Glossary

General

LLVM backend

A (*Clang*) compiler backend that converts the LLVM Intermediate Representation (IR) to code for a specified machine or other languages. In the case of Emscripten, the specified target is JavaScript.

Load-store consistency

Load-Store Consistency (LSC), is the requirement that after a value with a specific type is written to a memory location, loads from that memory location will be of the same type. So if a variable contains a 32-bit floating point number, then both loads and stores to that variable will be of 32-bit floating point values, and not 16-bit unsigned integers or anything else.

> **Note**
>
> This definition is taken from Emscripten: An LLVM-to-JavaScript Compiler (section 2.1.1). There is additional detail in that paper.

Minifying

Minification in JavaScript is the process of removing all unnecessary characters from source code without changing its functionality. At higher optimisation levels Emscripten uses the Closure Compiler to minify Emscripten code.

Relooping

Recreate high-level loop and `if` structures from the low-level labels and branches that appear in LLVM assembly (definition taken from this paper).

SDL

Simple DirectMedia Layer (SDL) is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D.

The original compiler supported a number of other memory models and compilation modes (see Code Generation Modes) but *Typed Arrays Mode 2* proved to have, among other benefits, the greatest support for arbitrary code.

XHR

Contraction of `XMLHttpRequest`. Emscripten uses XHRs for asynchronously downloading binary data.

Emscripten tools and dependencies

Binaryen

Binaryen is a WebAssembly compiler toolkit, which Emscripten uses to modify and optimize Wasm.

Clang

Clang is a compiler front end for C, C++, and other programming languages that uses LLVM as its back end.

Closure Compiler

The closure compiler is used to minify Emscripten-generated code at higher optimisations.

Compiler Configuration File

The Compiler Configuration File stores the active tools and SDKs as defined using emsdk activate.

emcc

The Emscripten Compiler Frontend (emcc). Emscripten's drop-in replacement for a compiler like *gcc*.

Emscripten Command Prompt

The Emscripten Windows Command Prompt (emcmdprompt.bat) is used to call Emscripten tools from the command line on Windows.

Fastcomp

Fastcomp was Emscripten's second compiler core, after the JS compiler and before the new LLVM Wasm backend.

Git

Git is a distributed revision control system. Emscripten is hosted on GitHub and can be updated and modified using a git client.

GitHub

GitHub is a Git repository web-based hosting service that also offers project-based collaboration features including wikis, task management, and bug tracking.

The Emscripten project is hosted on GitHub.

Java

Java is a programming language and computing platform. It is used by Emscripten for the code that performs some advanced optimisations. The required version is listed in the toolchain requirements.

JavaScript

JavaScript (ECMAScript) is a programming language that is primarily used as part of a web browser, providing programmatic access to objects within a host environment. With node.js, it is also being used in server-side network programming.

The asm.js subset of JavaScript is Emscripten's target output language.

lli LLVM Interpreter

The LLVM interpreter (LLI) executes programs from LLVM bitcode. This tool is not maintained and has odd errors and crashes.

LLVM

LLVM is a compiler infrastructure designed to allow optimization of programs written in arbitrary programming languages.

node.js

Node.js is a cross-platform runtime environment for server-side and networking applications written in JavaScript. Essentially it allows you to run JavaScript applications outside of a browser context.

Python

Python is a scripting language used to write many of Emscripten's tools. The required version is listed in the toolchain requirements.

SDK Terms

The following terms are used when referring to the SDK and Emscripten SDK (emsdk):

emsdk

The Emscripten SDK (emsdk) is used to perform all SDK maintenance and can install, update, add, remove and activate SDKs and tools. Most operations are of the form `./emsdk command`. To access the *emsdk* script, launch the Emscripten Command Prompt.

Tool

The basic unit of software bundled in the SDK. A Tool has a name and a version. For example, **clang-3.2-32bit** is a tool that contains the 32-bit version of the *Clang* v3.2 compiler. Other tools used by *Emscripten* include Java, Git, node.js, etc.

SDK

A set of tools. For example, **sdk-1.5.6-32bit** is an SDK consisting of the tools: clang-3.2-32bit, node-0.10.17-32bit, python-2.7.5.1-32bit and emscripten-1.5.6.

There are a number of different Emscripten SDK packages. These can be downloaded from here.

Active Tool/SDK

The emsdk can store multiple versions of tools and SDKs. The active tools/SDK is the set of tools that are used by default on the *Emscripten Command Prompt*. This compiler configuration is stored in an emsdk-specific config file (**.emscripten**) and can be changed using *emsdk*.

emsdk root directory

The emsdk can manage any number of tools and SDKs, and these are stored in subdirectories of the *emsdk root directory*. The **emsdk root** is the directory specified when you first installed an SDK.

SDK root directory

The emsdk can store any number of tools and SDKs. The *SDK root directory* is the directory used to store a particular SDK. It is located as follows, with respect to the emsdk root directory: **<emsdk root>\emscripten\<sdk root directory>**

Site / Sphinx

reStructured text

Markup language used to define content on this site. See the reStructured text primer.