/\*\*

\* The `tty` module provides the `tty.ReadStream` and `tty.WriteStream` classes.

\* In most cases, it will not be necessary or possible to use this module directly.

\* However, it can be accessed using:

\*

\* ```js

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

\* ```

\*

\* When Node.js detects that it is being run with a text terminal ("TTY")

\* attached, `process.stdin` will, by default, be initialized as an instance of`tty.ReadStream` and both `process.stdout` and `process.stderr` will, by

\* default, be instances of `tty.WriteStream`. The preferred method of determining

\* whether Node.js is being run within a TTY context is to check that the value of

\* the `process.stdout.isTTY` property is `true`:

\*

\* ```console

\* $ node -p -e "Boolean(process.stdout.isTTY)"

\* true

\* $ node -p -e "Boolean(process.stdout.isTTY)" | cat

\* false

\* ```

\*

\* In most cases, there should be little to no reason for an application to

\* manually create instances of the `tty.ReadStream` and `tty.WriteStream`classes.

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

\*/

declare module 'tty' {

import \* as net from 'node:net';

/\*\*

\* The `tty.isatty()` method returns `true` if the given `fd` is associated with

\* a TTY and `false` if it is not, including whenever `fd` is not a non-negative

\* integer.

\* @since v0.5.8

\* @param fd A numeric file descriptor

\*/

function isatty(fd: number): boolean;

/\*\*

\* Represents the readable side of a TTY. In normal circumstances `process.stdin` will be the only `tty.ReadStream` instance in a Node.js

\* process and there should be no reason to create additional instances.

\* @since v0.5.8

\*/

class ReadStream extends net.Socket {

constructor(fd: number, options?: net.SocketConstructorOpts);

/\*\*

\* A `boolean` that is `true` if the TTY is currently configured to operate as a

\* raw device. Defaults to `false`.

\* @since v0.7.7

\*/

isRaw: boolean;

/\*\*

\* Allows configuration of `tty.ReadStream` so that it operates as a raw device.

\*

\* When in raw mode, input is always available character-by-character, not

\* including modifiers. Additionally, all special processing of characters by the

\* terminal is disabled, including echoing input characters.Ctrl+C will no longer cause a `SIGINT` when in this mode.

\* @since v0.7.7

\* @param mode If `true`, configures the `tty.ReadStream` to operate as a raw device. If `false`, configures the `tty.ReadStream` to operate in its default mode. The `readStream.isRaw`

\* property will be set to the resulting mode.

\* @return The read stream instance.

\*/

setRawMode(mode: boolean): this;

/\*\*

\* A `boolean` that is always `true` for `tty.ReadStream` instances.

\* @since v0.5.8

\*/

isTTY: boolean;

}

/\*\*

\* -1 - to the left from cursor

\* 0 - the entire line

\* 1 - to the right from cursor

\*/

type Direction = -1 | 0 | 1;

/\*\*

\* Represents the writable side of a TTY. In normal circumstances,`process.stdout` and `process.stderr` will be the only`tty.WriteStream` instances created for a Node.js process and there

\* should be no reason to create additional instances.

\* @since v0.5.8

\*/

class WriteStream extends net.Socket {

constructor(fd: number);

addListener(event: string, listener: (...args: any[]) => void): this;

addListener(event: 'resize', listener: () => void): this;

emit(event: string | symbol, ...args: any[]): boolean;

emit(event: 'resize'): boolean;

on(event: string, listener: (...args: any[]) => void): this;

on(event: 'resize', listener: () => void): this;

once(event: string, listener: (...args: any[]) => void): this;

once(event: 'resize', listener: () => void): this;

prependListener(event: string, listener: (...args: any[]) => void): this;

prependListener(event: 'resize', listener: () => void): this;

prependOnceListener(event: string, listener: (...args: any[]) => void): this;

prependOnceListener(event: 'resize', listener: () => void): this;

/\*\*

\* `writeStream.clearLine()` clears the current line of this `WriteStream` in a

\* direction identified by `dir`.

\* @since v0.7.7

\* @param callback Invoked once the operation completes.

\* @return `false` if the stream wishes for the calling code to wait for the `'drain'` event to be emitted before continuing to write additional data; otherwise `true`.

\*/

clearLine(dir: Direction, callback?: () => void): boolean;

/\*\*

\* `writeStream.clearScreenDown()` clears this `WriteStream` from the current

\* cursor down.

