/\*\*

\* The `dns` module enables name resolution. For example, use it to look up IP

\* addresses of host names.

\*

\* Although named for the [Domain Name System (DNS)](https://en.wikipedia.org/wiki/Domain\_Name\_System), it does not always use the

\* DNS protocol for lookups. {@link lookup} uses the operating system

\* facilities to perform name resolution. It may not need to perform any network

\* communication. To perform name resolution the way other applications on the same

\* system do, use {@link lookup}.

\*

\* ```js

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

\*

\* dns.lookup('example.org', (err, address, family) => {

\* console.log('address: %j family: IPv%s', address, family);

\* });

\* // address: "93.184.216.34" family: IPv4

\* ```

\*

\* All other functions in the `dns` module connect to an actual DNS server to

\* perform name resolution. They will always use the network to perform DNS

\* queries. These functions do not use the same set of configuration files used by {@link lookup} (e.g. `/etc/hosts`). Use these functions to always perform

\* DNS queries, bypassing other name-resolution facilities.

\*

\* ```js

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

\*

\* dns.resolve4('archive.org', (err, addresses) => {

\* if (err) throw err;

\*

\* console.log(`addresses: ${JSON.stringify(addresses)}`);

\*

\* addresses.forEach((a) => {

\* dns.reverse(a, (err, hostnames) => {

\* if (err) {

\* throw err;

\* }

\* console.log(`reverse for ${a}: ${JSON.stringify(hostnames)}`);

\* });

\* });

\* });

\* ```

\*

\* See the `Implementation considerations section` for more information.

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

\*/

declare module 'dns' {

import \* as dnsPromises from 'node:dns/promises';

// Supported getaddrinfo flags.

export const ADDRCONFIG: number;

export const V4MAPPED: number;

/\*\*

\* If `dns.V4MAPPED` is specified, return resolved IPv6 addresses as

\* well as IPv4 mapped IPv6 addresses.

\*/

export const ALL: number;

export interface LookupOptions {

family?: number | undefined;

hints?: number | undefined;

all?: boolean | undefined;

/\*\*

\* @default true

\*/

verbatim?: boolean | undefined;

}

export interface LookupOneOptions extends LookupOptions {

all?: false | undefined;

}

export interface LookupAllOptions extends LookupOptions {

all: true;

}

export interface LookupAddress {

address: string;

family: number;

}

/\*\*

\* Resolves a host name (e.g. `'nodejs.org'`) into the first found A (IPv4) or

\* AAAA (IPv6) record. All `option` properties are optional. If `options` is an

\* integer, then it must be `4` or `6` – if `options` is not provided, then IPv4

\* and IPv6 addresses are both returned if found.

\*

\* With the `all` option set to `true`, the arguments for `callback` change to`(err, addresses)`, with `addresses` being an array of objects with the

\* properties `address` and `family`.

\*

\* On error, `err` is an `Error` object, where `err.code` is the error code.

\* Keep in mind that `err.code` will be set to `'ENOTFOUND'` not only when

\* the host name does not exist but also when the lookup fails in other ways

\* such as no available file descriptors.

\*

\* `dns.lookup()` does not necessarily have anything to do with the DNS protocol.

\* The implementation uses an operating system facility that can associate names

\* with addresses, and vice versa. This implementation can have subtle but

\* important consequences on the behavior of any Node.js program. Please take some

\* time to consult the `Implementation considerations section` before using`dns.lookup()`.

\*

\* Example usage:

\*

\* ```js

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

\* const options = {

\* family: 6,

\* hints: dns.ADDRCONFIG | dns.V4MAPPED,

\* };

\* dns.lookup('example.com', options, (err, address, family) =>

\* console.log('address: %j family: IPv%s', address, family));

\* // address: "2606:2800:220:1:248:1893:25c8:1946" family: IPv6

\*

\* // When options.all is true, the result will be an Array.

