/\*\*

\* The `url` module provides utilities for URL resolution and parsing. It can be

\* accessed using:

\*

\* ```js

\* import url from 'url';

\* ```

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

\*/

declare module 'url' {

import { Blob } from 'node:buffer';

import { ClientRequestArgs } from 'node:http';

import { ParsedUrlQuery, ParsedUrlQueryInput } from 'node:querystring';

// Input to `url.format`

interface UrlObject {

auth?: string | null | undefined;

hash?: string | null | undefined;

host?: string | null | undefined;

hostname?: string | null | undefined;

href?: string | null | undefined;

pathname?: string | null | undefined;

protocol?: string | null | undefined;

search?: string | null | undefined;

slashes?: boolean | null | undefined;

port?: string | number | null | undefined;

query?: string | null | ParsedUrlQueryInput | undefined;

}

// Output of `url.parse`

interface Url {

auth: string | null;

hash: string | null;

host: string | null;

hostname: string | null;

href: string;

path: string | null;

pathname: string | null;

protocol: string | null;

search: string | null;

slashes: boolean | null;

port: string | null;

query: string | null | ParsedUrlQuery;

}

interface UrlWithParsedQuery extends Url {

query: ParsedUrlQuery;

}

interface UrlWithStringQuery extends Url {

query: string | null;

}

/\*\*

\* The `url.parse()` method takes a URL string, parses it, and returns a URL

\* object.

\*

\* A `TypeError` is thrown if `urlString` is not a string.

\*

\* A `URIError` is thrown if the `auth` property is present but cannot be decoded.

\*

\* Use of the legacy `url.parse()` method is discouraged. Users should

\* use the WHATWG `URL` API. Because the `url.parse()` method uses a

\* lenient, non-standard algorithm for parsing URL strings, security

\* issues can be introduced. Specifically, issues with [host name spoofing](https://hackerone.com/reports/678487) and

\* incorrect handling of usernames and passwords have been identified.

\* @since v0.1.25

\* @deprecated Legacy: Use the WHATWG URL API instead.

\* @param urlString The URL string to parse.

\* @param [parseQueryString=false] If `true`, the `query` property will always be set to an object returned by the {@link querystring} module's `parse()` method. If `false`, the `query` property

\* on the returned URL object will be an unparsed, undecoded string.

\* @param [slashesDenoteHost=false] If `true`, the first token after the literal string `//` and preceding the next `/` will be interpreted as the `host`. For instance, given `//foo/bar`, the

\* result would be `{host: 'foo', pathname: '/bar'}` rather than `{pathname: '//foo/bar'}`.

\*/

function parse(urlString: string): UrlWithStringQuery;

function parse(urlString: string, parseQueryString: false | undefined, slashesDenoteHost?: boolean): UrlWithStringQuery;

function parse(urlString: string, parseQueryString: true, slashesDenoteHost?: boolean): UrlWithParsedQuery;

function parse(urlString: string, parseQueryString: boolean, slashesDenoteHost?: boolean): Url;

/\*\*

\* The `url.format()` method returns a formatted URL string derived from`urlObject`.

\*

\* ```js

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

\* url.format({

\* protocol: 'https',

\* hostname: 'example.com',

\* pathname: '/some/path',

\* query: {

\* page: 1,

\* format: 'json'

\* }

\* });

\*

\* // => 'https://example.com/some/path?page=1&#x26;format=json'

\* ```

\*

\* If `urlObject` is not an object or a string, `url.format()` will throw a `TypeError`.

\*

\* The formatting process operates as follows:

\*

\* \* A new empty string `result` is created.

\* \* If `urlObject.protocol` is a string, it is appended as-is to `result`.

\* \* Otherwise, if `urlObject.protocol` is not `undefined` and is not a string, an `Error` is thrown.

\* \* For all string values of `urlObject.protocol` that \_do not end\_ with an ASCII

\* colon (`:`) character, the literal string `:` will be appended to `result`.

\* \* If either of the following conditions is true, then the literal string `//`will be appended to `result`:

\* \* `urlObject.slashes` property is true;

\* \* `urlObject.protocol` begins with `http`, `https`, `ftp`, `gopher`, or`file`;

\* \* If the value of the `urlObject.auth` property is truthy, and either`urlObject.host` or `urlObject.hostname` are not `undefined`, the value of`urlObject.auth` will be coerced into a string

\* and appended to `result`followed by the literal string `@`.

