The GNOME™ Conference
GUADEC

Writing applications using GTK 4 & Rust

Bilal Elmoussaoui bilelmoussaoui@gnome.org
Slides at `https://github.com/bilelmoussaoui/guadec-2021`

Introduction
What's Rust?

A language empowering everyone to build reliable and efficient software

- Memory / Thread safety
- Great docs integration with rustdoc
- Awesome tooling: rust-analyzer, clippy, rustfmt
- IDE support: GNOME Builder, VSCode and others
- Flatpak support with rust-stable/rust-nightly extensions

Introduction
What's GTK?

- GTK is a C library that uses OOP through GObject
- Large choice of languages you can use thanks to GObject Introspection
- Rust bindings are also generated using the introspection data

GTK 4 bindings
The Rust bindings

Composed of two parts:

- Auto: automatically generated code using a Rust tool called gir
- Manual: mostly for subclassing support

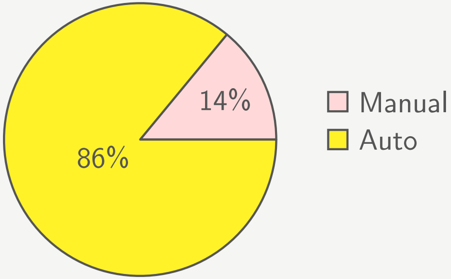

Figure: Total lines of Rust code in gtk4-rs crates

GTK 4 bindings
Hello world

- Add the gtk4 crate as a dependency

```
[dependencies]
gtk = {package = "gtk4", version = "0.2"}
```

- Create your first window

```
use gtk::prelude::*;

fn main() {
    let app = gtk::Application::new(Some("org.guadec.gtk-rs.talk"), Default::default());
    app.connect_activate(build_ui);
    app.run();
}

fn build_ui(app: &gtk::Application) {
    let window = gtk::ApplicationWindow::builder()
        .title("Hello world")
        .application(app)
        .build();
    let button = gtk::Button::with_label("Click me!");

    window.set_child(Some(&button));
    window.show();
}
```

GTK 4 bindings

Current status

- A WIP Book (https://gtk-rs.org/gtk4-rs/git/book/) to help you get started with writing GTK & Rust applications
- Approximately 1600 commits by 29 contributors
- Various examples (contributions are welcome!)
- Used in a ton of apps: Shortwave, Video Trimmer, Helvum...
- The majority of the C API is covered
- Composite Templates
- Almost complete subclassing support

GTK 4 bindings

Custom types

Subclassing a type or implementing an interface is replaced by implementing a trait

```rust
// imp.rs, equivalent of CustomWidgetPrivate in C
use gtk::{gdk, graphene, glib};
use gtk::{prelude::*, subclass::prelude::*};

#[derive(Debug, Default)]
pub struct CustomWidget {}

#[glib::object_subclass]
impl ObjectSubclass for CustomWidget {
    const NAME: &'static str = "CustomWidget";
    type Type = super::CustomWidget;
    type ParentType = gtk::Widget;
}

impl ObjectImpl for CustomWidget {}

impl WidgetImpl for CustomWidget {
    fn snapshot(&self, widget: &Self::Type, snapshot: &gtk::Snapshot) {
        snapshot.append_color(
            &gdk::RGBA::black(),
            &graphene::Rect::new(0.0, 0.0, widget.width() as f32, widget.height() as f32),
        );
    }
}
```

GTK 4 bindings
Custom types

We can now define our public API

```rust
// mod.rs, register the new type "CustomWidget"
mod imp;
use gtk::glib;

glib::wrapper! {
    pub struct CustomWidget(ObjectSubclass<imp::CustomWidget>) @extends gtk::Widget;
}

impl CustomWidget {
    pub fn new() -> Self {
        glib::Object::new(&[]).unwrap()
    }
}
```

GTK 4 bindings
Composite templates

Better code organization: smaller widgets

```
<!-- custom_widget.ui -->
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <template class="CustomWidget" parent="GtkWidget">
    <child>
      <object class="GtkLabel" id="label">
        <property name="label">Hello world!</property>
      </object>
    </child>
  </template>
</interface>
```

GTK 4 bindings
Composite templates

```rust
// imp.rs
#[derive(Debug, Default, CompositeTemplate)]
#[template(file = "custom_widget.ui")] // file, resource or string
pub struct CustomWidget {
    #[template_child]
    pub label: TemplateChild<gtk::Label>,
}

#[glib::object_subclass]
impl ObjectSubclass for CustomWidget {
    const NAME: &'static str = "CustomWidget";
    type Type = super::CustomWidget;
    type ParentType = gtk::Widget;

    fn class_init(klass: &mut Self::Class) {
        Self::bind_template(klass);
    }

    fn instance_init(obj: &glib::subclass::InitializingObject<Self>) {
        obj.init_template();
    }
}

impl ObjectImpl for CustomWidget {
    fn dispose(&self, obj: &Self::Type) {
        self.label.unparent();
    }
}

impl WidgetImpl for CustomWidget {}
```

Ecosystem

Bindings:

- Multimedia: GStreamer, PipeWire, GEGL
- GUI: GtkSourceView, Libadwaita
- Core: Cairo, Pango, GdkPixbuf, Graphene, GLib/GIO
- Other: Tracker

Rust:

- zbus
- librsvg
- plenty of other crates on crates.io

Ecosystem
Portals

Modern applications should use portals. ashpd provides a zbus based wrapper

- Integrate with either GTK 3 / GTK 4 (maybe other toolkits in the future?)
- Async API
- Comes with a WIP demo built using gtk4-rs

Example: Ask the compositor to pick a color

```rust
use ashpd::desktop::screenshot;

async fn run() -> Result<(), ashpd::Error> {
    let color = screenshot::pick_color(Default::default()).await?;
    println!("({}, {}, {})", color.red(), color.green(), color.blue());

    Ok(())
}
```

The future

- GtkBuilderScope integration
- Proc macro for generating GObject properties / signals, GActions
- Easier integration with other future executors: tokio / async-std for example
- Improve the generated docs
- Reduce boilerplate in subclassing code

Reach out

- Website: `https://gtk-rs.org`
- GitHub: `https://github.com/gtk-rs`
- Matrix: `https://matrix.to/#/#rust:gnome.org`

Thank you

Any questions?