deno file system apideno file system api

目录

[1 Deno.env 1](#_Toc114973963)

[2 Deno.connect 1](#_Toc114973964)

[2.1 Parameters 1](#_Toc114973965)

[2.2 Return Type 1](#_Toc114973966)

[3 Deno.connectTls 1](#_Toc114973967)

[3.1 Parameters 1](#_Toc114973968)

[3.2 Return Type 2](#_Toc114973969)

[4 Deno.startTls 2](#_Toc114973970)

[4.1 Parameters 2](#_Toc114973971)

[4.2 Return Type 2](#_Toc114973972)

[5 Deno.resolveDns 2](#_Toc114973973)

[5.1 Parameters 3](#_Toc114973974)

[5.2 Return Type 3](#_Toc114973975)

[5.3 Parameters 3](#_Toc114973976)

[5.4 Return Type 4](#_Toc114973977)

[5.5 Parameters 4](#_Toc114973978)

[5.6 Return Type 4](#_Toc114973979)

[5.7 Parameters 4](#_Toc114973980)

[5.8 Return Type 4](#_Toc114973981)

[5.9 Parameters 4](#_Toc114973982)

[5.10 Return Type 5](#_Toc114973983)

[5.11 Parameters 5](#_Toc114973984)

[5.12 Return Type 5](#_Toc114973985)

[5.13 Parameters 5](#_Toc114973986)

[5.14 Return Type 5](#_Toc114973987)

[5.15 Parameters 6](#_Toc114973988)

[5.16 Return Type 6](#_Toc114973989)

[6 Deno.cwd 6](#_Toc114973990)

[6.1 Return Type 6](#_Toc114973991)

[7 Deno.readDir 6](#_Toc114973992)

[7.1 Parameters 7](#_Toc114973993)

[7.2 Return Type 7](#_Toc114973994)

[8 Deno.readFile 7](#_Toc114973995)

[8.1 Parameters 7](#_Toc114973996)

[8.2 Return Type 7](#_Toc114973997)

[9 Deno.readTextFile 7](#_Toc114973998)

[9.1 Parameters 7](#_Toc114973999)

[9.2 Return Type 8](#_Toc114974000)

[10 Deno.open 8](#_Toc114974001)

[10.1 Parameters 8](#_Toc114974002)

[10.2 Return Type 8](#_Toc114974003)

[11 Deno.stat 8](#_Toc114974004)

[11.1 Parameters 8](#_Toc114974005)

[11.2 Return Type 8](#_Toc114974006)

[12 Deno.lstat 8](#_Toc114974007)

[12.1 Parameters 9](#_Toc114974008)

[12.2 Return Type 9](#_Toc114974009)

[13 Deno.realPath 9](#_Toc114974010)

[13.1 Parameters 9](#_Toc114974011)

[13.2 Return Type 9](#_Toc114974012)

# Deno.env

<https://doc.deno.land/deno/stable/~/Deno.env>

const env: {

get(key: string): string | undefined;

set(key: string, value: string): void;

delete(key: string): void;

toObject(): {

[index: string]: string;

};

};

# Deno.connect

<https://doc.deno.land/deno/stable/~/Deno.connect>

Connects to the hostname (default is "127.0.0.1") and port on the named transport (default is "tcp"), and resolves to the connection (Conn).

const conn1 = await Deno.connect({ port: 80 });

const conn2 = await Deno.connect({ hostname: "192.0.2.1", port: 80 });

const conn3 = await Deno.connect({ hostname: "[2001:db8::1]", port: 80 });

const conn4 = await Deno.connect({ hostname: "golang.org", port: 80, transport: "tcp" });

Requires allow-net permission for "tcp".

function connect(options: [ConnectOptions](https://doc.deno.land/deno/stable/~/Deno.ConnectOptions)): Promise<[TcpConn](https://doc.deno.land/deno/stable/~/Deno.TcpConn)>;

connect(options: [ConnectOptions](https://doc.deno.land/deno/stable/~/Deno.ConnectOptions)): Promise<[TcpConn](https://doc.deno.land/deno/stable/~/Deno.TcpConn)>

## Parameters

options: [ConnectOptions](https://doc.deno.land/deno/stable/~/Deno.ConnectOptions)

