Rust & Wasm

To build a browser based metronome

Vineet Naik

Rust Bangalore meetup

20 July 2024

About me

- Using Rust for about 8-9 months
- Mostly a backend dev
  - Have done full stack dev prior to 2014
- No experience with audio programming
- Hobbyist guitar player

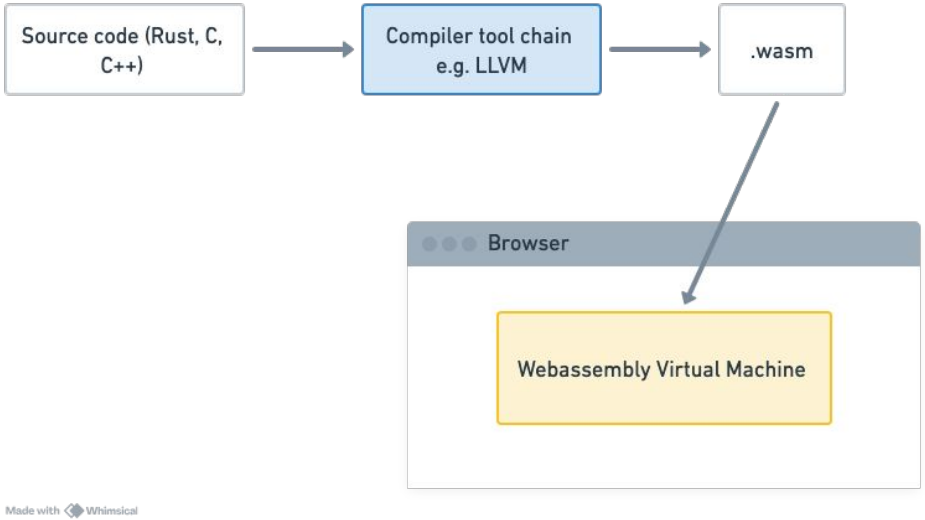
Outline

- What is Webassembly?
- Rust + Webassemby
  - Crates and tools for compiling rust to wasm
  - How to get rust code working inside the browser?
- What is a Metronome?
  - Demo
- Brief intro to WebAudio API
- Code examples
  - Debugging rust code at runtime
  - Error handling
  - Function callbacks and closures

Webassembly

What is Webassembly?

- Compact binary instruction format
- Not to be written by hand
  - Compilation target for Rust, C, C++
- Runs in modern web browsers
  - Designed to interop with Javascript both ways
  - But doesn't make any web specific assumptions
- Abbrev: *Wasm* in short



Advantages

- Near-native performance
  - Intermediate representation that’s closer to machine code than JS source code
  - No garbage collection
- Compact binary format that’s faster to transmit over network
  - Faster than even compressed JS
  - Tools available to further shrink the wasm binaries
- Can be used with existing JS code bases
  - No need to choose between Wasm and JS

Rust → Wasm → Browser

Two strategies

1. Build entire app in Rust
    a. Yew framework
2. Wasm + JS
    a. Build part of the app in Rust and integrate it into existing JS frontend

Tools

wasm-pack

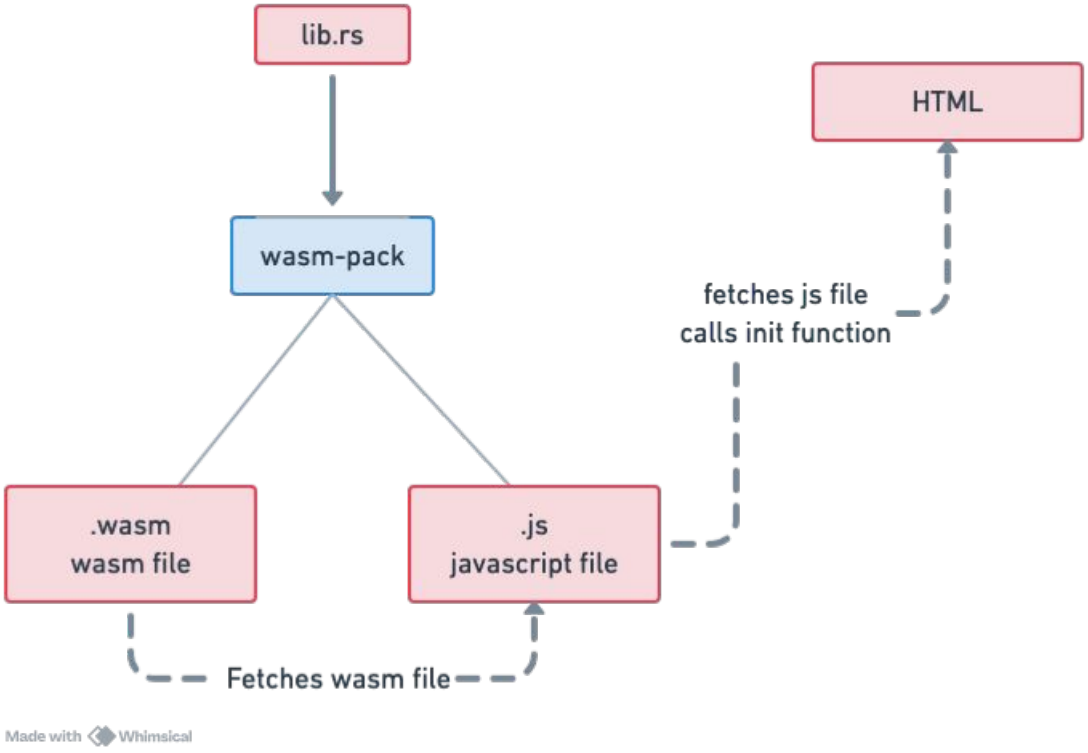
- Tool to compile Rust → Wasm
- Meant for interop with Javascript
  - Either browser (web) or nodejs

```
$ cargo install wasm-pack
```

