# Don’t Cross the Streams

MICROSOFT

JS

## A Show of Hands…

Who here has used Node.js ever?

## A Show of Hands…

Who here has used Streams in Node.js?

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, **scalable network applications**. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

```
var http = require('http');
http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');
```

## Why Node.js?

JIFASNIF: JavaScript is Fun and so Node.js is Fun.

- Isaac Schlueter (@izs)
Node.js Maintainer

https://twitter.com/izs/status/187639633641865216

## A History Lesson

To put my strongest concerns into a nutshell:

1. We should have some ways of connecting programs like garden hose--screw in another segment when it becomes when it becomes necessary to massage data in another way. This is the way of IO also.

M. D. McIlroy
October 11, 1964

## The Unix Way

```
cat in.txt | tr '[A-Z]' '[a-z]' > out.txt
```

## The Node.js Way

```
fs.createReadStream('in.txt')
  .pipe(transformStream())
  .pipe(fs.createWriteStream('out.txt'));
```

## Streams are…

…an abstraction of IO…

…incremental data in time with back pressure…

…are like Lego blocks that you can put together…

## Why Streams?

Improve Latency

Reduce memory footprint

Expand Possibilities

Enable Real-Time

## Why Use Streams?

```
var http = require('http'),
    fs = require('fs');

http.createServer( function (req, res) {
    fs.readFile('file.txt', function (err, data) {
        if (err) {
            res.statusCode = 500;
            res.end(err.toString());
        }
        else res.end(data);
    });
});
```

## Why not?

```
var http = require('http'),
    fs = require('fs');

http.createServer(function (req, res) {
    var s = fs.createReadStream('file.txt');
    s.on('error', function () {
        res.statusCode = 500;
        res.end(err.toString());
    };

    s.pipe(res);
});
```

## Why not?

```
var http = require('http'),
    fs = require('fs'),
    oppressor = require('oppressor');

http.createServer(function (req, res) {
    var s = fs.createReadStream('file.txt');
    s.on('error', function () {
        res.statusCode = 500;
        res.end(err.toString());
    };

    s.pipe(oppressor(req)).pipe(res);
});
```

## The Streams1 Class

- Special kind of Event Emitter
- Composition through pipe

```
var Stream = require('stream');

var s = new Stream();

...

s.pipe(process.stdout);
```

Well, let's say this Twinkie represents the normal amount of power in Node.js. Using the power of streams, it would be a Twinkie... thirty-five feet long, weighing approximately six hundred pounds.

## Types of Streams

Readable

Writable

Transform

Duplex

## Readable Stream

- Emit many data events and a single end event
- Implement pause/resume yourself

```
var s = new Stream();
s.readable = true;

var count = 0;
var id = setInterval(function () {
    s.emit('data', count);
    if (++count === 5) {
        s.emit('end');
        clearInterval(id);
    }
}, 1000);
```

## Writeable Stream

- Implement write, end and destroy methods

```
stream.writable = true;

s.write = function (data) { ... };

s.end = function (data) {
    if (arguments.length) s.write(data);
    this.destroy();
};

s.destroy = function () {
    this.writable = false;
};
```

## Back pressure

- Ensure Readable streams don't emit faster than Writeable streams can consume
- Drastically changing with Node >= 0.9

```
writer.write() === false   reader.pause()

writer.emit('drain')       reader.resume()
```

## Pipe

- Glues together readable and writable streams
- Handles back pressure

```
a.pipe(b).pipe(c).pipe(d)
```

## Transform streams

- Both readable and writable
- Transform input and produce result

```
readable.pipe(transform).pipe(writable)
```

## Duplex Streams

- Both readable and writable
- Both ends of the engage in a two-way interaction

```
stream1.pipe(stream2).pipe(stream1);
```

## Built-In Streams

- process.stdin, stdout, stderr
- net
- http
- fs
- child_process
- zlib

request, filed, JSONStream, mux-demux, shoe, pause-stream, emit-stream, through, scuttlebutt, tar, dnode

Who you gonna call?

## STREAM ALL THE THINGS!

## Streams on Devices

### with node-serialport

```
var sp = new SerialPort('COM5', {
    baudRate: 9600,
    dataBits: 8,
    parity: 'none',
    stopBits: 1,
    flowControl: false
});

serialPort.pipe(process.stdout);

serialPort.write('OMG IT WORKS\r');
```

https://github.com/voodootikigod/node-serialport

# Complex Event Processing
## with Beam-JS

```
var Beam = require('beam');

var source = Beam.Source();
var sink = Beam.Sink();

var even = Beam.Operator.filter(isEven);
var square = Beam.Operator.transform(sq);

source.pipe(even).pipe(square).pipe(sink);

sink.on('data', printData);

// Supply inputs
for (var i = 0; i <= 10; i++) source.push(i);
```

https://github.com/darach/beam-js

## Calling Remote Functions
### with dnode

```
var dnode = require('dnode');
var net = require('net');

var d = dnode();
d.on('remote', function (remote) {
    remote.yell('hi', function (s) {
        console.log(s);
        d.end();
    });
});

var c = net.connect(5004);
c.pipe(d).pipe(c);
```

https://github.com/substack/dnode

## Streams in the Browser
### with Browserify

```
var stream = require('stream')
var util = require('util')

function XHRStream(xhr) {
  stream.Stream.call(this)
  xhr.onreadystatechange = function () {
    ...
  };
  xhr.send(null);
}

util.inherits(XHRStream, stream.Stream)
```

https://github.com/substack/node-browserify

## Distributed Streams
### with Scuttlebutt

```
var Model = require('scuttlebutt/model')

var a = new Model();
var b = new Model();

a.set(key, value);

b.on('update', console.log);

var s = a.createStream();
s.pipe(b.createStream()).pipe(s);
```

https://github.com/dominictarr/scuttlebutt

All is well in Stream-land

...but Streams have big problems!

## Why Streams1 are bad

- Data eagerly fired whether ready or not
- Implement pause/resume yourself
- Pause still only advisory – so it might not…
- Buffering is too hard
- Overeager Backpressure

Streams2 to the rescue!

...coming in v0.1.0

http://blog.nodejs.org/2012/12/21/streams2/

## Readable Stream

- Eliminates pause/resume
- Adds read method and readable event
- From push based data event to pull based

```
function flow() {
  var chunk;
  while ((chunk = r.read()) !== null) {
    process(chunk);
  }
  r.once('readable', flow);
}
flow();
```

## Stream Symmetry

| Readable | Writable |
|---|---|
| read() => Buffer or null | write() => true/false |
| “readable” after read null | “drain” after false |
| “end” event | end() |

## Transform Stream

- Transform input using _transform
- Process input with output function
- Call callback when finished

```
..._transform = function (c, output, cb) {
    var s = String(c);
    output(new Buffer(s.toUpperCase()));
    cb();
}
```

Will all my old modules work?

YEP!

(mostly)

## Stream Handbook

https://github.com/substack/stream-handbook

substack
dominictarr
maxogden
mikeal

isaacs
raynos
fent
tootallnate

We’re ready to believe you!

# stream.end('Thank You!');

http://github.com/mattpodwysocki/SDC2013

## Credits

- Proton Stream: http://current.com/technology/90461049_las-vegas-ghostbusters-proton-stream-test.htm
- Twinkie: http://www.pics-site.com/2011/01/27/a-twinkie-in-a-ct-scanner/