/\*\*

\* The `util` module supports the needs of Node.js internal APIs. Many of the

\* utilities are useful for application and module developers as well. To access

\* it:

\*

\* ```js

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

\* ```

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

\*/

declare module 'util' {

import \* as types from 'node:util/types';

export interface InspectOptions {

/\*\*

\* If set to `true`, getters are going to be

\* inspected as well. If set to `'get'` only getters without setter are going

\* to be inspected. If set to `'set'` only getters having a corresponding

\* setter are going to be inspected. This might cause side effects depending on

\* the getter function.

\* @default `false`

\*/

getters?: 'get' | 'set' | boolean | undefined;

showHidden?: boolean | undefined;

/\*\*

\* @default 2

\*/

depth?: number | null | undefined;

colors?: boolean | undefined;

customInspect?: boolean | undefined;

showProxy?: boolean | undefined;

maxArrayLength?: number | null | undefined;

/\*\*

\* Specifies the maximum number of characters to

\* include when formatting. Set to `null` or `Infinity` to show all elements.

\* Set to `0` or negative to show no characters.

\* @default 10000

\*/

maxStringLength?: number | null | undefined;

breakLength?: number | undefined;

/\*\*

\* Setting this to `false` causes each object key

\* to be displayed on a new line. It will also add new lines to text that is

\* longer than `breakLength`. If set to a number, the most `n` inner elements

\* are united on a single line as long as all properties fit into

\* `breakLength`. Short array elements are also grouped together. Note that no

\* text will be reduced below 16 characters, no matter the `breakLength` size.

\* For more information, see the example below.

\* @default `true`

\*/

compact?: boolean | number | undefined;

sorted?: boolean | ((a: string, b: string) => number) | undefined;

}

export type Style = 'special' | 'number' | 'bigint' | 'boolean' | 'undefined' | 'null' | 'string' | 'symbol' | 'date' | 'regexp' | 'module';

export type CustomInspectFunction = (depth: number, options: InspectOptionsStylized) => string;

export interface InspectOptionsStylized extends InspectOptions {

stylize(text: string, styleType: Style): string;

}

/\*\*

\* The `util.format()` method returns a formatted string using the first argument

\* as a `printf`\-like format string which can contain zero or more format

\* specifiers. Each specifier is replaced with the converted value from the

\* corresponding argument. Supported specifiers are:

\*

\* If a specifier does not have a corresponding argument, it is not replaced:

\*

\* ```js

\* util.format('%s:%s', 'foo');

\* // Returns: 'foo:%s'

\* ```

\*

\* Values that are not part of the format string are formatted using`util.inspect()` if their type is not `string`.

\*

\* If there are more arguments passed to the `util.format()` method than the

\* number of specifiers, the extra arguments are concatenated to the returned

\* string, separated by spaces:

\*

\* ```js

\* util.format('%s:%s', 'foo', 'bar', 'baz');

\* // Returns: 'foo:bar baz'

\* ```

\*

\* If the first argument does not contain a valid format specifier, `util.format()`returns a string that is the concatenation of all arguments separated by spaces:

\*

\* ```js

\* util.format(1, 2, 3);

\* // Returns: '1 2 3'

\* ```

\*

\* If only one argument is passed to `util.format()`, it is returned as it is

\* without any formatting:

\*

\* ```js

\* util.format('%% %s');

\* // Returns: '%% %s'

\* ```

\*

\* `util.format()` is a synchronous method that is intended as a debugging tool.

\* Some input values can have a significant performance overhead that can block the

\* event loop. Use this function with care and never in a hot code path.

\* @since v0.5.3

\* @param format A `printf`-like format string.

\*/

export function format(format?: any, ...param: any[]): string;

/\*\*

\* This function is identical to {@link format}, except in that it takes

\* an `inspectOptions` argument which specifies options that are passed along to {@link inspect}.

\*

\* ```js

\* util.formatWithOptions({ colors: true }, 'See object %O', { foo: 42 });

\* // Returns 'See object { foo: 42 }', where `42` is colored as a number

\* // when printed to a terminal.

\* ```

\* @since v10.0.0

\*/

export function formatWithOptions(inspectOptions: InspectOptions, format?: any, ...param: any[]): string;

/\*\*

\* Returns the string name for a numeric error code that comes from a Node.js API.

\* The mapping between error codes and error names is platform-dependent.

\* See `Common System Errors` for the names of common errors.

\*

\* ```js

\* fs.access('file/that/does/not/exist', (err) => {

\* const name = util.getSystemErrorName(err.errno);

\* console.error(name); // ENOENT

\* });

\* ```

\* @since v9.7.0

\*/

export function getSystemErrorName(err: number): string;

/\*\*

\* Returns a Map of all system error codes available from the Node.js API.

\* The mapping between error codes and error names is platform-dependent.

\* See `Common System Errors` for the names of common errors.

\*

\* ```js

\* fs.access('file/that/does/not/exist', (err) => {

\* const errorMap = util.getSystemErrorMap();

\* const name = errorMap.get(err.errno);

\* console.error(name); // ENOENT

\* });

\* ```

\* @since v16.0.0, v14.17.0

\*/

export function getSystemErrorMap(): Map<number, [string, string]>;

/\*\*

\* The `util.log()` method prints the given `string` to `stdout` with an included

\* timestamp.

\*

\* ```js

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

\*

\* util.log('Timestamped message.');

\* ```

\* @since v0.3.0

\* @deprecated Since v6.0.0 - Use a third party module instead.

\*/

export function log(string: string): void;

/\*\*

\* Returns the `string` after replacing any surrogate code points

\* (or equivalently, any unpaired surrogate code units) with the

\* Unicode "replacement character" U+FFFD.

\* @since v16.8.0, v14.18.0

\*/

export function toUSVString(string: string): string;

/\*\*

\* The `util.inspect()` method returns a string representation of `object` that is

\* intended for debugging. The output of `util.inspect` may change at any time

\* and should not be depended upon programmatically. Additional `options` may be

\* passed that alter the result.`util.inspect()` will use the constructor's name and/or `@@toStringTag` to make

\* an identifiable tag for an inspected value.