wasm-bindgen

- A crate that wasm-pack depends on
- To provide a bridge between JS types and Rust types
- Comprises of multiple crates:
  - js_sys
    - bindings for std js objects only
  - web_sys
    - bindings for all the Web APIs that the browsers provide.
    - It's optional – all APIs are gated by cargo features
  -

How it works?




```
.
├── Cargo.lock
├── Cargo.toml
├── index.html
├── pkg
│   ├── hello_wasm.d.ts
│   ├── hello_wasm.js
│   ├── hello_wasm_bg.wasm
│   ├── hello_wasm_bg.wasm.d.ts
│   └── package.json
├── src
│   └── lib.rs
└── target
    ├── CACHEDIR.TAG
    ├── debug
    ├── release
    └── wasm32-unknown-unknown
```

Cargo.toml

```toml
[package]
name = "hello-wasm"
version = "0.1.0"
edition = "2021"

[lib]
crate-type = ["cdylib"]

[dependencies]
wasm-bindgen = "0.2"
```

```
$ wasm-pack build --target web
```

```
.
├── Cargo.lock
├── Cargo.toml
├── index.html
├── pkg
│   ├── hello_wasm.d.ts
│   ├── hello_wasm.js
│   ├── hello_wasm_bg.wasm
│   ├── hello_wasm_bg.wasm.d.ts
│   └── package.json
├── src
│   └── lib.rs
└── target
    ├── CACHEDIR.TAG
    ├── debug
    ├── release
    └── wasm32-unknown-unknown
```

lib.rs

```rust
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
extern {
    pub fn alert(s: &str);
}

#[wasm_bindgen]
pub fn greet(name: &str) {
    alert(&format!("Hello, {}!", name));
}
```

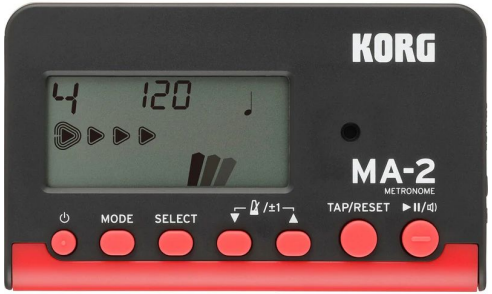
index.html

```html
<!doctype html>
<html lang="en-US">
  <head>
    <meta charset="utf-8" />
    <title>hello-wasm example</title>
  </head>
  <body>
    <script type="module">
      import init, { greet } from "./pkg/hello_wasm.js";
      init().then(() => {
        greet("WebAssembly");
      });
    </script>
  </body>
</html>
```

Building a browser based metronome

Metronome

- Device that gives an audible click or beep at a set rate
- Used by musicians during practice
- Speed is measured in Beats Per Minute or BPM (also known as tempo)
  - 60 bpm = 1 beat per sec
  - 240 bpm = 4 beats per sec
  - 150 bpm = 2.5 beats per sec