\* \* If the `urlObject.host` property is `undefined` then:

\* \* If the `urlObject.hostname` is a string, it is appended to `result`.

\* \* Otherwise, if `urlObject.hostname` is not `undefined` and is not a string,

\* an `Error` is thrown.

\* \* If the `urlObject.port` property value is truthy, and `urlObject.hostname`is not `undefined`:

\* \* The literal string `:` is appended to `result`, and

\* \* The value of `urlObject.port` is coerced to a string and appended to`result`.

\* \* Otherwise, if the `urlObject.host` property value is truthy, the value of`urlObject.host` is coerced to a string and appended to `result`.

\* \* If the `urlObject.pathname` property is a string that is not an empty string:

\* \* If the `urlObject.pathname`\_does not start\_ with an ASCII forward slash

\* (`/`), then the literal string `'/'` is appended to `result`.

\* \* The value of `urlObject.pathname` is appended to `result`.

\* \* Otherwise, if `urlObject.pathname` is not `undefined` and is not a string, an `Error` is thrown.

\* \* If the `urlObject.search` property is `undefined` and if the `urlObject.query`property is an `Object`, the literal string `?` is appended to `result`followed by the output of calling the

\* `querystring` module's `stringify()`method passing the value of `urlObject.query`.

\* \* Otherwise, if `urlObject.search` is a string:

\* \* If the value of `urlObject.search`\_does not start\_ with the ASCII question

\* mark (`?`) character, the literal string `?` is appended to `result`.

\* \* The value of `urlObject.search` is appended to `result`.

\* \* Otherwise, if `urlObject.search` is not `undefined` and is not a string, an `Error` is thrown.

\* \* If the `urlObject.hash` property is a string:

\* \* If the value of `urlObject.hash`\_does not start\_ with the ASCII hash (`#`)

\* character, the literal string `#` is appended to `result`.

\* \* The value of `urlObject.hash` is appended to `result`.

\* \* Otherwise, if the `urlObject.hash` property is not `undefined` and is not a

\* string, an `Error` is thrown.

\* \* `result` is returned.

\* @since v0.1.25

\* @deprecated Legacy: Use the WHATWG URL API instead.

\* @param urlObject A URL object (as returned by `url.parse()` or constructed otherwise). If a string, it is converted to an object by passing it to `url.parse()`.

\*/

function format(urlObject: URL, options?: URLFormatOptions): string;

/\*\*

\* The `url.format()` method returns a formatted URL string derived from`urlObject`.

\*

\* ```js

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

\* url.format({

\* protocol: 'https',

\* hostname: 'example.com',

\* pathname: '/some/path',

\* query: {

\* page: 1,

\* format: 'json'

\* }

\* });

\*

\* // => 'https://example.com/some/path?page=1&#x26;format=json'

\* ```

\*

\* If `urlObject` is not an object or a string, `url.format()` will throw a `TypeError`.

\*

\* The formatting process operates as follows:

\*

\* \* A new empty string `result` is created.

\* \* If `urlObject.protocol` is a string, it is appended as-is to `result`.

\* \* Otherwise, if `urlObject.protocol` is not `undefined` and is not a string, an `Error` is thrown.

\* \* For all string values of `urlObject.protocol` that \_do not end\_ with an ASCII

\* colon (`:`) character, the literal string `:` will be appended to `result`.

\* \* If either of the following conditions is true, then the literal string `//`will be appended to `result`:

\* \* `urlObject.slashes` property is true;

\* \* `urlObject.protocol` begins with `http`, `https`, `ftp`, `gopher`, or`file`;

\* \* If the value of the `urlObject.auth` property is truthy, and either`urlObject.host` or `urlObject.hostname` are not `undefined`, the value of`urlObject.auth` will be coerced into a string

\* and appended to `result`followed by the literal string `@`.

\* \* If the `urlObject.host` property is `undefined` then:

\* \* If the `urlObject.hostname` is a string, it is appended to `result`.

\* \* Otherwise, if `urlObject.hostname` is not `undefined` and is not a string,

\* an `Error` is thrown.

\* \* If the `urlObject.port` property value is truthy, and `urlObject.hostname`is not `undefined`:

\* \* The literal string `:` is appended to `result`, and

\* \* The value of `urlObject.port` is coerced to a string and appended to`result`.