\* options.all = true;

\* dns.lookup('example.com', options, (err, addresses) =>

\* console.log('addresses: %j', addresses));

\* // addresses: [{"address":"2606:2800:220:1:248:1893:25c8:1946","family":6}]

\* ```

\*

\* If this method is invoked as its `util.promisify()` ed version, and `all`is not set to `true`, it returns a `Promise` for an `Object` with `address` and`family` properties.

\* @since v0.1.90

\*/

export function lookup(hostname: string, family: number, callback: (err: NodeJS.ErrnoException | null, address: string, family: number) => void): void;

export function lookup(hostname: string, options: LookupOneOptions, callback: (err: NodeJS.ErrnoException | null, address: string, family: number) => void): void;

export function lookup(hostname: string, options: LookupAllOptions, callback: (err: NodeJS.ErrnoException | null, addresses: LookupAddress[]) => void): void;

export function lookup(hostname: string, options: LookupOptions, callback: (err: NodeJS.ErrnoException | null, address: string | LookupAddress[], family: number) => void): void;

export function lookup(hostname: string, callback: (err: NodeJS.ErrnoException | null, address: string, family: number) => void): void;

export namespace lookup {

function \_\_promisify\_\_(hostname: string, options: LookupAllOptions): Promise<LookupAddress[]>;

function \_\_promisify\_\_(hostname: string, options?: LookupOneOptions | number): Promise<LookupAddress>;

function \_\_promisify\_\_(hostname: string, options: LookupOptions): Promise<LookupAddress | LookupAddress[]>;

}

/\*\*

\* Resolves the given `address` and `port` into a host name and service using

\* the operating system's underlying `getnameinfo` implementation.

\*

\* If `address` is not a valid IP address, a `TypeError` will be thrown.

\* The `port` will be coerced to a number. If it is not a legal port, a `TypeError`will be thrown.

\*

\* On an error, `err` is an `Error` object, where `err.code` is the error code.

\*

\* ```js

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

\* dns.lookupService('127.0.0.1', 22, (err, hostname, service) => {

\* console.log(hostname, service);

\* // Prints: localhost ssh

\* });

\* ```

\*

\* If this method is invoked as its `util.promisify()` ed version, it returns a`Promise` for an `Object` with `hostname` and `service` properties.

\* @since v0.11.14

\*/

export function lookupService(address: string, port: number, callback: (err: NodeJS.ErrnoException | null, hostname: string, service: string) => void): void;

export namespace lookupService {

function \_\_promisify\_\_(

address: string,

port: number

): Promise<{

hostname: string;

service: string;

}>;

}

export interface ResolveOptions {

ttl: boolean;

}

export interface ResolveWithTtlOptions extends ResolveOptions {

ttl: true;

}

export interface RecordWithTtl {

address: string;

ttl: number;

}

/\*\* @deprecated Use `AnyARecord` or `AnyAaaaRecord` instead. \*/

export type AnyRecordWithTtl = AnyARecord | AnyAaaaRecord;

export interface AnyARecord extends RecordWithTtl {

type: 'A';

}

export interface AnyAaaaRecord extends RecordWithTtl {

type: 'AAAA';

}

export interface CaaRecord {

critial: number;

issue?: string | undefined;

issuewild?: string | undefined;

iodef?: string | undefined;

contactemail?: string | undefined;

contactphone?: string | undefined;

}

export interface MxRecord {

priority: number;

exchange: string;

}

export interface AnyMxRecord extends MxRecord {

type: 'MX';

}

export interface NaptrRecord {

flags: string;

service: string;

regexp: string;

replacement: string;

order: number;

preference: number;

}

export interface AnyNaptrRecord extends NaptrRecord {

type: 'NAPTR';

}

export interface SoaRecord {

nsname: string;

hostmaster: string;

serial: number;

refresh: number;

retry: number;

expire: number;

minttl: number;

}

export interface AnySoaRecord extends SoaRecord {

type: 'SOA';

}

export interface SrvRecord {

priority: number;

weight: number;

port: number;

name: string;

