GTK+

- Gimp Tool Kit
- Toolkit zur GUI-Programmierung
- Verfügbar für Unix/Linuxsysteme, einschl. Mac und Windows
- Basiert auf C
  https://developer.gnome.org/gtk3/stable/gtk-getting-started.html
- Tutorial:
-
- Installation unter ubuntu mittels
  sudo apt-get install libgtk-3-dev
- Alle Beispiele basieren auf dem o.g. Tutorium

Der Build Prozess

Ein einfaches Programm wird mit nachfolgendem Kommando kompilert:

```
gcc gtk1.c -o gtk1 `pkg-config --cflags --libs gtk+-3.0`
```

Dabei erzeugt `pkg-config --cflags --libs gtk+-3.0` Commandlineoptions, hauptsächlich zu Includefiles ,ihren Verzeichnissen und Bibliotheken:

```
pkg-config --cflags --libs gtk+-3.0
```

```
-pthread -I/usr/include/gtk-3.0 -I/usr/include/atk-1.0
-I/usr/include/at-spi2-atk/2.0 -I/usr/include/pango-1.0 -I/usr/
include/gio-unix-2.0/ -I/usr/include/cairo -I/usr/include/gdk-
pixbuf-2.0 -I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/
glib-2.0/include -I/usr/include/harfbuzz
-I/usr/include/freetype2 -I/usr/include/pixman-1
-I/usr/include/libpng12  -lgtk-3 -lgdk-3 -latk-1.0 -lgio-2.0 -
lpangocairo-1.0 -lgdk_pixbuf-2.0 -lcairo-gobject -lpango-1.0 -
lcairo -lgobject-2.0 -lglib-2.0
```

Alles dreht sich um Widgets – was ist das?

- Zusammensetzung aus
  - Window (Fenster)
  - Gadget (Vorrichtung, Gerät, Dingsbums, Apparatur, techn. Spielerei, ...)
- Letzlich - Fenster mit einer speziellen Funktionalität
- Widgets sind Buttons, Checkboxes, Eingabefelder (ein-/mehrzeilig) … oder Container
- Synonyme Bezeichnungen sind Controlls, Components (java), Bedienelemente

Helloprogramm

```
#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;
}
```

gtk_init(gint *argc, gchar ***argv)

GtkWidget *gtk_window_new (GtkWindowType type);
Erzeugt ein Toplevel-Window

void gtk_widget_show (GtkWidget *widget);
Wird als sichtbar markiert

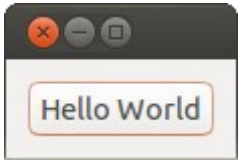
void gtk_main (void);
Startet die mainloop der Applikation.
Programm bleibt in dieser Funktion
Bis zum Beenden

Ergänzungen

```
int main( int   argc,
          char *argv[] )
{
 int i;
 GtkWidget *window;
 gtk_init (&argc, &argv);

 for (i=0; i<argc; i++) puts(argv[i]);

 window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

 //gtk_window_set_title((GtkWindow *)window,"GTK-Spass");
 gtk_window_set_title(GTK_WINDOW (window),"GTK-Spass");
 g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);

 gtk_widget_show  (window);
 gtk_main ();
 return 0;
}
```

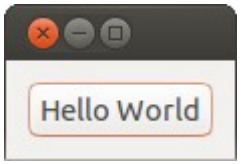
#define g_signal_connect(instance, detailed_signal, c_handler, data)
Bewirkt, dass unser Window bei Auftreten eines „destoy“-Ereignisses die Funktion gtk_main_quit(NULL) aufruft und damit nicht nur das Hauptfenster schließt, sondern auch das Programm ordentlich beendet.

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

  gtk_init (&argc, &argv);

  for (i=0; i<argc;i++)puts(argv[i]);
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW (window),"GTK-Spass");
  g_signal_connect (window, "delete-event", G_CALLBACK (on_delete_event), NULL);
  g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);
  button = gtk_button_new_with_label ("Hello World");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);
  g_signal_connect_swapped (button, "clicked", G_CALLBACK (gtk_widget_destroy), window);
  //g_signal_connect_swapped (button, "clicked", G_CALLBACK (on_delete_event), window);
  gtk_container_add (GTK_CONTAINER (window), button);
  gtk_widget_show (button);
  gtk_widget_show (window);
  gtk_main ();
  return 0;
}
```



Callback für Eventhandling

