
PRAIRIE
DEV CON
WEB | DEV | CLOUD | AI


NORTHFIELD


Jg
Johnston
group


Lotlinx


online
business systems


LAMBERT
NEMEC
GROUP
EXCELLENCE IN RECRUITMENT


CONQUEST


ATLASSIAN

SSR Web Components

EVERYWHERE

with **WASM**

Simon MacDonald
@macdonst@mastodon.online

Web Components

Web Components

- Custom Elements

Web Components

- Custom Elements
- Shadow DOM

Web Components

- Custom Elements
- Shadow DOM
- HTML Templates

Custom Elements

https://codepen.io/macdonst/embed/ZErMzrr?default-tab=html%2Cresult&editable=true

Shadow DOM

https://codepen.io/macdonst/embed/oNdpNKQ?default-tab=html%2Cresult&editable=true

Slots and Templates

https://codepen.io/macdonst/embed/vYjpYpV?default-tab=html%2Cresult&editable=true

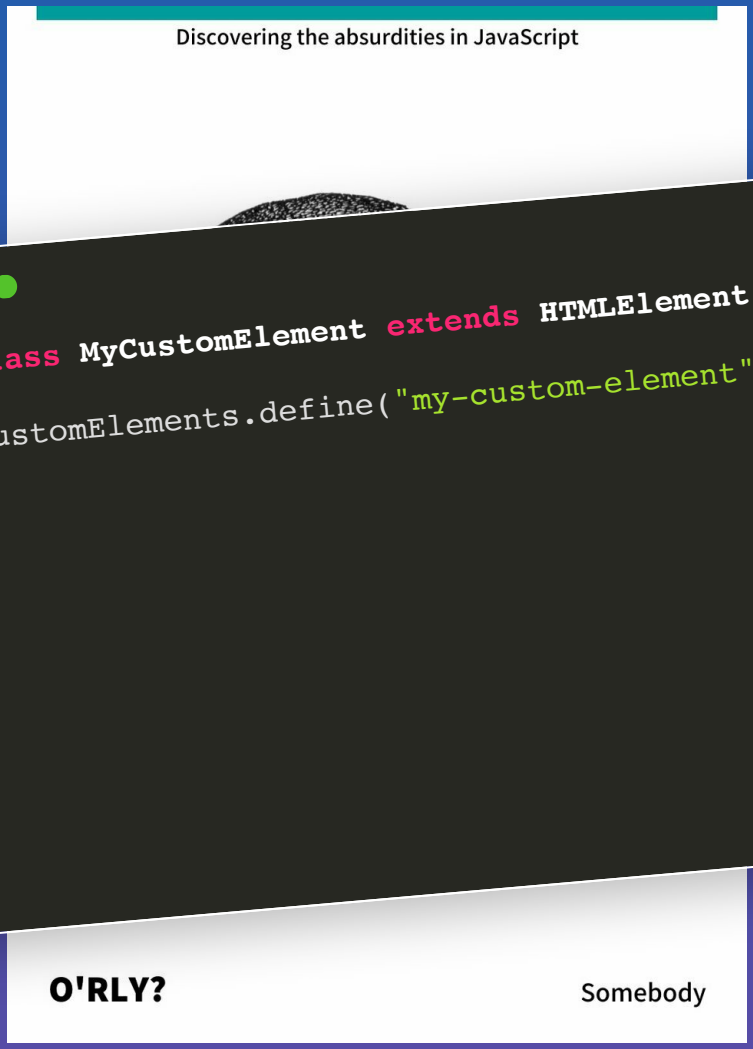
Full Example

https://codepen.io/macdonst/embed/oNrXXeV?default-tab=html%2Cresult&editable=true

Browser Native Technology

JavaScript Limitations

JavaScript




```
class MyCustomElement extends HTMLElement { … }

customElements.define("my-custom-element", MyCustomElement);
```

Server Side Rendering

Server Side Rendering

+

+

+node.js

+

+node.js =

HTML

Web Assembly

Web Assembly

- Is a binary machine code language

Web Assembly

- Is a binary machine code language
- You don't write web assembly

Web Assembly

- Is a binary machine code language
- You don't write web assembly
- Compile other languages into .wasm

Web Assembly

- Is a binary machine code language
- You don't write web assembly
- Compile other languages into .wasm
- Run anywhere!

