// Declare "static" methods in Error

interface ErrorConstructor {

/\*\* Create .stack property on a target object \*/

captureStackTrace(targetObject: object, constructorOpt?: Function): void;

/\*\*

\* Optional override for formatting stack traces

\*

\* @see https://v8.dev/docs/stack-trace-api#customizing-stack-traces

\*/

prepareStackTrace?: ((err: Error, stackTraces: NodeJS.CallSite[]) => any) | undefined;

stackTraceLimit: number;

}

/\*-----------------------------------------------\*

\* \*

\* GLOBAL \*

\* \*

------------------------------------------------\*/

// For backwards compability

interface NodeRequire extends NodeJS.Require { }

interface RequireResolve extends NodeJS.RequireResolve { }

interface NodeModule extends NodeJS.Module { }

declare var process: NodeJS.Process;

declare var console: Console;

declare var \_\_filename: string;

declare var \_\_dirname: string;

declare var require: NodeRequire;

declare var module: NodeModule;

// Same as module.exports

declare var exports: any;

/\*\*

\* Only available if `--expose-gc` is passed to the process.

\*/

declare var gc: undefined | (() => void);

//#region borrowed

// from https://github.com/microsoft/TypeScript/blob/38da7c600c83e7b31193a62495239a0fe478cb67/lib/lib.webworker.d.ts#L633 until moved to separate lib

/\*\* A controller object that allows you to abort one or more DOM requests as and when desired. \*/

interface AbortController {

/\*\*

\* Returns the AbortSignal object associated with this object.

\*/

readonly signal: AbortSignal;

/\*\*

\* Invoking this method will set this object's AbortSignal's aborted flag and signal to any observers that the associated activity is to be aborted.

\*/

abort(): void;

}

/\*\* A signal object that allows you to communicate with a DOM request (such as a Fetch) and abort it if required via an AbortController object. \*/

interface AbortSignal {

/\*\*

\* Returns true if this AbortSignal's AbortController has signaled to abort, and false otherwise.

\*/

readonly aborted: boolean;

}

declare var AbortController: {

prototype: AbortController;

new(): AbortController;

};

declare var AbortSignal: {

prototype: AbortSignal;

new(): AbortSignal;

// TODO: Add abort() static

};

//#endregion borrowed

//#region ArrayLike.at()

interface RelativeIndexable<T> {

/\*\*

\* Takes an integer value and returns the item at that index,

\* allowing for positive and negative integers.

\* Negative integers count back from the last item in the array.

\*/

at(index: number): T | undefined;

}

interface String extends RelativeIndexable<string> {}

interface Array<T> extends RelativeIndexable<T> {}

interface Int8Array extends RelativeIndexable<number> {}

interface Uint8Array extends RelativeIndexable<number> {}

interface Uint8ClampedArray extends RelativeIndexable<number> {}

interface Int16Array extends RelativeIndexable<number> {}

interface Uint16Array extends RelativeIndexable<number> {}

interface Int32Array extends RelativeIndexable<number> {}

interface Uint32Array extends RelativeIndexable<number> {}

interface Float32Array extends RelativeIndexable<number> {}

interface Float64Array extends RelativeIndexable<number> {}

interface BigInt64Array extends RelativeIndexable<bigint> {}

interface BigUint64Array extends RelativeIndexable<bigint> {}

//#endregion ArrayLike.at() end

/\*----------------------------------------------\*

\* \*

\* GLOBAL INTERFACES \*

\* \*

\*-----------------------------------------------\*/

declare namespace NodeJS {

interface CallSite {

/\*\*

\* Value of "this"

\*/

getThis(): unknown;

/\*\*

\* Type of "this" as a string.

\* This is the name of the function stored in the constructor field of

\* "this", if available. Otherwise the object's [[Class]] internal

\* property.

\*/

getTypeName(): string | null;

/\*\*

\* Current function

\*/

getFunction(): Function | undefined;

/\*\*

\* Name of the current function, typically its name property.