## Return Type

Promise<[TcpConn](https://doc.deno.land/deno/stable/~/Deno.TcpConn)>

# Deno.connectTls

<https://doc.deno.land/deno/stable/~/Deno.connectTls>

Establishes a secure connection over TLS (transport layer security) using an optional cert file, hostname (default is "127.0.0.1") and port. The cert file is optional and if not included Mozilla's root certificates will be used (see also <https://github.com/ctz/webpki-roots> for specifics)

const caCert = await Deno.readTextFile("./certs/my\_custom\_root\_CA.pem");

const conn1 = await Deno.connectTls({ port: 80 });

const conn2 = await Deno.connectTls({ caCerts: [caCert], hostname: "192.0.2.1", port: 80 });

const conn3 = await Deno.connectTls({ hostname: "[2001:db8::1]", port: 80 });

const conn4 = await Deno.connectTls({ caCerts: [caCert], hostname: "golang.org", port: 80});

Requires allow-net permission.

function connectTls(options: [ConnectTlsOptions](https://doc.deno.land/deno/stable/~/Deno.ConnectTlsOptions)): Promise<[TlsConn](https://doc.deno.land/deno/stable/~/Deno.TlsConn)>;

connectTls(options: [ConnectTlsOptions](https://doc.deno.land/deno/stable/~/Deno.ConnectTlsOptions)): Promise<[TlsConn](https://doc.deno.land/deno/stable/~/Deno.TlsConn)>

## Parameters

options: [ConnectTlsOptions](https://doc.deno.land/deno/stable/~/Deno.ConnectTlsOptions)

## Return Type

Promise<[TlsConn](https://doc.deno.land/deno/stable/~/Deno.TlsConn)>

# Deno.startTls

<https://doc.deno.land/deno/stable/~/Deno.startTls>

Start TLS handshake from an existing connection using an optional list of CA certificates, and hostname (default is "127.0.0.1"). Specifying CA certs is optional. By default the configured root certificates are used. Using this function requires that the other end of the connection is prepared for a TLS handshake.

const conn = await Deno.connect({ port: 80, hostname: "127.0.0.1" });

const caCert = await Deno.readTextFile("./certs/my\_custom\_root\_CA.pem");

const tlsConn = await Deno.startTls(conn, { caCerts: [caCert], hostname: "localhost" });

Requires allow-net permission.

function startTls(conn: [Conn](https://doc.deno.land/deno/stable/~/Deno.Conn), options?: [StartTlsOptions](https://doc.deno.land/deno/stable/~/Deno.StartTlsOptions)): Promise<[TlsConn](https://doc.deno.land/deno/stable/~/Deno.TlsConn)>;

startTls(conn: [Conn](https://doc.deno.land/deno/stable/~/Deno.Conn), options?: [StartTlsOptions](https://doc.deno.land/deno/stable/~/Deno.StartTlsOptions)): Promise<[TlsConn](https://doc.deno.land/deno/stable/~/Deno.TlsConn)>

## Parameters

conn: [Conn](https://doc.deno.land/deno/stable/~/Deno.Conn)

options?: [StartTlsOptions](https://doc.deno.land/deno/stable/~/Deno.StartTlsOptions) **optional**

## Return Type

Promise<[TlsConn](https://doc.deno.land/deno/stable/~/Deno.TlsConn)>

# Deno.resolveDns

<https://doc.deno.land/deno/stable/~/Deno.resolveDns>

function resolveDns(

query: string,

recordType:

| "A"

| "AAAA"

| "ANAME"

| "CNAME"

| "NS"

| "PTR"

,

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions),

): Promise<string[]>;

function resolveDns(

query: string,

recordType: "CAA",

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions),

): Promise<[CAARecord](https://doc.deno.land/deno/stable/~/Deno.CAARecord)[]>;

function resolveDns(

query: string,

recordType: "MX",

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions),

): Promise<[MXRecord](https://doc.deno.land/deno/stable/~/Deno.MXRecord)[]>;

function resolveDns(

query: string,

recordType: "NAPTR",

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions),

): Promise<[NAPTRRecord](https://doc.deno.land/deno/stable/~/Deno.NAPTRRecord)[]>;

function resolveDns(

query: string,

recordType: "SOA",

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions),

): Promise<[SOARecord](https://doc.deno.land/deno/stable/~/Deno.SOARecord)[]>;

