# Überblick

## Fahrplan für heute

- Rückblick – letzer Termin
- Präsentation(en): Home-Workshop
- Bastelaufgabe

dazwischen: PAUSE

- interaktiver Vortrag
  - Erstellung einer (einfachen) Klasse
- SOS: SVN ??

… vermutlich für den Home-Workshop

- Projektplanung
  - Analyse
  - Recherche Komponenten
  - Präsentation Ergebnisse

## Vortragsreihe - gobject Framework

- **Typsystem**

## Teil I (heute)

- Klassen + Instanzen
- Properties
- Parameter und Werte
- Typinformation

## Teil II

- Interfaces
- Signale
- Marshaller und Akkumulatoren

# Wo genau sind wir eigentlich?

## Komponenten

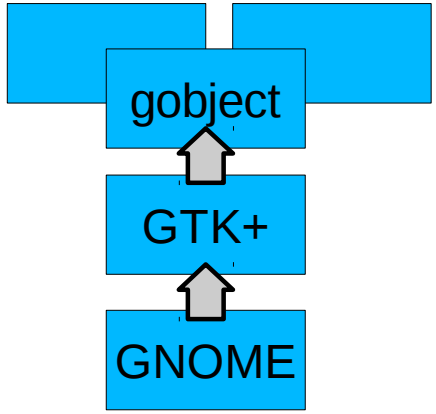
- auf gtk.org gelistet

| | | |
|---|---|---|
| ! | **Glib** | Utility Library |
| → | **GObject** | Objekt System Library |
| | GIO | Virtual Filesystem (VFS) Library |
| | Pango | Text Rendering Library |
| | ATK | Accessibility Toolkit |
| | GdkPixbuf | Pixelgrafik |
| | GDK | GIMP Drawing Kit |
| ✓ | **GTK** | GIMP Toolkit |

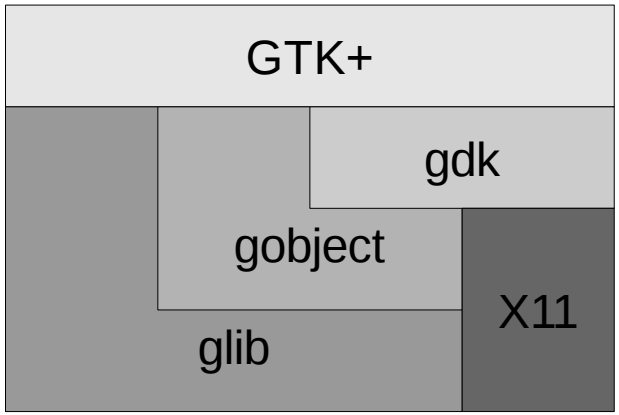
- … nicht zu vergessen

| | |
|---|---|
| cairo | Vektorgrafik |

Code-Wanderung




Aufbau



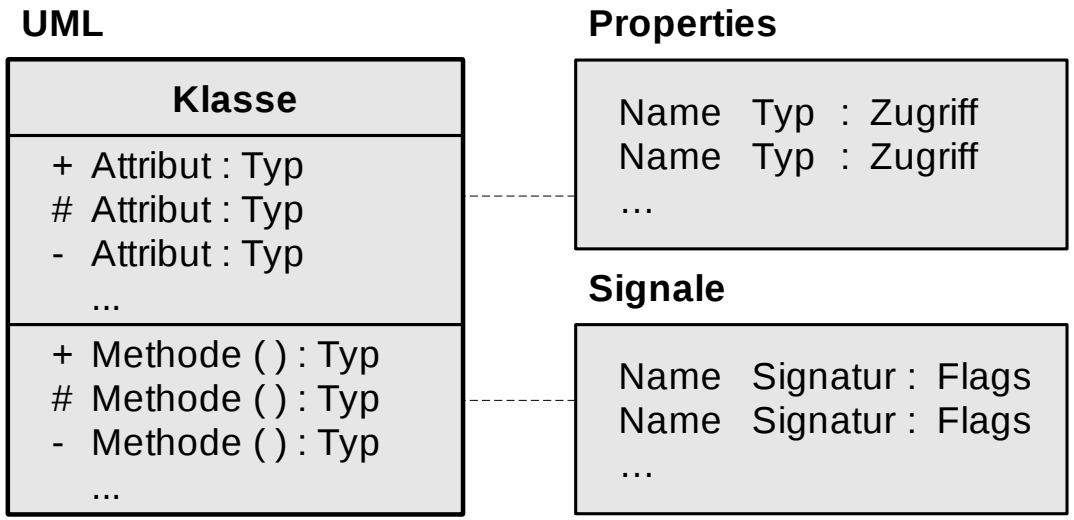
# 1 Klassen und Objekte

## benötigte Komponenten

- Typ / Klasse
- Attribute
- Properties
- Methoden
- Signale

## beschrieben durch

- Typinformation
- Instanz-Struktur
  - Attribute
- Klassen-Struktur
  - virtuelle Methoden
  - Signale



