regelmäßigen Termine. Ich werde jedoch an dem Tutorial arbeiten. 😊

1.1 Was ist ein Widget

In diesem Tutorial wird sehr häufig das Wort Widget fallen. Nun stellt sich sicher der ein oder andere die Frage: "Was ist ein "Widget" denn überhaupt?"

Als Widgets bezeichnet man meist Steuerelemente. Zu den am häufigsten genutzten Steuerelementen gehören Schaltflächen und Bildlaufleisten. Daneben gibt es eine Vielzahl mehr oder minder häufig genutzter Steuerelemente.[\[1\]](#)

1.2 Was ist GTK+ eigentlich?

GTK steht für "Gimp Tool Kit". Das Gimp Tool Kit ist eine in C geschriebene Bibliothek für eine grafische Benutzerschnittstelle, die ursprünglich als Alternative für Motif unter Linux entwickelt wurde, um somit dem Bildbearbeitungsprogramm GIMP eine Alternative GUI (Graphical User Interface, zu Deutsch: grafische Benutzerschnittstelle) bereitzustellen.

Im Laufe der Zeit hat sich GTK+ zu einer der erfolgreichsten Widget-Bibliotheken entwickelt und ist auf X-Server basierenden Systemen neben dem QT Toolkit auch einer der meist benutzten. GTK+ ist auch die Basis der GNOME Desktop Environment, die der eine oder andere von euch sicherlich kennt.

GTK+ ist zwar in C implementiert, aber mit einem objektorientierten Ansatz, der die Anbindung an objektorientierte Sprachen (wie z.B. Python, Perl und C++ und andere Sprachen) stark vereinfacht, da dies von Anfang an berücksichtigt wurde.

GTKmm ist eine solche Anbindung an GTK+ für C++. Im Gegensatz zum originalen GTK+ besitzt GTKmm eine Klassenhierarchie, die die Entwicklung und Weiterentwicklung neuer Widgets stark vereinfacht. Die Typensicherheit von GTKmm erleichtert dem Programmierer die Arbeit enorm. Hierdurch werden einige Fehler im Quelltext bereits zur Kompilierzeit erkannt, was bei der C-Variante nicht der Fall ist und die Fehler somit z. B. womöglich erst durch undefiniertes Verhalten auftreten.

In GTK+ werden ausschließlich Zeiger verwendet, wovon man als C++-Programmierer Abstand nehmen und stattdessen vermehrt auf Membervariablen zurückgreifen sollte. Dies vereinfacht die Speicherverwaltung enorm.

1.3 GTKmm ist ein Wrapper

GTKmm ist keine native Implementierung eines Toolkits wie es QT z. B. ist. Aber das hat auch Vorteile. So können sich die GTKmm-Entwickler mehr Gedanken darüber machen, eine saubere und transparente Schnittstelle zu schaffen, und müssen sich nicht auf Grund von technischen Details auf Kompromisse einlassen. Ein Beispiel hierfür wäre, dass man auf Grund von technischen Details, Abstriche in der Flexibilität von GTKmm machen müsste.

1.4 Bestandteile von GTKmm

Da GTKmm die GTK+ C Bibliotheken verwendet sind diese ein wichtiger Bestandteil von GTKmm. GTKmm kapselt die C Bibliothek so gut, dass es die Verwendung gegenüber GTK+ sogar noch etwas vereinfacht. Hier eine Auflistung und kurze Erklärung der einzelnen Bestandteile:

1.4.1 GTK+ Bibliotheken

Da GTKmm nur ein Wrapper einer C Bibliothek ist, baut es vor allem erst einmal auf diversen C Bibliotheken auf.

Es handelt sich hier bei um folgende C Bibliotheken:

- libglib-2.0
- libatk-1.0
- libpango-1.0
- libgtk-2.0
- libgdk-2.0
- libintl
- libiconv

und je nach Implementierung auch -libcairo

Glib, Pango, ATK, GDK sind alle Bestandteile von **GTK+** selbst, wurden aber unterteilt. Das macht die ganze Sache somit etwas flexibler.

Glib ist der Kern der ganzen Library und bildet die Basis von GTK+ und GNOME. Es erleichtert den Umgang mit Datenstrukturen in C, liefert portable Wrapper für verschiedene betriebssystemabhängige Funktionen und enthält Schnittstellen zur Laufzeitfunktionalität wie z.B. Ereignisschleifen, Threads, dynamisches Laden und ein Objektsystem.

Pango ist eine Layout- und Textrenderingbibliothek mit dem Schwerpunkt auf Internationalisierung. Somit bildet es den Kernteil für den Umgang mit Text- und Schriftarten in GTK+ 2.x.