function resolveDns(

query: string,

recordType: "SRV",

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions),

): Promise<[SRVRecord](https://doc.deno.land/deno/stable/~/Deno.SRVRecord)[]>;

function resolveDns(

query: string,

recordType: "TXT",

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions),

): Promise<string[][]>;

function resolveDns(

query: string,

recordType: [RecordType](https://doc.deno.land/deno/stable/~/Deno.RecordType),

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions),

): Promise<string[] | [CAARecord](https://doc.deno.land/deno/stable/~/Deno.CAARecord)[] | [MXRecord](https://doc.deno.land/deno/stable/~/Deno.MXRecord)[] | [NAPTRRecord](https://doc.deno.land/deno/stable/~/Deno.NAPTRRecord)[] | [SOARecord](https://doc.deno.land/deno/stable/~/Deno.SOARecord)[] | [SRVRecord](https://doc.deno.land/deno/stable/~/Deno.SRVRecord)[] | string[][]>;

resolveDns(query: string, recordType: "A" | "AAAA" | "ANAME" | "CNAME" | "NS" | "PTR", options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions)): Promise<string[]>

## Parameters

query: string

recordType: "A" | "AAAA" | "ANAME" | "CNAME" | "NS" | "PTR"

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions) **optional**

## Return Type

Promise<string[]>

resolveDns(query: string, recordType: "CAA", options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions)): Promise<[CAARecord](https://doc.deno.land/deno/stable/~/Deno.CAARecord)[]>

## Parameters

query: string

recordType: "CAA"

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions) **optional**

## Return Type

Promise<[CAARecord](https://doc.deno.land/deno/stable/~/Deno.CAARecord)[]>

resolveDns(query: string, recordType: "MX", options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions)): Promise<[MXRecord](https://doc.deno.land/deno/stable/~/Deno.MXRecord)[]>

## Parameters

query: string

recordType: "MX"

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions) **optional**

## Return Type

Promise<[MXRecord](https://doc.deno.land/deno/stable/~/Deno.MXRecord)[]>

resolveDns(query: string, recordType: "NAPTR", options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions)): Promise<[NAPTRRecord](https://doc.deno.land/deno/stable/~/Deno.NAPTRRecord)[]>

## Parameters

query: string

recordType: "NAPTR"

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions) **optional**

## Return Type

Promise<[NAPTRRecord](https://doc.deno.land/deno/stable/~/Deno.NAPTRRecord)[]>

resolveDns(query: string, recordType: "SOA", options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions)): Promise<[SOARecord](https://doc.deno.land/deno/stable/~/Deno.SOARecord)[]>

## Parameters

query: string

recordType: "SOA"

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions) **optional**

## Return Type

Promise<[SOARecord](https://doc.deno.land/deno/stable/~/Deno.SOARecord)[]>

resolveDns(query: string, recordType: "SRV", options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions)): Promise<[SRVRecord](https://doc.deno.land/deno/stable/~/Deno.SRVRecord)[]>

## Parameters

query: string

recordType: "SRV"

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions) **optional**

## Return Type

Promise<[SRVRecord](https://doc.deno.land/deno/stable/~/Deno.SRVRecord)[]>

resolveDns(query: string, recordType: "TXT", options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions)): Promise<string[][]>

## Parameters

query: string

recordType: "TXT"

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions) **optional**

## Return Type

Promise<string[][]>

resolveDns(query: string, recordType: [RecordType](https://doc.deno.land/deno/stable/~/Deno.RecordType), options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions)): Promise<string[] | [CAARecord](https://doc.deno.land/deno/stable/~/Deno.CAARecord)[] | [MXRecord](https://doc.deno.land/deno/stable/~/Deno.MXRecord)[] | [NAPTRRecord](https://doc.deno.land/deno/stable/~/Deno.NAPTRRecord)[] | [SOARecord](https://doc.deno.land/deno/stable/~/Deno.SOARecord)[] | [SRVRecord](https://doc.deno.land/deno/stable/~/Deno.SRVRecord)[] | string[][]>

Performs DNS resolution against the given query, returning resolved records. Fails in the cases such as:

* the query is in invalid format
* the options have an invalid parameter, e.g. nameServer.port is beyond the range of 16-bit unsigned integer
* timed out

const a = await Deno.resolveDns("example.com", "A");

const aaaa = await Deno.resolveDns("example.com", "AAAA", {

nameServer: { ipAddr: "8.8.8.8", port: 53 },

});

Requires allow-net permission.

