How GNOME supports multiple programming languages
(The story of GObject-Introspection)

Federico Mena Quintero
(he/him)
federico@gnome.org
@federicomena@mstdn.mx

GNOME Onboard
Africa Virtual 2020
Q: Why are assembly programmers always wet?

A: because they work below C-level.
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk

win = Gtk.Window()
win.set_title("Hello World")

label = Gtk.Label.new("I am written in Python")
win.add(label);

win.show_all()
Gtk.main()
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk

win = Gtk.Window()
win.set_title("Hello World")

label = Gtk.Label.new("I am written in Python")
win.add(label);

win.show_all()
Gtk.main()
imports.gi.versions.Gtk = "3.0";
const Gtk = imports.gi.Gtk;

Gtk.init(null);

let win = new Gtk.Window();
win.set_title("Hello World");

let label = Gtk.Label.new("I am written in Javascript")
win.add(label);

win.show_all();
Gtk.main();
```javascript
import gi;

gi.version = "3.0";
const Gtk = gi.Gtk;

Gtk.init(null);

let win = new Gtk.Window();
win.set_title("Hello World");

let label = Gtk.Label.new("I am written in Javascript")
win.add(label);

win.show_all();
Gtk.main();
```
use gtk::{self, prelude::*};

fn main() {
    gtk::init().unwrap();

    let win = Window::new(WindowType::Toplevel);
    win.set_title("Hello World");

    let label = Label::new(Some("I am written in Rust"));
    win.add(&label);

    win.show_all();
    gtk::main();
}
use gtk::{self, prelude::*};

fn main() {
    gtk::init().unwrap();

    let win = Window::new(WindowType::Toplevel);
    win.set_title("Hello World");

    let label = Label::new(Some("I am written in Rust"));
    win.add(&label);

    win.show_all();
    gtk::main();
}
But GTK is written in C

GtkWidget*
gtk_window_new (GtkWindowType type)
{
    ...
}

void
gtk_window_set_title (GtkWidget  *window,
                      const gchar  *title)
{
    ...
}
Object-oriented programming in C

```c
void
gtk_widget_show_all (GtkWidget *widget)
{
    GtkWidgetClass *class;

    class = GTK_WIDGET_GET_CLASS (widget);

    if (class->show_all)
        class->show_all (widget);
}

void
gtk_container_add (GtkContainer *container,
                    GtkWidget    *widget)
{
    ...
}
```

class GtkWidget:
    def show_all(self):
        ...

class GtkContainer:
    def add(self, widget):
        ...
```
Object-oriented programming in C

```c
void gtk_widget_show_all (GtkWidget *widget) {
    GtkWidgetClass *class;

    class = GTK_WIDGET_GET_CLASS (widget);

    if (class->show_all)
        class->show_all (widget);
}

void gtk_container_add (GtkContainer *container,
                        GtkWidget   *widget)
{
    ...
}
```
Object-oriented programming in C

```c
void gtk_widget_show_all (GtkWidget *widget)
{
    GtkWidgetClass *class;

    class = GTK_WIDGET_GET_CLASS (widget);

    if (class->show_all)          // If the method is implemented
        class->show_all (widget);  // call the method
}

void gtk_container_add (GtkContainer *container,
                        GtkWidget    *widget)
{
    ...
}
```
Extending Python from C

```c
static PyObject *
say_hello(PyObject *self, PyObject *args)
{
    const char *name;

    if (!PyArg_ParseTuple(args, "s", &name))
        return NULL;

    printf("hello %s\n", name);

    Py_INCREF(Py_None);
    return Py_None;
}
```

everything is a PyObject
static PyObject *
say_hello(PyObject *self, PyObject *args)
{
    const char *name;

    if (!PyArg_ParseTuple(args, "s", &name))
        return NULL;

    printf("hello %s\n", name);

    Py_INCREF(Py_None);
    return Py_None;
}
static PyObject *
say_hello(PyObject *self, PyObject *args)
{
    const char *name;

    if (!PyArg_ParseTuple(args, "s", &name))
        return NULL;

    printf("hello %s\n", name);

    Py_INCREF(Py_None);
    return Py_None;
}
static PyObject *
say_hello(PyObject *self, PyObject *args)
{
    const char *name;

    if (!PyArg_ParseTuple(args, "s", &name))
        return NULL;

    printf("hello %s\n", name);

    Py_INCREF(Py_None);
    return Py_None;
}
class Label(Widget):
    def set_text(self, text):
        ...

label.set_text("hello")
A hand-written wrapper for gtk_label_set_text()

```c
static PyObject *
wrap_label_set_text(PyObject *self, PyObject *args)
{
    GtkWidget *label;
    const char *text;

    label = get_gtklabel_from_self(self);

    if (!PyArg_ParseTuple(args, "s", &text))
        return NULL;

    gtk_label_set_text(label, text);

    Py_INCREF(Py_None);
    return Py_None;
}
```
What is a language binding?

- A language-specific wrapper for a library.
- Ideally make it feel like a library written for that language.
Back in 1997

- Hand-written bindings.
- C++, Objective-C, Python (?), Perl
- Wrap each function by hand, as needed.
- Error-prone, lots of duplication.
1998 / 1999

- Red Hat rewrites their installer in Python.
- With a GTK interface.
- Lots of work put into PyGTK.
Machine-readable descriptions

(class GtkWidget
   ;; void gtk_widget_show (GtkWidget *widget);
   (method show
       (args nil)
       (retval nil))

   ;; void gtk_widget_hide (GtkWidget *widget);
   (method hide
       (args nil)
       (retval nil)))
A method with one argument

(class GtkContainer
  ;; void gtk_container_add (GtkContainer *container,
  ;; GtkWidget   *child);

  (method add
    (args GtkWidget)
    (retval nil)))
typedef struct {
    guchar r;
    guchar g;
    guchar b;
} GdkColor;

(struct GdkColor
    (field r (type 'guchar))
    (field g (type 'guchar))
    (field b (type 'guchar)))
GObject

• An object and type system for C.

• Register types at runtime.

• Lots of things for memory management.
  – How does a value for a type get copied?
  – Are values reference-counted?
  – Etc.
Example: Signals and their argument types