ATK liefert diverse Schnittstellen für die Zugänglichkeit. Durch das Unterstützen von ATK Schnittstellen kann eine Anwendung oder ein Toolkit mit Werkzeugen wie Bildschirmlesern, Lupen oder alternativen Eingabegeräten benutzt werden.

GDK ist eine Schnittstelle zu nativen Zeichenfunktionen eines Betriebssystems und erhöht damit die Portabilität der Zeichnung von Widgets. Es gibt dafür portable Backends (diverse Bibliotheken mit einem identischen Interface, die man austauschen kann, nennt man in der Regel Backends). GTK+ for Win32 ist z.B. ein solches GDK-Backend für Windows.

GTK+ Implementiert die Widgets und deren Verhalten mit Hilfe der zuvor genannten Bibliotheken.

1.4.2 Bestandteile von GTKmm

GTKmm benötigt neben den eben genannten C Bibliotheken auch noch diverse andere Bibliotheken.:

- libglibmm
- libgtkmm
- libsigc++-2.0

Es gibt noch andere Bibliotheken, die aber bei libgtkmm schon dabei sind:

- libatkmm
- libgdkmm
- libpangomm

Wie es bereits auffällt, haben die meisten dieser Bibliotheken einfach nur ein mm als Suffix und sind einfach nur C++-Wrapper der C-Bibliotheken. Da ich in 1.4.1 bereits erwähnt habe, was die einzelnen Bibliotheken tun, werde ich das einfach an dieser Stelle auslassen und direkt auf SigC++ zu sprechen kommen, das die einzige Bibliothek ist, die nicht auf den C-Bibliotheken aufbaut.

SigC++ implementiert ein typensicheres Callback-System in Standard C++. Es erlaubt dem Programmierer, Signale zu definieren und diese mit irgendeiner Callback-Funktion zu verbinden. Dabei können nicht nur Funktionen, sondern auch Methoden verwendet werden, selbst wenn diese statisch oder virtuell sind. Es beinhaltet auch Adapter, die es dem Programmierer ermöglichen, unterschiedliche Callbacks zu verbinden. Es besteht sogar die Möglichkeit, Standardparameter zu übergeben und damit eine Funktion mit mehreren Parametern an eine bestimmte Anforderung anzupassen. Mehr dazu wird es zu einem späteren Zeitpunkt geben, da die Signalbehandlung in GTKmm ein größeres Kapitel brauchen wird.

2. Voraussetzung für dieses Tutorial

Ich setze für dieses Tutorial voraus, dass sich eine Entwicklerversion von GTKmm 2.4.x oder höher auf dem System befindet und dass der Leser die Grundlagen von C++ beherrscht und sich im Klaren ist, um was es sich bei Vererbung, Polymorphie und auch SmartPointern handelt. Fragen bezüglich der Programmierung mit GTKmm können im C/C++-Forum unter der Adresse <http://www.c-plusplus.de/forum> gestellt werden.

Bezüglich der Installation wird es evtl. ein gesonderte Anleitung geben.

2.1 Bezugsmöglichkeiten für Windows

Unter Windows gibt es dafür Installer, die dem Programmierer alle notwendigen Bibliotheken auf dem System installieren.

Den Installer für alle GTK+ Bibliotheken bekommt man hier:

<http://gladewin32.sourceforge.net/>

Nach der Installation der GTK+ Bibliotheken braucht man noch die GTKmm Bibliotheken die man auf:

http://www.pcpm.ucl.ac.be/~gustin/win32_ports/

bekommt. Klickt dort im Menü auf „gtkmm on win32“ und auf den Link unter "Developer (Full) Environment"

(runtime + headers, import libraries, demo, doc).

Solltet ihr vorhaben, mit der DevC++ IDE zu programmieren, empfehle ich euch nach der Installation der beiden Pakete meinen GTKmm-Template-Installer für DevC++ zu verwenden, der euch eine Vorlage in DevC++ installiert, die bereits sämtliche benötigten Includeverzeichnisse und Bibliotheken in das Projekt einbindet.

Ihr findet den Installer auf <http://www.evillissimo-softdev.de/downloads.html>.

2.2 Bezugsmöglichkeiten für Linux

Bei Linux hängt es von eurer Distribution ab und ihr solltet euch diesbezüglich noch einmal extra informieren. Fragen zu diesem Thema sind natürlich im GUI Forum des C/C++ Boards willkommen. (<http://www.c-plusplus.de/forum/viewforum-var-f-is-51.html>)

Da dies ein Tutorial zur Programmierung mit GTKmm werden soll, werden wir an dieser Stelle mit den Nebensächlichkeiten aufhören und mit der Einführung in die Programmierung beginnen.