}

export interface AnySrvRecord extends SrvRecord {

type: 'SRV';

}

export interface AnyTxtRecord {

type: 'TXT';

entries: string[];

}

export interface AnyNsRecord {

type: 'NS';

value: string;

}

export interface AnyPtrRecord {

type: 'PTR';

value: string;

}

export interface AnyCnameRecord {

type: 'CNAME';

value: string;

}

export type AnyRecord = AnyARecord | AnyAaaaRecord | AnyCnameRecord | AnyMxRecord | AnyNaptrRecord | AnyNsRecord | AnyPtrRecord | AnySoaRecord | AnySrvRecord | AnyTxtRecord;

/\*\*

\* Uses the DNS protocol to resolve a host name (e.g. `'nodejs.org'`) into an array

\* of the resource records. The `callback` function has arguments`(err, records)`. When successful, `records` will be an array of resource

\* records. The type and structure of individual results varies based on `rrtype`:

\*

\* <omitted>

\*

\* On error, `err` is an `Error` object, where `err.code` is one of theDNS error codes.

\* @since v0.1.27

\* @param hostname Host name to resolve.

\* @param [rrtype='A'] Resource record type.

\*/

export function resolve(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export function resolve(hostname: string, rrtype: 'A', callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export function resolve(hostname: string, rrtype: 'AAAA', callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export function resolve(hostname: string, rrtype: 'ANY', callback: (err: NodeJS.ErrnoException | null, addresses: AnyRecord[]) => void): void;

export function resolve(hostname: string, rrtype: 'CNAME', callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export function resolve(hostname: string, rrtype: 'MX', callback: (err: NodeJS.ErrnoException | null, addresses: MxRecord[]) => void): void;

export function resolve(hostname: string, rrtype: 'NAPTR', callback: (err: NodeJS.ErrnoException | null, addresses: NaptrRecord[]) => void): void;

export function resolve(hostname: string, rrtype: 'NS', callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export function resolve(hostname: string, rrtype: 'PTR', callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export function resolve(hostname: string, rrtype: 'SOA', callback: (err: NodeJS.ErrnoException | null, addresses: SoaRecord) => void): void;

export function resolve(hostname: string, rrtype: 'SRV', callback: (err: NodeJS.ErrnoException | null, addresses: SrvRecord[]) => void): void;

export function resolve(hostname: string, rrtype: 'TXT', callback: (err: NodeJS.ErrnoException | null, addresses: string[][]) => void): void;

export function resolve(

hostname: string,

rrtype: string,

callback: (err: NodeJS.ErrnoException | null, addresses: string[] | MxRecord[] | NaptrRecord[] | SoaRecord | SrvRecord[] | string[][] | AnyRecord[]) => void

): void;

export namespace resolve {

function \_\_promisify\_\_(hostname: string, rrtype?: 'A' | 'AAAA' | 'CNAME' | 'NS' | 'PTR'): Promise<string[]>;

function \_\_promisify\_\_(hostname: string, rrtype: 'ANY'): Promise<AnyRecord[]>;

function \_\_promisify\_\_(hostname: string, rrtype: 'MX'): Promise<MxRecord[]>;

function \_\_promisify\_\_(hostname: string, rrtype: 'NAPTR'): Promise<NaptrRecord[]>;

function \_\_promisify\_\_(hostname: string, rrtype: 'SOA'): Promise<SoaRecord>;

function \_\_promisify\_\_(hostname: string, rrtype: 'SRV'): Promise<SrvRecord[]>;

function \_\_promisify\_\_(hostname: string, rrtype: 'TXT'): Promise<string[][]>;

function \_\_promisify\_\_(hostname: string, rrtype: string): Promise<string[] | MxRecord[] | NaptrRecord[] | SoaRecord | SrvRecord[] | string[][] | AnyRecord[]>;

}

/\*\*

\* Uses the DNS protocol to resolve a IPv4 addresses (`A` records) for the`hostname`. The `addresses` argument passed to the `callback` function

\* will contain an array of IPv4 addresses (e.g.`['74.125.79.104', '74.125.79.105', '74.125.79.106']`).