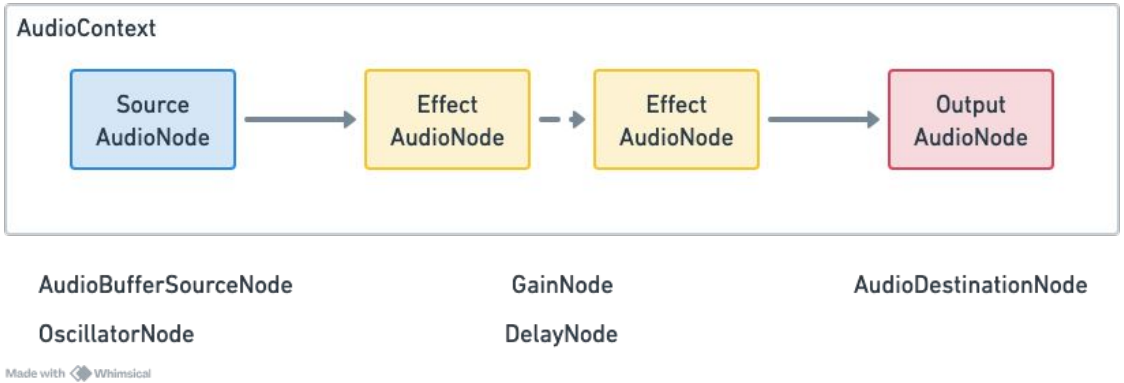


Demo

https://naiquevin.github.io/rustick/

WebAudio API

- Browser API for controlling audio
- AudioContext
  - currentTime
- AudioNode
  - Source nodes
  - Effect nodes
  - Destination nodes
- AudioParam



Code structure

main.js

```js
const m = new Metronome()
    .set_tempo(120)
    .set_time_signature(4, 4)
    .set_on_ended(() => {
      console.log('A sound was played')
    });

const timer = setInterval(
  () => m.schedule(),
  m.beat_interval
);

// Stop
// clearInterval(timer);
```

lib.rs

```rust
use wasm_bindgen::prelude::*;

#[wasm_bindgen]
pub struct Metronome {
    // ...
}

#[wasm_bindgen]
impl Metronome {
    #[wasm_bindgen(constructor)]
    pub fn new() -> Self {
        // ...
    }

    pub fn set_tempo(mut self, bpm: u16) -> Self {
        // ...
    }

    // ...

    /// Schedule clicks for the next bar
    pub fn schedule(&mut self) -> {
        // ...
    }
}
```

WebAudio code in Rust

Cargo.toml

```toml
[dependencies.web-sys]
version = "0.3.69"
features = [
    "AudioContext",
    "AudioDestinationNode",
    "AudioParam",
    "GainNode",
    "OscillatorNode"
]
```