3. Grundlagen

3.1 Ein einfaches Beispiel

Das erste GTKmm Programm, das ich euch zeigen möchte, ist ein einfaches Fenster von 200 x 200 Pixel.

Hiermit möchte ich verdeutlichen, wie einfach es sein kann, eine GTKmm Anwendung zu schreiben.

```
#include <gtkmm.h>

int main(int argc, char *argv[])
{
    Gtk::Main main_obj(argc, argv);
    Gtk::Window window_obj;
    main_obj.run(window_obj);
    return 0;
}
```

Nun es ist nicht viel Code, aber es erzeugt schon ein Fenster. Nun werde ich detaillierter auf die einzelnen Zeilen eingehen und ein paar Bemerkungen dazu machen.

```
#include <gtkmm.h>
```

Wir binden mit dieser Zeile alle Headerdateien ein, die zu GTKmm gehören; dies gilt dann auch für GDKmm, Glibmm, Pangomm und SigC++.

Der Nachteil dieser Zeile ist, dass sie bei größeren Projekten zu längeren Kompilierzeiten führt, da erst einmal alle Header analysiert werden müssen. Daher ist es empfehlenswert, aber nicht zwingend notwendig, nur die benötigten Header einzubinden. In unserem Fall wären das die folgenden:

```
#include <gtkmm/main.h>

#include <gtkmm/window.h>
```

In der Dokumentation auf <http://www.gtkmm.org/docs/gtkmm-2.4/docs/> findet ihr heraus in welchen Headern sich die verwendeten Klassen befinden.

```
int main(int argc, char *argv[])
```

Der normale Programmeinstiegspunkt, der aus C++, aber auch aus C Programmen bekannt sein sollte.

```
Gtk::Main main_obj(argc, argv);
```

Mit dieser Zeile starten wir das GTK+- und GTKmm-Subsystem und stellen durch Übergeben der Parameter argc und argv sicher, dass auch unsere Anwendung wie alle GTK+ Anwendungen bestimmte Standardparameter annimmt und auswertet. Damit passen wir unser Programm nur an das Verhalten anderer GTK+ Programme an. Mehr müssen wir darüber nicht wissen, es sei denn, wir möchten selber Parameter übergeben - dann solltet ihr eure Auswertung der Parameter erst nach GTKmm durchführen, da GTKmm sie zur Initialisierung benötigt!

Merke: *Immer erst das GTKmm Subsystem starten.*

Die restlichen zwei Zeilen sind nicht sonderlich spektakulär, aber ich werde trotzdem kurz erklären, was diese tun.

```
Gtk::Window window_obj;
```

Erstellt ein Fensterobjekt, das wir mir der Zeile:

```
main_obj.run(window_obj);
```

starten.

Nachdem wir den Sourcecode in eine Sourcedatei gepackt haben - ich nenne sie hier mal einfach "beispiel1.cpp" - werden wir die Anwendung kompilieren.

Je nach OS wird anders kompiliert. Die Windows-User mit DevC++ drücken vielleicht einfach nur F9. Unix/Linux/*BSD/Whatever-User schreiben dann eher

```
g++ beispiel1.cpp `pkg-config gtkmm-2.4 --cflags --libs` -o beispiel1
```

Da dies sehr unterschiedlich ist, möchte ich an dieser Stelle nicht tiefer darauf eingehen. Diese Fragen werden gerne im GUI-Forum auf dem C/C++ Board (<http://www.c-plusplus.de/forum/viewforum-var-f-is-51.html>) beantwortet und erhalten dann sicherlich auch Einzug in die FAQ. 😊

Und so sieht das dann z.B. unter Window XP aus:



Soviel zum ersten Teil des Tutorials. Ich freue mich schon auf euer Feedback. Im nächsten Teil werde ich euch die ersten Widgets näher bringen. Darunter fallen Label, Buttons und das GTK Boxen System. Auch eine kleine Einführung in die Verwendung von Signalen. Mehr dazu im nächsten Teil.

Vinzenz 'evilissimo' Feenstra

Verweise und Referenzen:

- [1] [Widget \(GUI\) Beschreibung von Wikipedia](#)
- [2] [Homepage von GTK+](#)
- [3] [Homepage von GTKmm](#)
- [4] [GTKmm Dokumentation](#)
- [5] [GTKmm Template installer für DevC++ von evilissimo](#)
- [6] [gladewin32 Installer \(GTK+ Entwicklungsbibliotheken für GTK+ \)](#)
- [7] [GTKmm Installer für Windows \(Entwicklungsbibliotheken für GTKmm\)](#)