# delayed-stream

Buffers events from a stream until you are ready to handle them.

## Installation

npm install delayed-stream

## Usage

The following example shows how to write a http echo server that delays its response by 1000 ms.

var DelayedStream = require('delayed-stream');

var http = require('http');

http.createServer(function(req, res) {

var delayed = DelayedStream.create(req);

setTimeout(function() {

res.writeHead(200);

delayed.pipe(res);

}, 1000);

});

If you are not using Stream#pipe, you can also manually release the buffered events by calling delayedStream.resume():

var delayed = DelayedStream.create(req);

setTimeout(function() {

// Emit all buffered events and resume underlaying source

delayed.resume();

}, 1000);

## Implementation

In order to use this meta stream properly, here are a few things you should know about the implementation.

### Event Buffering / Proxying

All events of the source stream are hijacked by overwriting the source.emit method. Until node implements a catch-all event listener, this is the only way.

However, delayed-stream still continues to emit all events it captures on the source, regardless of whether you have released the delayed stream yet or not.

Upon creation, delayed-stream captures all source events and stores them in an internal event buffer. Once delayedStream.release() is called, all buffered events are emitted on the delayedStream, and the event buffer is cleared. After that, delayed-stream merely acts as a proxy for the underlaying source.

### Error handling

Error events on source are buffered / proxied just like any other events. However, delayedStream.create attaches a no-op 'error' listener to the source. This way you only have to handle errors on the delayedStream object, rather than in two places.

### Buffer limits

delayed-stream provides a maxDataSize property that can be used to limit the amount of data being buffered. In order to protect you from bad source streams that don't react to source.pause(), this feature is enabled by default.

## API

### DelayedStream.create(source, [options])

Returns a new delayedStream. Available options are:

* pauseStream
* maxDataSize

The description for those properties can be found below.

### delayedStream.source

The source stream managed by this object. This is useful if you are passing your delayedStream around, and you still want to access properties on the source object.

### delayedStream.pauseStream = true

Whether to pause the underlaying source when calling DelayedStream.create(). Modifying this property afterwards has no effect.

### delayedStream.maxDataSize = 1024 \* 1024

The amount of data to buffer before emitting an error.

If the underlaying source is emitting Buffer objects, the maxDataSize refers to bytes.

If the underlaying source is emitting JavaScript strings, the size refers to characters.

If you know what you are doing, you can set this property to Infinity to disable this feature. You can also modify this property during runtime.

### delayedStream.dataSize = 0

The amount of data buffered so far.

### delayedStream.readable

An ECMA5 getter that returns the value of source.readable.

### delayedStream.resume()

If the delayedStream has not been released so far, delayedStream.release() is called.

In either case, source.resume() is called.

### delayedStream.pause()

Calls source.pause().

### delayedStream.pipe(dest)

Calls delayedStream.resume() and then proxies the arguments to source.pipe.

### delayedStream.release()

Emits and clears all events that have been buffered up so far. This does not resume the underlaying source, use delayedStream.resume() instead.

## License

delayed-stream is licensed under the MIT license.