WASM Example

helloWorld.js

```
1 function helloWorld() {
2   console.log("Hello World from WebAssembly!")
3 }
4
5 exports { helloWorld }
```

WASM Example

helloWorld.js

```
1 functio
2   conso
3 }
4
5 exports
```

WASM Example

helloWorld.js

```
1 function helloWorld() {
2   console.log("Hello World from WebAssembly!")
3 }
4
5 exports { helloWorld }
```

WASM Example

helloWorld.js

```js
function helloWorld() {
  console.log("Hello World from WebAssembly!")
}

exports { helloWorld }
```

helloWorld.wat

```
(module
    ;; Imports from JavaScript namespace
    (import  "console"  "log" (func  $log (param  i32  i32))) ;; Import log function
    (import  "js"  "mem" (memory  1)) ;; Import 1 page of memory (54kb)

    ;; Data section of our module
    (data (i32.const 0) "Hello World from WebAssembly!")

    ;; Function declaration: Exported as helloWorld(), no arguments
    (func (export  "helloWorld")
        i32.const 0  ;; pass offset 0 to log
        i32.const 29  ;; pass length 29 to log (strlen of sample text)
        call  $log
        )
)
```

WASM Example

helloWorld.js

```
1 function helloWorld() {
2   console.log("Hello World from WebAssembly!")
3 }
4
5 exports { helloWorld }
```

helloWorld.wat

```
1  (module
2      ;; Imports from JavaScript namespace
3      (import  "console"  "log" (func  $log (param  i32  i32))) ;; Import log function
4      (import  "js"  "mem" (memory  1)) ;; Import 1 page of memory (54kb)
5
6      ;; Data section of our module
7      (data (i32.const 0) "Hello World  Assembly!")
8
9      ;; Function declaration: Exported  oWorld(), no arguments
10     (func (export  "helloWorld")
11         i32.const 0  ;; pass offse  to log
12         i32.const 29  ;; pass length 29 to log (strlen of sample text)
13         call  $log
14         )
15 )
```

WASM Example

Compile to WASM

```
javy compile helloWorld.js -o helloWorld.wasm
```

WASM Example

Compile to WASM

```
javy compile helloWorld.js -o helloWorld.wasm
```

Test using wasmtime

```
wasmtime run helloWorld.wasm

Hello World from WebAssembly!
```

WASM Example

Running WASM on a webpage

```
<script>
const memory = new WebAssembly.Memory({initial:1});

function  consoleLogString(offset, length) {
  const bytes = new Uint8Array(memory.buffer, offset, length);
  const string = new TextDecoder('utf8').decode(bytes);
  console.log(string);
};

const importObject = {
  console: {
    log:  consoleLogString
  },
  js : {
    mem:  memory
  }
};

WebAssembly.instantiateStreaming(fetch('helloworld.wasm'), importObject)
  .then(obj  => {
    obj.instance.exports.helloWorld();
  }
);
</script>
```

WASM Example

Running WASM on a webpage

```
<script>
const memory = new WebAssembly.Memory({initial:1});

function  consoleLogString(offset, length) {
  const bytes = new Uint8Array(memory.buffer, offset, length);
  const string = new TextDecoder('utf8').decode(bytes);
  console.log(string);
};

const importObject = {
  console: {
    log:  consoleLogString
  },
  js : {
    mem:  memory
  }
};

WebAssembly.instantiateStreaming(fetch('helloworld.wasm'), importObject)
  .then(obj  => {
    obj.instance.exports.helloWorld();
  }
);
</script>
```

WASM Example

Running WASM on a webpage

```
<script>
const memory = new WebAssembly.Memory({initial:1});

function  consoleLogString(offset, length) {
  const bytes = new Uint8Array(memory.buffer, offset, length);
  const string = new TextDecoder('utf8').decode(bytes);
  console.log(string);
};

const importObject = {
  console: {
    log:  consoleLogString
  },
  js : {
    mem:  memory
  }
};

WebAssembly.instantiateStreaming(fetch('helloworld.wasm'), importObject)
  .then(obj  => {
    obj.instance.exports.helloWorld();
  }
);
</script>
```

