BACK TO THE FRONT

NODE AND THE UI

NODE.JS

NODE

- As a web server
- As a data service
- As a build tool
- As a robotics controller
- As UI controller (e.g. CLI apps, desktop apps)
- As version control system
- …As operating system (node-os)

JAVASCRIPT

- As a web server
- As a data service
- As a build tool
- As a robotics controller
- As UI controller
- As version control system
- …As operating system (runtime.js)

MONOLINGUISTIC ARCHITECTURE

FULL STACK

- Allows for code reuse/sharing
- Eliminates cognitive context switching
- Removes artificial hurdles and blockers
- Generates unified discipline dev teams

FULL STACK JS

PayPal's Full Stack Story

- Less Lines of Code – 3-5 fold shrinkage
- Greater performance –10x throughput in scale
- Faster development cycles – 3x-10x faster

PARADIGM SOLIDARITY

BRINGING NODE CONCEPTS INTO THE UI

COMMONJS MODULARITY

```
function myMod() {

}

myMod.subMeth = function () { }

module.exports = myMod;
```

```
var pubMod = require('publishedMod');
var myMod  = require('./myMod');

pubMod.doSomething('foo', function (err, data) {
  var x = myMod.subMeth(data);

});
```

CommonJS Modularity

- Prevent global scope pollution
- Unified architectural approach for back end and front end
- Synchronous loading provides dependency system simplicity
    - Where async has advantages (e.g. lazy loading of non-critical code) you simply create multiple bundles - code can be shared between bundles with `externalise` module
    - AMD optimisations end up in the same place (concatenated and synchronous)

COMMON JS MODULARITY

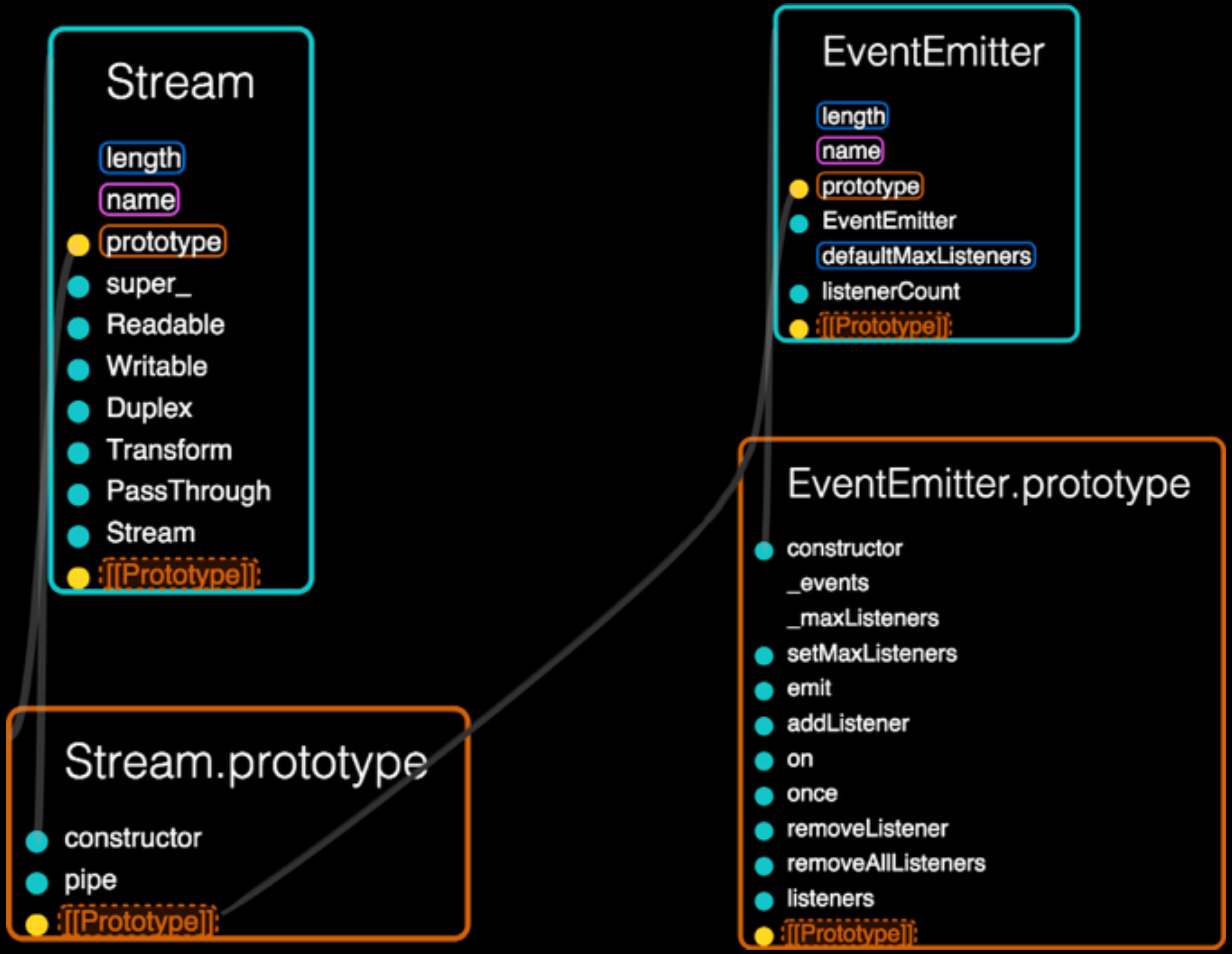
- Can be used alongside UI frameworks
- Even if they have a modular system, common js can compliment that system
- For instance, Angular.js:

```
var someDep = require('./aDep.js');
var ctrlSomething = require('./controllers/ctrlSomething');

var myMod = angular.module('app.myMod', [someDep.name])
  .controller('ctrlSomething', ctrlSomething);

module.exports = myMod;
```