# 1.1 Instanz- und Klassen-Struktur

## Idee

### Klassen-Struktur

- Typkennung
  - GTypeClass als „erstes Element“
- Klassenvariablen
- virtuelle Methoden
  - Signale

```
struct _GTypeClass {
  GType g_type;
};
```

### Instanz-Struktur

- Klasseninformation
  - GtypeInstance als „erstes Element“
- Instanzvariablen
  - Properties

```
struct _GTypeInstance {
  GTypeClass *g_class;
};
```

# 1.2 Beispiel

## Strukturen

```
typedef struct {
  GTypeClass class;
  gint       value_a;
} AClass;
```

```
typedef struct {
  GTypeInstance instance;
  gdouble       value_b;
} A;
```

```
struct _GTypeClass {
  GType g_type;
};
```

```
struct _GTypeInstance {
  GTypeClass *g_class;
};
```

```
typedef struct {
  AClass     parent;
  void (*method_b)(void);
} BClass;
```

```
typedef struct {
  A       parent;
  gchar  *string_b;
} B;
```

A *a;

**Klasse / Typ**

((GTypeInstance*)a)->g_class

((GTypeInstance*)a)->g_class->g_type

**Klassenvariablen**

((AClass*)((GTypeInstance*)a)->g_class)->value_a

**Instanzvariablen**

a->value_b

B *b;

((GTypeInstance*)b)->g_class

((GTypeInstance*)b)->g_class->g_type

((AClass*)((GTypeInstance*)b)->g_class)->value_a

b->string_b

((A*) b)->value_a

# 1.3 GObject

## Instanz-Struktur

- (Instanz-Struktur Oberklasse)
- Instanzvariablen
- Properties

```
struct _Gobject {
  GTypeInstance     g_type_instance;
  volatile guint    ref_count;
  GData            *qdata;
};
```

```
typedef struct _Gobject GObject;
```

## Klassen-Struktur

- (Klassen-Struktur Oberklasse)
- Klassenvariablen
- virtuelle Methoden
- Signale

```
struct  _GobjectClass {
  GTypeClass   g_type_class;
  GSList      *construct_properties;

  GObject* (*constructor)    (Gtype type, guint n_construct_properties,
                              GObjectConstructParam *construct_properties);
  void     (*set_property)   (GObject *object, guint property_id,
                              const GValue *value, GparamSpec *pspec);
  void     (*get_property)   (GObject *object, guint property_id,
                              GValue *value, GparamSpec *pspec);
  void     (*dispose)        (GObject *object);
  void     (*finalize)       (GObject *object);
  void     (*dispatch_properties_changed) (GObject *object, guint n_pspecs,
                                           GParamSpec  **pspecs);
  void     (*notify)         (GObject *object, GParamSpec *pspec);
  /* called when done constructing */
  void     (*constructed)    (GObject *object);
  ...
```