## Parameters

query: string

recordType: [RecordType](https://doc.deno.land/deno/stable/~/Deno.RecordType)

options?: [ResolveDnsOptions](https://doc.deno.land/deno/stable/~/Deno.ResolveDnsOptions) **optional**

## Return Type

Promise<string[] | [CAARecord](https://doc.deno.land/deno/stable/~/Deno.CAARecord)[] | [MXRecord](https://doc.deno.land/deno/stable/~/Deno.MXRecord)[] | [NAPTRRecord](https://doc.deno.land/deno/stable/~/Deno.NAPTRRecord)[] | [SOARecord](https://doc.deno.land/deno/stable/~/Deno.SOARecord)[] | [SRVRecord](https://doc.deno.land/deno/stable/~/Deno.SRVRecord)[] | string[][]>

# Deno.cwd

<https://doc.deno.land/deno/stable/~/Deno.cwd>

Return a string representing the current working directory.

If the current directory can be reached via multiple paths (due to symbolic links), cwd() may return any one of them.

const currentWorkingDirectory = Deno.cwd();

Throws Deno.errors.NotFound if directory not available.

Requires --allow-read

function cwd(): string;

cwd(): string

## Return Type

string

# Deno.readDir

<https://doc.deno.land/deno/stable/~/Deno.readDir>

Reads the directory given by path and returns an async iterable of Deno.DirEntry.

for await (const dirEntry of Deno.readDir("/")) {

console.log(dirEntry.name);

}

Throws error if path is not a directory.

Requires allow-read permission.

function readDir(path: string | [URL](https://doc.deno.land/deno/stable/~/URL)): AsyncIterable<[DirEntry](https://doc.deno.land/deno/stable/~/Deno.DirEntry)>;

readDir(path: string | [URL](https://doc.deno.land/deno/stable/~/URL)): AsyncIterable<[DirEntry](https://doc.deno.land/deno/stable/~/Deno.DirEntry)>

## Parameters

path: string | [URL](https://doc.deno.land/deno/stable/~/URL)

## Return Type

AsyncIterable<[DirEntry](https://doc.deno.land/deno/stable/~/Deno.DirEntry)>

# Deno.readFile

<https://doc.deno.land/deno/stable/~/Deno.readFile>

Reads and resolves to the entire contents of a file as an array of bytes. TextDecoder can be used to transform the bytes to string if required. Reading a directory returns an empty data array.

const decoder = new TextDecoder("utf-8");

const data = await Deno.readFile("hello.txt");

console.log(decoder.decode(data));

Requires allow-read permission.

function readFile(path: string | [URL](https://doc.deno.land/deno/stable/~/URL), options?: [ReadFileOptions](https://doc.deno.land/deno/stable/~/Deno.ReadFileOptions)): Promise<Uint8Array>;

readFile(path: string | [URL](https://doc.deno.land/deno/stable/~/URL), options?: [ReadFileOptions](https://doc.deno.land/deno/stable/~/Deno.ReadFileOptions)): Promise<Uint8Array>

## Parameters

path: string | [URL](https://doc.deno.land/deno/stable/~/URL)

options?: [ReadFileOptions](https://doc.deno.land/deno/stable/~/Deno.ReadFileOptions) **optional**

## Return Type

Promise<Uint8Array>

# Deno.readTextFile

<https://doc.deno.land/deno/stable/~/Deno.readTextFile>

Asynchronously reads and returns the entire contents of a file as utf8 encoded string. Reading a directory throws an error.

const data = await Deno.readTextFile("hello.txt");

console.log(data);

Requires allow-read permission.

function readTextFile(path: string | [URL](https://doc.deno.land/deno/stable/~/URL), options?: [ReadFileOptions](https://doc.deno.land/deno/stable/~/Deno.ReadFileOptions)): Promise<string>;

readTextFile(path: string | [URL](https://doc.deno.land/deno/stable/~/URL), options?: [ReadFileOptions](https://doc.deno.land/deno/stable/~/Deno.ReadFileOptions)): Promise<string>

## Parameters

path: string | [URL](https://doc.deno.land/deno/stable/~/URL)

options?: [ReadFileOptions](https://doc.deno.land/deno/stable/~/Deno.ReadFileOptions) **optional**