WASM Example

Running WASM on a webpage

```
<script>
const memory = new WebAssembly.Memory({initial:1});

function  consoleLogString(offset, length) {
  const bytes = new Uint8Array(memory.buffer, offset, length);
  const string = new TextDecoder('utf8').decode(bytes);
  console.log(string);
};

const importObject = {
  console: {
    log:  consoleLogString
  },
  js : {
    mem:  memory
  }
};

WebAssembly.instantiateStreaming(fetch('helloworld.wasm'), importObject)
  .then(obj  => {
    obj.instance.exports.helloWorld();
  }
);
</script>
```

WASM Example

Running WASM on a webpage

```
<script>
const memory = new WebAssembly.Memory({initial:1});

function  consoleLogString(offset, length) {
  const bytes = new Uint8Array(memory.buffer, offset, length);
  const string = new TextDecoder('utf8').decode(bytes);
  console.log(string);
};

const importObject = {
  console: {
    log:  consoleLogString
  },
  js : {
    mem:  memory
  }
};

WebAssembly.instantiateStreaming(fetch('helloworld.wasm'), importObject)
  .then(obj  => {
    obj.instance.exports.helloWorld();
  }
);
</script>
```

WASM Example

Running WASM on a webpage

```
<script>
const memory = new WebAssembly.Memory({initial:1});

function  consoleLogString(offset, length) {
  const bytes = new Uint8Array(memory.buffer, offset, length);
  const string = new TextDecoder('utf8').decode(bytes);
  console.log(string);
};

const importObject = {
  console: {
    log:  consoleLogString
  },
  js : {
    mem:  memory
  }
};

WebAssembly.instantiateStreaming(fetch('helloworld.wasm'), importObject)
  .then(obj  => {
    obj.instance.exports.helloWorld();
  }
);
</script>
```

EXTISM

Extism Example

Install Extism

```
curl -O https://raw.githubusercontent.com/extism/js-pdk/main/install.sh
sh install.sh
```

Extism Example

Install Extism

```
curl -O https://raw.githubusercontent.com/extism/js-pdk/main/install.sh
sh install.sh
```

Create an Extism plugin

```
extism gen plugin -l JavaScript -o plugin
cd plugin
npm install
// write some code
```

Extism Example

Install Extism

```
curl -O https://raw.githubusercontent.com/extism/js-pdk/main/install.sh
sh install.sh
```

Create an Extism plugin

```
extism gen plugin -l JavaScript -o plugin
cd plugin
npm install
// write some code
```

Build & Test Extism plugin

```
npm run build
extism call ./dist/plugin.wasm count_vowels --input "Hello, World!" --wasi
// => '{"count":3,"total":3,"vowels":"aeiouAEIOU"}'
```

Extism Example

```
const createPlugin = require("@extism/extism")

const plugin = await createPlugin(
  'https://github.com/extism/plugins/releases/latest/download/count_vowels.wasm',
  { useWasi: true }
);

let out = await plugin.call("count_vowels", "Hello, World!");
console.log(out.text())
// => '{"count":3,"total":3,"vowels":"aeiouAEIOU"}'
```

Extism Example

```
const createPlugin = require("@extism/extism")

const plugin = await createPlugin(
  'https://github.com/extism/plugins/releases/latest/download/count_vowels.wasm',
  { useWasi: true }
);

let out = await plugin.call("count_vowels", "Hello, World!");
console.log(out.text())
// => '{"count":3,"total":3,"vowels":"aeiouAEIOU"}'
```

Extism Example

```
const createPlugin = require("@extism/extism")

const plugin = await createPlugin(
  'https://github.com/extism/plugins/releases/latest/download/count_vowels.wasm',
  { useWasi: true }
);

let out = await plugin.call("count_vowels", "Hello, World!");
console.log(out.text())
// => '{"count":3,"total":3,"vowels":"aeiouAEIOU"}'
```

Extism Example

```
const createPlugin = require("@extism/extism")

const plugin = await createPlugin(
  'https://github.com/extism/plugins/releases/latest/download/count_vowels.wasm',
  { useWasi: true }
);

let out = await plugin.call("count_vowels", "Hello, World!");
console.log(out.text())
// => '{"count":3,"total":3,"vowels":"aeiouAEIOU"}'
```

