declare module 'path/posix' {

import path = require('path');

export = path;

}

declare module 'path/win32' {

import path = require('path');

export = path;

}

/\*\*

\* The `path` module provides utilities for working with file and directory paths.

\* It can be accessed using:

\*

\* ```js

\* const path = require('path');

\* ```

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

\*/

declare module 'path' {

namespace path {

/\*\*

\* A parsed path object generated by path.parse() or consumed by path.format().

\*/

interface ParsedPath {

/\*\*

\* The root of the path such as '/' or 'c:\'

\*/

root: string;

/\*\*

\* The full directory path such as '/home/user/dir' or 'c:\path\dir'

\*/

dir: string;

/\*\*

\* The file name including extension (if any) such as 'index.html'

\*/

base: string;

/\*\*

\* The file extension (if any) such as '.html'

\*/

ext: string;

/\*\*

\* The file name without extension (if any) such as 'index'

\*/

name: string;

}

interface FormatInputPathObject {

/\*\*

\* The root of the path such as '/' or 'c:\'

\*/

root?: string | undefined;

/\*\*

\* The full directory path such as '/home/user/dir' or 'c:\path\dir'

\*/

dir?: string | undefined;

/\*\*

\* The file name including extension (if any) such as 'index.html'

\*/

base?: string | undefined;

/\*\*

\* The file extension (if any) such as '.html'

\*/

ext?: string | undefined;

/\*\*

\* The file name without extension (if any) such as 'index'

\*/

name?: string | undefined;

}

interface PlatformPath {

/\*\*

\* Normalize a string path, reducing '..' and '.' parts.

\* When multiple slashes are found, they're replaced by a single one; when the path contains a trailing slash, it is preserved. On Windows backslashes are used.

\*

\* @param p string path to normalize.

\*/

normalize(p: string): string;

/\*\*

\* Join all arguments together and normalize the resulting path.

\* Arguments must be strings. In v0.8, non-string arguments were silently ignored. In v0.10 and up, an exception is thrown.

\*

\* @param paths paths to join.

\*/

join(...paths: string[]): string;

/\*\*

\* The right-most parameter is considered {to}. Other parameters are considered an array of {from}.

\*

\* Starting from leftmost {from} parameter, resolves {to} to an absolute path.

\*

\* If {to} isn't already absolute, {from} arguments are prepended in right to left order,

\* until an absolute path is found. If after using all {from} paths still no absolute path is found,

\* the current working directory is used as well. The resulting path is normalized,

\* and trailing slashes are removed unless the path gets resolved to the root directory.

\*

\* @param pathSegments string paths to join. Non-string arguments are ignored.

\*/

resolve(...pathSegments: string[]): string;

/\*\*

\* Determines whether {path} is an absolute path. An absolute path will always resolve to the same location, regardless of the working directory.

\*

\* @param path path to test.

\*/

isAbsolute(p: string): boolean;

/\*\*

\* Solve the relative path from {from} to {to}.

\* At times we have two absolute paths, and we need to derive the relative path from one to the other. This is actually the reverse transform of path.resolve.

\*/

relative(from: string, to: string): string;

/\*\*

\* Return the directory name of a path. Similar to the Unix dirname command.

\*

\* @param p the path to evaluate.

\*/

dirname(p: string): string;

/\*\*

\* Return the last portion of a path. Similar to the Unix basename command.

\* Often used to extract the file name from a fully qualified path.

\*

\* @param p the path to evaluate.

\* @param ext optionally, an extension to remove from the result.

\*/

basename(p: string, ext?: string): string;

/\*\*

\* Return the extension of the path, from the last '.' to end of string in the last portion of the path.

\* If there is no '.' in the last portion of the path or the first character of it is '.', then it returns an empty string

\*

\* @param p the path to evaluate.

\*/

extname(p: string): string;

/\*\*

\* The platform-specific file separator. '\\' or '/'.

\*/

readonly sep: string;

/\*\*

\* The platform-specific file delimiter. ';' or ':'.

\*/

readonly delimiter: string;

/\*\*

\* Returns an object from a path string - the opposite of format().

\*

\* @param pathString path to evaluate.

\*/

parse(p: string): ParsedPath;

/\*\*

\* Returns a path string from an object - the opposite of parse().

\*

\* @param pathString path to evaluate.

\*/

format(pP: FormatInputPathObject): string;

/\*\*

\* On Windows systems only, returns an equivalent namespace-prefixed path for the given path.

\* If path is not a string, path will be returned without modifications.

\* This method is meaningful only on Windows system.

\* On POSIX systems, the method is non-operational and always returns path without modifications.

\*/

toNamespacedPath(path: string): string;

/\*\*

\* Posix specific pathing.

\* Same as parent object on posix.

\*/

readonly posix: PlatformPath;

/\*\*

\* Windows specific pathing.

\* Same as parent object on windows

\*/

readonly win32: PlatformPath;

}

}

const path: path.PlatformPath;

export = path;

}

declare module 'node:path' {

import path = require('path');

export = path;

}

declare module 'node:path/posix' {

import path = require('path/posix');

export = path;

}

declare module 'node:path/win32' {

import path = require('path/win32');

export = path;

}