# 1.4 Vererbung

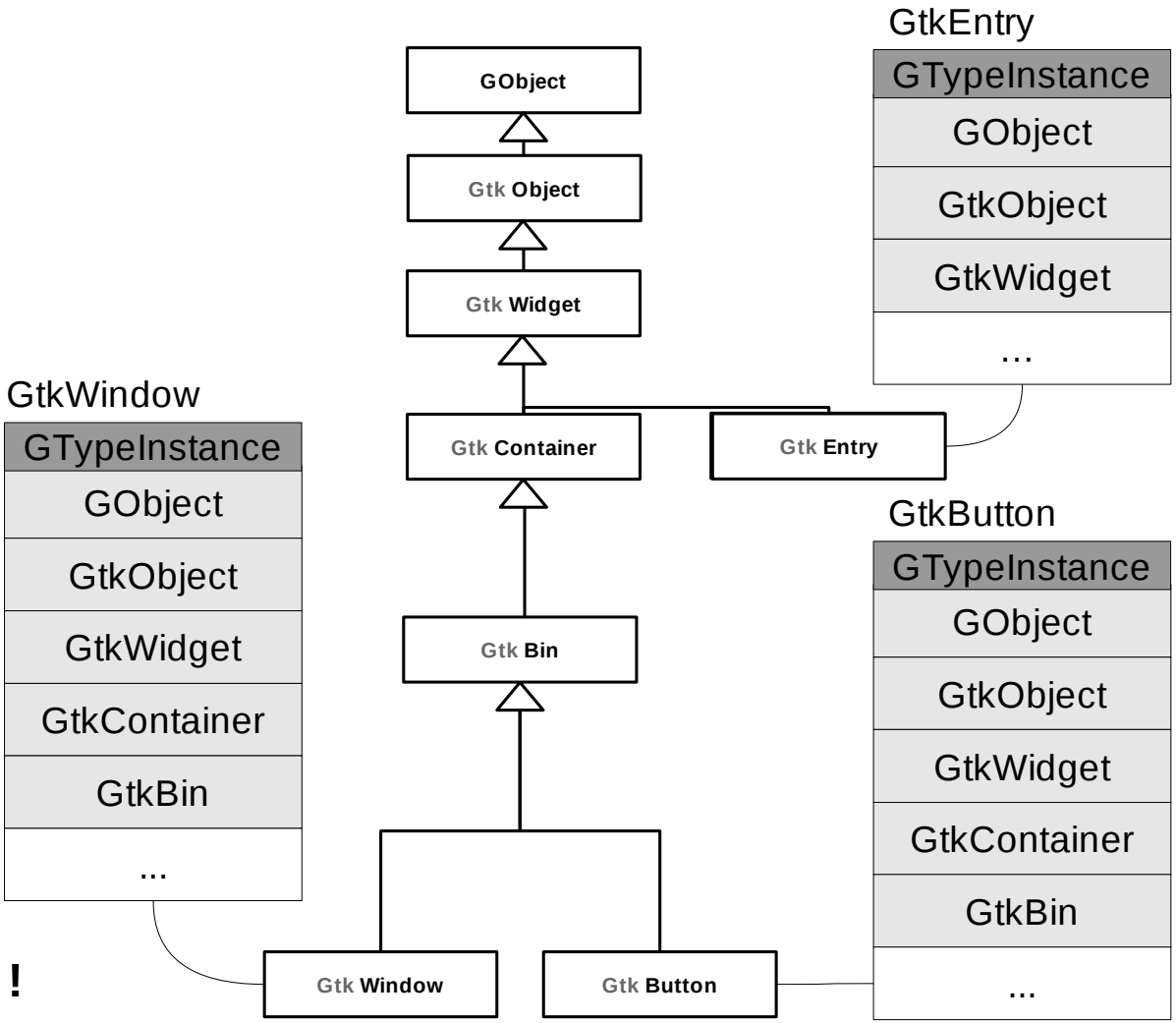
## ... hier für Instanz-Strukturen

```
struct _GtkWidget {
  GtkObject object;
  guint16 GSEAL (fprivate_flags);
  ...
};
```

```
struct _GtkContainer {
  GtkWidget widget;
  GtkWidget *GSEAL (focus_child);
  ...
};
```

```
struct _GtkBin {
  GtkContainer container;
  GtkWidget *GSEAL (child);
};
```

```
struct _GtkWindow {
  GtkBin bin;
  gchar *GSEAL (title);
  ...
};
```




## ... für Klassen-Strukturen analog !

# 2 Typ / Klasseninformation

## Typkennung / Klasse

- **TYPE_...**
  (g_type_register_... ())
- **..._GET_CLASS (instance)**
  G_TYPE_INSTANCE_GET_CLASS(instance, g_type, c_type)

## Casting-Makros

- **... (instance)**
  G_TYPE_CHECK_INSTANCE_CAST(instance, g_type, c_type)
- **..._CLASS (class)**
  G_TYPE_CHECK_CLASS_CAST(g_class, g_type, c_type)

## Test-Makros

- **IS_... (instance)**
  G_TYPE_CHECK_INSTANCE_TYPE(instance, g_type)
- **IS_..._CLASS (class)**
  G_TYPE_CHECK_CLASS_TYPE(g_class, g_type)

# 2.1 Typ-Registrierung

## Registrierung

- g_type_register_static ()
- g_type_register_static_simple ()

- g_type_register_dynamic ()
- g_type_register_fundamental ()

```
Gtype  g_type_register_static_simple  ( Gtype parent_type, const gchar *type_name,
                                        guint class_size, GClassInitFunc class_init,
                                        guint instance_size, GInstanceInitFunc instance_init,
                                        GtypeFlags flags );

    void (*GClassInitFunc)      ( gpointer g_class, gpointer class_data );
    void (*GinstanceInitFunc)  ( GTypeInstance *instance, gpointer g_class);

    typedef enum {
          G_TYPE_FLAG_ABSTRACT            = (1 << 4),
          G_TYPE_FLAG_VALUE_ABSTRACT   = (1 << 5)
    } GTypeFlags;
```

