The coding language for the web

JavaScript

JS
Code for the web with high-level programming languages

WebAssembly (wasm)
Major languages supported by WebAssembly

- C, C++, and Rust
- C#/.NET, Go, D, Python, Java, Kotlin, Ruby, PHP
Supported in all major browsers

With asm.js supports also and other older browsers
What are the benefits?

• Use your **preferred language** to target the web (no JS)
• Leverage **existing tools and libraries** – portability
• Paradigms like **strong typing, functional programming, OOP**
• Better **performance**, faster startup time
• Safe **sand-boxed** environment (browser's same-origin and permissions policies)
• **JavaScript interop** – complement rather than replace
Products using WebAssembly

- Figma
- Google Earth
- SketchUp
- Soundation
- Unity
- PSPDFKit
Let’s dive deeper
Designed by a web standards team

• W3C Community Group
  www.w3.org/community/webassembly/
  Includes members of all major browsers

• W3C Working Group
  www.w3.org/wasm/
Progress

MVP
• Binary module format for execution
• Feature parity with asm.js

After the MVP
• Threads
• Exception handling
• Garbage Collection
• ECMAScript module integration
• ...

Shipped in major browsers Oct 2017

https://webassembly.org/docs/future-features/
How the compiler works

An Abridged Cartoon Introduction To WebAssembly

**QUICK SUMMARY**

There's a lot of hype about WebAssembly in JavaScript circles today. People talk about how blazingly fast it is, and how it's going to revolutionize web development. But most conversations don't go into the details of why it's fast. In this article, I want to help you understand what exactly it is about WebAssembly that makes it fast. But first, what is it? WebAssembly is a way of taking code written in programming languages other than JavaScript and running that code in the browser.
How the compiler works
How the compiler works
Resource loading
At runtime – Initiating

WIP: `<script type="module">` or ES2015 import, use loader script

// With streaming support
WebAssembly.instantiateStreaming(
    fetch('simple.wasm'), importObject)
.then(results => {
    /* Do something with the results! */
});

// WebAssembly 1.0 browsers
fetch('module.wasm')
.then(response => response.arrayBuffer())
.then(bytes => WebAssembly.instantiate(bytes, importObject))
.then(results => {
    /* Do something with the results! */
});
Demo: C to Wasm
Challenges

• No direct access to the DOM
• Memory management – numeric
• JS interop overhead
• Tree shaking

• Early days – tooling in development
• Debugging
• Multi-treading
• Source maps
• Garbage collection (GC)
.NET in the browser
A story about SPA frameworks

Blazor
(.NET/C#)

Yew
(Rust)
Adding the missing pieces to Wasm

- **DOM & Browser APIs**
  - A component model for building composable UI
  - Routing
  - Layouts
  - Forms and validation
  - Dependency injection
  - JavaScript interop

- **Live reloading in the browser during development**
- **Server-side rendering**
- **Full .NET debugging both in browsers and in the IDE**
- **Rich IntelliSense and tooling**
- **Ability to run on older (non-WebAssembly) browsers via asm.js**
- **Publishing and app size trimming**
NEW POSSIBILITIES for WEB APPS

Service Workers

Web Assembly

New low-level code format
- Compact, verifiable, highly optimisable
- Bring your own language, runtime, etc.

Web Apps can’t really do *that*, can they? - Steve Sanderson

136,608 views
Hello WebAssembly

Miguel de Icaza  August 9, 2017  runtime

We have been experimenting with a couple of approaches to bring Mono to the web using WebAssembly - a technology that can efficiently and safely execute code in web browsers without being limited to Javascript. Running code written in C or C++ inside the browser has been a big motivator, but most major programming languages have plans to target WebAssembly as a platform.

WebAssembly has been out for a few months on desktop computers and Android, and with the introduction of iOS 11 it will become nearly universal.

We have done some exploratory work to identify what needs to be done to run Mono on the browser. The early experiments are promising, let me talk about those.

Mono supports various execution modes, it ranges from the traditional fully just-in-time compiled, to fully statically compiled with a couple of hybrid modes in between (statically compiled with JIT, and statically compiled with an interpreter).

Today we have two prototypes running in WebAssembly.
Today I’m excited to announce a new experimental project from the ASP.NET team called Blazor. Blazor is an experimental web UI framework based on C#, Razor, and HTML that runs in the browser via WebAssembly. Blazor promises to greatly simplify the task of building fast and beautiful single-page applications that run in any browser. It does this by enabling developers to write .NET-based web apps that run client-side in web browsers using open web standards.

If you already use .NET, this completes the picture: you’ll be able to use your skills for browser-based development in addition to existing scenarios for server and cloud-based services, native mobile/desktop apps, and games. If you don’t yet use .NET, our hope is that the productivity and simplicity benefits of Blazor will be compelling enough that you will try it.
An experiment: Blazor

• Jun 2017
  Steve Sanderson showcases Blazor hacky experiment

• Aug 2017
  Mono project announces experimental WebAssembly compilation target

• Nov 2017
  Steve Sanderson switches Blazor to Mono runtime

• Feb 2018
  ASP.NET Team announces that they continue the experiment
Interpreted vs Ahead-of-Time (AOT)

Dev time

C# / Razor source files

mono.wasm
WebAssembly binary executed natively

YourApp.dll
netstandard .NET assembly files

Runtime (in browser)

Browser APIs
visible DOM, HTTP requests, etc.

C# / Razor source files

YourApp.dll

mono.wasm
WebAssembly binary executed natively

YourApp.wasm
WebAssembly binary executed natively

Runtime (in browser)

Browser APIs
visible DOM, HTTP requests, etc.

http://blog.stevensanderson.com/2018/02/06/blazor-intro/
Demo: Blazor
A word about performance

Blazor is still an experiment
Your mileage may vary
Closing thoughts
Further reading

- **WebAssembly**
  www.webassembly.org

- **Lin Clark’s An Abridged Cartoon Introduction To WebAssembly**
  www.smashingmagazine.com/2017/05/abridged-cartoon-introduction-webassembly/

- **Steve Sanderson’s first Blazor reveal**
  www.youtube.com/watch?v=MiLAE6HMr10

- **Yew – Rust based SPA framework**
  github.com/DenisKolodin/yew

- **WebAssembly languages**
  github.com/appcypher/awesome-wasm-langs
Thanks for listening

@bgever