```
g_signal_new ("button-press-event",
    gtk_widget_get_type(), /* type of object for which this
    * signal is being created
    */
    ...
    G_TYPE_BOOLEAN,  /* type of return value */
    1,             /* number of arguments */
    GDK_TYPE_EVENT); /* type of first and only argument */

```

```
g_signal_query(class, signal)
```
Static vs. Dynamic

- Static languages (Rust, C++)
  - Cannot `g_signal_query()` and construct classes at runtime
  - Everything needs to be pre-generated

- Dynamic languages (Python, Javascript)
  - Can generate new classes and methods at runtime
GNOME 2

- Lots of machine-readable descriptions of APIs.
- Cut&paste from binding to binding.
- Lots of manual work on top of automation.
GNOME 3

- GObject-Introspection.
- Write a library in C.
- Generate a machine-readable API/ABI description from it.
- Bindings are language-specific glue from the machine-readable descriptions to the C library.
/**
 * gtk_widget_get_parent:
 * @widget: a GtkWidget
 *
 * Returns the parent container of @widget.
 *
 * Returns: (transfer none) (nullable): the parent container of @widget, or %NULL
 **/

GtkWidget *
gtk_widget_get_parent (GtkWidget *widget)
{
    ...
}

/**
 * gtk_widget_get_parent:
 * @widget: a #GtkWidget
 *
 * Returns the parent container of @widget.
 *
 * Returns: (transfer none) (nullable): the parent container of @widget, or %NULL
 **/

GtkWidget *
gtk_widget_get_parent (GtkWidget *widget)
{
    ...
}

Inline documentation written in comments
/**
 * gtk_widget_get_parent:
 * @widget: a GtkWidget
 *
 * Returns the parent container of @widget.
 *
 * Returns: (transfer none) (nullable): the parent container of @widget, or %NULL
 **/

GtkWidget *
gtk_widget_get_parent (GtkWidget *widget)
{
    ...
}

(transfer none) - no extra reference count is added to the returned object
gtk_widget_get_parent:

@widget: a GtkWidget

Returns the parent container of @widget.

Returns: (transfer none) (nullable): the parent container of @widget, or %NULL

GtkWidget *
gtk_widget_get_parent (GtkWidget *widget)
{
    ...
}

(nullable) - may return NULL
<repository version="1.2" ...>

<namespace name="Gtk"
  version="3.0"
  shared-library="libgtk-3.so.0,libgdk-3.so.0"
  c:identifier-prefixes="Gtk"
  c:symbol-prefixes="gtk">
GtkEntry class

The #GtkEntry widget is a single line text entry widget. A fairly large set of key bindings are supported by default. If the entered text is longer than the...
What does GtkEntry implement?

<implements name="Atk.ImplementorIface"/>
<implements name="Buildable"/>
<implements name="CellEditable"/>
<implements name="Editable"/>
const gchar*  
gtk_entry_get_text (GtkEntry *entry);

<method name="get_text" c:identifier="gtk_entry_get_text">
  <doc>Retrieves the contents of the entry widget. ... </doc>

  <return-value transfer-ownership="none">
    <type name="utf8" c:type="const gchar*"/>
  </return-value>

  <parameters>
    <instance-parameter name="entry" transfer-ownership="none">
      <type name="Entry" c:type="GtkEntry*"/>
    </instance-parameter>
  </parameters>
</method>
typedef struct {
    int x, y;
    int width, height;
} GdkRectangle;

<record name="Rectangle"
  c:type="GdkRectangle"
  glib:type-name="GdkRectangle"
  glib:get-type="gdk_rectangle_get_type"
  c:symbol-prefix="rectangle">
  <field name="x" writable="1">
    <type name="gint" c:type="int"/>
  </field>
  <field name="y" writable="1">
    <type name="gint" c:type="int"/>
  </field>
  ...
</record>
Compiled descriptions

- .gir files are huge XML files!
  - Look in /usr/share/gir-1.0/

- Don’t want to parse megabytes at startup of every application.

- *.gir -> compiled to *.typelib
  - Look in /usr/lib64/girepository-1.0/
  - can be memory-mapped and shared across processes
The Unwritten Future

Here's a rant (opinion?) that I'll turn in to a blog post soon.

I think Rust should have a third ABI, not repr(C) or repr(Rust), and it should be used to slowly phase out the C ABI for a better one. Want to hear more of my unreasonable thoughts about ABIs? Read on.

4:22 PM · Jul 23, 2020 · Twitter Web App

https://twitter.com/bitshiftmask/status/1286411475658178561
GNOME already has part of this!

Fifth: The ABI should include the ability to encode information about types and layouts, not just function names. This should be used at compile/runtime to provide type information. Compilers should be able to import ABI libraries directly, or provide codegen tools to get headers.

4:35 PM · Jul 23, 2020 · Twitter Web App

Work on tools, save the world!
Thanks

- federico@gnome.org