Watching Files

The watch() API connects globs to tasks using a file system watcher. It watches for changes to files that match the globs and executes the task when a change occurs. If the task doesn't signal Async Completion, it will never be run a second time.

This API provides built-in delay and queueing based on most-common-use defaults.

```
const { watch, series } = require('gulp');

function clean(cb) {
  // body omitted
  cb();
}

function javascript(cb) {
  // body omitted
  cb();
}

function css(cb) {
  // body omitted
  cb();
}

exports.default = function() {
  // You can use a single task
  watch('src/*.css', css);
  // Or a composed task
  watch('src/*.js', series(clean, javascript));
};
```

Warning: avoid synchronous

A watcher's task cannot be synchronous, like tasks registered into the task system. If you pass a sync task, the completion can't be determined and the task won't run again - it is assumed to still be running.

There is no error or warning message provided because the file watcher keeps your Node process running. Since the process doesn't exit, it cannot be determined whether the task is done or just taking a really, really long time to run.

Watched events

By default, the watcher executes tasks whenever a file is created, changed, or deleted. If you need to use different events, you can use the events option when calling watch(). The available events are 'add', 'addDir', 'change', 'unlink', 'unlinkDir', 'ready', 'error'. Additionally 'all' is available, which represents all events other than 'ready' and 'error'.

```
const { watch } = require('gulp');

exports.default = function() {
  // All events will be watched
  watch('src/*.js', { events: 'all' }, function(cb) {
    // body omitted
    cb();
  });
};
```

Initial execution

Upon calling watch(), the tasks won't be executed, instead they'll wait for the first file change.

To execute tasks before the first file change, set the ignoreInitialoption to false.

```
const { watch } = require('gulp');

exports.default = function() {
  // The task will be executed upon startup
  watch('src/*.js', { ignoreInitial: false }, function(cb) {
    // body omitted
    cb();
  });
};
```

Queueing

Each watch() guarantees that its currently running task won't execute again concurrently. When a file change is made while a watcher task is running, another execution will queue up to run when the task finishes. Only one run can be queued up at a time.
To disable queueing, set the queue option to false.

```
const { watch } = require('gulp');

exports.default = function() {
  // The task will be run (concurrently) for every change made
  watch('src/*.js', { queue: false }, function(cb) {
    // body omitted
    cb();
  });
};
```

Delay

Upon file change, a watcher task won't run until a 200ms delay has elapsed. This is to avoid starting a task too early when many files are being changed at once - like find-and-replace.
To adjust the delay duration, set the delay option to a positive integer.

```
const { watch } = require('gulp');

exports.default = function() {
  // The task won't be run until 500ms have elapsed since the first c
  watch('src/*.js', { delay: 500 }, function(cb) {
    // body omitted
    cb();
  });
};
```

Using the watcher instance

You likely won't use this feature, but if you need full control over changed files - like access to paths or metadata - use

the chokidarinstance returned from watch().
Be careful: The returned chokidar instance doesn't have queueing, delay, or async completion features.

Optional dependency

Gulp has an optional dependency called fsevents, which is a Mac-specific file watcher. If you see an installation warning for fsevents - *"npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents"* - it is not an issue. If fsevents installation is skipped, a fallback watcher will be used and any errors occurring in your gulpfile aren't related to this warning.