\* @since v0.1.16

\* @param hostname Host name to resolve.

\*/

export function resolve4(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export function resolve4(hostname: string, options: ResolveWithTtlOptions, callback: (err: NodeJS.ErrnoException | null, addresses: RecordWithTtl[]) => void): void;

export function resolve4(hostname: string, options: ResolveOptions, callback: (err: NodeJS.ErrnoException | null, addresses: string[] | RecordWithTtl[]) => void): void;

export namespace resolve4 {

function \_\_promisify\_\_(hostname: string): Promise<string[]>;

function \_\_promisify\_\_(hostname: string, options: ResolveWithTtlOptions): Promise<RecordWithTtl[]>;

function \_\_promisify\_\_(hostname: string, options?: ResolveOptions): Promise<string[] | RecordWithTtl[]>;

}

/\*\*

\* Uses the DNS protocol to resolve a IPv6 addresses (`AAAA` records) for the`hostname`. The `addresses` argument passed to the `callback` function

\* will contain an array of IPv6 addresses.

\* @since v0.1.16

\* @param hostname Host name to resolve.

\*/

export function resolve6(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export function resolve6(hostname: string, options: ResolveWithTtlOptions, callback: (err: NodeJS.ErrnoException | null, addresses: RecordWithTtl[]) => void): void;

export function resolve6(hostname: string, options: ResolveOptions, callback: (err: NodeJS.ErrnoException | null, addresses: string[] | RecordWithTtl[]) => void): void;

export namespace resolve6 {

function \_\_promisify\_\_(hostname: string): Promise<string[]>;

function \_\_promisify\_\_(hostname: string, options: ResolveWithTtlOptions): Promise<RecordWithTtl[]>;

function \_\_promisify\_\_(hostname: string, options?: ResolveOptions): Promise<string[] | RecordWithTtl[]>;

}

/\*\*

\* Uses the DNS protocol to resolve `CNAME` records for the `hostname`. The`addresses` argument passed to the `callback` function

\* will contain an array of canonical name records available for the `hostname`(e.g. `['bar.example.com']`).

\* @since v0.3.2

\*/

export function resolveCname(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export namespace resolveCname {

function \_\_promisify\_\_(hostname: string): Promise<string[]>;

}

/\*\*

\* Uses the DNS protocol to resolve `CAA` records for the `hostname`. The`addresses` argument passed to the `callback` function

\* will contain an array of certification authority authorization records

\* available for the `hostname` (e.g. `[{critical: 0, iodef: 'mailto:pki@example.com'}, {critical: 128, issue: 'pki.example.com'}]`).

\* @since v15.0.0, v14.17.0

\*/

export function resolveCaa(hostname: string, callback: (err: NodeJS.ErrnoException | null, records: CaaRecord[]) => void): void;

export namespace resolveCaa {

function \_\_promisify\_\_(hostname: string): Promise<CaaRecord[]>;

}

/\*\*

\* Uses the DNS protocol to resolve mail exchange records (`MX` records) for the`hostname`. The `addresses` argument passed to the `callback` function will

\* contain an array of objects containing both a `priority` and `exchange`property (e.g. `[{priority: 10, exchange: 'mx.example.com'}, ...]`).

\* @since v0.1.27

\*/

export function resolveMx(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: MxRecord[]) => void): void;

export namespace resolveMx {

function \_\_promisify\_\_(hostname: string): Promise<MxRecord[]>;

}

/\*\*

\* Uses the DNS protocol to resolve regular expression based records (`NAPTR`records) for the `hostname`. The `addresses` argument passed to the `callback`function will contain an array of

\* objects with the following properties:

\*

\* \* `flags`

\* \* `service`

\* \* `regexp`

\* \* `replacement`

\* \* `order`

\* \* `preference`

\*

\* ```js

\* {

\* flags: 's',

\* service: 'SIP+D2U',

\* regexp: '',

\* replacement: '\_sip.\_udp.example.com',

\* order: 30,

\* preference: 100

\* }

\* ```

\* @since v0.9.12

\*/

export function resolveNaptr(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: NaptrRecord[]) => void): void;

export namespace resolveNaptr {

function \_\_promisify\_\_(hostname: string): Promise<NaptrRecord[]>;

}

/\*\*

\* Uses the DNS protocol to resolve name server records (`NS` records) for the`hostname`. The `addresses` argument passed to the `callback` function will

\* contain an array of name server records available for `hostname`(e.g. `['ns1.example.com', 'ns2.example.com']`).

\* @since v0.1.90

\*/

export function resolveNs(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export namespace resolveNs {

function \_\_promisify\_\_(hostname: string): Promise<string[]>;

}

/\*\*

\* Uses the DNS protocol to resolve pointer records (`PTR` records) for the`hostname`. The `addresses` argument passed to the `callback` function will

\* be an array of strings containing the reply records.

\* @since v6.0.0

\*/

export function resolvePtr(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: string[]) => void): void;

export namespace resolvePtr {

function \_\_promisify\_\_(hostname: string): Promise<string[]>;

}

/\*\*

\* Uses the DNS protocol to resolve a start of authority record (`SOA` record) for

\* the `hostname`. The `address` argument passed to the `callback` function will

\* be an object with the following properties:

\*

\* \* `nsname`

\* \* `hostmaster`

\* \* `serial`

\* \* `refresh`

\* \* `retry`

\* \* `expire`

\* \* `minttl`

\*

\* ```js

\* {

\* nsname: 'ns.example.com',

\* hostmaster: 'root.example.com',

\* serial: 2013101809,

\* refresh: 10000,

\* retry: 2400,

\* expire: 604800,

\* minttl: 3600

\* }

\* ```

\* @since v0.11.10

\*/

export function resolveSoa(hostname: string, callback: (err: NodeJS.ErrnoException | null, address: SoaRecord) => void): void;

export namespace resolveSoa {

function \_\_promisify\_\_(hostname: string): Promise<SoaRecord>;

}

/\*\*

\* Uses the DNS protocol to resolve service records (`SRV` records) for the`hostname`. The `addresses` argument passed to the `callback` function will

\* be an array of objects with the following properties:

\*

\* \* `priority`

\* \* `weight`

\* \* `port`

\* \* `name`

\*

\* ```js

\* {

\* priority: 10,

\* weight: 5,

\* port: 21223,

\* name: 'service.example.com'

\* }

\* ```

\* @since v0.1.27

\*/

export function resolveSrv(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: SrvRecord[]) => void): void;

export namespace resolveSrv {

function \_\_promisify\_\_(hostname: string): Promise<SrvRecord[]>;

}

/\*\*

\* Uses the DNS protocol to resolve text queries (`TXT` records) for the`hostname`. The `records` argument passed to the `callback` function is a

\* two-dimensional array of the text records available for `hostname` (e.g.`[ ['v=spf1 ip4:0.0.0.0 ', '~all' ] ]`). Each sub-array contains TXT chunks of

\* one record. Depending on the use case, these could be either joined together or

\* treated separately.

\* @since v0.1.27

\*/

export function resolveTxt(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: string[][]) => void): void;

export namespace resolveTxt {

function \_\_promisify\_\_(hostname: string): Promise<string[][]>;

}

/\*\*

\* Uses the DNS protocol to resolve all records (also known as `ANY` or `\*` query).