\*

\* ```js

\* class Foo {

\* get [Symbol.toStringTag]() {

\* return 'bar';

\* }

\* }

\*

\* class Bar {}

\*

\* const baz = Object.create(null, { [Symbol.toStringTag]: { value: 'foo' } });

\*

\* util.inspect(new Foo()); // 'Foo [bar] {}'

\* util.inspect(new Bar()); // 'Bar {}'

\* util.inspect(baz); // '[foo] {}'

\* ```

\*

\* Circular references point to their anchor by using a reference index:

\*

\* ```js

\* const { inspect } = require('util');

\*

\* const obj = {};

\* obj.a = [obj];

\* obj.b = {};

\* obj.b.inner = obj.b;

\* obj.b.obj = obj;

\*

\* console.log(inspect(obj));

\* // <ref \*1> {

\* // a: [ [Circular \*1] ],

\* // b: <ref \*2> { inner: [Circular \*2], obj: [Circular \*1] }

\* // }

\* ```

\*

\* The following example inspects all properties of the `util` object:

\*

\* ```js

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

\*

\* console.log(util.inspect(util, { showHidden: true, depth: null }));

\* ```

\*

\* The following example highlights the effect of the `compact` option:

\*

\* ```js

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

\*

\* const o = {

\* a: [1, 2, [[

\* 'Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit, sed do ' +

\* 'eiusmod \ntempor incididunt ut labore et dolore magna aliqua.',

\* 'test',

\* 'foo']], 4],

\* b: new Map([['za', 1], ['zb', 'test']])

\* };

\* console.log(util.inspect(o, { compact: true, depth: 5, breakLength: 80 }));

\*

\* // { a:

\* // [ 1,

\* // 2,

\* // [ [ 'Lorem ipsum dolor sit amet,\nconsectetur [...]', // A long line

\* // 'test',

\* // 'foo' ] ],

\* // 4 ],

\* // b: Map(2) { 'za' => 1, 'zb' => 'test' } }

\*

\* // Setting `compact` to false or an integer creates more reader friendly output.

\* console.log(util.inspect(o, { compact: false, depth: 5, breakLength: 80 }));

\*

\* // {

\* // a: [

\* // 1,

\* // 2,

\* // [

\* // [

\* // 'Lorem ipsum dolor sit amet,\n' +

\* // 'consectetur adipiscing elit, sed do eiusmod \n' +

\* // 'tempor incididunt ut labore et dolore magna aliqua.',

\* // 'test',

\* // 'foo'

\* // ]

\* // ],

\* // 4

\* // ],

\* // b: Map(2) {

\* // 'za' => 1,

\* // 'zb' => 'test'

\* // }

\* // }

\*

\* // Setting `breakLength` to e.g. 150 will print the "Lorem ipsum" text in a

\* // single line.

\* ```

\*

\* The `showHidden` option allows [`WeakMap`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/WeakMap) and

\* [`WeakSet`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/WeakSet) entries to be

\* inspected. If there are more entries than `maxArrayLength`, there is no

\* guarantee which entries are displayed. That means retrieving the same [`WeakSet`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/WeakSet) entries twice may

\* result in different output. Furthermore, entries

\* with no remaining strong references may be garbage collected at any time.

\*

\* ```js

\* const { inspect } = require('util');

\*

\* const obj = { a: 1 };

\* const obj2 = { b: 2 };

\* const weakSet = new WeakSet([obj, obj2]);

\*

\* console.log(inspect(weakSet, { showHidden: true }));

\* // WeakSet { { a: 1 }, { b: 2 } }

\* ```

\*

\* The `sorted` option ensures that an object's property insertion order does not

\* impact the result of `util.inspect()`.

\*

\* ```js

\* const { inspect } = require('util');

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

\*

\* const o1 = {

\* b: [2, 3, 1],

\* a: '`a` comes before `b`',

\* c: new Set([2, 3, 1])

\* };

\* console.log(inspect(o1, { sorted: true }));

\* // { a: '`a` comes before `b`', b: [ 2, 3, 1 ], c: Set(3) { 1, 2, 3 } }

\* console.log(inspect(o1, { sorted: (a, b) => b.localeCompare(a) }));

\* // { c: Set(3) { 3, 2, 1 }, b: [ 2, 3, 1 ], a: '`a` comes before `b`' }

\*

\* const o2 = {

\* c: new Set([2, 1, 3]),

\* a: '`a` comes before `b`',

\* b: [2, 3, 1]

\* };

\* assert.strict.equal(

\* inspect(o1, { sorted: true }),

\* inspect(o2, { sorted: true })

\* );

\* ```

\*

\* `util.inspect()` is a synchronous method intended for debugging. Its maximum

\* output length is approximately 128 MB. Inputs that result in longer output will

\* be truncated.

\* @since v0.3.0

\* @param object Any JavaScript primitive or `Object`.

\* @return The representation of `object`.

\*/

export function inspect(object: any, showHidden?: boolean, depth?: number | null, color?: boolean): string;

export function inspect(object: any, options: InspectOptions): string;

export namespace inspect {

let colors: NodeJS.Dict<[number, number]>;

let styles: {

[K in Style]: string;

};

let defaultOptions: InspectOptions;

/\*\*

\* Allows changing inspect settings from the repl.

\*/

let replDefaults: InspectOptions;

/\*\*

\* That can be used to declare custom inspect functions.

\*/

const custom: unique symbol;

}

/\*\*

\* Alias for [`Array.isArray()`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array/isArray).

\*

\* Returns `true` if the given `object` is an `Array`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isArray([]);

\* // Returns: true

\* util.isArray(new Array());

\* // Returns: true

\* util.isArray({});

\* // Returns: false

\* ```

\* @since v0.6.0

\* @deprecated Since v4.0.0 - Use `isArray` instead.

\*/

export function isArray(object: unknown): object is unknown[];

/\*\*

\* Returns `true` if the given `object` is a `RegExp`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isRegExp(/some regexp/);

\* // Returns: true

\* util.isRegExp(new RegExp('another regexp'));

\* // Returns: true

\* util.isRegExp({});

\* // Returns: false

\* ```

\* @since v0.6.0

\* @deprecated Since v4.0.0 - Deprecated

\*/

export function isRegExp(object: unknown): object is RegExp;

/\*\*

\* Returns `true` if the given `object` is a `Date`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isDate(new Date());

\* // Returns: true

\* util.isDate(Date());

\* // false (without 'new' returns a String)

\* util.isDate({});

\* // Returns: false

\* ```

\* @since v0.6.0

\* @deprecated Since v4.0.0 - Use {@link types.isDate} instead.

\*/

export function isDate(object: unknown): object is Date;

/\*\*

\* Returns `true` if the given `object` is an `Error`. Otherwise, returns`false`.

\*

\* ```js

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

\*

\* util.isError(new Error());

\* // Returns: true

\* util.isError(new TypeError());

\* // Returns: true

\* util.isError({ name: 'Error', message: 'an error occurred' });

\* // Returns: false

\* ```

\*

\* This method relies on `Object.prototype.toString()` behavior. It is

\* possible to obtain an incorrect result when the `object` argument manipulates`@@toStringTag`.

\*

\* ```js

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

\* const obj = { name: 'Error', message: 'an error occurred' };

\*

\* util.isError(obj);

\* // Returns: false

\* obj[Symbol.toStringTag] = 'Error';

\* util.isError(obj);

\* // Returns: true

\* ```

\* @since v0.6.0

\* @deprecated Since v4.0.0 - Use {@link types.isNativeError} instead.

