WebAssembly: Status & Web IDL Bindings

W3C Games Workshop - June, 2019

Luke Wagner

Based on joint Mozilla/Google presentation to WebAssembly CG last week (link)
WebAssembly Status

- 2017: "MVP" ships in 4 browsers \o/
- Immediately continued work on a pipeline of proposed additions
- Based on TC39 stages process
- Post-MVP Roadmap
- https://github.com/webassembly/proposals
* Bindings Proposal History

- **2017 - Ship the WebAssembly "MVP"**
  - Only 4 value types: i32, i64, f32, f64
  - *How can WebAssembly call Web APIs?*

- **2017 - Take 1: "Host Bindings" ([CG pres](#), [TPAC pres](#))**
  - All host values go in wasm tables (wasm linear memory requires exposing raw bits)
  - Automatically convert between table indices and host values at the interface
  - Increasingly awkward as we worked through use cases; also not efficient

- **2018 - Reference Types ([CG pres](#), [explainer](#))**
  - Subtype hierarchy: anyref, funcref, ref T (where T = func(X→Y), struct{x:A,y:B}, array(T), ...)
  - Gives wasm first-class host values

- **2019 - Take 2: "Web IDL Bindings" ([CG pres](#), [explainer](#))**
  - Let's focus just on efficiently binding to Web IDL, building on reference types
  - "Efficiently" means eliminating copies, garbage, auxiliary calls ("Host Bindings" didn't)
  - "Web IDL" allows us to focus on Web IDL's types, avoid Hard(TM) problems
In the MVP, ... ... when calling wasm ... when calling a Web API
With the proposal...  ... when calling wasm  ... when calling a Web API

Web IDL Bindings "wrap" the core module
- Defines new custom section
- Can 100% polyfill by generating JS glue from custom section

Incoming binding exprs
- can place values into linear memory

Outgoing binding exprs
- can extract values from linear memory

Web IDL values

With the proposal...  ... when calling wasm  ... when calling a Web API

Web IDL Bindings "wrap" the core module
- Defines new custom section
- Can 100% polyfill by generating JS glue from custom section

Incoming binding exprs
- can place values into linear memory

Outgoing binding exprs
- can extract values from linear memory

Web IDL values

With the proposal...  ... when calling wasm  ... when calling a Web API

Web IDL Bindings "wrap" the core module
- Defines new custom section
- Can 100% polyfill by generating JS glue from custom section

Incoming binding exprs
- can place values into linear memory

Outgoing binding exprs
- can extract values from linear memory

Web IDL values

With the proposal...  ... when calling wasm  ... when calling a Web API

Web IDL Bindings "wrap" the core module
- Defines new custom section
- Can 100% polyfill by generating JS glue from custom section

Incoming binding exprs
- can place values into linear memory

Outgoing binding exprs
- can extract values from linear memory

Web IDL values

With the proposal...  ... when calling wasm  ... when calling a Web API

Web IDL Bindings "wrap" the core module
- Defines new custom section
- Can 100% polyfill by generating JS glue from custom section

Incoming binding exprs
- can place values into linear memory

Outgoing binding exprs
- can extract values from linear memory

Web IDL values

With the proposal...  ... when calling wasm  ... when calling a Web API

Web IDL Bindings "wrap" the core module
- Defines new custom section
- Can 100% polyfill by generating JS glue from custom section

Incoming binding exprs
- can place values into linear memory

Outgoing binding exprs
- can extract values from linear memory

Web IDL values

With the proposal...  ... when calling wasm  ... when calling a Web API

Web IDL Bindings "wrap" the core module
- Defines new custom section
- Can 100% polyfill by generating JS glue from custom section

Incoming binding exprs
- can place values into linear memory

Outgoing binding exprs
- can extract values from linear memory

Web IDL values

With the proposal...  ... when calling wasm  ... when calling a Web API

Web IDL Bindings "wrap" the core module
- Defines new custom section
- Can 100% polyfill by generating JS glue from custom section

