# combined-stream

A stream that emits multiple other streams one after another.

**NB** Currently combined-stream works with streams version 1 only. There is ongoing effort to switch this library to streams version 2. Any help is welcome. :) Meanwhile you can explore other libraries that provide streams2 support with more or less compatibility with combined-stream.

* [combined-stream2](https://www.npmjs.com/package/combined-stream2): A drop-in streams2-compatible replacement for the combined-stream module.
* [multistream](https://www.npmjs.com/package/multistream): A stream that emits multiple other streams one after another.

## Installation

npm install combined-stream

## Usage

Here is a simple example that shows how you can use combined-stream to combine two files into one:

var CombinedStream = require('combined-stream');

var fs = require('fs');

var combinedStream = CombinedStream.create();

combinedStream.append(fs.createReadStream('file1.txt'));

combinedStream.append(fs.createReadStream('file2.txt'));

combinedStream.pipe(fs.createWriteStream('combined.txt'));

While the example above works great, it will pause all source streams until they are needed. If you don't want that to happen, you can set pauseStreams to false:

var CombinedStream = require('combined-stream');

var fs = require('fs');

var combinedStream = CombinedStream.create({pauseStreams: false});

combinedStream.append(fs.createReadStream('file1.txt'));

combinedStream.append(fs.createReadStream('file2.txt'));

combinedStream.pipe(fs.createWriteStream('combined.txt'));

However, what if you don't have all the source streams yet, or you don't want to allocate the resources (file descriptors, memory, etc.) for them right away? Well, in that case you can simply provide a callback that supplies the stream by calling a next() function:

var CombinedStream = require('combined-stream');

var fs = require('fs');

var combinedStream = CombinedStream.create();

combinedStream.append(function(next) {

next(fs.createReadStream('file1.txt'));

});

combinedStream.append(function(next) {

next(fs.createReadStream('file2.txt'));

});

combinedStream.pipe(fs.createWriteStream('combined.txt'));

## API

### CombinedStream.create([options])

Returns a new combined stream object. Available options are:

* maxDataSize
* pauseStreams

The effect of those options is described below.

### combinedStream.pauseStreams = true

Whether to apply back pressure to the underlaying streams. If set to false, the underlaying streams will never be paused. If set to true, the underlaying streams will be paused right after being appended, as well as when delayedStream.pipe() wants to throttle.

### combinedStream.maxDataSize = 2 \* 1024 \* 1024

The maximum amount of bytes (or characters) to buffer for all source streams. If this value is exceeded, combinedStream emits an 'error' event.

### combinedStream.dataSize = 0

The amount of bytes (or characters) currently buffered by combinedStream.

### combinedStream.append(stream)

Appends the given stream to the combinedStream object. If pauseStreams is set to `true, this stream will also be paused right away.

streams can also be a function that takes one parameter called next. next is a function that must be invoked in order to provide the next stream, see example above.

Regardless of how the stream is appended, combined-stream always attaches an 'error' listener to it, so you don't have to do that manually.

Special case: stream can also be a String or Buffer.

### combinedStream.write(data)

You should not call this, combinedStream takes care of piping the appended streams into itself for you.

### combinedStream.resume()

Causes combinedStream to start drain the streams it manages. The function is idempotent, and also emits a 'resume' event each time which usually goes to the stream that is currently being drained.

### combinedStream.pause();

If combinedStream.pauseStreams is set to false, this does nothing. Otherwise a 'pause' event is emitted, this goes to the stream that is currently being drained, so you can use it to apply back pressure.

### combinedStream.end();

Sets combinedStream.writable to false, emits an 'end' event, and removes all streams from the queue.

### combinedStream.destroy();

Same as combinedStream.end(), except it emits a 'close' event instead of 'end'.

## License

combined-stream is licensed under the MIT license.