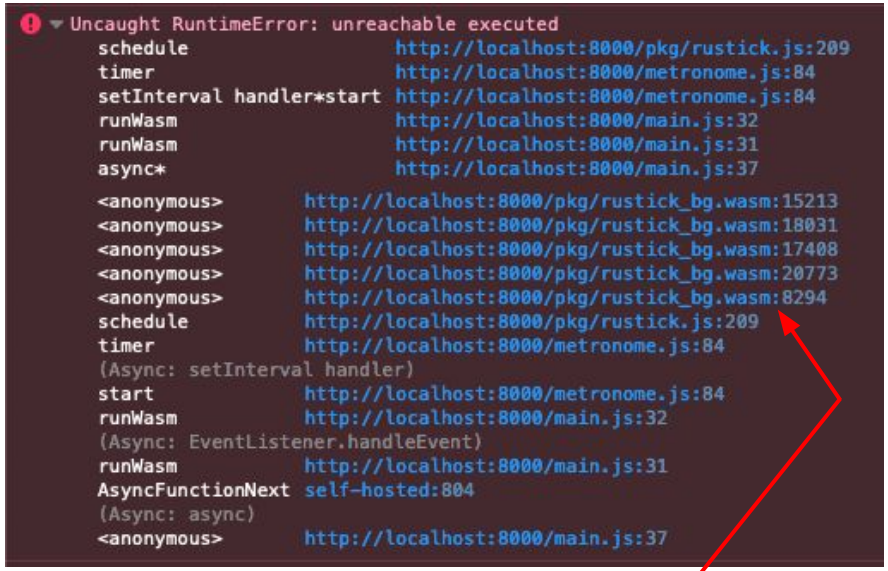
lib.rs

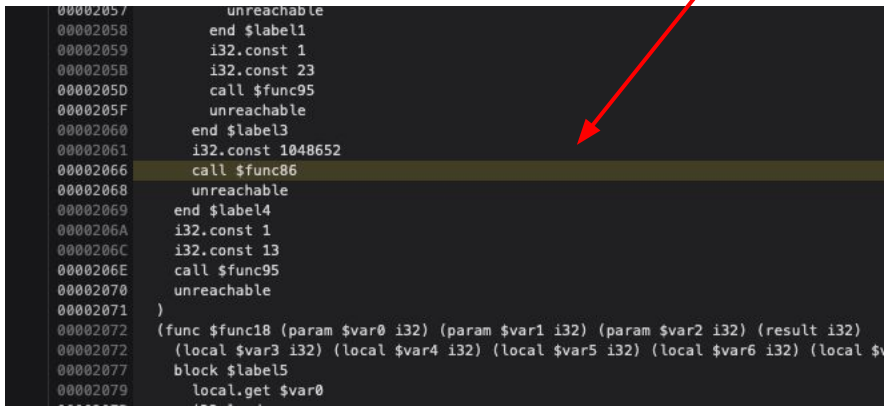
```rust
// OscillatorNode (source)
let osc = self.ctx.create_oscillator.unwrap();
osc.frequency().set_value(800.0);  // hertz

// GainNode (effect)
let envelope = self.ctx.create_gain().unwrap();
envelope.gain().set_value(1.0);

// Internal state to track time
let play_time = self.scheduled_upto + self.beat_interval;

envelope
    .gain()
    .exponential_ramp_to_value_at_time(1.0, play_time + 0.001)
    .unwrap();

envelope
    .gain()
    .exponential_ramp_to_value_at_time(0.001, play_time + 0.02)
    .unwrap();

// Link osc -> gain
osc.connect_with_audio_node(&envelope).unwrap();

// Link gain -> destination
envelope
    .connect_with_audio_node(&self.ctx.destination())
    .unwrap();

// Start audio at `play_time`
osc.start_with_when(play_time + 0.0).unwrap();

// Stop audio after 0.03s
osc.stop_with_when(play_time + 0.03).unwrap();

// Internal state to track time
self.scheduled_upto = play_time;
```

What if rust code panics?

- Panic converted to `RuntimeError` exception
- The stacktrace does point to text representation of wasm
- But symbols such as `$func86` are hardly of any help

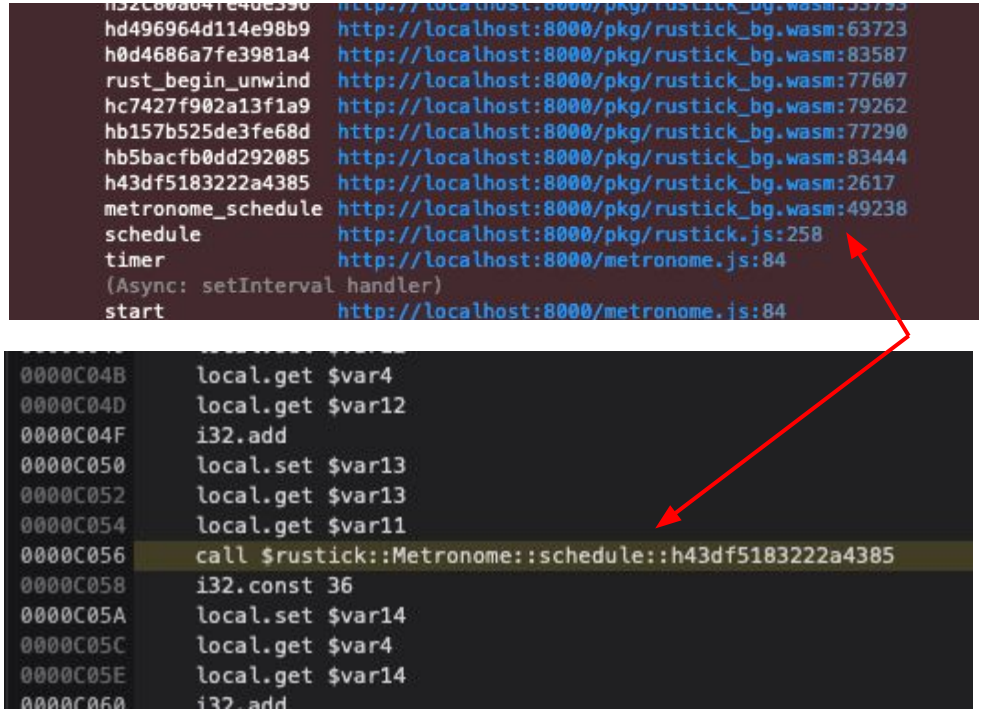
Aside: Wasm text representation is a Lisp!