\* \* Otherwise, if the `urlObject.host` property value is truthy, the value of`urlObject.host` is coerced to a string and appended to `result`.

\* \* If the `urlObject.pathname` property is a string that is not an empty string:

\* \* If the `urlObject.pathname`\_does not start\_ with an ASCII forward slash

\* (`/`), then the literal string `'/'` is appended to `result`.

\* \* The value of `urlObject.pathname` is appended to `result`.

\* \* Otherwise, if `urlObject.pathname` is not `undefined` and is not a string, an `Error` is thrown.

\* \* If the `urlObject.search` property is `undefined` and if the `urlObject.query`property is an `Object`, the literal string `?` is appended to `result`followed by the output of calling the

\* `querystring` module's `stringify()`method passing the value of `urlObject.query`.

\* \* Otherwise, if `urlObject.search` is a string:

\* \* If the value of `urlObject.search`\_does not start\_ with the ASCII question

\* mark (`?`) character, the literal string `?` is appended to `result`.

\* \* The value of `urlObject.search` is appended to `result`.

\* \* Otherwise, if `urlObject.search` is not `undefined` and is not a string, an `Error` is thrown.

\* \* If the `urlObject.hash` property is a string:

\* \* If the value of `urlObject.hash`\_does not start\_ with the ASCII hash (`#`)

\* character, the literal string `#` is appended to `result`.

\* \* The value of `urlObject.hash` is appended to `result`.

\* \* Otherwise, if the `urlObject.hash` property is not `undefined` and is not a

\* string, an `Error` is thrown.

\* \* `result` is returned.

\* @since v0.1.25

\* @deprecated Legacy: Use the WHATWG URL API instead.

\* @param urlObject A URL object (as returned by `url.parse()` or constructed otherwise). If a string, it is converted to an object by passing it to `url.parse()`.

\*/

function format(urlObject: UrlObject | string): string;

/\*\*

\* The `url.resolve()` method resolves a target URL relative to a base URL in a

\* manner similar to that of a Web browser resolving an anchor tag HREF.

\*

\* ```js

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

\* url.resolve('/one/two/three', 'four'); // '/one/two/four'

\* url.resolve('http://example.com/', '/one'); // 'http://example.com/one'

\* url.resolve('http://example.com/one', '/two'); // 'http://example.com/two'

\* ```

\*

\* You can achieve the same result using the WHATWG URL API:

\*

\* ```js

\* function resolve(from, to) {

\* const resolvedUrl = new URL(to, new URL(from, 'resolve://'));

\* if (resolvedUrl.protocol === 'resolve:') {

\* // `from` is a relative URL.

\* const { pathname, search, hash } = resolvedUrl;

\* return pathname + search + hash;

\* }

\* return resolvedUrl.toString();

\* }

\*

\* resolve('/one/two/three', 'four'); // '/one/two/four'

\* resolve('http://example.com/', '/one'); // 'http://example.com/one'

\* resolve('http://example.com/one', '/two'); // 'http://example.com/two'

\* ```

\* @since v0.1.25

\* @deprecated Legacy: Use the WHATWG URL API instead.

\* @param from The Base URL being resolved against.

\* @param to The HREF URL being resolved.

\*/

function resolve(from: string, to: string): string;

/\*\*

\* Returns the [Punycode](https://tools.ietf.org/html/rfc5891#section-4.4) ASCII serialization of the `domain`. If `domain` is an

\* invalid domain, the empty string is returned.

\*

\* It performs the inverse operation to {@link domainToUnicode}.

\*

\* This feature is only available if the `node` executable was compiled with `ICU` enabled. If not, the domain names are passed through unchanged.

\*

\* ```js

\* import url from 'url';

\*

\* console.log(url.domainToASCII('español.com'));

\* // Prints xn--espaol-zwa.com

\* console.log(url.domainToASCII('中文.com'));

\* // Prints xn--fiq228c.com

\* console.log(url.domainToASCII('xn--iñvalid.com'));

\* // Prints an empty string

\* ```

\* @since v7.4.0, v6.13.0

\*/

function domainToASCII(domain: string): string;

/\*\*

\* Returns the Unicode serialization of the `domain`. If `domain` is an invalid

\* domain, the empty string is returned.