+


WA

+

+

Any
Runtime

+

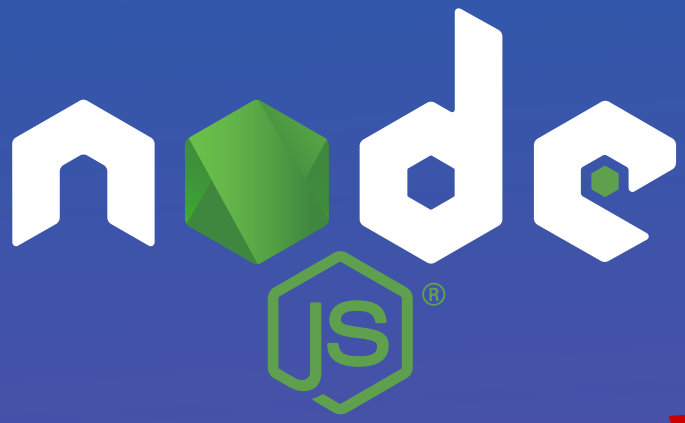
+ Any Runtime =


django


.NET

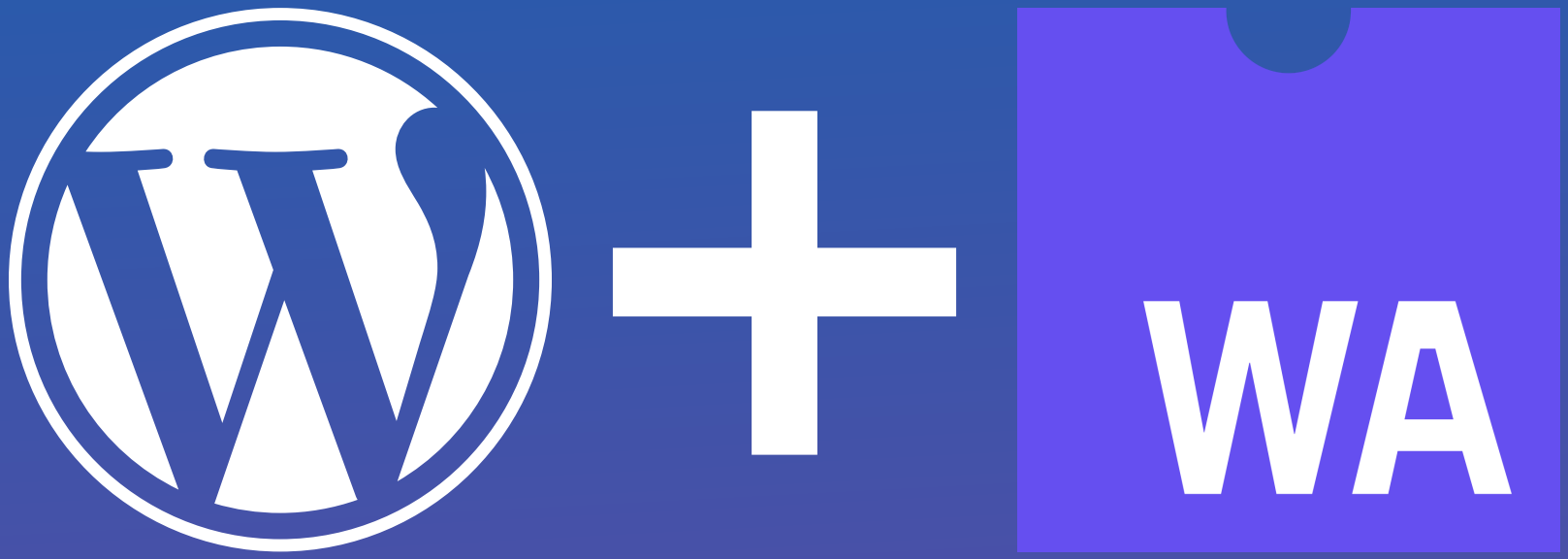

GO


Java


node
JS


php


RAILS


WA

+


WA

+


WA

+

+


WA

+

=


HTML

Extism Example

```php
<?php
require "../../../vendor/autoload.php";

use Enhance\EnhanceWASM;
use Enhance\Elements;

$elementPath = "../resources";
$elements = new Elements($elementPath, ["wasm" => true]);
$enhance = new EnhanceWASM(["elements" => $elements->wasmElements]);

$input = [
    "markup" => "<my-header>Hello World</my-header>",
    "initialState" => [],
];

$output = $enhance->ssr($input);

$htmlDocument = $output->document;

echo $htmlDocument . "\n";
```