\* If a name property is not available an attempt will be made to try

\* to infer a name from the function's context.

\*/

getFunctionName(): string | null;

/\*\*

\* Name of the property [of "this" or one of its prototypes] that holds

\* the current function

\*/

getMethodName(): string | null;

/\*\*

\* Name of the script [if this function was defined in a script]

\*/

getFileName(): string | null;

/\*\*

\* Current line number [if this function was defined in a script]

\*/

getLineNumber(): number | null;

/\*\*

\* Current column number [if this function was defined in a script]

\*/

getColumnNumber(): number | null;

/\*\*

\* A call site object representing the location where eval was called

\* [if this function was created using a call to eval]

\*/

getEvalOrigin(): string | undefined;

/\*\*

\* Is this a toplevel invocation, that is, is "this" the global object?

\*/

isToplevel(): boolean;

/\*\*

\* Does this call take place in code defined by a call to eval?

\*/

isEval(): boolean;

/\*\*

\* Is this call in native V8 code?

\*/

isNative(): boolean;

/\*\*

\* Is this a constructor call?

\*/

isConstructor(): boolean;

}

interface ErrnoException extends Error {

errno?: number | undefined;

code?: string | undefined;

path?: string | undefined;

syscall?: string | undefined;

}

interface ReadableStream extends EventEmitter {

readable: boolean;

read(size?: number): string | Buffer;

setEncoding(encoding: BufferEncoding): this;

pause(): this;

resume(): this;

isPaused(): boolean;

pipe<T extends WritableStream>(destination: T, options?: { end?: boolean | undefined; }): T;

unpipe(destination?: WritableStream): this;

unshift(chunk: string | Uint8Array, encoding?: BufferEncoding): void;

wrap(oldStream: ReadableStream): this;

[Symbol.asyncIterator](): AsyncIterableIterator<string | Buffer>;

}

interface WritableStream extends EventEmitter {

writable: boolean;

write(buffer: Uint8Array | string, cb?: (err?: Error | null) => void): boolean;

write(str: string, encoding?: BufferEncoding, cb?: (err?: Error | null) => void): boolean;

end(cb?: () => void): void;

end(data: string | Uint8Array, cb?: () => void): void;

end(str: string, encoding?: BufferEncoding, cb?: () => void): void;

}

interface ReadWriteStream extends ReadableStream, WritableStream { }

interface RefCounted {

ref(): this;

unref(): this;

}

type TypedArray =

| Uint8Array

| Uint8ClampedArray

| Uint16Array

| Uint32Array

| Int8Array

| Int16Array

| Int32Array

| BigUint64Array

| BigInt64Array

| Float32Array

| Float64Array;

type ArrayBufferView = TypedArray | DataView;

interface Require {

(id: string): any;

resolve: RequireResolve;

cache: Dict<NodeModule>;

/\*\*

\* @deprecated

\*/

extensions: RequireExtensions;

main: Module | undefined;

}

interface RequireResolve {

(id: string, options?: { paths?: string[] | undefined; }): string;

paths(request: string): string[] | null;

}

interface RequireExtensions extends Dict<(m: Module, filename: string) => any> {

'.js': (m: Module, filename: string) => any;

'.json': (m: Module, filename: string) => any;

'.node': (m: Module, filename: string) => any;

}

interface Module {

/\*\*

\* `true` if the module is running during the Node.js preload

\*/

isPreloading: boolean;

exports: any;

require: Require;

id: string;

filename: string;

loaded: boolean;

/\*\* @deprecated since 14.6.0 Please use `require.main` and `module.children` instead. \*/

parent: Module | null | undefined;

children: Module[];

/\*\*

\* @since 11.14.0

\*

\* The directory name of the module. This is usually the same as the path.dirname() of the module.id.

\*/

path: string;

paths: string[];

}

interface Dict<T> {

[key: string]: T | undefined;

}

interface ReadOnlyDict<T> {

readonly [key: string]: T | undefined;

}

}