\*

\* It performs the inverse operation to {@link domainToASCII}.

\*

\* This feature is only available if the `node` executable was compiled with `ICU` enabled. If not, the domain names are passed through unchanged.

\*

\* ```js

\* import url from 'url';

\*

\* console.log(url.domainToUnicode('xn--espaol-zwa.com'));

\* // Prints español.com

\* console.log(url.domainToUnicode('xn--fiq228c.com'));

\* // Prints 中文.com

\* console.log(url.domainToUnicode('xn--iñvalid.com'));

\* // Prints an empty string

\* ```

\* @since v7.4.0, v6.13.0

\*/

function domainToUnicode(domain: string): string;

/\*\*

\* This function ensures the correct decodings of percent-encoded characters as

\* well as ensuring a cross-platform valid absolute path string.

\*

\* ```js

\* import { fileURLToPath } from 'url';

\*

\* const \_\_filename = fileURLToPath(import.meta.url);

\*

\* new URL('file:///C:/path/').pathname; // Incorrect: /C:/path/

\* fileURLToPath('file:///C:/path/'); // Correct: C:\path\ (Windows)

\*

\* new URL('file://nas/foo.txt').pathname; // Incorrect: /foo.txt

\* fileURLToPath('file://nas/foo.txt'); // Correct: \\nas\foo.txt (Windows)

\*

\* new URL('file:///你好.txt').pathname; // Incorrect: /%E4%BD%A0%E5%A5%BD.txt

\* fileURLToPath('file:///你好.txt'); // Correct: /你好.txt (POSIX)

\*

\* new URL('file:///hello world').pathname; // Incorrect: /hello%20world

\* fileURLToPath('file:///hello world'); // Correct: /hello world (POSIX)

\* ```

\* @since v10.12.0

\* @param url The file URL string or URL object to convert to a path.

\* @return The fully-resolved platform-specific Node.js file path.

\*/

function fileURLToPath(url: string | URL): string;

/\*\*

\* This function ensures that `path` is resolved absolutely, and that the URL

\* control characters are correctly encoded when converting into a File URL.

\*

\* ```js

\* import { pathToFileURL } from 'url';

\*

\* new URL('/foo#1', 'file:'); // Incorrect: file:///foo#1

\* pathToFileURL('/foo#1'); // Correct: file:///foo%231 (POSIX)

\*

\* new URL('/some/path%.c', 'file:'); // Incorrect: file:///some/path%.c

\* pathToFileURL('/some/path%.c'); // Correct: file:///some/path%25.c (POSIX)

\* ```

\* @since v10.12.0

\* @param path The path to convert to a File URL.

\* @return The file URL object.

\*/

function pathToFileURL(path: string): URL;

/\*\*

\* This utility function converts a URL object into an ordinary options object as

\* expected by the `http.request()` and `https.request()` APIs.

\*

\* ```js

\* import { urlToHttpOptions } from 'url';

\* const myURL = new URL('https://a:b@測試?abc#foo');

\*

\* console.log(urlToHttpOptions(myURL));

\*

\* {

\* protocol: 'https:',

\* hostname: 'xn--g6w251d',

\* hash: '#foo',

\* search: '?abc',

\* pathname: '/',

\* path: '/?abc',

\* href: 'https://a:b@xn--g6w251d/?abc#foo',

\* auth: 'a:b'

\* }

\*

\* ```

\* @since v15.7.0, v14.18.0

\* @param url The `WHATWG URL` object to convert to an options object.

\* @return Options object

\*/

function urlToHttpOptions(url: URL): ClientRequestArgs;

interface URLFormatOptions {

auth?: boolean | undefined;

fragment?: boolean | undefined;

search?: boolean | undefined;

unicode?: boolean | undefined;

}

/\*\*

\* Browser-compatible `URL` class, implemented by following the WHATWG URL

\* Standard. [Examples of parsed URLs](https://url.spec.whatwg.org/#example-url-parsing) may be found in the Standard itself.

\* The `URL` class is also available on the global object.

\*

\* In accordance with browser conventions, all properties of `URL` objects

\* are implemented as getters and setters on the class prototype, rather than as

\* data properties on the object itself. Thus, unlike `legacy urlObject` s,

\* using the `delete` keyword on any properties of `URL` objects (e.g. `delete myURL.protocol`, `delete myURL.pathname`, etc) has no effect but will still

\* return `true`.