\* The `ret` argument passed to the `callback` function will be an array containing

\* various types of records. Each object has a property `type` that indicates the

\* type of the current record. And depending on the `type`, additional properties

\* will be present on the object:

\*

\* <omitted>

\*

\* Here is an example of the `ret` object passed to the callback:

\*

\* ```js

\* [ { type: 'A', address: '127.0.0.1', ttl: 299 },

\* { type: 'CNAME', value: 'example.com' },

\* { type: 'MX', exchange: 'alt4.aspmx.l.example.com', priority: 50 },

\* { type: 'NS', value: 'ns1.example.com' },

\* { type: 'TXT', entries: [ 'v=spf1 include:\_spf.example.com ~all' ] },

\* { type: 'SOA',

\* nsname: 'ns1.example.com',

\* hostmaster: 'admin.example.com',

\* serial: 156696742,

\* refresh: 900,

\* retry: 900,

\* expire: 1800,

\* minttl: 60 } ]

\* ```

\*

\* DNS server operators may choose not to respond to `ANY`queries. It may be better to call individual methods like {@link resolve4},{@link resolveMx}, and so on. For more details, see [RFC

\* 8482](https://tools.ietf.org/html/rfc8482).

\*/

export function resolveAny(hostname: string, callback: (err: NodeJS.ErrnoException | null, addresses: AnyRecord[]) => void): void;

export namespace resolveAny {

function \_\_promisify\_\_(hostname: string): Promise<AnyRecord[]>;

}

/\*\*

\* Performs a reverse DNS query that resolves an IPv4 or IPv6 address to an

\* array of host names.

\*

\* On error, `err` is an `Error` object, where `err.code` is

\* one of the `DNS error codes`.

\* @since v0.1.16

\*/

export function reverse(ip: string, callback: (err: NodeJS.ErrnoException | null, hostnames: string[]) => void): void;

/\*\*

\* Sets the IP address and port of servers to be used when performing DNS

\* resolution. The `servers` argument is an array of [RFC 5952](https://tools.ietf.org/html/rfc5952#section-6) formatted

\* addresses. If the port is the IANA default DNS port (53) it can be omitted.

\*

\* ```js

\* dns.setServers([

\* '4.4.4.4',

\* '[2001:4860:4860::8888]',

\* '4.4.4.4:1053',

\* '[2001:4860:4860::8888]:1053',

\* ]);

\* ```

\*

\* An error will be thrown if an invalid address is provided.

\*

\* The `dns.setServers()` method must not be called while a DNS query is in

\* progress.

\*

\* The {@link setServers} method affects only {@link resolve},`dns.resolve\*()` and {@link reverse} (and specifically \_not\_ {@link lookup}).

\*

\* This method works much like [resolve.conf](https://man7.org/linux/man-pages/man5/resolv.conf.5.html).

\* That is, if attempting to resolve with the first server provided results in a`NOTFOUND` error, the `resolve()` method will \_not\_ attempt to resolve with

\* subsequent servers provided. Fallback DNS servers will only be used if the

\* earlier ones time out or result in some other error.

\* @since v0.11.3

\* @param servers array of `RFC 5952` formatted addresses

\*/

export function setServers(servers: ReadonlyArray<string>): void;

/\*\*

\* Returns an array of IP address strings, formatted according to [RFC 5952](https://tools.ietf.org/html/rfc5952#section-6),

\* that are currently configured for DNS resolution. A string will include a port

\* section if a custom port is used.

\*

\* ```js

\* [

\* '4.4.4.4',

\* '2001:4860:4860::8888',

\* '4.4.4.4:1053',

\* '[2001:4860:4860::8888]:1053',

\* ]

\* ```

\* @since v0.11.3

\*/

export function getServers(): string[];

/\*\*

\* Set the default value of `verbatim` in {@link lookup} and `dnsPromises.lookup()`. The value could be:

\*

\* \* `ipv4first`: sets default `verbatim` `false`.

\* \* `verbatim`: sets default `verbatim` `true`.

\*

\* The default is `ipv4first` and {@link setDefaultResultOrder} have higher

\* priority than `--dns-result-order`. When using `worker threads`,{@link setDefaultResultOrder} from the main thread won't affect the default

\* dns orders in workers.

\* @since v16.4.0, v14.18.0

\* @param order must be `'ipv4first'` or `'verbatim'`.

