/\*\*

\* The `timer` module exposes a global API for scheduling functions to

\* be called at some future period of time. Because the timer functions are

\* globals, there is no need to call `require('timers')` to use the API.

\*

\* The timer functions within Node.js implement a similar API as the timers API

\* provided by Web Browsers but use a different internal implementation that is

\* built around the Node.js [Event Loop](https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/#setimmediate-vs-settimeout).

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

\*/

declare module 'timers' {

import { Abortable } from 'node:events';

import { setTimeout as setTimeoutPromise, setImmediate as setImmediatePromise, setInterval as setIntervalPromise } from 'node:timers/promises';

interface TimerOptions extends Abortable {

/\*\*

\* Set to `false` to indicate that the scheduled `Timeout`

\* should not require the Node.js event loop to remain active.

\* @default true

\*/

ref?: boolean | undefined;

}

let setTimeout: typeof global.setTimeout;

let clearTimeout: typeof global.clearTimeout;

let setInterval: typeof global.setInterval;

let clearInterval: typeof global.clearInterval;

let setImmediate: typeof global.setImmediate;

let clearImmediate: typeof global.clearImmediate;

global {

namespace NodeJS {

// compatibility with older typings

interface Timer extends RefCounted {

hasRef(): boolean;

refresh(): this;

[Symbol.toPrimitive](): number;

}

interface Immediate extends RefCounted {

/\*\*

\* If true, the `Immediate` object will keep the Node.js event loop active.

\* @since v11.0.0

\*/

hasRef(): boolean;

\_onImmediate: Function; // to distinguish it from the Timeout class

}

interface Timeout extends Timer {

/\*\*

\* If true, the `Timeout` object will keep the Node.js event loop active.

\* @since v11.0.0

\*/

hasRef(): boolean;

/\*\*

\* Sets the timer's start time to the current time, and reschedules the timer to

\* call its callback at the previously specified duration adjusted to the current

\* time. This is useful for refreshing a timer without allocating a new

\* JavaScript object.

\*

\* Using this on a timer that has already called its callback will reactivate the

\* timer.

\* @since v10.2.0

\* @return a reference to `timeout`

\*/

refresh(): this;

[Symbol.toPrimitive](): number;

}

}

function setTimeout<TArgs extends any[]>(callback: (...args: TArgs) => void, ms?: number, ...args: TArgs): NodeJS.Timeout;

// util.promisify no rest args compability

// tslint:disable-next-line void-return

function setTimeout(callback: (args: void) => void, ms?: number): NodeJS.Timeout;

namespace setTimeout {

const \_\_promisify\_\_: typeof setTimeoutPromise;

}

function clearTimeout(timeoutId: NodeJS.Timeout): void;

function setInterval<TArgs extends any[]>(callback: (...args: TArgs) => void, ms?: number, ...args: TArgs): NodeJS.Timer;

// util.promisify no rest args compability

// tslint:disable-next-line void-return

function setInterval(callback: (args: void) => void, ms?: number): NodeJS.Timer;

namespace setInterval {

const \_\_promisify\_\_: typeof setIntervalPromise;

}

function clearInterval(intervalId: NodeJS.Timeout): void;

function setImmediate<TArgs extends any[]>(callback: (...args: TArgs) => void, ...args: TArgs): NodeJS.Immediate;

// util.promisify no rest args compability

// tslint:disable-next-line void-return

function setImmediate(callback: (args: void) => void): NodeJS.Immediate;

namespace setImmediate {

const \_\_promisify\_\_: typeof setImmediatePromise;

}

function clearImmediate(immediateId: NodeJS.Immediate): void;

function queueMicrotask(callback: () => void): void;

}

}

declare module 'node:timers' {

export \* from 'timers';

}