Incoming binding exprs
- can place values into linear memory

Outgoing binding exprs
- can extract values from linear memory

Web IDL values
Binding Types/Values sketch

reftype ::= ... all the core wasm reference types

numtype ::= s8 | u8 | s16 | u16 | s32 | u32 | s64 | u64 | f32 | f64  // signedness matters

bindingtype ::=  // Web IDL type
    reftype |  // ↔ any, Interface, Promise, ...
    numtype |  // ↔ byte, octet, short, ...
    string |  // ↔ DOMString
    bytes |  // ↔ ArrayBuffer
    numtype view |  // ↔ Int8Array, Uint8Array, ...
    bindingtype list |  // ↔ Sequence
    record{ (lbl: bindingtype)* } |  // ↔ Dictionary
    variant{ (lbl: bindingtype)* } |  // ↔ Union, Enumeration
    func(bindingtype* → bindingtype*)  // ↔ Callback function
Binding Expressions sampler

wasm type  outgoing expression  binding type  incoming expression  wasm type

ref T  as  ref T  as  ref T

i32  utf8-mem-str  string  alloc-utf8-mem-str $alloc $free  i32

i32  utf8-gc-str  as  alloc-utf8-gc-str  as  i32

ref array u8  as  ref array u8

ref T  T=string

ref T  as  ref struct ...

ref struct ...  as  string

string  as  record

record { x:string, y:string }  as  record

record { x:string, y:string }  as  field "x"

field "y"  as  string

string  as  string

...  as  ...

...  as  ...

T=string
Two facets of "direct" Web IDL access from wasm

Importing

// JS loader glue code:

const importObj = {
    Document: {
        createElement: 
            Document.prototype.createElement
    }
};

WebAssembly.instantiate(module, importObj).then(...)

This JS glue will be removed by a combination of:
- WebAssembly ESM-integration
- Built-in modules + import-maps
- get-originals

Calling

// JS runtime glue code

var memory = ...
var td = new TextDecoder();

function createElement_glue(doc, tagOff, tagLen) {
    var buf = memory.buffer;
    var bytes = new Uint8Array(buf, tagOff, tagLen);
    return doc.createElement(td.decode(bytes));
}

// wasm caller

(import "Document" "createElement"
    (func (param anyref i32 i32) (result anyref)))

Removing this type of JS glue is the focus of Web IDL Bindings
C++ Prototype

https://github.com/jgravelle-google/wasm-webidl-polyfill

Building a webIDL.js module, reads a custom section and fixes up import + export dicts at runtime

Goals:
- Prototype the design, prove feasibility of polyfilling
- Polyfillability in general is a useful property because developers can ship the real bytes early, and the browser can support that natively at a later time
- Having a prepackaged chunk of JS makes this easier to include in arbitrary toolchains
Rust: wasm-bindgen

```
#![wasm_bindgen(js_namespace = console)]
fn log(s: &str);
```
WebGL Prototype

- Animometer Benchmark uses ~20 OpenGL functions.
- 7 function calls in a hot loop.

```c
// repeated for 20,000 primitives

glUniform1f(uScale, uniformData[i].scale);
glUniform1f(uTime, uniformData[i].time);
glUniform1f(uOffsetX, uniformData[i].offsetX);
glUniform1f(uOffsetY, uniformData[i].offsetY);
glUniform1f(uScalar, uniformData[i].scalar);
glUniform1f(uScalarOffset, uniformData[i].scalarOffset);

glDrawArrays(GL_TRIANGLES, 0, 3);
```
WebGL Prototype Experimental Results

Loop Time - No Rendering (ms)

- Benchmark (20000)
- Benchmark (100000)

Native
Javascript
Debugging

- Discussion at CG meeting last week ([minutes](#))
- Converging on new debugging interfaces allowing portable debuggers
  - Goal: don't require building DWARF into all browsers
- Expect renewed activity in WebAssembly debugging subgroup ([link](#))
  - Join!
Discussion