

WebAssembly

Embedders, Modules, WABT, Primitives, Functions, Linear Memory, Tables, Control Flow, JS API

All the major browser vendors - Google, Mozilla, Microsoft and Apple - have cooperated to define an agreed standard called WebAssembly that specifies how an executable looks like for embedders to execute.

WebAssembly allows you to run code written directly in assembly or in a high-level language (e.g. C/C++) compiled into assembly in a browser without plug-ins.

WebAssembly is a virtual Instruction Set Architecture that runs in an embedder. Currently the most popular embedder is the modern standard web browser and on the server the latest Node.js 11 supports it. Other more

specialist embedders include the OCaml based [spec interpreter](#) and the C++ based [WABT interpreter](#).

WebAssembly modules are binary, low-level (e.g. support 64-bit integers, unlike JavaScript), and are very fast both to load and run. Support for calling WebIDL-defined APIs (e.g. the DOM) will be added in future.

Web developers with requirements for very high performing applications will benefit from exploring WebAssembly as it the basis for considerable industry innovation and has wide cross-browser support.

Contents of One-Day Training Course	
<p>Target Audience Developers wishing to get the very best performance out of their web browser and web server code</p> <p>Prerequisites Knowledge of C/C++ /JavaScript/TypeScript with some background experience of working with assembly</p>	<p>WebAssembly Overview Part of the modern web platform Designed for the web (security, etc.) Tour of WebAssembly concepts</p> <p>WebAssembly Modules Binary files with .wasm suffix Can be written by hand (assembly programming) or compiled from C/C++ Overview of assembly syntax S-Expressions</p> <p>Embedder Role of embedders Embedding in web browsers Embedding on server (Node.js 11/v8 7.0) Spec and WABT interpreters Custom embedders in your applications</p> <p>WABT – WebAssembly Binary Toolkit Low-level CLI tools to work with wasm modules Assembler, disassembler, interpreter, linker, extractor etc.</p> <p>Primitive Types Just four!! - i32 / i64 / f32 / f64 Working with primitives Primitive operations</p> <p>Functions Defining and calling functions Function parameters and return value The call opcode The start function Importing/exporting functions</p> <p>Linear Memory Handling strings in linear memory Memory imports and exports Data section</p> <p>Structured Control Flow Hierarchical targets Don't jump to specific address, rather move up levels in hierarchy block/loop/if/else/br_table opcode</p> <p>Tables Indirect function calls and security Tables and the call_indirect opcode Table imports and exports</p> <p>JavaScript API Think of a WebAssembly module as a low-level representation that is passed to web browser for internal/local compilation Calling from JS to wasm / wasm to JS</p> <p>Module Binary Format Binary format with well-defined layout Sections are either named or indexed Extensible – can add custom sections LEB128, opcodes, ..</p> <p>WebAssembly Threads Chrome 70 introduces wasn threads Shared state via shared array buffer</p> <p>Future Enhancements Future enhancements coming in areas such as SIMD, exceptions, garbage collection and synchronization</p> <p>Project Using all the major constructs of WebAssembly in a large project</p>