STREAMS


Stream
length
name
prototype
super_
Readable
Writable
Duplex
Transform
PassThrough
Stream
[[Prototype]]
Stream.prototype
constructor
pipe
[[Prototype]]
EventEmitter
length
name
prototype
EventEmitter
defaultMaxListeners
listenerCount
[[Prototype]]
EventEmitter.prototype
constructor
_events
_maxListeners
setMaxListeners
emit
addListener
on
once
removeListener
removeAllListeners
listeners
[[Prototype]]

STREAMS

```
readadable.pipe(transform).pipe(writable)

duplex.pipe(transform).pipe(duplex)
```

STREAMS

```
var request = require('request');
var JSONStream = require('JSONStream');

var parser = JSONStream.parse('*.name', function (name) {
  return name + '\n';
});

request('http://registry.npmjs.org/-/all')
  .pipe(parser)
  .pipe(process.stdout)
```

STREAMS

```
require('net').createServer(function(socket) {

  socket.pipe(socket);

}).listen(1337);
```

Streams

- Divide everything up into input, transform, output
- Simple connectable parts with a standard interface
    - Easily recompose parts
    - Move the middle parts between browser and server environments

STREAMS

- Architecture becomes very straightforward
    - Draw a line
        - Maybe fork lines from the main line
        - Maybe make it a circle
        - Maybe add a perpendicular line to represent layer between Node and UI

STREAMS

- Performance is not the primary goal of streams
- Stability is the primary goal, it's about resource management
- In a single pass benchmark, blocking IO is likely to beat a stream (less set up overhead)
- However under load, blocking IO slows significantly (no threads), but streams perform consistently at relatively high speed.

BROWSERIFY

- Recursively walks **`require`** calls for a given file
- Builds a dependency graph
- Bundles everything into a single file, with each source file in it's own closure
- Supplies a **`require`** replacement that references dependencies from within the bundle

BROWSERIFY

- Allows core modules, like `stream`, to be used in the browser
- System specific core modules, like `fs`, can also be browserified with plugins that use client side API's
  - e.g. `fs` can use `localStorage` browser API
- Therefore, many ecosystem Node modules can also be browserified.
- Engenders a unified, modular and consistent to full stack development

STREAM ALL

THE THINGS

TRANSPORTS

- EventSource
- Websockets
- DataChannels
- HTTP Keep-Alive
- Combination Frameworks (real time libraries)

EventSource

- Server push
- Basically native long polling
- Easily polyfilled
- Not used too much, people tend to jump straight to WebSockets
- May be handy for light use cases
  - But only for browsers supporting it, fallbacks will consume memory because XHR keep alive accumulates all data
- Easily replaceable if wrapped in a stream

EVENTSOURCE

Server

```
var sse = require('sse-stream')('/sse');
var rs = require('random-stream');
var http = require('http');
var st = require('st')(__dirname + '/static');

sse.install(http.createServer(st).listen(1337))

sse.on('connection', function(socket) {
  rs().pipe(socket);
})
```