# 2.2 Abkürzungen

## Makro

- G_DEFINE_TYPE (TN, t_n, T_P)
  … WITH_CODE (TN, t_n, T_P, _C_)

| | |
|---|---|
| TN | TypeName |
| t_n | type_name |
| T_P | TYPE_PARENT |
| _f_ | flags : GTypeFlags |
| _C_ | code |

## zu implementieren

- static void t_n_init (TN *self)
- static void t_n_class_init (TN *klass)

## … weitere Abküzungen

- G_DEFINE_TYPE_EXTENDED (TN, t_n, T_P, _f_, _C_)
- G_DEFINE_ABSTRACT_TYPE[1)] (TN, t_n, T_P)
- G_DEFINE_INTERFACE[1)] (TN, t_n, T_P)

- G_IMPLEMENT_INTERFACE (TYPE_IFACE, iface_init)

- G_DEFINE_BOXED_TYPE[1)] (TypeName, type_name, copy_func, free_func)
- G_DEFINE_POINTER_TYPE[1)] (TypeName, type_name

[1)] …_WITH_CODE (… , _C_)

# 2.3 … und Umwege

```
typedef struct _GtypeInfo GTypeInfo;

struct _GtypeInfo {
  /* interface types, classed types, instantiated types */
  guint16             class_size;

  GbaseInitFunc       base_init;
  GbaseFinalizeFunc   base_finalize;

  /* classed types, instantiated types */
  GClassInitFunc            class_init;
  GClassFinalizeFunc        class_finalize;
  gconstpointer             class_data;

  /* instantiated types */
  guint16                   instance_size;
  guint16                   n_preallocs;
  GInstanceInitFunc         instance_init;

  /* value handling */
  const GtypeValueTable     *value_table;
};
```

```
void      (*GbaseInitFunc)        (gpointer g_class);
void      (*GbaseFinalizeFunc)    (gpointer g_class);

void      (*GclassInitFunc)       (gpointer g_class, gpointer class_data);
void      (*GclassFinalizeFunc)   (gpointer g_class, gpointer class_data);

void      (*GinstanceInitFunc)    (GTypeInstance *instance, gpointer g_class);
```

# 3 Properties

## Registrierung

void g_object_class_install_property ( GObjectClass *oclass, guint property_id,
GParamSpec *pspec);

### GparamSpec

- g_param_spec_int ()
- g_param_spec_string ()
- g_param_spec_boolean ()
- ...

## Implementierung

… überschreibe geerbte Methoden:

- set_property ()
- get_property ()

```
struct  _GobjectClass {
 ...
  void (*set_property) (GObject *object, guint property_id,
                        const GValue *value, GparamSpec *pspec);
  void (*get_property) (GObject *object, guint property_id,
                        GValue *value, GparamSpec *pspec);
  ...
```

# 3.1 Beispiel

```
enum {
    PROP_0,      // dummy
    PROP_SPEED,
};
```

---

```
static void fahrzeug_set_property (GObject *object, guint property_id, const GValue *value, GParamSpec *pspec) {
    Fahrzeug *self = FAHRZEUG (object);
    switch (property_id) {
    case PROP_SPEED:
        self->speed = g_value_get_int (value); break;
    default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, property_id, pspec); break;
    }
}
```

---

```
static void fahrzeug_get_property (GObject *object, guint property_id, GValue *value, GParamSpec *pspec) {
    Fahrzeug *self = FAHRZEUG (object);
    switch (property_id) {
    case PROP_SPEED:
        g_value_set_int (value, self->speed); break;
    default:
        G_OBJECT_WARN_INVALID_PROPERTY_ID (object, property_id, pspec); break;
    }
}
```

# 4 Parameter und Werte

## GValue

- **generisch**

```
Gvalue*   g_value_init     (GValue *value, GType g_type);
void      g_value_copy     (const GValue *src_value, GValue *dest_value);
GValue *  g_value_reset    (GValue *value);
void      g_value_unset    (GValue *value);
```

- **typspezifisch**

```
void      g_value_set_int  (GValue *value, gint v_int);
gint      g_value_get_int  (const GValue *value);
…
```

```
typedef struct {
} GValue;
```

## GparamSpec

- **Information**

```
const gchar *   g_param_spec_get_name  (GParamSpec *pspec);
const gchar *   g_param_spec_get_nick  (GParamSpec *pspec);
const gchar *   g_param_spec_get_blurb (GParamSpec *pspec);
```

