

Oxidising GStreamer

Rust out your multimedia!

GStreamer Conference 2017

22 October 2017, Prague

Sebastian 'slomo' Dröge

< sebastian@centricular.com >



Introduction

Who?

What?



+



What is Rust?

Type-safe, memory-safe *systems* programming
language

Low-level programming with a high-level look and
feel

Web, game, server/network, OS, microcontroller, ...
development

Used and backed by the industry

<https://www.rust-lang.org/friends.html>

Rust is what C++ should have been

Why Rust?

Writing safe C/C++ code is hard

Let the compiler help you writing *correct* and *fast*
code

Escape hatch: unsafe

- opt-in
- can do *everything* C can do

Feels like an high-level language,
not glorified assembly

Why should we care?

Parsing of complicated media formats

... from untrusted sources!

Multi-threading is hard

... especially in C!

Programming like it's 2017

But: *Not* a magic bullet

All non-trivial code has bugs

Status Last Year

GStreamer bindings

- Manually written
- Not integrating well with other Rust code
- Required usage of unsafe code
- Diverging from GStreamer concepts
- Incomplete

GStreamer crate for writing plugins

- Manually written
- Lots of missing features
- Incomplete and difficult to extend

A lot has happened

Let's talk about that in detail now

Writing GStreamer Applications in Rust

New GStreamer bindings

- (mostly) autogenerated from GI
- No unsafe code for apps
- Covering almost all of core and others

Idiomatic Rust

(mostly)

... but still directly mapping GStreamer
API/concepts

Objects

- Semi-automatic, safe reference counting
- Inheritance via traits
- Compiler-enforced thread-safety
- All the standard GStreamer & GObject API

MiniObjects

- Compiler-enforced writability / COW
- Feel like proper Rust types
 - Incl. caps/structure fields with special types

What's missing?

- GstMemory, GstAllocator, GstMeta, GstCapsFeatures
- Typefinders
- GstControlBinding and related
- PbUtils, other libraries incomplete

Is it useful?

Yes!

**Let's look at some code
snippets**

Element creation

```
let pipeline = gst::Pipeline::new(None);

let src = gst::ElementFactory::make("filesrc", None)
    .ok_or(MyError::ElementNotFound("filesrc"))?;

let dbin = gst::ElementFactory::make("decodebin", None)
    .ok_or(MyError::ElementNotFound("decodebin"))?;
```

Caps creation

```
let caps = gst::Caps::new_simple(  
    "video/x-raw",  
    &[  
        ("format", "BGRA"),  
        ("width", &(1080i32)),  
        ("height", &(720i32)),  
        ("framerate", &gst::Fraction::new(30, 1)),  
    ],  
);
```

Element Linking

```
gst::Element::link_many([&src, &decodebin])?;  
element1.link_pads("src", &element2, "sink")?;
```

pad-added signal

```
let pipeline = _;  
decodebin.connect_pad_added(move |dbin, src_pad| {  
    let sink = gst::ElementFactory::make(  
        "fakesink",  
        None  
    ).unwrap();  
    pipeline.add(&sink);  
  
    let sink_pad = sink.get_static_pad("sink").unwrap();  
    src_pad.link(&sink_pad);  
  
    sink.sync_state_with_parent();  
});
```

Buffer mapping

```
let mut buffer = gst::Buffer::with_size(320*240*4).unwrap();
{
    let buffer = buffer.get_mut().unwrap();
    let mut data = buffer.map_writable().unwrap();

    for p in data.as_mut_slice().chunks_mut(4) {
        p[0] = b; p[1] = g;
        p[2] = r; p[3] = 0;
    }
}
```

Bus & Messages

```
while let Some(msg) = bus.timed_pop(gst::CLOCK_TIME_NONE) {
    use gst::MessageView;

    match msg.view() {
        MessageView::Eos(..) => break,
        MessageView::Error(err) => {
            println!(
                "Error from {}: {} ({:??})",
                msg.get_src().get_path_string(),
                err.get_error(),
                err.get_debug()
            );
            break;
        }
    }
    => ()
```

AppSrc

```
let appsrc = _;  
thread::spawn(move || {  
    for i in 0..100 {  
        let buffer = _;  
        if appsrc.push_buffer(buffer) != gst::FlowReturn::Ok {  
            break;  
        }  
    }  
    appsrc.end_of_stream();  
});
```

AppSink

```
appsink.set_callbacks(gst_app::AppSinkCallbacks::new(  
  /* eos */  
  |_| {},  
  
  /* new_preroll */  
  |_| gst::FlowReturn::Ok,  
  
  /* new_sample */  
  |appsink| {  
    let sample = match appsink.pull_sample() {  
      None => return gst::FlowReturn::Eos,  
      Some(sample) => sample,  
    };  
  
    let buffer = match sample.get_buffer() {
```