\*/

export function isError(object: unknown): object is Error;

/\*\*

\* Usage of `util.inherits()` is discouraged. Please use the ES6 `class` and`extends` keywords to get language level inheritance support. Also note

\* that the two styles are [semantically incompatible](https://github.com/nodejs/node/issues/4179).

\*

\* Inherit the prototype methods from one [constructor](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Object/constructor) into another. The

\* prototype of `constructor` will be set to a new object created from`superConstructor`.

\*

\* This mainly adds some input validation on top of`Object.setPrototypeOf(constructor.prototype, superConstructor.prototype)`.

\* As an additional convenience, `superConstructor` will be accessible

\* through the `constructor.super\_` property.

\*

\* ```js

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

\* const EventEmitter = require('events');

\*

\* function MyStream() {

\* EventEmitter.call(this);

\* }

\*

\* util.inherits(MyStream, EventEmitter);

\*

\* MyStream.prototype.write = function(data) {

\* this.emit('data', data);

\* };

\*

\* const stream = new MyStream();

\*

\* console.log(stream instanceof EventEmitter); // true

\* console.log(MyStream.super\_ === EventEmitter); // true

\*

\* stream.on('data', (data) => {

\* console.log(`Received data: "${data}"`);

\* });

\* stream.write('It works!'); // Received data: "It works!"

\* ```

\*

\* ES6 example using `class` and `extends`:

\*

\* ```js

\* const EventEmitter = require('events');

\*

\* class MyStream extends EventEmitter {

\* write(data) {

\* this.emit('data', data);

\* }

\* }

\*

\* const stream = new MyStream();

\*

\* stream.on('data', (data) => {

\* console.log(`Received data: "${data}"`);

\* });

\* stream.write('With ES6');

\* ```

\* @since v0.3.0

\* @deprecated Legacy: Use ES2015 class syntax and `extends` keyword instead.

\*/

export function inherits(constructor: unknown, superConstructor: unknown): void;

export type DebugLoggerFunction = (msg: string, ...param: unknown[]) => void;

export interface DebugLogger extends DebugLoggerFunction {

enabled: boolean;

}

/\*\*

\* The `util.debuglog()` method is used to create a function that conditionally

\* writes debug messages to `stderr` based on the existence of the `NODE\_DEBUG`environment variable. If the `section` name appears within the value of that

\* environment variable, then the returned function operates similar to `console.error()`. If not, then the returned function is a no-op.

\*

\* ```js

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

\* const debuglog = util.debuglog('foo');

\*

\* debuglog('hello from foo [%d]', 123);

\* ```

\*

\* If this program is run with `NODE\_DEBUG=foo` in the environment, then

\* it will output something like:

\*

\* ```console

\* FOO 3245: hello from foo [123]

\* ```

\*

\* where `3245` is the process id. If it is not run with that

\* environment variable set, then it will not print anything.

\*

\* The `section` supports wildcard also:

\*

\* ```js

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

\* const debuglog = util.debuglog('foo-bar');

\*

\* debuglog('hi there, it\'s foo-bar [%d]', 2333);

\* ```

\*

\* if it is run with `NODE\_DEBUG=foo\*` in the environment, then it will output

\* something like:

\*

\* ```console

\* FOO-BAR 3257: hi there, it's foo-bar [2333]

\* ```

\*

\* Multiple comma-separated `section` names may be specified in the `NODE\_DEBUG`environment variable: `NODE\_DEBUG=fs,net,tls`.

\*

\* The optional `callback` argument can be used to replace the logging function

\* with a different function that doesn't have any initialization or

\* unnecessary wrapping.

\*

\* ```js

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

\* let debuglog = util.debuglog('internals', (debug) => {

\* // Replace with a logging function that optimizes out

\* // testing if the section is enabled

\* debuglog = debug;

\* });

\* ```

\* @since v0.11.3

\* @param section A string identifying the portion of the application for which the `debuglog` function is being created.

\* @param callback A callback invoked the first time the logging function is called with a function argument that is a more optimized logging function.

\* @return The logging function

\*/

export function debuglog(section: string, callback?: (fn: DebugLoggerFunction) => void): DebugLogger;

export const debug: typeof debuglog;

/\*\*

\* Returns `true` if the given `object` is a `Boolean`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isBoolean(1);

\* // Returns: false

\* util.isBoolean(0);

\* // Returns: false

\* util.isBoolean(false);

\* // Returns: true

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `typeof value === 'boolean'` instead.

\*/

export function isBoolean(object: unknown): object is boolean;

/\*\*

\* Returns `true` if the given `object` is a `Buffer`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isBuffer({ length: 0 });

\* // Returns: false

\* util.isBuffer([]);

\* // Returns: false

\* util.isBuffer(Buffer.from('hello world'));

\* // Returns: true

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `isBuffer` instead.

\*/

export function isBuffer(object: unknown): object is Buffer;

/\*\*

\* Returns `true` if the given `object` is a `Function`. Otherwise, returns`false`.

\*

\* ```js

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

\*

\* function Foo() {}

\* const Bar = () => {};

\*

\* util.isFunction({});

\* // Returns: false

\* util.isFunction(Foo);

\* // Returns: true

\* util.isFunction(Bar);

\* // Returns: true

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `typeof value === 'function'` instead.

\*/

export function isFunction(object: unknown): boolean;

/\*\*

\* Returns `true` if the given `object` is strictly `null`. Otherwise, returns`false`.

\*

\* ```js

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

\*

\* util.isNull(0);

\* // Returns: false

\* util.isNull(undefined);

\* // Returns: false

\* util.isNull(null);

\* // Returns: true

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `value === null` instead.

\*/

export function isNull(object: unknown): object is null;

/\*\*

\* Returns `true` if the given `object` is `null` or `undefined`. Otherwise,

\* returns `false`.

\*

\* ```js

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

\*

\* util.isNullOrUndefined(0);

\* // Returns: false

\* util.isNullOrUndefined(undefined);

\* // Returns: true

\* util.isNullOrUndefined(null);

\* // Returns: true

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `value === undefined || value === null` instead.

\*/

export function isNullOrUndefined(object: unknown): object is null | undefined;

/\*\*

\* Returns `true` if the given `object` is a `Number`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isNumber(false);

\* // Returns: false

\* util.isNumber(Infinity);

\* // Returns: true

\* util.isNumber(0);

\* // Returns: true

\* util.isNumber(NaN);

\* // Returns: true

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `typeof value === 'number'` instead.

\*/

export function isNumber(object: unknown): object is number;

/\*\*

\* Returns `true` if the given `object` is strictly an `Object`\*\*and\*\* not a`Function` (even though functions are objects in JavaScript).

\* Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isObject(5);

\* // Returns: false

\* util.isObject(null);

\* // Returns: false

\* util.isObject({});

\* // Returns: true

\* util.isObject(() => {});

\* // Returns: false

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Deprecated: Use `value !== null && typeof value === 'object'` instead.

\*/

export function isObject(object: unknown): boolean;

/\*\*

\* Returns `true` if the given `object` is a primitive type. Otherwise, returns`false`.