- **eigene Typhierarchie**

```
GType  g_param_type_register_static (const gchar *name, const GParamSpecTypeInfo *pspec_info);
```

```
struct _GParamSpec {
  GTypeInstance  g_type_instance;

  gchar          *name;
  GParamFlags     flags;
  GType           value_type;
  GType           owner_type;
};
```

# 4.1 GParamSpecs

## Beispiel

```
GParamSpec *   g_param_spec_int ( const gchar *name, const gchar *nick, const gchar *blurb,
                                  gint minimum, gint maximum, gint default_value,
                                  GParamFlags flags);
```

## Flags

```
typedef enum {
  G_PARAM_READABLE            = 1 << 0,
  G_PARAM_WRITABLE            = 1 << 1,
  G_PARAM_CONSTRUCT           = 1 << 2,
  G_PARAM_CONSTRUCT_ONLY      = 1 << 3,
  G_PARAM_LAX_VALIDATION      = 1 << 4,
  G_PARAM_STATIC_NAME         = 1 << 5,
  G_PARAM_STATIC_NICK         = 1 << 6,
  G_PARAM_STATIC_BLURB        = 1 << 7,

  G_PARAM_DEPRECATED          = 1 << 31
} GParamFlags;
```

## … weitere Generatoren

- g_param_spec_char (…)
- g_param_spec_uchar (…)
- g_param_spec_uint (…)
- …
- g_param_spec_object (…)
- …

# 5 Typinformation

… nur ein kleiner Ausschnitt. [1)]

## Makros

```
G_TYPE_FROM_INSTANCE      (instance)
G_TYPE_FROM_CLASS         (g_class)
G_TYPE_FROM_INTERFACE     (g_iface)
```

## Funktionen

```
GTypeInstance *  g_type_create_instance        (GType type);
void             g_type_free_instance          (GTypeInstance *instance);

const gchar *    g_type_name                   (GType type);
GType            g_type_from_name              (const gchar *name);
GType            g_type_parent                 (GType type);

guint            g_type_depth                  (GType type);
gboolean         g_type_is_a                   (GType type, GType is_a_type);

GType *          g_type_children               (GType type, guint *n_children);

GType *          g_type_interfaces             (GType type, guint *n_interfaces);
GType *          g_type_interface_prerequisites  (GType interface_type, guint *n_prerequisites);

void             g_type_query                  (GType type, GTypeQuery *query);
```

[1)] viel mehr unter: http://developer.gnome.org/gobject/stable/gobject-Type-Information.html

# 5.1 … ausserdem haben wir geerbt

## Properties

```
GParamSpec *   g_object_class_find_property     (GObjectClass *oclass, const gchar *property_name);
GParamSpec **  g_object_class_list_properties   (GObjectClass *oclass, guint *n_properties);

void           g_object_set_property            (GObject *object, const gchar *property_name, const GValue *value);
void           g_object_get_property            (GObject *object, const gchar *property_name, GValue *value);
void           g_object_set                     (gpointer object, const gchar *first_property_name, ...);
void           g_object_get                     (gpointer object, const gchar *first_property_name, ...);
```

## Erzeugung

```
gpointer       g_object_new                     (GType object_type, const gchar *first_property_name, ...);
gpointer       g_object_newv                    (GType object_type, guint n_parameters, GParameter *parameters);
```

## Referenzzähler

```
gpointer       g_object_ref                     (gpointer object);
void           g_object_unref                   (gpointer object);
void           g_clear_object                   (volatile GObject **object_ptr);

gboolean       g_object_is_floating             (gpointer object);
gpointer       g_object_ref_sink                (gpointer object);
void           g_object_force_floating          (GObject *object);
```

# 5.2 … die brauchen wir erst nächstes Mal !

## Interfaces

```
void          g_object_class_override_property (GObjectClass *oclass, guint property_id, const gchar *name);

void          g_object_interface_install_property     (gpointer g_iface, GParamSpec *pspec);
GParamSpec *  g_object_interface_find_property        (gpointer g_iface, const gchar *property_name);
GParamSpec ** g_object_interface_list_properties      (gpointer g_iface, guint *n_properties_p);
```

## Signale

```
void          g_object_notify              (GObject *object, const gchar *property_name);
void          g_object_notify_by_pspec     (GObject *object, GParamSpec *pspec);
void          g_object_freeze_notify       (GObject *object);
void          g_object_thaw_notify         (GObject *object);

gpointer      g_object_connect             (gpointer object, const gchar *signal_spec, ...);
void          g_object_disconnect          (gpointer object, const gchar *signal_spec, ...);
```