EVT-SRC-STREAM

npm install evt-src-stream

```
var EventSource = require('event-source');
var Readable = require('stream').Readable;
var util = require('util');

util.inherits(EvtSrcStream, Readable)
function EvtSrcStream (url) {
  if ( (!this instanceof EvtSrcStream) ) { return (new EvtSrcStream(url)); }
  var self = this;
  Readable.call(self, {objectMode: true});

  self.es = new EventSource(url);

  self.es.onmessage = function(ev) {
   ev.toString = function () { return ev.data + ''; }
    self.push(ev);
  }
}
EvtSrcStream.prototype._read = function() { }

module.exports = EvtSrcStream;
```

EVENTSOURCE

Client

```
var EvtSrcStream = require('evt-src-stream');

var es = new EvtSrcStream('http://localhost:1337/sse');

es.on('data', function (data) {
  document.body.innerHTML += data;
});
```

WEBSOCKETS

- Two way real time communication
- Fits a Duplex Stream paradigm perfectly
- Native API looks like it was designed in the 90's
- The **`websocket-stream`** module supplies a stream API on top of the native web socket API

WEBSOCKETS

Server

```
var WebSocketServer = require('ws').Server
var rs = require('random-stream');
var http = require('http');
var st = require('st')(__dirname + '/static');
var server = http.createServer(st).listen(1337);
var wsStream = require('websocket-stream')
var wss = new WebSocketServer({server: server});
var StrStream = require('to-string-stream');
var strStream = new StrStream();

wss.on('connection', function(socket) {
  rs().pipe(strStream).pipe(wsStream(socket));
})
```

WEBSOCKETS

Client

```
var websocket = require('websocket-stream')
var ws = websocket('ws://localhost:1337/')

ws.on('data', function (data) {
  document.body.innerHTML += data;
});
```

DataChannels

- Exciting new development in modern browsers
- Part of the WebRTC API
- Supplies video, audio and data streaming
- Data streaming is accomplished with DataChannels
- The browsers native concept of a DataChannel "stream" is a different, we still need to wrap it in a node stream API for compatibility

DATAHCHANNELS

Peer

```
var rtcDataStream = require('rtc-data-stream')
var quickconnect = require('rtc-quickconnect')
var rs = require('random-stream');
var StrStream = require('to-string-stream');
var strStream = new StrStream();

quickconnect('http://rtc.io/switchboard', {room: 'yay'})
  .createDataChannel('randomness')
  .on('channel:opened:randomness', function(id, dc) {
    var stream = rtcDataStream(dc);

    stream.on('data', function(data) {
        document.body.innerHTML += data;
    })

    rs().pipe(strStream).pipe(stream);
  });
```

DATAHCHANNELS

In Full Technicolor™

```
// ^^^ required stuff ^^^

var me = quickconnect('http://rtc.io/switchboard', {room: 'yay'})
  .createDataChannel('randomness')
  .on('channel:opened:randomness', function(id, dc) {
    var color = '#' + id.substr(-6);

    var stream = rtcDataStream(dc);
    stream.on('data', function(data) {
      document.body.innerHTML += '<span style="color:' +
        color + '">' + data + '</span>';
    })

    rs().pipe(strStream).pipe(stream);
  });

document.getElementById('me')
  .style.background = '#' + me.id.substr(-6);
```

Templates

- ReadStream
  - Initialise with template and locals
  - Output stream of generated HTML
- WriteStream - DOM Sink
  - Pipe template to stream
  - Stream modifies DOM

DUST.JS

```
var dust = require('dustjs-linkedin');
var fs = require('fs');
var data = require('./data.json')
var content = fs.readFileSync('./index.tmpl').toString();

dust.renderSource(content, data).pipe(process.stdout)
```

Templates

- Syntax Transform
  - Initialise with locals
  - Pipe in template content
  - Output stream of generated HTML

JADESTREAM

```
p Hello #{name}! You have #{count} new messages.
```

```
var jadeStream = require('jade-stream'),
  fs = require('fs');

fs.createReadStream('./index.jade')
  .pipe(jadeStream(require('./data.json')))
  .pipe(process.stdout);
```

TEMPLATES

- Data Transform
  - Initialise with template
  - Pipe data in
  - Output stream of generated HTML