\*

\* ```js

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

\*

\* util.isPrimitive(5);

\* // Returns: true

\* util.isPrimitive('foo');

\* // Returns: true

\* util.isPrimitive(false);

\* // Returns: true

\* util.isPrimitive(null);

\* // Returns: true

\* util.isPrimitive(undefined);

\* // Returns: true

\* util.isPrimitive({});

\* // Returns: false

\* util.isPrimitive(() => {});

\* // Returns: false

\* util.isPrimitive(/^$/);

\* // Returns: false

\* util.isPrimitive(new Date());

\* // Returns: false

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `(typeof value !== 'object' && typeof value !== 'function') || value === null` instead.

\*/

export function isPrimitive(object: unknown): boolean;

/\*\*

\* Returns `true` if the given `object` is a `string`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isString('');

\* // Returns: true

\* util.isString('foo');

\* // Returns: true

\* util.isString(String('foo'));

\* // Returns: true

\* util.isString(5);

\* // Returns: false

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `typeof value === 'string'` instead.

\*/

export function isString(object: unknown): object is string;

/\*\*

\* Returns `true` if the given `object` is a `Symbol`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* util.isSymbol(5);

\* // Returns: false

\* util.isSymbol('foo');

\* // Returns: false

\* util.isSymbol(Symbol('foo'));

\* // Returns: true

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `typeof value === 'symbol'` instead.

\*/

export function isSymbol(object: unknown): object is symbol;

/\*\*

\* Returns `true` if the given `object` is `undefined`. Otherwise, returns `false`.

\*

\* ```js

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

\*

\* const foo = undefined;

\* util.isUndefined(5);

\* // Returns: false

\* util.isUndefined(foo);

\* // Returns: true

\* util.isUndefined(null);

\* // Returns: false

\* ```

\* @since v0.11.5

\* @deprecated Since v4.0.0 - Use `value === undefined` instead.

\*/

export function isUndefined(object: unknown): object is undefined;

/\*\*

\* The `util.deprecate()` method wraps `fn` (which may be a function or class) in

\* such a way that it is marked as deprecated.

\*

\* ```js

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

\*

\* exports.obsoleteFunction = util.deprecate(() => {

\* // Do something here.

\* }, 'obsoleteFunction() is deprecated. Use newShinyFunction() instead.');

\* ```

\*

\* When called, `util.deprecate()` will return a function that will emit a`DeprecationWarning` using the `'warning'` event. The warning will

\* be emitted and printed to `stderr` the first time the returned function is

\* called. After the warning is emitted, the wrapped function is called without

\* emitting a warning.

\*

\* If the same optional `code` is supplied in multiple calls to `util.deprecate()`,

\* the warning will be emitted only once for that `code`.

\*

\* ```js

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

\*

\* const fn1 = util.deprecate(someFunction, someMessage, 'DEP0001');

\* const fn2 = util.deprecate(someOtherFunction, someOtherMessage, 'DEP0001');

\* fn1(); // Emits a deprecation warning with code DEP0001

\* fn2(); // Does not emit a deprecation warning because it has the same code

\* ```

\*

\* If either the `--no-deprecation` or `--no-warnings` command-line flags are

\* used, or if the `process.noDeprecation` property is set to `true`\_prior\_ to

\* the first deprecation warning, the `util.deprecate()` method does nothing.

\*

\* If the `--trace-deprecation` or `--trace-warnings` command-line flags are set,

\* or the `process.traceDeprecation` property is set to `true`, a warning and a

\* stack trace are printed to `stderr` the first time the deprecated function is

\* called.

\*

\* If the `--throw-deprecation` command-line flag is set, or the`process.throwDeprecation` property is set to `true`, then an exception will be

\* thrown when the deprecated function is called.

\*

\* The `--throw-deprecation` command-line flag and `process.throwDeprecation`property take precedence over `--trace-deprecation` and`process.traceDeprecation`.

\* @since v0.8.0

\* @param fn The function that is being deprecated.

\* @param msg A warning message to display when the deprecated function is invoked.

\* @param code A deprecation code. See the `list of deprecated APIs` for a list of codes.

\* @return The deprecated function wrapped to emit a warning.

\*/

export function deprecate<T extends Function>(fn: T, msg: string, code?: string): T;

/\*\*

\* Returns `true` if there is deep strict equality between `val1` and `val2`.

\* Otherwise, returns `false`.

\*

\* See `assert.deepStrictEqual()` for more information about deep strict

\* equality.

\* @since v9.0.0

\*/

export function isDeepStrictEqual(val1: unknown, val2: unknown): boolean;

/\*\*

\* Returns `str` with any ANSI escape codes removed.

\*

\* ```js

\* console.log(util.stripVTControlCharacters('\u001B[4mvalue\u001B[0m'));

\* // Prints "value"

\* ```

\* @since v16.11.0

\*/

export function stripVTControlCharacters(str: string): string;

/\*\*

\* Takes an `async` function (or a function that returns a `Promise`) and returns a

\* function following the error-first callback style, i.e. taking

\* an `(err, value) => ...` callback as the last argument. In the callback, the

\* first argument will be the rejection reason (or `null` if the `Promise`resolved), and the second argument will be the resolved value.

\*

\* ```js

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

\*

\* async function fn() {

\* return 'hello world';

\* }

\* const callbackFunction = util.callbackify(fn);

\*

\* callbackFunction((err, ret) => {

\* if (err) throw err;

\* console.log(ret);

\* });

\* ```

\*

\* Will print:

\*

\* ```text

\* hello world

\* ```

\*

\* The callback is executed asynchronously, and will have a limited stack trace.

\* If the callback throws, the process will emit an `'uncaughtException'` event, and if not handled will exit.

\*

\* Since `null` has a special meaning as the first argument to a callback, if a

\* wrapped function rejects a `Promise` with a falsy value as a reason, the value

\* is wrapped in an `Error` with the original value stored in a field named`reason`.

\*

\* ```js

\* function fn() {

\* return Promise.reject(null);

\* }

\* const callbackFunction = util.callbackify(fn);

\*

\* callbackFunction((err, ret) => {

\* // When the Promise was rejected with `null` it is wrapped with an Error and

\* // the original value is stored in `reason`.