\* @since v7.0.0, v6.13.0

\*/

class URL {

/\*\*

\* Creates a `'blob:nodedata:...'` URL string that represents the given `Blob` object and can be used to retrieve the `Blob` later.

\*

\* ```js

\* const {

\* Blob,

\* resolveObjectURL,

\* } = require('buffer');

\*

\* const blob = new Blob(['hello']);

\* const id = URL.createObjectURL(blob);

\*

\* // later...

\*

\* const otherBlob = resolveObjectURL(id);

\* console.log(otherBlob.size);

\* ```

\*

\* The data stored by the registered `Blob` will be retained in memory until`URL.revokeObjectURL()` is called to remove it.

\*

\* `Blob` objects are registered within the current thread. If using Worker

\* Threads, `Blob` objects registered within one Worker will not be available

\* to other workers or the main thread.

\* @since v16.7.0

\* @experimental

\*/

static createObjectURL(blob: Blob): string;

/\*\*

\* Removes the stored `Blob` identified by the given ID.

\* @since v16.7.0

\* @experimental

\* @param id A `'blob:nodedata:...` URL string returned by a prior call to `URL.createObjectURL()`.

\*/

static revokeObjectURL(objectUrl: string): void;

constructor(input: string, base?: string | URL);

/\*\*

\* Gets and sets the fragment portion of the URL.

\*

\* ```js

\* const myURL = new URL('https://example.org/foo#bar');

\* console.log(myURL.hash);

\* // Prints #bar

\*

\* myURL.hash = 'baz';

\* console.log(myURL.href);

\* // Prints https://example.org/foo#baz

\* ```

\*

\* Invalid URL characters included in the value assigned to the `hash` property

\* are `percent-encoded`. The selection of which characters to

\* percent-encode may vary somewhat from what the {@link parse} and {@link format} methods would produce.

\*/

hash: string;

/\*\*

\* Gets and sets the host portion of the URL.

\*

\* ```js

\* const myURL = new URL('https://example.org:81/foo');

\* console.log(myURL.host);

\* // Prints example.org:81

\*

\* myURL.host = 'example.com:82';

\* console.log(myURL.href);

\* // Prints https://example.com:82/foo

\* ```

\*

\* Invalid host values assigned to the `host` property are ignored.

\*/

host: string;

/\*\*

\* Gets and sets the host name portion of the URL. The key difference between`url.host` and `url.hostname` is that `url.hostname` does \_not\_ include the

\* port.

\*

\* ```js

\* const myURL = new URL('https://example.org:81/foo');

\* console.log(myURL.hostname);

\* // Prints example.org

\*

\* // Setting the hostname does not change the port

\* myURL.hostname = 'example.com:82';

\* console.log(myURL.href);

\* // Prints https://example.com:81/foo

\*

\* // Use myURL.host to change the hostname and port

\* myURL.host = 'example.org:82';

\* console.log(myURL.href);

\* // Prints https://example.org:82/foo

\* ```

\*

\* Invalid host name values assigned to the `hostname` property are ignored.

\*/

hostname: string;

/\*\*

\* Gets and sets the serialized URL.

\*

\* ```js

\* const myURL = new URL('https://example.org/foo');

\* console.log(myURL.href);

\* // Prints https://example.org/foo

\*

\* myURL.href = 'https://example.com/bar';

\* console.log(myURL.href);

\* // Prints https://example.com/bar

\* ```

\*

\* Getting the value of the `href` property is equivalent to calling {@link toString}.

\*

\* Setting the value of this property to a new value is equivalent to creating a

\* new `URL` object using `new URL(value)`. Each of the `URL`object's properties will be modified.

\*

\* If the value assigned to the `href` property is not a valid URL, a `TypeError`will be thrown.

\*/

href: string;

/\*\*

\* Gets the read-only serialization of the URL's origin.

\*

\* ```js

\* const myURL = new URL('https://example.org/foo/bar?baz');

\* console.log(myURL.origin);

\* // Prints https://example.org

\* ```

\*

\* ```js

\* const idnURL = new URL('https://測試');

\* console.log(idnURL.origin);

\* // Prints https://xn--g6w251d

\*

\* console.log(idnURL.hostname);

\* // Prints xn--g6w251d

\* ```

\*/

readonly origin: string;

/\*\*

\* Gets and sets the password portion of the URL.