```
static gboolean
on_delete_event (GtkWidget *widget,
                 GdkEvent  *event,
                 gpointer   data)
{
  g_print ("delete event occurred\n");
  //return TRUE;
  return FALSE;
}

static void
print_hello (GtkWidget *widget,
             gpointer   data)
{
  g_print ("Hello World\n");
}
```



Mehrere Buttons

- `GtkWidget * gtk_grid_new (void);`
  - Erzeugt ein neues Widget (GridContainerWidget)
  - Es können mehrere Widgets eingefügt werden
  - Die Widgets werden in eine Gitterstruktur einsortiert
- `void gtk_container_add (GtkContainer *container,`
  `GtkWidget *widget);`
  - Fügt einen Container in ein anderes Widget ein, bei uns in das Hauptfenster
- `void gtk_grid_attach (GtkGrid *grid, GtkWidget *child, gint left, gint top, gint width, gint height);`
  - Fügt ein Widget (child) in den GridContainer ein


```
static void
print_hello (GtkWidget *widget,
             gpointer   data)
{
  g_print ("%s\n",(char*)data);
}
```

```
Int main (int   argc,
      char *argv[])
{
  GtkWidget *window;
  GtkWidget *grid;
  GtkWidget *button;

  gtk_init (&argc, &argv);
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "Grid");
  g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);
  grid = gtk_grid_new ();
  gtk_container_add (GTK_CONTAINER (window), grid);
  button = gtk_button_new_with_label ("Button 1");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), "Hello 1");
  gtk_grid_attach (GTK_GRID (grid), button, 0, 0, 1, 1);
  button = gtk_button_new_with_label ("Button 2");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), "Hello 2");
  gtk_grid_attach (GTK_GRID (grid), button, 1, 0, 1, 1);
  button = gtk_button_new_with_label ("Quit");
  g_signal_connect (button, "clicked", G_CALLBACK (gtk_main_quit), NULL);
  gtk_grid_attach (GTK_GRID (grid), button, 0, 1, 2, 1);
  gtk_widget_show_all (window);
  gtk_main ();
  return 0;
}
```

Anderes Layout

- Widget GtkImage erlaubt das Einfügen von Bildern
- Durch Einsetzen eines anderen Containers an Stelle von Grid wird ein anderes Layout erzeugt.
- Andere Contaioner sind
  - **GtkListBox**
  - GtkFlowBox
  - GtkPaned
  - GtkNotebook
  - . . .

```
int main (int   argc, char *argv[])
{
  GtkWidget *window;
  GtkWidget *list;
  GtkWidget *button;
  gtk_init (&argc, &argv);
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title (GTK_WINDOW (window), "Grid");
  g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);
  gtk_container_set_border_width (GTK_CONTAINER (window), 10);
  list = gtk_list_box_new ();
  gtk_container_add (GTK_CONTAINER (window), list);
  button = gtk_button_new_with_label ("Button 1");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), "Hello 1");
  gtk_list_box_insert (GTK_LIST_BOX (list), button,0);
  button = gtk_button_new_with_label ("Button 2");
  g_signal_connect (button, "clicked", G_CALLBACK (print_hello), "Hello 2");
  gtk_list_box_insert (GTK_LIST_BOX (list), button,1);
  button = gtk_button_new_with_label ("Quit");
  g_signal_connect (button, "clicked", G_CALLBACK (gtk_main_quit), NULL);
  gtk_list_box_insert (GTK_LIST_BOX (list), button,2);
  GtkWidget *image;
  image = gtk_image_new_from_file ("lampe.jpg");
  gtk_list_box_insert (GTK_LIST_BOX (list), image,3);
  gtk_widget_show_all (window);
  gtk_main ();
  return 0;
}
```

Layout generieren

- Das gesamte Layout kann in einer XML-Datei beschrieben werden
- Mittels GtkBuilder wird die gesamte Oberfläche generiert.
- Die Applikation reduziert sich dann auf das Eventhandling

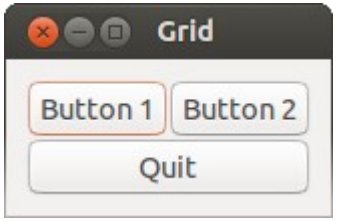
```
GtkBuilder *builder;
builder = gtk_builder_new ();
gtk_builder_add_from_file (builder, "gtk5.ui", NULL);
```

```
int main (int   argc, char *argv[])
{
 GtkBuilder *builder;
 GObject *window;
 GObject *button;

 gtk_init (&argc, &argv);
 builder = gtk_builder_new ();
 gtk_builder_add_from_file (builder, "gtk5.ui", NULL);
 window = gtk_builder_get_object (builder, "window");
 g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);

 button = gtk_builder_get_object (builder, "button1");
 g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);

 button = gtk_builder_get_object (builder, "button2");
 g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);

 button = gtk_builder_get_object (builder, "quit");
 g_signal_connect (button, "clicked", G_CALLBACK (gtk_main_quit), NULL);

 gtk_main ();

