TTY

> *Stability: 2 - Stable*

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:

```
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`:

```
$ 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.

Class: `tty.ReadStream`

- Extends: {net.Socket}

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.

`readStream.isRaw`

A `boolean` that is `true` if the TTY is currently configured to operate as a raw device. Defaults to `false`.

`readStream.isTTY`

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

`readStream.setRawMode(mode)`

- `mode` {boolean} 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.
- Returns: {this} The read stream instance.

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.

Class: `tty.WriteStream`

- Extends: {net.Socket}

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.

Event: `'resize'`

The `'resize'` event is emitted whenever either of the `writeStream.columns` or `writeStream.rows` properties have changed. No arguments are passed to the listener callback when called.

```
process.stdout.on('resize', () => {
  console.log('screen size has changed!');
  console.log(`${process.stdout.columns}x${process.stdout.rows}`);
});
```

`writeStream.clearLine(dir[, callback])`

- `dir` {number}
  - `-1` : to the left from cursor
  - `1` : to the right from cursor
  - `0` : the entire line
- `callback` {Function} Invoked once the operation completes.
- Returns: {boolean} `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` .

`writeStream.clearLine()` clears the current line of this `WriteStream` in a direction identified by `dir` .

`writeStream.clearScreenDown([callback])`

- `callback` {Function} Invoked once the operation completes.
- Returns: {boolean} `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` .

`writeStream.clearScreenDown()` clears this `WriteStream` from the current cursor down.

`writeStream.columns`

A `number` specifying the number of columns the TTY currently has. This property is updated whenever the `'resize'` event is emitted.

`writeStream.cursorTo(x[, y][, callback])`

- `x` {number}
- `y` {number}
- `callback` {Function} Invoked once the operation completes.
- Returns: {boolean} `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` .

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

`writeStream.getColorDepth([env])`

- `env` {Object} An object containing the environment variables to check. This enables simulating the usage of a specific terminal. **Default:** `process.env`.
- Returns: {number}

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.

`writeStream.getWindowSize()`

- Returns: {number[]}

`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.

`writeStream.hasColors([count][, env])`

- `count` {integer} The number of colors that are requested (minimum 2). **Default:** 16.
- `env` {Object} An object containing the environment variables to check. This enables simulating the usage of a specific terminal. **Default:** `process.env`.
- Returns: {boolean}

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).
```

`writeStream.isTTY`

A `boolean` that is always `true`.

`writeStream.moveCursor(dx, dy[, callback])`

- `dx` {number}
- `dy` {number}
- `callback` {Function} Invoked once the operation completes.
- Returns: {boolean} `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`.

`writeStream.moveCursor()` moves this `WriteStream`'s cursor *relative* to its current position.

`writeStream.rows`

A `number` specifying the number of rows the TTY currently has. This property is updated whenever the `'resize'` event is emitted.

`tty.isatty(fd)`

- `fd` {number} A numeric file descriptor
- Returns: {boolean}

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.