Some Links

- Bindings: <https://github.com/sdroege/gstreamer-rs>
- Examples: [gstreamer-rs/examples](#)
- Tutorials: [gstreamer-rs/tutorials](#)

Writing GStreamer Plugins in Rust

Object / Element infrastructure

- Sub-classing, virtual methods
- Properties
- Manually written on top of the bindings
 - To be improved
- No unsafe Rust for implementors
- Goal: Create elements by implementing traits only

Existing base classes

- Element
- BaseSrc, BaseSink, BaseTransform
- Soon hopefully: VideoDecoder
- Panics cause error messages on the bus

Existing elements

- FLV demuxer
- HTTP source
- File source/sink
- Amazon S3 source/sink
- Audio echo
- Soon hopefully: (animated) GIF decoder

Simplified traits

- Source, sink, demuxer
- Experiments for nicer base classes

Status?

- Still in its early stages
- Ready to start getting used now
- Missing features to be added when needed

It's the perfect time to write your next
GStreamer element in Rust

**Let's look at some code
snippets**

Element registration

```
pub fn register(plugin: &gst::Plugin) {  
    let type_ = register_type(AudioEchoStatic);  
    gst::Element::register(plugin, "rsaudioecho", 0, type_);  
}
```

Element registration (2)

```
struct AudioEchoStatic;

impl ImplTypeStatic<RsBaseTransform> for AudioEchoStatic {
    fn get_name(&self) -> &str {
        "AudioEcho"
    }

    fn new(&self, element: &RsBaseTransform)
        -> Box<BaseTransformImpl<RsBaseTransform>> {
        AudioEcho::init(element)
    }

    fn class_init(&self, klass: &mut RsBaseTransformClass) {
        AudioEcho::class_init(klass);
    }
}
```

Element class initialization

```
struct AudioEcho { ... }

impl AudioEcho {
    fn class_init(klass: &mut RsBaseTransformClass) {
        klass.set_metadata(...);

        let src_pad_template = gst::PadTemplate::new(
            "src",
            gst::PadDirection::Src,
            gst::PadPresence::Always,
            &caps,
        );
        klass.add_pad_template(src_pad_template);

        klass.install_properties(&PROPERTIES);
    }
}
```

Properties

```
static PROPERTIES: [Property; 4] = [  
    Property::UInt64(  
        "max-delay",  
        "Maximum Delay",  
        "Maximum delay ...",  
        (0, u64::MAX),  
        DEFAULT_MAX_DELAY,  
        PropertyMutability::ReadWrite,  
    ),  
    ...  
];
```

Properties (2)

```
impl ObjectImpl<RsBaseTransform> for AudioEcho {
    fn set_property(&self, _obj: &glib::Object,
        id: u32, value: &glib::Value) {
        let prop = &PROPERTIES[id as usize];

        match *prop {
            Property::UInt64("max-delay", ..) => {
                let mut settings = self.settings.lock().unwrap();
                settings.max_delay = value.get().unwrap();
            },
            ...
        }
    }
}
```

Caps Handling

```
impl BaseTransformImpl<RsBaseTransform> for AudioEcho {
    fn set_caps(
        &self,
        _element: &RsBaseTransform,
        incaps: &gst::Caps,
        outcaps: &gst::Caps,
    ) -> bool {
        let info = match gst_audio::AudioInfo::from_caps(incaps) {
            None => return false,
            Some(info) => info,
        };

        ...

        *self.state.lock().unwrap() = Some(State {
```

Transform

```
fn transform_ip(
    &self,
    _element: &RsBaseTransform,
    buf: &mut gst::BufferRef,
) -> gst::FlowReturn {
    let mut settings = *self.settings.lock().unwrap();

    let mut state_guard = self.state.lock().unwrap();
    let state = match *state_guard {
        None => return gst::FlowReturn::NotNegotiated,
        Some(ref mut state) => state,
    };

    let mut map = match buf.map_writable() {
        None => return gst::FlowReturn::Error,
```


Some Links

- Code: <https://github.com/sdroege/gst-plugin-rs>
- All plugins are inside that same repository currently

Future

Write more code in Rust
... and replace C code with Rust

Get more people excited and involved
... like you!

Don't write new projects in C

Thanks

Questions?

Some useful links:

<https://www.rust-lang.org>

<https://github.com/sdroege/gstreamer-rs>

<https://github.com/sdroege/gst-plugin-rs/>