\* err &#x26;&#x26; err.hasOwnProperty('reason') &#x26;&#x26; err.reason === null; // true

\* });

\* ```

\* @since v8.2.0

\* @param original An `async` function

\* @return a callback style function

\*/

export function callbackify(fn: () => Promise<void>): (callback: (err: NodeJS.ErrnoException) => void) => void;

export function callbackify<TResult>(fn: () => Promise<TResult>): (callback: (err: NodeJS.ErrnoException, result: TResult) => void) => void;

export function callbackify<T1>(fn: (arg1: T1) => Promise<void>): (arg1: T1, callback: (err: NodeJS.ErrnoException) => void) => void;

export function callbackify<T1, TResult>(fn: (arg1: T1) => Promise<TResult>): (arg1: T1, callback: (err: NodeJS.ErrnoException, result: TResult) => void) => void;

export function callbackify<T1, T2>(fn: (arg1: T1, arg2: T2) => Promise<void>): (arg1: T1, arg2: T2, callback: (err: NodeJS.ErrnoException) => void) => void;

export function callbackify<T1, T2, TResult>(fn: (arg1: T1, arg2: T2) => Promise<TResult>): (arg1: T1, arg2: T2, callback: (err: NodeJS.ErrnoException | null, result: TResult) => void) => void;

export function callbackify<T1, T2, T3>(fn: (arg1: T1, arg2: T2, arg3: T3) => Promise<void>): (arg1: T1, arg2: T2, arg3: T3, callback: (err: NodeJS.ErrnoException) => void) => void;

export function callbackify<T1, T2, T3, TResult>(

fn: (arg1: T1, arg2: T2, arg3: T3) => Promise<TResult>

): (arg1: T1, arg2: T2, arg3: T3, callback: (err: NodeJS.ErrnoException | null, result: TResult) => void) => void;

export function callbackify<T1, T2, T3, T4>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4) => Promise<void>

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4, callback: (err: NodeJS.ErrnoException) => void) => void;

export function callbackify<T1, T2, T3, T4, TResult>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4) => Promise<TResult>

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4, callback: (err: NodeJS.ErrnoException | null, result: TResult) => void) => void;

export function callbackify<T1, T2, T3, T4, T5>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5) => Promise<void>

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5, callback: (err: NodeJS.ErrnoException) => void) => void;

export function callbackify<T1, T2, T3, T4, T5, TResult>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5) => Promise<TResult>

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5, callback: (err: NodeJS.ErrnoException | null, result: TResult) => void) => void;

export function callbackify<T1, T2, T3, T4, T5, T6>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5, arg6: T6) => Promise<void>

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5, arg6: T6, callback: (err: NodeJS.ErrnoException) => void) => void;

export function callbackify<T1, T2, T3, T4, T5, T6, TResult>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5, arg6: T6) => Promise<TResult>

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5, arg6: T6, callback: (err: NodeJS.ErrnoException | null, result: TResult) => void) => void;

export interface CustomPromisifyLegacy<TCustom extends Function> extends Function {

\_\_promisify\_\_: TCustom;

}

export interface CustomPromisifySymbol<TCustom extends Function> extends Function {

[promisify.custom]: TCustom;

}

export type CustomPromisify<TCustom extends Function> = CustomPromisifySymbol<TCustom> | CustomPromisifyLegacy<TCustom>;

/\*\*

\* Takes a function following the common error-first callback style, i.e. taking

\* an `(err, value) => ...` callback as the last argument, and returns a version

\* that returns promises.

\*

\* ```js

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

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

\*

\* const stat = util.promisify(fs.stat);

\* stat('.').then((stats) => {

\* // Do something with `stats`

\* }).catch((error) => {

\* // Handle the error.

\* });

\* ```

\*

\* Or, equivalently using `async function`s:

\*

\* ```js

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

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

\*

\* const stat = util.promisify(fs.stat);

\*

\* async function callStat() {

\* const stats = await stat('.');

\* console.log(`This directory is owned by ${stats.uid}`);

\* }

\* ```

\*

\* If there is an `original[util.promisify.custom]` property present, `promisify`will return its value, see `Custom promisified functions`.

\*

\* `promisify()` assumes that `original` is a function taking a callback as its

\* final argument in all cases. If `original` is not a function, `promisify()`will throw an error. If `original` is a function but its last argument is not

\* an error-first callback, it will still be passed an error-first

\* callback as its last argument.

\*

\* Using `promisify()` on class methods or other methods that use `this` may not

\* work as expected unless handled specially:

\*

\* ```js

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

\*

\* class Foo {

\* constructor() {

\* this.a = 42;

\* }

\*

\* bar(callback) {

\* callback(null, this.a);

\* }

\* }

\*

\* const foo = new Foo();

\*

\* const naiveBar = util.promisify(foo.bar);

\* // TypeError: Cannot read property 'a' of undefined

\* // naiveBar().then(a => console.log(a));

\*

\* naiveBar.call(foo).then((a) => console.log(a)); // '42'

\*

\* const bindBar = naiveBar.bind(foo);

\* bindBar().then((a) => console.log(a)); // '42'

\* ```

\* @since v8.0.0

\*/

export function promisify<TCustom extends Function>(fn: CustomPromisify<TCustom>): TCustom;

export function promisify<TResult>(fn: (callback: (err: any, result: TResult) => void) => void): () => Promise<TResult>;

export function promisify(fn: (callback: (err?: any) => void) => void): () => Promise<void>;

export function promisify<T1, TResult>(fn: (arg1: T1, callback: (err: any, result: TResult) => void) => void): (arg1: T1) => Promise<TResult>;

export function promisify<T1>(fn: (arg1: T1, callback: (err?: any) => void) => void): (arg1: T1) => Promise<void>;

export function promisify<T1, T2, TResult>(fn: (arg1: T1, arg2: T2, callback: (err: any, result: TResult) => void) => void): (arg1: T1, arg2: T2) => Promise<TResult>;

export function promisify<T1, T2>(fn: (arg1: T1, arg2: T2, callback: (err?: any) => void) => void): (arg1: T1, arg2: T2) => Promise<void>;

export function promisify<T1, T2, T3, TResult>(fn: (arg1: T1, arg2: T2, arg3: T3, callback: (err: any, result: TResult) => void) => void): (arg1: T1, arg2: T2, arg3: T3) => Promise<TResult>;

export function promisify<T1, T2, T3>(fn: (arg1: T1, arg2: T2, arg3: T3, callback: (err?: any) => void) => void): (arg1: T1, arg2: T2, arg3: T3) => Promise<void>;

export function promisify<T1, T2, T3, T4, TResult>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4, callback: (err: any, result: TResult) => void) => void

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4) => Promise<TResult>;

export function promisify<T1, T2, T3, T4>(fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4, callback: (err?: any) => void) => void): (arg1: T1, arg2: T2, arg3: T3, arg4: T4) => Promise<void>;

export function promisify<T1, T2, T3, T4, T5, TResult>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5, callback: (err: any, result: TResult) => void) => void

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5) => Promise<TResult>;

export function promisify<T1, T2, T3, T4, T5>(

fn: (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5, callback: (err?: any) => void) => void

): (arg1: T1, arg2: T2, arg3: T3, arg4: T4, arg5: T5) => Promise<void>;

export function promisify(fn: Function): Function;

export namespace promisify {

/\*\*

\* That can be used to declare custom promisified variants of functions.

\*/

const custom: unique symbol;

}

/\*\*

\* An implementation of the [WHATWG Encoding Standard](https://encoding.spec.whatwg.org/) `TextDecoder` API.