 return 0;
}
```

```
<interface>
  <object id="window" class="GtkWindow">
    <property name="visible">True</property>
    <property name="title">Grid</property>
    <property name="border-width">10</property>
    <child>
      <object id="grid" class="GtkGrid">
        <property name="visible">True</property>
        <child>
          <object id="button1" class="GtkButton">
            <property name="visible">True</property>
            <property name="label">Button 1</property>
          </object>
          <packing>
            <property name="left-attach">0</property>
            <property name="top-attach">0</property>
          </packing>
        </child>
        <child>
          <object id="button2" class="GtkButton">
            <property name="visible">True</property>
            <property name="label">Button 2</property>
          </object>
          <packing>
            <property name="left-attach">1</property>
            <property name="top-attach">0</property>
          </packing>
        </child>
        <child>
          <object id="quit" class="GtkButton">
            <property name="visible">True</property>
            <property name="label">Quit</property>
          </object>
          <packing>
            <property name="left-attach">0</property>
            <property name="top-attach">1</property>
            <property name="width">2</property>
          </packing>
        </child>
      </object>
      <packing>
      </packing>
    </child>
  </object>
</interface>
```

GtkListBox

GtkGrid



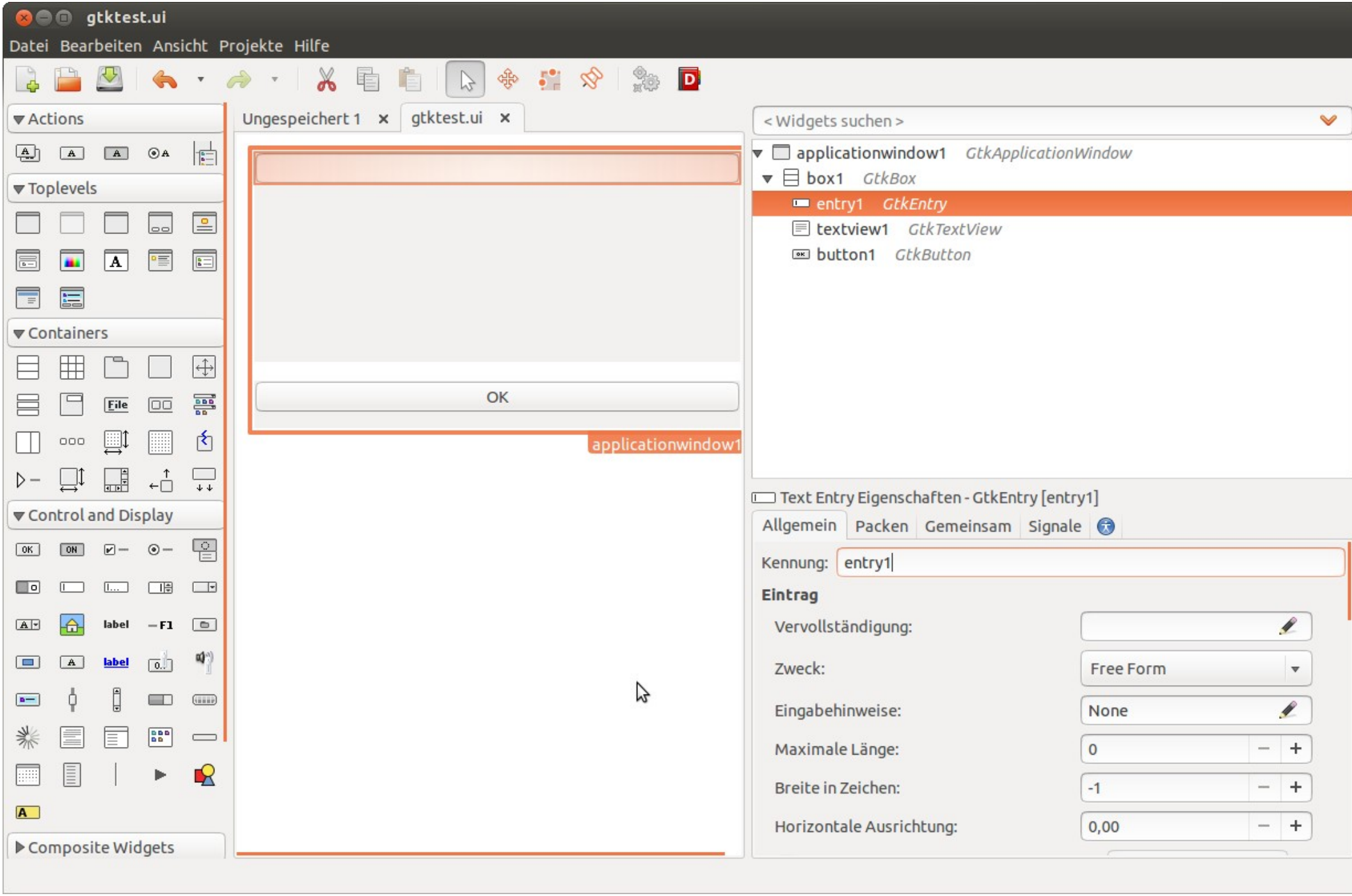
- Nicht validierendes XML
- Datenauszeichnungssprache
- Inhalte werden in 'Tags' verpackt
- Ähnlich html
- Ermöglicht den Einsatz von Oberflächengeneratoren
- Ermöglicht Änderung der Oberfläche ohne neu zu kompilieren

Oberflächengenerator glade

- Installation aus Package
- Tutorial:

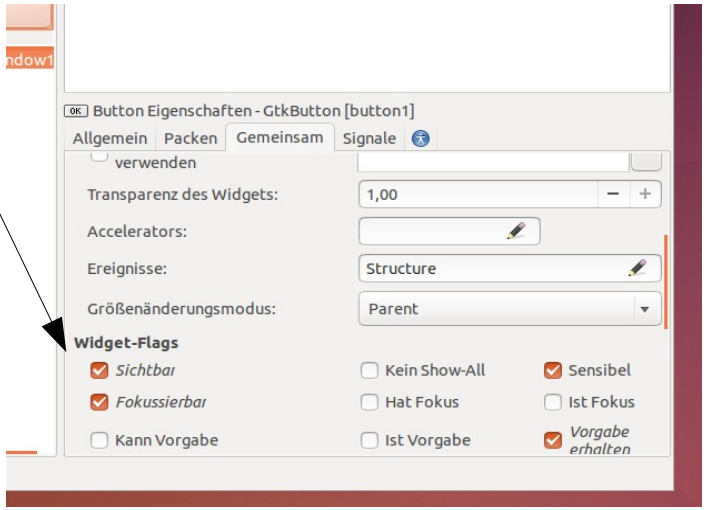
  http://www.micahcarrick.com/gtk-glade-tutorial-part-1.html

  (Versionsprobleme gtk+2.0 / gtk+3.0)
- Die gespeicherte Datei liegt in xml vor, sie hat die Extension .glade, .ui oder .xml.
  - `gcc -Wall -g -o tutorial main.c `pkg-config --cflags --libs gtk+-3.0` -export-dynamic`
- Das c-Programm ähnelt dem von Folie 13


gtktest.ui
Datei Bearbeiten Ansicht Projekte Hilfe
Actions
Toplevels
Containers
Control and Display
Composite Widgets
Ungespeichert 1
gtktest.ui
OK
applicationwindow1
< Widgets suchen >
applicationwindow1 GtkApplicationWindow
box1 GtkBox
entry1 GtkEntry
textview1 GtkTextView
button1 GtkButton
Text Entry Eigenschaften - GtkEntry [entry1]
Allgemein
Packen
Gemeinsam
Signale
Kennung: entry1
Eintrag
Vervollständigung:
Zweck: Free Form
Eingabehinweise: None
Maximale Länge: 0
Breite in Zeichen: -1
Horizontale Ausrichtung: 0,00

Arbeitsschritte

- Applikationsfenster von Auswahl nach Arbeitsfläche ziehen
- Eigenschaften einstellen (wichtig: Gemeinsam: sichtbar)
- Weitere Controlls und Layouts einfügen
- Speichern unter .ui, .xml oder .glade