Extism Example

```php
<?php
require "../../../vendor/autoload.php";

use Enhance\EnhanceWASM;
use Enhance\Elements;

$elementPath = "../resources";
$elements = new Elements($elementPath, ["wasm" => true]);
$enhance = new EnhanceWASM(["elements" => $elements->wasmElements]);

$input = [
    "markup" => "<my-header>Hello World</my-header>",
    "initialState" => [],
];

$output = $enhance->ssr($input);

$htmlDocument = $output->document;

echo $htmlDocument . "\n";
```

Extism Example

```php
<?php
require "../../../vendor/autoload.php";

use Enhance\EnhanceWASM;
use Enhance\Elements;

$elementPath = "../resources";
$elements = new Elements($elementPath, ["wasm" => true]);
$enhance = new EnhanceWASM(["elements" => $elements->wasmElements]);

$input = [
    "markup" => "<my-header>Hello World</my-header>",
    "initialState" => [],
];

$output = $enhance->ssr($input);

$htmlDocument = $output->document;

echo $htmlDocument . "\n";
```

Extism Example

```php
<?php
require "../../../vendor/autoload.php";

use Enhance\EnhanceWASM;
use Enhance\Elements;

$elementPath = "../resources";
$elements = new Elements($elementPath, ["wasm" => true]);
$enhance = new EnhanceWASM(["elements" => $elements->wasmElements]);

$input = [
    "markup" => "<my-header>Hello World</my-header>",
    "initialState" => [],
];

$output = $enhance->ssr($input);

$htmlDocument = $output->document;

echo $htmlDocument . "\n";
```

Extism Example

```php
<?php
require "../../../vendor/autoload.php";

use Enhance\EnhanceWASM;
use Enhance\Elements;

$elementPath = "../resources";
$elements = new Elements($elementPath, ["wasm" => true]);
$enhance = new EnhanceWASM(["elements" => $elements->wasmElements]);

$input = [
    "markup" => "<my-header>Hello World</my-header>",
    "initialState" => [],
];

$output = $enhance->ssr($input);

$htmlDocument = $output->document;

echo $htmlDocument . "\n";
```

Demo

Some Problems

Some Problems

- Extism is a shared library

Some Problems

- Extism is a shared library
- Installation of the shared library requires sudo/root access

Some Problems

- Extism is a shared library
- Installation of the shared library requires sudo/root access
- Needs "ffi.enable=true"


Thanks!

+


php

+


php

+


+

php
+

=

HTML

Enhance WP Plugin

server side render any enhance custom elements in the wordpress site. These can be added in PHP templates, raw HTML blocks in the editor, or as predefined blocks.

Enhance WP Plugin

server side render any enhance custom elements in the wordpress site. These can be added in PHP templates, raw HTML blocks in the editor, or as predefined blocks.

Enhance Blocks Plugin

demonstrates wrapping an Enhance component for use in the block editor. This works with the SSR plugin. These blocks are stored in the WP database as HTML (i.e. Hi) and then the SSR plugin will render them wherever they are used.

Demo 2

Get Involved

Enhance is an open source initiative, and we're looking for collaborators to join us on our mission of making server side rendered web components accessible to all web developers.

Whatever your choice of programming language or framework may be, we'd love assistance with optimizing our existing implementations, creating new adapters, improving example applications, reviewing pull requests, and everything in between!

If you're enthusiastic about getting involved, let's start talking!

https://enhance.dev/

Key Takeaways

Key Takeaways

- Web Components are awesome

Key Takeaways

- Web Components are awesome
- WASM is great for bringing native like performance to the web

Key Takeaways

- Web Components are awesome
- WASM is great for bringing native like performance to the web
- Never write WASM by hand

Key Takeaways

- Web Components are awesome
- WASM is great for bringing native like performance to the web
- Never write WASM by hand
- Use WASM to support cross platform SDK's


Thanks,
For real this time!