// TypeScript Version: 3.0

/// <reference types="node" />

import \* as fs from "fs";

import { EventEmitter } from "events";

import { Matcher } from 'anymatch';

export class FSWatcher extends EventEmitter implements fs.FSWatcher {

options: WatchOptions;

/\*\*

\* Constructs a new FSWatcher instance with optional WatchOptions parameter.

\*/

constructor(options?: WatchOptions);

/\*\*

\* Add files, directories, or glob patterns for tracking. Takes an array of strings or just one

\* string.

\*/

add(paths: string | ReadonlyArray<string>): this;

/\*\*

\* Stop watching files, directories, or glob patterns. Takes an array of strings or just one

\* string.

\*/

unwatch(paths: string | ReadonlyArray<string>): this;

/\*\*

\* Returns an object representing all the paths on the file system being watched by this

\* `FSWatcher` instance. The object's keys are all the directories (using absolute paths unless

\* the `cwd` option was used), and the values are arrays of the names of the items contained in

\* each directory.

\*/

getWatched(): {

[directory: string]: string[];

};

/\*\*

\* Removes all listeners from watched files.

\*/

close(): Promise<void>;

on(event: 'add'|'addDir'|'change', listener: (path: string, stats?: fs.Stats) => void): this;

on(event: 'all', listener: (eventName: 'add'|'addDir'|'change'|'unlink'|'unlinkDir', path: string, stats?: fs.Stats) => void): this;

/\*\*

\* Error occurred

\*/

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

/\*\*

\* Exposes the native Node `fs.FSWatcher events`

\*/

on(event: 'raw', listener: (eventName: string, path: string, details: any) => void): this;

/\*\*

\* Fires when the initial scan is complete

\*/

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

on(event: 'unlink'|'unlinkDir', listener: (path: string) => void): this;

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

}

export interface WatchOptions {

/\*\*

\* Indicates whether the process should continue to run as long as files are being watched. If

\* set to `false` when using `fsevents` to watch, no more events will be emitted after `ready`,

\* even if the process continues to run.

\*/

persistent?: boolean;

/\*\*

\* ([anymatch](https://github.com/micromatch/anymatch)-compatible definition) Defines files/paths to

\* be ignored. The whole relative or absolute path is tested, not just filename. If a function

\* with two arguments is provided, it gets called twice per path - once with a single argument

\* (the path), second time with two arguments (the path and the

\* [`fs.Stats`](https://nodejs.org/api/fs.html#fs\_class\_fs\_stats) object of that path).

\*/

ignored?: Matcher;

/\*\*

\* If set to `false` then `add`/`addDir` events are also emitted for matching paths while

\* instantiating the watching as chokidar discovers these file paths (before the `ready` event).

\*/

ignoreInitial?: boolean;

/\*\*

\* When `false`, only the symlinks themselves will be watched for changes instead of following

\* the link references and bubbling events through the link's path.

\*/

followSymlinks?: boolean;

/\*\*

\* The base directory from which watch `paths` are to be derived. Paths emitted with events will

\* be relative to this.

\*/

cwd?: string;

/\*\*

\* If set to true then the strings passed to .watch() and .add() are treated as literal path

\* names, even if they look like globs. Default: false.

\*/

disableGlobbing?: boolean;

/\*\*

\* Whether to use fs.watchFile (backed by polling), or fs.watch. If polling leads to high CPU

\* utilization, consider setting this to `false`. It is typically necessary to \*\*set this to

\* `true` to successfully watch files over a network\*\*, and it may be necessary to successfully

\* watch files in other non-standard situations. Setting to `true` explicitly on OS X overrides

\* the `useFsEvents` default.

\*/

usePolling?: boolean;

/\*\*

\* Whether to use the `fsevents` watching interface if available. When set to `true` explicitly

\* and `fsevents` is available this supercedes the `usePolling` setting. When set to `false` on

\* OS X, `usePolling: true` becomes the default.

\*/

useFsEvents?: boolean;

/\*\*

\* If relying upon the [`fs.Stats`](https://nodejs.org/api/fs.html#fs\_class\_fs\_stats) object that

\* may get passed with `add`, `addDir`, and `change` events, set this to `true` to ensure it is

\* provided even in cases where it wasn't already available from the underlying watch events.

\*/

alwaysStat?: boolean;

/\*\*

\* If set, limits how many levels of subdirectories will be traversed.

\*/

depth?: number;

/\*\*

\* Interval of file system polling.

\*/

interval?: number;

/\*\*

\* Interval of file system polling for binary files. ([see list of binary extensions](https://gi

\* thub.com/sindresorhus/binary-extensions/blob/master/binary-extensions.json))

\*/

binaryInterval?: number;

/\*\*

\* Indicates whether to watch files that don't have read permissions if possible. If watching

\* fails due to `EPERM` or `EACCES` with this set to `true`, the errors will be suppressed

\* silently.

\*/

ignorePermissionErrors?: boolean;

/\*\*

\* `true` if `useFsEvents` and `usePolling` are `false`). Automatically filters out artifacts

\* that occur when using editors that use "atomic writes" instead of writing directly to the

\* source file. If a file is re-added within 100 ms of being deleted, Chokidar emits a `change`

\* event rather than `unlink` then `add`. If the default of 100 ms does not work well for you,

\* you can override it by setting `atomic` to a custom value, in milliseconds.

\*/

atomic?: boolean | number;

/\*\*

\* can be set to an object in order to adjust timing params:

\*/

awaitWriteFinish?: AwaitWriteFinishOptions | boolean;

}

export interface AwaitWriteFinishOptions {

/\*\*

\* Amount of time in milliseconds for a file size to remain constant before emitting its event.

\*/

stabilityThreshold?: number;

/\*\*

\* File size polling interval.

\*/

pollInterval?: number;

}

/\*\*

\* produces an instance of `FSWatcher`.

\*/

export function watch(

paths: string | ReadonlyArray<string>,

options?: WatchOptions

): FSWatcher;