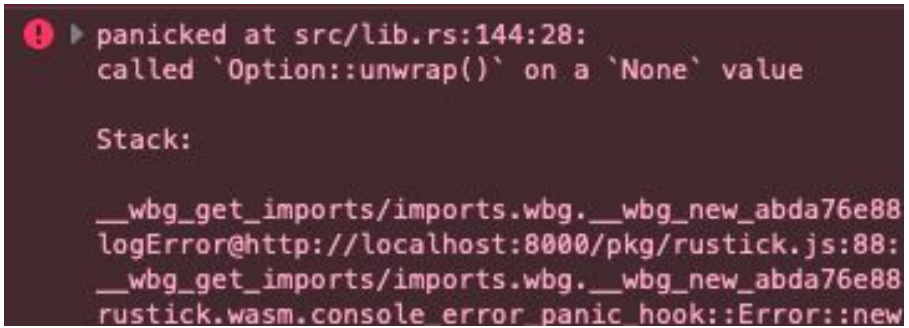
Build with --debug flag

```
$ wasm-pack build --debug --target web
```



External crate: console_error_panic_hook

- Include the crate as a dependency
- Add following line in some code path that's guaranteed to execute e.g. `Metronome::new` function




```
console_error_panic_hook::set_once()
;
```

https://github.com/rustwasm/console_error_panic_hook

Using console.log during development

```
4   #[wasm_bindgen]
5   extern "C" {
6       // Use `js_namespace` here to bind `console.log(..)` instead of just
7       // `log(..)`
8       #[wasm_bindgen(js_namespace = console)]
9       fn log(s: &str);
10
11      // // Multiple arguments too!
12      // #[wasm_bindgen(js_namespace = console, js_name = log)]
13      // fn log_specific(a: u8, b: f32);
14  }
```

Or using helpers provided by the web-sys crate

- `Web_sys::console::log` // array of values
- `Web_sys::console::log_1` // single value
- `Web_sys::console::log_2` // two values

Error handling

```rust
#[derive(Debug)]
pub enum MetronomeError {
    Uninitialized(String),
    WebAudio(JsValue),
}

impl Into<JsValue> for MetronomeError {
    fn into(self) -> JsValue {
        match self {
            Self::Uninitialized(s) => JsValue::from_str(&format!("Param '{s}' uninitialized")),
            Self::WebAudio(jv) => jv,
        }
    }
}
```

```rust
#[wasm_bindgen]
pub fn schedule(&mut self) -> Result<(), MetronomeError> {
    let schedule_ahead_interval =
        self.schedule_ahead_interval
        .ok_or(MetronomeError::Uninitialized(
            "schedule_ahead_interval".to_string(),
        ))?;
```

```rust
let osc = self
    .ctx
    .create_oscillator()
    .map_err(MetronomeError::WebAudio)?;
let envelope = self.ctx.create_gain().map_err(MetronomeError::WebAudio)?;
```

`JsValue`: Rust representation of an object owned by Javascript

Callback functions / Closures

lib.rs

```rust
use web_sys::js_sys::Function;

#[wasm_bindgen]
pub struct Metronome {
    // ...
    on_ended: Option<Function>,
}

// Inside the schedule method
osc.set_onended(self.on_ended.as_ref());

// Calling it directly
let f = self.on_ended.unwrap();
let context = JsValue::null(); // bind to this
f.call0(context).unwrap();

f.call1(context, JsValue::from_bool(true));
// Similarly, there is
f.call2(context, _, _);
f.call3(context, _, _, _);
```

example.rs

```rust
let closure: Closure<dyn Fn()> = Closure::new(move || {
    // Do something here
});

let window = web_sys::window().expect('window is expected');
window.set_interval_with_callback_and_timeout_and_arguments_0(
    closure.as_ref().unchecked_ref(),
    1000,
).unwrap();

// Following allows the object to be dropped without
// invalidating the underlying closure
// WARNING! Leaks memory in rust
closure.forget();
```

References

Rustic metronome

- Code: https://github.com/naiquevin/rustick
- Demo: https://naiquevin.github.io/rustick/

Webassembly

- https://webassembly.org/

Rust + Wasm

- https://rustwasm.github.io/docs/book/
- https://rustwasm.github.io/wasm-bindgen/introduction.html
- https://developer.mozilla.org/en-US/docs/WebAssembly/Rust_to_Wasm

Metronome & WebAudio API

- https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
- https://grantjam.es/creating-a-simple-metronome-using-javascript-and-the-web-audio-api/

Thank you!