\*

\* ```js

\* const myURL = new URL('https://abc:xyz@example.com');

\* console.log(myURL.password);

\* // Prints xyz

\*

\* myURL.password = '123';

\* console.log(myURL.href);

\* // Prints https://abc:123@example.com

\* ```

\*

\* Invalid URL characters included in the value assigned to the `password` property

\* are `percent-encoded`. The selection of which characters to

\* percent-encode may vary somewhat from what the {@link parse} and {@link format} methods would produce.

\*/

password: string;

/\*\*

\* Gets and sets the path portion of the URL.

\*

\* ```js

\* const myURL = new URL('https://example.org/abc/xyz?123');

\* console.log(myURL.pathname);

\* // Prints /abc/xyz

\*

\* myURL.pathname = '/abcdef';

\* console.log(myURL.href);

\* // Prints https://example.org/abcdef?123

\* ```

\*

\* Invalid URL characters included in the value assigned to the `pathname`property are `percent-encoded`. The selection of which characters

\* to percent-encode may vary somewhat from what the {@link parse} and {@link format} methods would produce.

\*/

pathname: string;

/\*\*

\* Gets and sets the port portion of the URL.

\*

\* The port value may be a number or a string containing a number in the range`0` to `65535` (inclusive). Setting the value to the default port of the`URL` objects given `protocol` will

\* result in the `port` value becoming

\* the empty string (`''`).

\*

\* The port value can be an empty string in which case the port depends on

\* the protocol/scheme:

\*

\* <omitted>

\*

\* Upon assigning a value to the port, the value will first be converted to a

\* string using `.toString()`.

\*

\* If that string is invalid but it begins with a number, the leading number is

\* assigned to `port`.

\* If the number lies outside the range denoted above, it is ignored.

\*

\* ```js

\* const myURL = new URL('https://example.org:8888');

\* console.log(myURL.port);

\* // Prints 8888

\*

\* // Default ports are automatically transformed to the empty string

\* // (HTTPS protocol's default port is 443)

\* myURL.port = '443';

\* console.log(myURL.port);

\* // Prints the empty string

\* console.log(myURL.href);

\* // Prints https://example.org/

\*

\* myURL.port = 1234;

\* console.log(myURL.port);

\* // Prints 1234

\* console.log(myURL.href);

\* // Prints https://example.org:1234/

\*

\* // Completely invalid port strings are ignored

\* myURL.port = 'abcd';

\* console.log(myURL.port);

\* // Prints 1234

\*

\* // Leading numbers are treated as a port number

\* myURL.port = '5678abcd';

\* console.log(myURL.port);

\* // Prints 5678

\*

\* // Non-integers are truncated

\* myURL.port = 1234.5678;

\* console.log(myURL.port);

\* // Prints 1234

\*

\* // Out-of-range numbers which are not represented in scientific notation

\* // will be ignored.

\* myURL.port = 1e10; // 10000000000, will be range-checked as described below

\* console.log(myURL.port);

\* // Prints 1234

\* ```

\*

\* Numbers which contain a decimal point,

\* such as floating-point numbers or numbers in scientific notation,

\* are not an exception to this rule.

\* Leading numbers up to the decimal point will be set as the URL's port,

\* assuming they are valid:

\*

\* ```js

\* myURL.port = 4.567e21;

\* console.log(myURL.port);

\* // Prints 4 (because it is the leading number in the string '4.567e21')

\* ```

\*/

port: string;

/\*\*

\* Gets and sets the protocol portion of the URL.

\*

\* ```js

\* const myURL = new URL('https://example.org');

\* console.log(myURL.protocol);

\* // Prints https:

\*

\* myURL.protocol = 'ftp';

\* console.log(myURL.href);

\* // Prints ftp://example.org/

\* ```

\*

\* Invalid URL protocol values assigned to the `protocol` property are ignored.

\*/

protocol: string;

/\*\*

\* Gets and sets the serialized query portion of the URL.