\*/

export function setDefaultResultOrder(order: 'ipv4first' | 'verbatim'): void;

// Error codes

export const NODATA: string;

export const FORMERR: string;

export const SERVFAIL: string;

export const NOTFOUND: string;

export const NOTIMP: string;

export const REFUSED: string;

export const BADQUERY: string;

export const BADNAME: string;

export const BADFAMILY: string;

export const BADRESP: string;

export const CONNREFUSED: string;

export const TIMEOUT: string;

export const EOF: string;

export const FILE: string;

export const NOMEM: string;

export const DESTRUCTION: string;

export const BADSTR: string;

export const BADFLAGS: string;

export const NONAME: string;

export const BADHINTS: string;

export const NOTINITIALIZED: string;

export const LOADIPHLPAPI: string;

export const ADDRGETNETWORKPARAMS: string;

export const CANCELLED: string;

export interface ResolverOptions {

timeout?: number | undefined;

/\*\*

\* @default 4

\*/

tries?: number;

}

/\*\*

\* An independent resolver for DNS requests.

\*

\* Creating a new resolver uses the default server settings. Setting

\* the servers used for a resolver using `resolver.setServers()` does not affect

\* other resolvers:

\*

\* ```js

\* const { Resolver } = require('dns');

\* const resolver = new Resolver();

\* resolver.setServers(['4.4.4.4']);

\*

\* // This request will use the server at 4.4.4.4, independent of global settings.

\* resolver.resolve4('example.org', (err, addresses) => {

\* // ...

\* });

\* ```

\*

\* The following methods from the `dns` module are available:

\*

\* \* `resolver.getServers()`

\* \* `resolver.resolve()`

\* \* `resolver.resolve4()`

\* \* `resolver.resolve6()`

\* \* `resolver.resolveAny()`

\* \* `resolver.resolveCaa()`

\* \* `resolver.resolveCname()`

\* \* `resolver.resolveMx()`

\* \* `resolver.resolveNaptr()`

\* \* `resolver.resolveNs()`

\* \* `resolver.resolvePtr()`

\* \* `resolver.resolveSoa()`

\* \* `resolver.resolveSrv()`

\* \* `resolver.resolveTxt()`

\* \* `resolver.reverse()`

\* \* `resolver.setServers()`

\* @since v8.3.0

\*/

export class Resolver {

constructor(options?: ResolverOptions);

/\*\*

\* Cancel all outstanding DNS queries made by this resolver. The corresponding

\* callbacks will be called with an error with code `ECANCELLED`.

\* @since v8.3.0

\*/

cancel(): void;

getServers: typeof getServers;

resolve: typeof resolve;

resolve4: typeof resolve4;

resolve6: typeof resolve6;

resolveAny: typeof resolveAny;

resolveCname: typeof resolveCname;

resolveMx: typeof resolveMx;

resolveNaptr: typeof resolveNaptr;

resolveNs: typeof resolveNs;

resolvePtr: typeof resolvePtr;

resolveSoa: typeof resolveSoa;

resolveSrv: typeof resolveSrv;

resolveTxt: typeof resolveTxt;

reverse: typeof reverse;

/\*\*

\* The resolver instance will send its requests from the specified IP address.

\* This allows programs to specify outbound interfaces when used on multi-homed

\* systems.

\*

\* If a v4 or v6 address is not specified, it is set to the default, and the

\* operating system will choose a local address automatically.

\*

\* The resolver will use the v4 local address when making requests to IPv4 DNS

\* servers, and the v6 local address when making requests to IPv6 DNS servers.

\* The `rrtype` of resolution requests has no impact on the local address used.

\* @since v15.1.0, v14.17.0

\* @param [ipv4='0.0.0.0'] A string representation of an IPv4 address.

\* @param [ipv6='::0'] A string representation of an IPv6 address.

\*/

setLocalAddress(ipv4?: string, ipv6?: string): void;

setServers: typeof setServers;

}

export { dnsPromises as promises };

}

declare module 'node:dns' {

export \* from 'dns';

}