\*

\* ```js

\* const decoder = new TextDecoder('shift\_jis');

\* let string = '';

\* let buffer;

\* while (buffer = getNextChunkSomehow()) {

\* string += decoder.decode(buffer, { stream: true });

\* }

\* string += decoder.decode(); // end-of-stream

\* ```

\* @since v8.3.0

\*/

export class TextDecoder {

/\*\*

\* The encoding supported by the `TextDecoder` instance.

\*/

readonly encoding: string;

/\*\*

\* The value will be `true` if decoding errors result in a `TypeError` being

\* thrown.

\*/

readonly fatal: boolean;

/\*\*

\* The value will be `true` if the decoding result will include the byte order

\* mark.

\*/

readonly ignoreBOM: boolean;

constructor(

encoding?: string,

options?: {

fatal?: boolean | undefined;

ignoreBOM?: boolean | undefined;

}

);

/\*\*

\* Decodes the `input` and returns a string. If `options.stream` is `true`, any

\* incomplete byte sequences occurring at the end of the `input` are buffered

\* internally and emitted after the next call to `textDecoder.decode()`.

\*

\* If `textDecoder.fatal` is `true`, decoding errors that occur will result in a`TypeError` being thrown.

\* @param input An `ArrayBuffer`, `DataView` or `TypedArray` instance containing the encoded data.

\*/

decode(

input?: NodeJS.ArrayBufferView | ArrayBuffer | null,

options?: {

stream?: boolean | undefined;

}

): string;

}

export interface EncodeIntoResult {

/\*\*

\* The read Unicode code units of input.

\*/

read: number;

/\*\*

\* The written UTF-8 bytes of output.

\*/

written: number;

}

export { types };

/\*\*

\* An implementation of the [WHATWG Encoding Standard](https://encoding.spec.whatwg.org/) `TextEncoder` API. All

\* instances of `TextEncoder` only support UTF-8 encoding.

\*

\* ```js

\* const encoder = new TextEncoder();

\* const uint8array = encoder.encode('this is some data');

\* ```

\*

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

\* @since v8.3.0

\*/

export class TextEncoder {

/\*\*

\* The encoding supported by the `TextEncoder` instance. Always set to `'utf-8'`.

\*/

readonly encoding: string;

/\*\*

\* UTF-8 encodes the `input` string and returns a `Uint8Array` containing the

\* encoded bytes.

\* @param [input='an empty string'] The text to encode.

\*/

encode(input?: string): Uint8Array;

/\*\*

\* UTF-8 encodes the `src` string to the `dest` Uint8Array and returns an object

\* containing the read Unicode code units and written UTF-8 bytes.

\*

\* ```js

\* const encoder = new TextEncoder();

\* const src = 'this is some data';

\* const dest = new Uint8Array(10);

\* const { read, written } = encoder.encodeInto(src, dest);

\* ```

\* @param src The text to encode.

\* @param dest The array to hold the encode result.

\*/

encodeInto(src: string, dest: Uint8Array): EncodeIntoResult;

}

}

declare module 'util/types' {

export \* from 'util/types';

}

declare module 'util/types' {

import { KeyObject, webcrypto } from 'node:crypto';

/\*\*

\* Returns `true` if the value is a built-in [`ArrayBuffer`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer) or

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

\*

\* See also `util.types.isArrayBuffer()` and `util.types.isSharedArrayBuffer()`.

\*

\* ```js

\* util.types.isAnyArrayBuffer(new ArrayBuffer()); // Returns true

\* util.types.isAnyArrayBuffer(new SharedArrayBuffer()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isAnyArrayBuffer(object: unknown): object is ArrayBufferLike;

/\*\*

\* Returns `true` if the value is an `arguments` object.

\*

\* ```js

\* function foo() {

\* util.types.isArgumentsObject(arguments); // Returns true

\* }

\* ```

\* @since v10.0.0

\*/

function isArgumentsObject(object: unknown): object is IArguments;

/\*\*

\* Returns `true` if the value is a built-in [`ArrayBuffer`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer) instance.

\* This does \_not\_ include [`SharedArrayBuffer`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/SharedArrayBuffer) instances. Usually, it is

\* desirable to test for both; See `util.types.isAnyArrayBuffer()` for that.

\*

\* ```js

\* util.types.isArrayBuffer(new ArrayBuffer()); // Returns true

\* util.types.isArrayBuffer(new SharedArrayBuffer()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isArrayBuffer(object: unknown): object is ArrayBuffer;

/\*\*

\* Returns `true` if the value is an instance of one of the [`ArrayBuffer`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer) views, such as typed

\* array objects or [`DataView`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/DataView). Equivalent to

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

\*

\* ```js

\* util.types.isArrayBufferView(new Int8Array()); // true

\* util.types.isArrayBufferView(Buffer.from('hello world')); // true

\* util.types.isArrayBufferView(new DataView(new ArrayBuffer(16))); // true

\* util.types.isArrayBufferView(new ArrayBuffer()); // false

\* ```

\* @since v10.0.0

\*/

function isArrayBufferView(object: unknown): object is NodeJS.ArrayBufferView;

/\*\*

\* Returns `true` if the value is an [async function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async\_function).

\* This only reports back what the JavaScript engine is seeing;

\* in particular, the return value may not match the original source code if

\* a transpilation tool was used.

\*

\* ```js

\* util.types.isAsyncFunction(function foo() {}); // Returns false

\* util.types.isAsyncFunction(async function foo() {}); // Returns true

\* ```

\* @since v10.0.0

\*/

function isAsyncFunction(object: unknown): boolean;

/\*\*

\* Returns `true` if the value is a `BigInt64Array` instance.

\*

\* ```js

\* util.types.isBigInt64Array(new BigInt64Array()); // Returns true

\* util.types.isBigInt64Array(new BigUint64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isBigInt64Array(value: unknown): value is BigInt64Array;

/\*\*

\* Returns `true` if the value is a `BigUint64Array` instance.

\*

\* ```js

\* util.types.isBigUint64Array(new BigInt64Array()); // Returns false

\* util.types.isBigUint64Array(new BigUint64Array()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isBigUint64Array(value: unknown): value is BigUint64Array;

/\*\*

\* Returns `true` if the value is a boolean object, e.g. created

\* by `new Boolean()`.

\*

\* ```js

\* util.types.isBooleanObject(false); // Returns false

\* util.types.isBooleanObject(true); // Returns false

\* util.types.isBooleanObject(new Boolean(false)); // Returns true

\* util.types.isBooleanObject(new Boolean(true)); // Returns true

\* util.types.isBooleanObject(Boolean(false)); // Returns false

\* util.types.isBooleanObject(Boolean(true)); // Returns false

\* ```

\* @since v10.0.0

\*/

function isBooleanObject(object: unknown): object is Boolean;

/\*\*

\* Returns `true` if the value is any boxed primitive object, e.g. created

\* by `new Boolean()`, `new String()` or `Object(Symbol())`.