\*

\* ```js

\* const myURL = new URL('https://example.org/abc?123');

\* console.log(myURL.search);

\* // Prints ?123

\*

\* myURL.search = 'abc=xyz';

\* console.log(myURL.href);

\* // Prints https://example.org/abc?abc=xyz

\* ```

\*

\* Any invalid URL characters appearing in the value assigned the `search`property will be `percent-encoded`. The selection of which

\* characters to percent-encode may vary somewhat from what the {@link parse} and {@link format} methods would produce.

\*/

search: string;

/\*\*

\* Gets the `URLSearchParams` object representing the query parameters of the

\* URL. This property is read-only but the `URLSearchParams` object it provides

\* can be used to mutate the URL instance; to replace the entirety of query

\* parameters of the URL, use the {@link search} setter. See `URLSearchParams` documentation for details.

\*

\* Use care when using `.searchParams` to modify the `URL` because,

\* per the WHATWG specification, the `URLSearchParams` object uses

\* different rules to determine which characters to percent-encode. For

\* instance, the `URL` object will not percent encode the ASCII tilde (`~`)

\* character, while `URLSearchParams` will always encode it:

\*

\* ```js

\* const myUrl = new URL('https://example.org/abc?foo=~bar');

\*

\* console.log(myUrl.search); // prints ?foo=~bar

\*

\* // Modify the URL via searchParams...

\* myUrl.searchParams.sort();

\*

\* console.log(myUrl.search); // prints ?foo=%7Ebar

\* ```

\*/

readonly searchParams: URLSearchParams;

/\*\*

\* Gets and sets the username portion of the URL.

\*

\* ```js

\* const myURL = new URL('https://abc:xyz@example.com');

\* console.log(myURL.username);

\* // Prints abc

\*

\* myURL.username = '123';

\* console.log(myURL.href);

\* // Prints https://123:xyz@example.com/

\* ```

\*

\* Any invalid URL characters appearing in the value assigned the `username`property will be `percent-encoded`. The selection of which

\* characters to percent-encode may vary somewhat from what the {@link parse} and {@link format} methods would produce.

\*/

username: string;

/\*\*

\* The `toString()` method on the `URL` object returns the serialized URL. The

\* value returned is equivalent to that of {@link href} and {@link toJSON}.

\*/

toString(): string;

/\*\*

\* The `toJSON()` method on the `URL` object returns the serialized URL. The

\* value returned is equivalent to that of {@link href} and {@link toString}.

\*

\* This method is automatically called when an `URL` object is serialized

\* with [`JSON.stringify()`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/JSON/stringify).

\*

\* ```js

\* const myURLs = [

\* new URL('https://www.example.com'),

\* new URL('https://test.example.org'),

\* ];

\* console.log(JSON.stringify(myURLs));

\* // Prints ["https://www.example.com/","https://test.example.org/"]

\* ```

\*/

toJSON(): string;

}

/\*\*

\* The `URLSearchParams` API provides read and write access to the query of a`URL`. The `URLSearchParams` class can also be used standalone with one of the

\* four following constructors.

\* The `URLSearchParams` class is also available on the global object.

\*

\* The WHATWG `URLSearchParams` interface and the `querystring` module have

\* similar purpose, but the purpose of the `querystring` module is more

\* general, as it allows the customization of delimiter characters (`&#x26;` and `=`).

\* On the other hand, this API is designed purely for URL query strings.

\*

\* ```js

\* const myURL = new URL('https://example.org/?abc=123');

\* console.log(myURL.searchParams.get('abc'));

\* // Prints 123

\*

\* myURL.searchParams.append('abc', 'xyz');

\* console.log(myURL.href);

\* // Prints https://example.org/?abc=123&#x26;abc=xyz

\*

\* myURL.searchParams.delete('abc');

\* myURL.searchParams.set('a', 'b');

\* console.log(myURL.href);

\* // Prints https://example.org/?a=b

\*

\* const newSearchParams = new URLSearchParams(myURL.searchParams);

\* // The above is equivalent to

\* // const newSearchParams = new URLSearchParams(myURL.search);

\*

\* newSearchParams.append('a', 'c');

\* console.log(myURL.href);

\* // Prints https://example.org/?a=b

\* console.log(newSearchParams.toString());

\* // Prints a=b&#x26;a=c

\*

\* // newSearchParams.toString() is implicitly called

\* myURL.search = newSearchParams;

\* console.log(myURL.href);

\* // Prints https://example.org/?a=b&#x26;a=c

\* newSearchParams.delete('a');

\* console.log(myURL.href);

\* // Prints https://example.org/?a=b&#x26;a=c

\* ```

\* @since v7.5.0, v6.13.0

\*/

class URLSearchParams implements Iterable<[string, string]> {

constructor(init?: URLSearchParams | string | Record<string, string | ReadonlyArray<string>> | Iterable<[string, string]> | ReadonlyArray<[string, string]>);

/\*\*

\* Append a new name-value pair to the query string.

