/\*\*

\* The WASI API provides an implementation of the [WebAssembly System Interface](https://wasi.dev/) specification. WASI gives sandboxed WebAssembly applications access to the

\* underlying operating system via a collection of POSIX-like functions.

\*

\* ```js

\* import { readFile } from 'fs/promises';

\* import { WASI } from 'wasi';

\* import { argv, env } from 'process';

\*

\* const wasi = new WASI({

\* args: argv,

\* env,

\* preopens: {

\* '/sandbox': '/some/real/path/that/wasm/can/access'

\* }

\* });

\*

\* // Some WASI binaries require:

\* // const importObject = { wasi\_unstable: wasi.wasiImport };

\* const importObject = { wasi\_snapshot\_preview1: wasi.wasiImport };

\*

\* const wasm = await WebAssembly.compile(

\* await readFile(new URL('./demo.wasm', import.meta.url))

\* );

\* const instance = await WebAssembly.instantiate(wasm, importObject);

\*

\* wasi.start(instance);

\* ```

\*

\* To run the above example, create a new WebAssembly text format file named`demo.wat`:

\*

\* ```text

\* (module

\* ;; Import the required fd\_write WASI function which will write the given io vectors to stdout

\* ;; The function signature for fd\_write is:

\* ;; (File Descriptor, \*iovs, iovs\_len, nwritten) -> Returns number of bytes written

\* (import "wasi\_snapshot\_preview1" "fd\_write" (func $fd\_write (param i32 i32 i32 i32) (result i32)))

\*

\* (memory 1)

\* (export "memory" (memory 0))

\*

\* ;; Write 'hello world\n' to memory at an offset of 8 bytes

\* ;; Note the trailing newline which is required for the text to appear

\* (data (i32.const 8) "hello world\n")

\*

\* (func $main (export "\_start")

\* ;; Creating a new io vector within linear memory

\* (i32.store (i32.const 0) (i32.const 8)) ;; iov.iov\_base - This is a pointer to the start of the 'hello world\n' string

\* (i32.store (i32.const 4) (i32.const 12)) ;; iov.iov\_len - The length of the 'hello world\n' string

\*

\* (call $fd\_write

\* (i32.const 1) ;; file\_descriptor - 1 for stdout

\* (i32.const 0) ;; \*iovs - The pointer to the iov array, which is stored at memory location 0

\* (i32.const 1) ;; iovs\_len - We're printing 1 string stored in an iov - so one.

\* (i32.const 20) ;; nwritten - A place in memory to store the number of bytes written

\* )

\* drop ;; Discard the number of bytes written from the top of the stack

\* )

\* )

\* ```

\*

\* Use [wabt](https://github.com/WebAssembly/wabt) to compile `.wat` to `.wasm`

\*

\* ```console

\* $ wat2wasm demo.wat

\* ```

\*

\* The `--experimental-wasi-unstable-preview1` CLI argument is needed for this

\* example to run.

\* @experimental

\* @see [source](https://github.com/nodejs/node/blob/v17.0.0/lib/wasi.js)

\*/

declare module 'wasi' {

interface WASIOptions {

/\*\*

\* An array of strings that the WebAssembly application will

\* see as command line arguments. The first argument is the virtual path to the

\* WASI command itself.

\*/

args?: string[] | undefined;

/\*\*

\* An object similar to `process.env` that the WebAssembly

\* application will see as its environment.

\*/

env?: object | undefined;

/\*\*

\* This object represents the WebAssembly application's

\* sandbox directory structure. The string keys of `preopens` are treated as

\* directories within the sandbox. The corresponding values in `preopens` are

\* the real paths to those directories on the host machine.

\*/

preopens?: NodeJS.Dict<string> | undefined;

/\*\*

\* By default, WASI applications terminate the Node.js

\* process via the `\_\_wasi\_proc\_exit()` function. Setting this option to `true`

\* causes `wasi.start()` to return the exit code rather than terminate the

\* process.

\* @default false

\*/

returnOnExit?: boolean | undefined;

/\*\*

\* The file descriptor used as standard input in the WebAssembly application.

\* @default 0

\*/

stdin?: number | undefined;

/\*\*

\* The file descriptor used as standard output in the WebAssembly application.

\* @default 1

\*/

stdout?: number | undefined;

/\*\*

\* The file descriptor used as standard error in the WebAssembly application.

\* @default 2

\*/

stderr?: number | undefined;

}

/\*\*

\* The `WASI` class provides the WASI system call API and additional convenience

\* methods for working with WASI-based applications. Each `WASI` instance

\* represents a distinct sandbox environment. For security purposes, each `WASI`instance must have its command-line arguments, environment variables, and

\* sandbox directory structure configured explicitly.

\* @since v13.3.0, v12.16.0

\*/

class WASI {

constructor(options?: WASIOptions);

/\*\*

\* Attempt to begin execution of `instance` as a WASI command by invoking its`\_start()` export. If `instance` does not contain a `\_start()` export, or if`instance` contains an `\_initialize()`

\* export, then an exception is thrown.

\*

\* `start()` requires that `instance` exports a [`WebAssembly.Memory`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/WebAssembly/Memory) named`memory`. If

\* `instance` does not have a `memory` export an exception is thrown.

\*

\* If `start()` is called more than once, an exception is thrown.

\* @since v13.3.0, v12.16.0

\*/

start(instance: object): void; // TODO: avoid DOM dependency until WASM moved to own lib.

/\*\*

\* Attempt to initialize `instance` as a WASI reactor by invoking its`\_initialize()` export, if it is present. If `instance` contains a `\_start()`export, then an exception is thrown.

\*

\* `initialize()` requires that `instance` exports a [`WebAssembly.Memory`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/WebAssembly/Memory) named`memory`.

\* If `instance` does not have a `memory` export an exception is thrown.

\*

\* If `initialize()` is called more than once, an exception is thrown.

\* @since v14.6.0, v12.19.0

\*/

initialize(instance: object): void; // TODO: avoid DOM dependency until WASM moved to own lib.

/\*\*

\* `wasiImport` is an object that implements the WASI system call API. This object

\* should be passed as the `wasi\_snapshot\_preview1` import during the instantiation

\* of a [`WebAssembly.Instance`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/WebAssembly/Instance).

\* @since v13.3.0, v12.16.0

\*/

readonly wasiImport: NodeJS.Dict<any>; // TODO: Narrow to DOM types

}

}

declare module 'node:wasi' {

export \* from 'wasi';

}