\*

\* For example:

\*

\* ```js

\* util.types.isBoxedPrimitive(false); // Returns false

\* util.types.isBoxedPrimitive(new Boolean(false)); // Returns true

\* util.types.isBoxedPrimitive(Symbol('foo')); // Returns false

\* util.types.isBoxedPrimitive(Object(Symbol('foo'))); // Returns true

\* util.types.isBoxedPrimitive(Object(BigInt(5))); // Returns true

\* ```

\* @since v10.11.0

\*/

function isBoxedPrimitive(object: unknown): object is String | Number | BigInt | Boolean | Symbol;

/\*\*

\* Returns `true` if the value is a built-in [`DataView`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/DataView) instance.

\*

\* ```js

\* const ab = new ArrayBuffer(20);

\* util.types.isDataView(new DataView(ab)); // Returns true

\* util.types.isDataView(new Float64Array()); // Returns false

\* ```

\*

\* See also [`ArrayBuffer.isView()`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer/isView).

\* @since v10.0.0

\*/

function isDataView(object: unknown): object is DataView;

/\*\*

\* Returns `true` if the value is a built-in [`Date`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Date) instance.

\*

\* ```js

\* util.types.isDate(new Date()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isDate(object: unknown): object is Date;

/\*\*

\* Returns `true` if the value is a native `External` value.

\*

\* A native `External` value is a special type of object that contains a

\* raw C++ pointer (`void\*`) for access from native code, and has no other

\* properties. Such objects are created either by Node.js internals or native

\* addons. In JavaScript, they are [frozen](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Object/freeze) objects with a`null` prototype.

\*

\* ```c

\* #include <js\_native\_api.h>

\* #include <stdlib.h>

\* napi\_value result;

\* static napi\_value MyNapi(napi\_env env, napi\_callback\_info info) {

\* int\* raw = (int\*) malloc(1024);

\* napi\_status status = napi\_create\_external(env, (void\*) raw, NULL, NULL, &#x26;result);

\* if (status != napi\_ok) {

\* napi\_throw\_error(env, NULL, "napi\_create\_external failed");

\* return NULL;

\* }

\* return result;

\* }

\* ...

\* DECLARE\_NAPI\_PROPERTY("myNapi", MyNapi)

\* ...

\* ```

\*

\* ```js

\* const native = require('napi\_addon.node');

\* const data = native.myNapi();

\* util.types.isExternal(data); // returns true

\* util.types.isExternal(0); // returns false

\* util.types.isExternal(new String('foo')); // returns false

\* ```

\*

\* For further information on `napi\_create\_external`, refer to `napi\_create\_external()`.

\* @since v10.0.0

\*/

function isExternal(object: unknown): boolean;

/\*\*

\* Returns `true` if the value is a built-in [`Float32Array`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Float32Array) instance.

\*

\* ```js

\* util.types.isFloat32Array(new ArrayBuffer()); // Returns false

\* util.types.isFloat32Array(new Float32Array()); // Returns true

\* util.types.isFloat32Array(new Float64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isFloat32Array(object: unknown): object is Float32Array;

/\*\*

\* Returns `true` if the value is a built-in [`Float64Array`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Float64Array) instance.

\*

\* ```js

\* util.types.isFloat64Array(new ArrayBuffer()); // Returns false

\* util.types.isFloat64Array(new Uint8Array()); // Returns false

\* util.types.isFloat64Array(new Float64Array()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isFloat64Array(object: unknown): object is Float64Array;

/\*\*

\* Returns `true` if the value is a generator function.

\* This only reports back what the JavaScript engine is seeing;

\* in particular, the return value may not match the original source code if

\* a transpilation tool was used.

\*

\* ```js

\* util.types.isGeneratorFunction(function foo() {}); // Returns false

\* util.types.isGeneratorFunction(function\* foo() {}); // Returns true

\* ```

\* @since v10.0.0

\*/

function isGeneratorFunction(object: unknown): object is GeneratorFunction;

/\*\*

\* Returns `true` if the value is a generator object as returned from a

\* built-in generator function.

\* This only reports back what the JavaScript engine is seeing;

\* in particular, the return value may not match the original source code if

\* a transpilation tool was used.

\*

\* ```js

\* function\* foo() {}

\* const generator = foo();

\* util.types.isGeneratorObject(generator); // Returns true

\* ```

\* @since v10.0.0

\*/

function isGeneratorObject(object: unknown): object is Generator;

/\*\*

\* Returns `true` if the value is a built-in [`Int8Array`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Int8Array) instance.

\*

\* ```js

\* util.types.isInt8Array(new ArrayBuffer()); // Returns false

\* util.types.isInt8Array(new Int8Array()); // Returns true

\* util.types.isInt8Array(new Float64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isInt8Array(object: unknown): object is Int8Array;

/\*\*

\* Returns `true` if the value is a built-in [`Int16Array`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Int16Array) instance.

\*

\* ```js

\* util.types.isInt16Array(new ArrayBuffer()); // Returns false

\* util.types.isInt16Array(new Int16Array()); // Returns true

\* util.types.isInt16Array(new Float64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isInt16Array(object: unknown): object is Int16Array;

/\*\*

\* Returns `true` if the value is a built-in [`Int32Array`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Int32Array) instance.

\*

\* ```js

\* util.types.isInt32Array(new ArrayBuffer()); // Returns false

\* util.types.isInt32Array(new Int32Array()); // Returns true

\* util.types.isInt32Array(new Float64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isInt32Array(object: unknown): object is Int32Array;

/\*\*

\* Returns `true` if the value is a built-in [`Map`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map) instance.

\*

\* ```js

\* util.types.isMap(new Map()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isMap<T>(object: T | {}): object is T extends ReadonlyMap<any, any> ? (unknown extends T ? never : ReadonlyMap<any, any>) : Map<unknown, unknown>;

/\*\*

\* Returns `true` if the value is an iterator returned for a built-in [`Map`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Map) instance.

\*

\* ```js

\* const map = new Map();

\* util.types.isMapIterator(map.keys()); // Returns true

\* util.types.isMapIterator(map.values()); // Returns true

\* util.types.isMapIterator(map.entries()); // Returns true

\* util.types.isMapIterator(map[Symbol.iterator]()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isMapIterator(object: unknown): boolean;

/\*\*

\* Returns `true` if the value is an instance of a [Module Namespace Object](https://tc39.github.io/ecma262/#sec-module-namespace-exotic-objects).

\*

\* ```js

\* import \* as ns from './a.js';

\*

\* util.types.isModuleNamespaceObject(ns); // Returns true

\* ```

\* @since v10.0.0

\*/

function isModuleNamespaceObject(value: unknown): boolean;

/\*\*

\* Returns `true` if the value is an instance of a built-in `Error` type.