\*/

append(name: string, value: string): void;

/\*\*

\* Remove all name-value pairs whose name is `name`.

\*/

delete(name: string): void;

/\*\*

\* Returns an ES6 `Iterator` over each of the name-value pairs in the query.

\* Each item of the iterator is a JavaScript `Array`. The first item of the `Array`is the `name`, the second item of the `Array` is the `value`.

\*

\* Alias for `urlSearchParams[@@iterator]()`.

\*/

entries(): IterableIterator<[string, string]>;

/\*\*

\* Iterates over each name-value pair in the query and invokes the given function.

\*

\* ```js

\* const myURL = new URL('https://example.org/?a=b&#x26;c=d');

\* myURL.searchParams.forEach((value, name, searchParams) => {

\* console.log(name, value, myURL.searchParams === searchParams);

\* });

\* // Prints:

\* // a b true

\* // c d true

\* ```

\* @param fn Invoked for each name-value pair in the query

\* @param thisArg To be used as `this` value for when `fn` is called

\*/

forEach<TThis = this>(callback: (this: TThis, value: string, name: string, searchParams: this) => void, thisArg?: TThis): void;

/\*\*

\* Returns the value of the first name-value pair whose name is `name`. If there

\* are no such pairs, `null` is returned.

\* @return or `null` if there is no name-value pair with the given `name`.

\*/

get(name: string): string | null;

/\*\*

\* Returns the values of all name-value pairs whose name is `name`. If there are

\* no such pairs, an empty array is returned.

\*/

getAll(name: string): string[];

/\*\*

\* Returns `true` if there is at least one name-value pair whose name is `name`.

\*/

has(name: string): boolean;

/\*\*

\* Returns an ES6 `Iterator` over the names of each name-value pair.

\*

\* ```js

\* const params = new URLSearchParams('foo=bar&#x26;foo=baz');

\* for (const name of params.keys()) {

\* console.log(name);

\* }

\* // Prints:

\* // foo

\* // foo

\* ```

\*/

keys(): IterableIterator<string>;

/\*\*

\* Sets the value in the `URLSearchParams` object associated with `name` to`value`. If there are any pre-existing name-value pairs whose names are `name`,

\* set the first such pair's value to `value` and remove all others. If not,

\* append the name-value pair to the query string.

\*

\* ```js

\* const params = new URLSearchParams();

\* params.append('foo', 'bar');

\* params.append('foo', 'baz');

\* params.append('abc', 'def');

\* console.log(params.toString());

\* // Prints foo=bar&#x26;foo=baz&#x26;abc=def

\*

\* params.set('foo', 'def');

\* params.set('xyz', 'opq');

\* console.log(params.toString());

\* // Prints foo=def&#x26;abc=def&#x26;xyz=opq

\* ```

\*/

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

/\*\*

\* Sort all existing name-value pairs in-place by their names. Sorting is done

\* with a [stable sorting algorithm](https://en.wikipedia.org/wiki/Sorting\_algorithm#Stability), so relative order between name-value pairs

\* with the same name is preserved.

\*

\* This method can be used, in particular, to increase cache hits.

\*

\* ```js

\* const params = new URLSearchParams('query[]=abc&#x26;type=search&#x26;query[]=123');

\* params.sort();

\* console.log(params.toString());

\* // Prints query%5B%5D=abc&#x26;query%5B%5D=123&#x26;type=search

\* ```

\* @since v7.7.0, v6.13.0

\*/

sort(): void;

/\*\*

\* Returns the search parameters serialized as a string, with characters

\* percent-encoded where necessary.

\*/

toString(): string;

/\*\*

\* Returns an ES6 `Iterator` over the values of each name-value pair.

\*/

values(): IterableIterator<string>;

[Symbol.iterator](): IterableIterator<[string, string]>;

}

}

declare module 'node:url' {

export \* from 'url';

}