Windowing Status Check

with rust
via wasm32-unknown-unknown

Motivation

- Coherent, cross-platform APIs lower the barrier to entry
- Improving platform support on crates with mindshare improves ecosystem disproportionately
- Rust is up to the challenge

Concepts

The Window

- What is a Window?
  - Size & DPI
  - Title & Icon & Decorations
  - Transparency & Always-On-Topedness & Fullscreenedness
- What does a Window do?
  - Events
  - Graphics

The Event Loop

- An loop that polls for events — well-named
- Creates synthetic events from underlying data changes
- Handles program control flow

Graphics

- A graphics context.
  - gfx-hal? Maybe.
  - wgpu? Eventually.
  - OpenGL/WebGL? Yeah, today.

Emscripten

The “don’t we have this already?” slide.

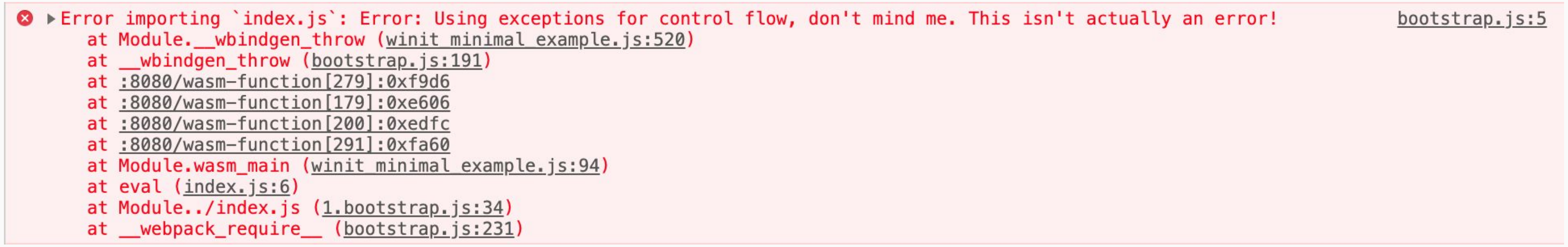
Emscripten

- Worth mentioning as a special case.
- Is wasm/web but, conceptually, is more like programming for native.
- Uses a special context setup (*Module*), a lot of JS glue, and forked dependencies to ease the pain.
- *wasm32-unknown-unknown* is a simpler pipeline and more... honest, as a platform.

Current Projects

winit

- Window
  - Creates a canvas element, applies attributes & styles
  - Exposes it raw for you to place in the DOM
- Event Loop
  - `pub fn run<F>(self, mut event_handler: F) -> !`
  - A little rough around the edges but API is coherent
- Web support is not on master yet



```
▸Error importing `index.js`: Error: Using exceptions for control flow, don't mind me. This isn't actually an error!    bootstrap.js:5
    at Module.__wbindgen_throw (winit_minimal_example.js:520)
    at __wbindgen_throw (bootstrap.js:191)
    at :8080/wasm-function[279]:0xf9d6
    at :8080/wasm-function[179]:0xe606
    at :8080/wasm-function[200]:0xedfc
    at :8080/wasm-function[291]:0xfa60
    at Module.wasm_main (winit_minimal_example.js:94)
    at eval (index.js:6)
    at Module../index.js (1.bootstrap.js:34)
    at __webpack_require__ (bootstrap.js:231)
```

glutin

- Backed by winit
- Requires `Send + Sync` be implemented for the Window context
  - Problematic on web platforms
  - Problematic in an `anyref` world
- `get_proc_address` can be emulated on web
  - But OpenGL and WebGL are different in significant way
  - Function signatures are very different

raw-window-handle

- Sleeper hit. New(er) crate.
- Doesn’t handle creation or management of windows.
- A common interface that window creation libraries (e.g. Winit, SDL) can use to easily talk with graphics libraries (e.g. gfx-hal).
- Will ease implementation limitations
- Localize hacks to here

GL Context

Glium

- GLES2 and old comments indicate an interest in WebGL support
- Web backend via `get_proc_address` might be possible
  - How are GLES2 and WebGL different? Let me count the ways.
  - OpenGL: `void glBindBuffer(GLenum target, GLuint buffer);`
  - WebGL: `void gl.bindBuffer(GLenum target, WebGLBuffer buffer);`

Glow

- Hand-crafted trait implementations
- Creates an abstraction over OpenGL, WebGL1 and WebGL2
  - Incurs some runtime cost (probably minimal)
  - Restricts features
- No stdweb implementation yet

gl-rs

- Not cross-platform in the sense that we’re describing
- Extremely useful building block

Higher Level Projects

- Kiss3d
  - Has web-specific implementations of core features
  - None of the demos work on WASM & native
- Quicksilver
  - Closest thing to out-of-the-box cross-platform window with GL context
  - Very curated 2D API

Conclusion

What To Use Today

- Winit, web branch, window + event loop
- Glow, OpenGL/WebGL abstraction
- Glutin on native only, feed to Glow
- wasm-bindgen

Links

- https://github.com/rust-windowing
- https://github.com/grovesNL/glow
- https://rustwasm.github.io/docs/wasm-bindgen/
- https://github.com/ryanisaacg/quicksilver