\*

\* ```js

\* util.types.isNativeError(new Error()); // Returns true

\* util.types.isNativeError(new TypeError()); // Returns true

\* util.types.isNativeError(new RangeError()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isNativeError(object: unknown): object is Error;

/\*\*

\* Returns `true` if the value is a number object, e.g. created

\* by `new Number()`.

\*

\* ```js

\* util.types.isNumberObject(0); // Returns false

\* util.types.isNumberObject(new Number(0)); // Returns true

\* ```

\* @since v10.0.0

\*/

function isNumberObject(object: unknown): object is Number;

/\*\*

\* Returns `true` if the value is a built-in [`Promise`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Promise).

\*

\* ```js

\* util.types.isPromise(Promise.resolve(42)); // Returns true

\* ```

\* @since v10.0.0

\*/

function isPromise(object: unknown): object is Promise<unknown>;

/\*\*

\* Returns `true` if the value is a [`Proxy`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Proxy) instance.

\*

\* ```js

\* const target = {};

\* const proxy = new Proxy(target, {});

\* util.types.isProxy(target); // Returns false

\* util.types.isProxy(proxy); // Returns true

\* ```

\* @since v10.0.0

\*/

function isProxy(object: unknown): boolean;

/\*\*

\* Returns `true` if the value is a regular expression object.

\*

\* ```js

\* util.types.isRegExp(/abc/); // Returns true

\* util.types.isRegExp(new RegExp('abc')); // Returns true

\* ```

\* @since v10.0.0

\*/

function isRegExp(object: unknown): object is RegExp;

/\*\*

\* Returns `true` if the value is a built-in [`Set`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Set) instance.

\*

\* ```js

\* util.types.isSet(new Set()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isSet<T>(object: T | {}): object is T extends ReadonlySet<any> ? (unknown extends T ? never : ReadonlySet<any>) : Set<unknown>;

/\*\*

\* Returns `true` if the value is an iterator returned for a built-in [`Set`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Set) instance.

\*

\* ```js

\* const set = new Set();

\* util.types.isSetIterator(set.keys()); // Returns true

\* util.types.isSetIterator(set.values()); // Returns true

\* util.types.isSetIterator(set.entries()); // Returns true

\* util.types.isSetIterator(set[Symbol.iterator]()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isSetIterator(object: unknown): boolean;

/\*\*

\* Returns `true` if the value is a built-in [`SharedArrayBuffer`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/SharedArrayBuffer) instance.

\* This does \_not\_ include [`ArrayBuffer`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer) instances. Usually, it is

\* desirable to test for both; See `util.types.isAnyArrayBuffer()` for that.

\*

\* ```js

\* util.types.isSharedArrayBuffer(new ArrayBuffer()); // Returns false

\* util.types.isSharedArrayBuffer(new SharedArrayBuffer()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isSharedArrayBuffer(object: unknown): object is SharedArrayBuffer;

/\*\*

\* Returns `true` if the value is a string object, e.g. created

\* by `new String()`.

\*

\* ```js

\* util.types.isStringObject('foo'); // Returns false

\* util.types.isStringObject(new String('foo')); // Returns true

\* ```

\* @since v10.0.0

\*/

function isStringObject(object: unknown): object is String;

/\*\*

\* Returns `true` if the value is a symbol object, created

\* by calling `Object()` on a `Symbol` primitive.

\*

\* ```js

\* const symbol = Symbol('foo');

\* util.types.isSymbolObject(symbol); // Returns false

\* util.types.isSymbolObject(Object(symbol)); // Returns true

\* ```

\* @since v10.0.0

\*/

function isSymbolObject(object: unknown): object is Symbol;

/\*\*

\* Returns `true` if the value is a built-in [`TypedArray`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/TypedArray) instance.

\*

\* ```js

\* util.types.isTypedArray(new ArrayBuffer()); // Returns false

\* util.types.isTypedArray(new Uint8Array()); // Returns true

\* util.types.isTypedArray(new Float64Array()); // Returns true

\* ```

\*

\* See also [`ArrayBuffer.isView()`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer/isView).

\* @since v10.0.0

\*/

function isTypedArray(object: unknown): object is NodeJS.TypedArray;

/\*\*

\* Returns `true` if the value is a built-in [`Uint8Array`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Uint8Array) instance.

\*

\* ```js

\* util.types.isUint8Array(new ArrayBuffer()); // Returns false

\* util.types.isUint8Array(new Uint8Array()); // Returns true

\* util.types.isUint8Array(new Float64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isUint8Array(object: unknown): object is Uint8Array;

/\*\*

\* Returns `true` if the value is a built-in [`Uint8ClampedArray`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Uint8ClampedArray) instance.

\*

\* ```js

\* util.types.isUint8ClampedArray(new ArrayBuffer()); // Returns false

\* util.types.isUint8ClampedArray(new Uint8ClampedArray()); // Returns true

\* util.types.isUint8ClampedArray(new Float64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isUint8ClampedArray(object: unknown): object is Uint8ClampedArray;

/\*\*

\* Returns `true` if the value is a built-in [`Uint16Array`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Uint16Array) instance.

\*

\* ```js

\* util.types.isUint16Array(new ArrayBuffer()); // Returns false

\* util.types.isUint16Array(new Uint16Array()); // Returns true

\* util.types.isUint16Array(new Float64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isUint16Array(object: unknown): object is Uint16Array;

/\*\*

\* Returns `true` if the value is a built-in [`Uint32Array`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Uint32Array) instance.

\*

\* ```js

\* util.types.isUint32Array(new ArrayBuffer()); // Returns false

\* util.types.isUint32Array(new Uint32Array()); // Returns true

\* util.types.isUint32Array(new Float64Array()); // Returns false

\* ```

\* @since v10.0.0

\*/

function isUint32Array(object: unknown): object is Uint32Array;

/\*\*

\* Returns `true` if the value is a built-in [`WeakMap`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/WeakMap) instance.

\*

\* ```js

\* util.types.isWeakMap(new WeakMap()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isWeakMap(object: unknown): object is WeakMap<object, unknown>;

/\*\*

\* Returns `true` if the value is a built-in [`WeakSet`](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/WeakSet) instance.

\*

\* ```js

\* util.types.isWeakSet(new WeakSet()); // Returns true

\* ```

\* @since v10.0.0

\*/

function isWeakSet(object: unknown): object is WeakSet<object>;

/\*\*

\* Returns `true` if `value` is a `KeyObject`, `false` otherwise.

\* @since v16.2.0

\*/

function isKeyObject(object: unknown): object is KeyObject;

/\*\*

\* Returns `true` if `value` is a `CryptoKey`, `false` otherwise.

\* @since v16.2.0

\*/

function isCryptoKey(object: unknown): object is webcrypto.CryptoKey;

}

declare module 'node:util' {

export \* from 'util';

}

declare module 'node:util/types' {

export \* from 'util/types';

}