

# Stringrefs

Reference-typed strings in  
WebAssembly

Andy Wingo | `wingo@igalia.com`

`https://github.com/wingo/  
stringrefs/`

# Agenda

Motivation

Overview: Goals, Requirements,  
Design, Proposal

Open questions / feedback

`https://github.com/wingo/  
stringrefs`

# Motivation

## Three examples of suboptimality

- ☛ C++ on the web: double copies, memory capability
- ☛ Java on web: DOM access expensive, code duplication
- ☛ Component model: from single copy to zero copy

# C++ on the web

```
https://github.com/emscripten-core/emscripten/blob/main/src/preamble.js

) 97 function ccall(ident, returnType, argTypes, args, opts) {
   98     // For fast lookup of conversion functions
   99     var toC = {
100         'string': function(str) {
101             var ret = 0;
102             if (str !== null && str !== undefined && str !== 0) { // null string
103                 // at most 4 bytes per UTF-8 code point, +1 for the trailing '\0'
104                 var len = (str.length << 2) + 1;
105                 ret = stackAlloc(len);
106                 stringToUTF8(str, ret, len);
107             }
108             return ret;
109         },
110         'arrav': function(arr) {
```

<https://github.com/emscripten-core/emscripten/blob/main/src/preamble.js#L100>

# C++ on the web

Double-copy (first to stack then to where you need it)

NUL termination (have to scan again for length)

Can't represent NUL codepoints

Requires read/write capability on whole memory

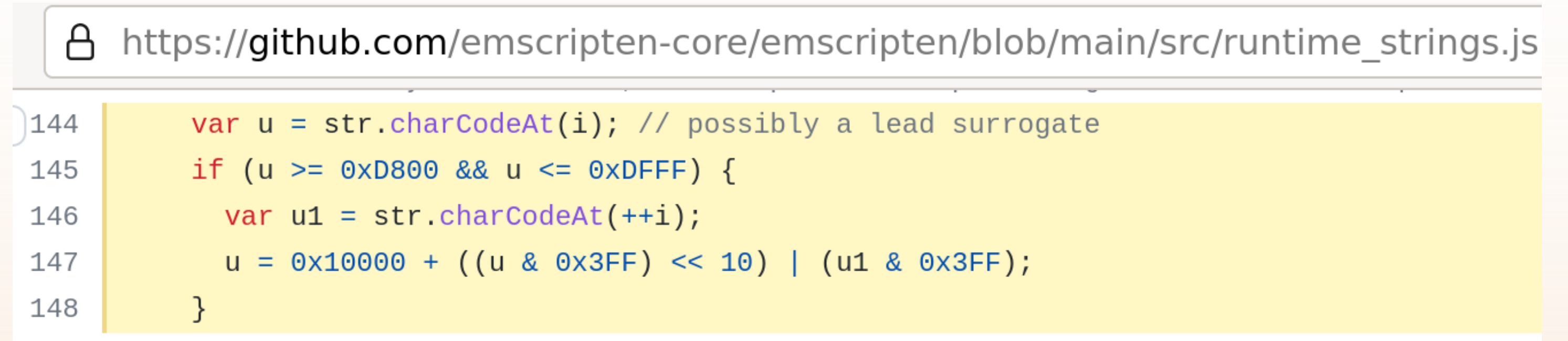
Requires that users wrangle malloc

Requires JS

Similar problems in other direction

C++  
on the  
web

Also it's buggy :)

A screenshot of a web browser displaying a GitHub repository. The address bar shows the URL: https://github.com/emscripten-core/emscripten/blob/main/src/runtime\_strings.js. The main content area shows a JavaScript code snippet with line numbers 144 through 148. The code is as follows:

```
144     var u = str.charCodeAt(i); // possibly a lead surrogate
145     if (u >= 0xD800 && u <= 0xDFFF) {
146         var u1 = str.charCodeAt(++i);
147         u = 0x10000 + ((u & 0x3FF) << 10) | (u1 & 0x3FF);
148     }
```

<https://github.com/emscripten-core/emscripten/issues/15324>

# Java on the web

JS strings are exactly what Java needs:  
immutable sequences of 16-bit code  
units

But all Java can do is GC array of u16 –  
GC allocation on Java/JS boundary

Penalizes access to DOM

Penalizes JS/Java interaction

Needlessly ships second string facility



# Component model

Components are isolated

- ☛ Communication via abstractly-typed interfaces
- ☛ JIT compilation of adapters for concrete representations