\* @since v0.7.7

\* @param callback Invoked once the operation completes.

\* @return `false` if the stream wishes for the calling code to wait for the `'drain'` event to be emitted before continuing to write additional data; otherwise `true`.

\*/

clearScreenDown(callback?: () => void): boolean;

/\*\*

\* `writeStream.cursorTo()` moves this `WriteStream`'s cursor to the specified

\* position.

\* @since v0.7.7

\* @param callback Invoked once the operation completes.

\* @return `false` if the stream wishes for the calling code to wait for the `'drain'` event to be emitted before continuing to write additional data; otherwise `true`.

\*/

cursorTo(x: number, y?: number, callback?: () => void): boolean;

cursorTo(x: number, callback: () => void): boolean;

/\*\*

\* `writeStream.moveCursor()` moves this `WriteStream`'s cursor \_relative\_ to its

\* current position.

\* @since v0.7.7

\* @param callback Invoked once the operation completes.

\* @return `false` if the stream wishes for the calling code to wait for the `'drain'` event to be emitted before continuing to write additional data; otherwise `true`.

\*/

moveCursor(dx: number, dy: number, callback?: () => void): boolean;

/\*\*

\* Returns:

\*

\* \* `1` for 2,

\* \* `4` for 16,

\* \* `8` for 256,

\* \* `24` for 16,777,216 colors supported.

\*

\* Use this to determine what colors the terminal supports. Due to the nature of

\* colors in terminals it is possible to either have false positives or false

\* negatives. It depends on process information and the environment variables that

\* may lie about what terminal is used.

\* It is possible to pass in an `env` object to simulate the usage of a specific

\* terminal. This can be useful to check how specific environment settings behave.

\*

\* To enforce a specific color support, use one of the below environment settings.

\*

\* \* 2 colors: `FORCE\_COLOR = 0` (Disables colors)

\* \* 16 colors: `FORCE\_COLOR = 1`

\* \* 256 colors: `FORCE\_COLOR = 2`

\* \* 16,777,216 colors: `FORCE\_COLOR = 3`

\*

\* Disabling color support is also possible by using the `NO\_COLOR` and`NODE\_DISABLE\_COLORS` environment variables.

\* @since v9.9.0

\* @param [env=process.env] An object containing the environment variables to check. This enables simulating the usage of a specific terminal.

\*/

getColorDepth(env?: object): number;

/\*\*

\* Returns `true` if the `writeStream` supports at least as many colors as provided

\* in `count`. Minimum support is 2 (black and white).

\*

\* This has the same false positives and negatives as described in `writeStream.getColorDepth()`.

\*

\* ```js

\* process.stdout.hasColors();

\* // Returns true or false depending on if `stdout` supports at least 16 colors.

\* process.stdout.hasColors(256);

\* // Returns true or false depending on if `stdout` supports at least 256 colors.

\* process.stdout.hasColors({ TMUX: '1' });

\* // Returns true.

\* process.stdout.hasColors(2 \*\* 24, { TMUX: '1' });

\* // Returns false (the environment setting pretends to support 2 \*\* 8 colors).

\* ```

\* @since v11.13.0, v10.16.0

\* @param [count=16] The number of colors that are requested (minimum 2).

\* @param [env=process.env] An object containing the environment variables to check. This enables simulating the usage of a specific terminal.

\*/

hasColors(count?: number): boolean;

hasColors(env?: object): boolean;

hasColors(count: number, env?: object): boolean;

/\*\*

\* `writeStream.getWindowSize()` returns the size of the TTY

\* corresponding to this `WriteStream`. The array is of the type`[numColumns, numRows]` where `numColumns` and `numRows` represent the number

\* of columns and rows in the corresponding TTY.

\* @since v0.7.7

\*/

getWindowSize(): [number, number];

/\*\*

\* A `number` specifying the number of columns the TTY currently has. This property

\* is updated whenever the `'resize'` event is emitted.

\* @since v0.7.7

\*/

columns: number;

/\*\*

\* A `number` specifying the number of rows the TTY currently has. This property

\* is updated whenever the `'resize'` event is emitted.

\* @since v0.7.7

\*/

rows: number;

/\*\*

\* A `boolean` that is always `true`.

\* @since v0.5.8

\*/

isTTY: boolean;

}

}

declare module 'node:tty' {

export \* from 'tty';

}