## Return Type

Promise<string>

# Deno.open

<https://doc.deno.land/deno/stable/~/Deno.open>

Open a file and resolve to an instance of Deno.FsFile. The file does not need to previously exist if using the create or createNew open options. It is the callers responsibility to close the file when finished with it.

const file = await Deno.open("/foo/bar.txt", { read: true, write: true });

// Do work with file

Deno.close(file.rid);

Requires allow-read and/or allow-write permissions depending on options.

function open(path: string | [URL](https://doc.deno.land/deno/stable/~/URL), options?: [OpenOptions](https://doc.deno.land/deno/stable/~/Deno.OpenOptions)): Promise<[FsFile](https://doc.deno.land/deno/stable/~/Deno.FsFile)>;

open(path: string | [URL](https://doc.deno.land/deno/stable/~/URL), options?: [OpenOptions](https://doc.deno.land/deno/stable/~/Deno.OpenOptions)): Promise<[FsFile](https://doc.deno.land/deno/stable/~/Deno.FsFile)>

## Parameters

path: string | [URL](https://doc.deno.land/deno/stable/~/URL)

options?: [OpenOptions](https://doc.deno.land/deno/stable/~/Deno.OpenOptions) **optional**

## Return Type

Promise<[FsFile](https://doc.deno.land/deno/stable/~/Deno.FsFile)>

# Deno.stat

<https://doc.deno.land/deno/stable/~/Deno.stat>

Resolves to a Deno.FileInfo for the specified path. Will always follow symlinks.

import { assert } from "https://deno.land/std/testing/asserts.ts";

const fileInfo = await Deno.stat("hello.txt");

assert(fileInfo.isFile);

Requires allow-read permission.

function stat(path: string | [URL](https://doc.deno.land/deno/stable/~/URL)): Promise<[FileInfo](https://doc.deno.land/deno/stable/~/Deno.FileInfo)>;

stat(path: string | [URL](https://doc.deno.land/deno/stable/~/URL)): Promise<[FileInfo](https://doc.deno.land/deno/stable/~/Deno.FileInfo)>

## Parameters

path: string | [URL](https://doc.deno.land/deno/stable/~/URL)

## Return Type

Promise<[FileInfo](https://doc.deno.land/deno/stable/~/Deno.FileInfo)>

# Deno.lstat

<https://doc.deno.land/deno/stable/~/Deno.lstat>

Resolves to a Deno.FileInfo for the specified path. If path is a symlink, information for the symlink will be returned instead of what it points to.

import { assert } from "https://deno.land/std/testing/asserts.ts";

const fileInfo = await Deno.lstat("hello.txt");

assert(fileInfo.isFile);

Requires allow-read permission.

function lstat(path: string | [URL](https://doc.deno.land/deno/stable/~/URL)): Promise<[FileInfo](https://doc.deno.land/deno/stable/~/Deno.FileInfo)>;

lstat(path: string | [URL](https://doc.deno.land/deno/stable/~/URL)): Promise<[FileInfo](https://doc.deno.land/deno/stable/~/Deno.FileInfo)>

## Parameters

path: string | [URL](https://doc.deno.land/deno/stable/~/URL)

## Return Type

Promise<[FileInfo](https://doc.deno.land/deno/stable/~/Deno.FileInfo)>

# Deno.realPath

<https://doc.deno.land/deno/stable/~/Deno.realPath>

Resolves to the absolute normalized path, with symbolic links resolved.

// e.g. given /home/alice/file.txt and current directory /home/alice

await Deno.symlink("file.txt", "symlink\_file.txt");

const realPath = await Deno.realPath("./file.txt");

const realSymLinkPath = await Deno.realPath("./symlink\_file.txt");

console.log(realPath); // outputs "/home/alice/file.txt"

console.log(realSymLinkPath); // outputs "/home/alice/file.txt"

Requires allow-read permission for the target path. Also requires allow-read permission for the CWD if the target path is relative.

function realPath(path: string | [URL](https://doc.deno.land/deno/stable/~/URL)): Promise<string>;

realPath(path: string | [URL](https://doc.deno.land/deno/stable/~/URL)): Promise<string>

## Parameters

path: string | [URL](https://doc.deno.land/deno/stable/~/URL)

## Return Type

Promise<string>