Linear memory strings always copied at least once between components

Strings in GC memory: same (because mutability)

Could do better if WebAssembly had immutable stringrefs



# Why not u16 arrays?

You can implement GC in linear memory, but it is terrible

On web, GC is right there, let's use it

Same argument for JS strings

Implies growing WebAssembly platform for non-JS hosts

☛ But, immutable stringrefs also good for component model

# Why not a library?

Duplication: Host already has strings

Duplication: Avoid library per module  
or component

Inefficiency: Module boundary is a  
barrier

Platform effects: Strings are interop  
MVP

# Goals

- Enable programs compiled to WebAssembly to efficiently create and consume JavaScript strings
- Provide a good string implementation that many languages implemented on top of the GC proposal would find useful

# Req'ts

- 🦋 Zero-copy string passing between JS and Wasm
- 🦋 No new string implementations on the web
- 🦋 Allow WTF-8 or WTF-16 internal representations
- 🦋 Allow WTF-16 code unit access
- 🦋 Allow string literals in element sections

# Design

The tension:

- ☛ Source languages: UTF-8 for Rust, WTF-16 for Java, codepoint access for Python...
- ☛ Implementations: WTF-16 for V8, UTF-8 for wasmtime...

Solve via common-denominator  
stringref plus encoding-specific  
stringviews

# Proposal

`stringref` is new opaque reference-typed value, like `externref`

A `stringref` is a sequence of Unicode scalar values and isolated surrogates

Can obtain WTF-8, WTF-16, codepoint iterator “views” on a `stringref`

# stringref

```
(string.new_wtf8 $memory ptr:address bytes:i32)
  -> str:stringref
(string.new_wtf16 $memory ptr:address codeunits:i32)
  -> str:stringref
(string.const contents:i32)
  -> str:stringref
(string.measure_utf8 str:stringref)
  -> bytes:i32
(string.measure_wtf8 str:stringref)
  -> bytes:i32
(string.measure_wtf16 str:stringref)
  -> bytes:i32
wtf8_policy ::= 'utf8' | 'wtf8' | 'replace'
(string.encode_wtf8 $memory $wtf8_policy
                    str:stringref ptr:address)
(string.encode_wtf16 $memory str:stringref ptr:address)
(string.concat a:stringref b:stringref) -> stringref
(string.eq a:stringref b:stringref) -> i32
(string.is_usv_sequence str:stringref)
  -> bool:i32
```



# stringview\_wtf8

```
(string.as_wtf8 str:stringref)
-> view:stringview_wtf8
(stringview_wtf8.advance view:stringview_wtf8
                        pos:i32 bytes:i32)

-> next_pos:i32
(stringview_wtf8.encode $memory $wtf8_policy
    view:stringview_wtf8 ptr:address pos:i32 bytes:i32)
-> next_pos:i32, bytes:i32
(stringview_wtf8.slice view:stringview_wtf8
                        start:i32 end:i32)

-> str:stringref
```

# stringview\_wtf16

[illegible]



# Relation to GC

Not dependent on GC MVP

Same family though

Best “like externref” formulation is in terms of heaptypes, from typed function references

Will want array u16, array u8 read/write

# Open questions

Type relationship of stringview  
variants

eq supertype or not?

Utility of WTF-8 view

Performance proof



<https://github.com/wingo/wasm-jit/blob/main/interp.py>

26

27 **def** write\_string(string):

28 utf8 = string.encode('utf-8')

29 ptr = interplib.allocateBytes(len(utf8) + 1)

30 ptr = interplib.allocateBytes(len(utf8) + 1)

31 dst = interplib.memory.data\_ptr(wasmtime.loader.store)

32 **for** i **in** range(0, len(utf8)):

33 dst[ptr + i] = utf8[i]

34 dst[ptr + len(utf8)] = 0

35 **return** ptr

26

# next steps

CG meeting 26 April: phase 1?

Move repo to WebAssembly org

Q2-Q3: Prototyping in V8

Q3-Q4: Toolchain (LLVM, Binaryen)

[https://github.com/wingo/  
stringrefs/](https://github.com/wingo/stringrefs/)

[wingo@igalia.com](mailto:wingo@igalia.com)