```
// aus Tutorial http://www.micahcarrick.com/gtk-glade-tutorial-part-1.html

// gcc -Wall -g -o gtkfolie gtkFolie.c `pkg-config --cflags --libs gtk+-3.0` -export-dynamic

#include <gtk/gtk.h>
void on_window_destroy (GtkWidget *object, gpointer user_data)
{
    gtk_main_quit ();
}
int
main (int argc, char *argv[])
{
    GtkBuilder      *builder;
    GtkWidget       *window;
    gtk_init (&argc, &argv);

    builder = gtk_builder_new ();
    gtk_builder_add_from_file (builder, "gtktest.xml", NULL);
    window = GTK_WIDGET (gtk_builder_get_object (builder, "window"));
    gtk_builder_connect_signals (builder, NULL);
    g_object_unref (G_OBJECT (builder));

    gtk_widget_show (window);
    gtk_main ();
    return 0;
}
```

GTK+ 2019/20

- Einführung:
  - https://developer.gnome.org/gtk3/stable/gtk-getting-started.html
- Glade(template/make)
  - https://prognotes.net/2015/07/gtk-3-glade-c-programming-template/
- Referenz
  - https://developer.gnome.org/gtk3/stable/
- Buildkommando für app. mit glade
  - gcc src/main.c -o gglade `pkg-config --cflags --libs gtk+-3.0` -export-dynamic