STREAMSTACHE

Close but no… pipe

```
<p> Hello #{name}! You have #{count} new messages. </p>
```

```
var streamstache = require('streamstache');
var fs = require('fs');
var data = require('./data.json');

var tmpl = streamstache(fs.readFileSync('./index.tmpl'));
tmpl.pipe(process.stdout);

tmpl.name = data.name;
setTimeout(function () {
  tmpl.count = data.count;
}, 2000);
```

STREAMPLATES

Make It So

```
<p> Hello #{name}! You have #{count} new messages. </p>
```

```
var streamplates = require('streamplates');
var fs = require('fs');

var tmpl = streamplates(fs.readFileSync('./index.tmpl'));

fs.createReadStream('./data.json')
  .pipe(tmpl)
  .pipe(process.stdout)
```

STREAMPLATES

```
var request = require('hyperquest');
var JSONStream = require('JSONStream');
var streamplates = require('../..');
var fs = require('fs');
var feed = 'https://skimdb.npmjs.com/registry/_all_docs'
var opts = '?include_docs=true';
var tmpl = streamplates.partial(fs.readFileSync('./partial.tmpl'));

var parse = JSONStream.parse('rows.*.doc', function (doc) {
    return {
      name: doc.name || '',
      description: doc.description ? doc.description || '' : '',
      author: doc.author ? doc.author.name || '' : '',
      url: doc.repository ? doc.repository.url || '' : ''
    }
  })

request(feed + opts)
  .pipe(parse)
  .pipe(tmpl)
  .pipe(process.stdout)
```

TO THE UI

STREAMPLATES

Serving a template stream

```
var htmlstream = require('./htmlstream');
var http = require('http');

var style = [

  '<style>',
   'dt {font-weight: bold}',
   'dl {padding:1em}',
   'dl:nth-child(even) {background:#ddd;}',
  '</style>'

].join('');

http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(style)
  htmlstream().pipe(res);
}).listen(9110)
```

STREAMPLATES

```
var request = require('request');
var JSONStream = require('JSONStream');
var streamplates = require('../..');
var fs = require('fs');
var partial = fs.readFileSync('./partial.tmpl');
var feed = 'https://skimdb.npmjs.com/registry/_all_docs';
var opts = '?include_docs=true';

module.exports = function () {
  var tmpl = streamplates.partial(partial);
  var parse = JSONStream.parse('rows.*.doc', function (doc) {
   return {
     name: doc.name || '',
     description: doc.description ? doc.description || '' : '',
     author: doc.author ? doc.author.name || '' : '',
     url: doc.repository ? doc.repository.url || '' : ''
   }
 })
 return request(feed + opts)
   .pipe(parse)
   .pipe(tmpl)
}
```

STREAMPLATES

```
var http = require('http');
var WebSocketServer = require('ws').Server
var wsStream = require('websocket-stream');
var st = require('st')(__dirname + '/static');
var server = http.createServer(st).listen(9110);
var wss = new WebSocketServer({server: server});
var request = require('request');
var feed = 'https://skimdb.npmjs.com/registry/_changes';

function init(res, socket) {
  var since = JSON.parse(res.body).last_seq - 4;
  var url = feed + '?include_docs=true&feed=continuous&since=' + since;
  var req = request(url)
  var stream = wsStream(socket);
  req.pipe(stream);
}

wss.on('connection', function(socket) {
 request(feed + '?descending=true&limit=1', function (err, res) {
    init(res, socket)
 })
})
```

STREAMPLATES

UI Template Stream

```
var websocket = require('websocket-stream')
var ws = websocket('ws://localhost:9110/')
var StrStream = require('to-string-stream');
var strStream = new StrStream();
var JSONStream = require('JSONStream')
var streamplates = require('../../..');
var fs = require('fs');
var partial = fs.readFileSync('./partial.tmpl');

var parse = JSONStream.parse('doc', function (doc) {
  return {
    name: doc.name || '',
    description: doc.description ? doc.description || '' : '',
    author: doc.author ? doc.author.name || '' : '',
    url: doc.repository ? doc.repository.url || '' : ''
  }
})
```

STREAMPLATES

UI Template Stream

```
var tmpl = streamplates.partial(partial);

var list = [];

ws.pipe(strStream).pipe(parse).pipe(tmpl)
  .on('data', function (data) {
    if (process.title === 'browser') {
      list.unshift(data+'');
      list.length = 3;
      document.body.innerHTML = '';
      list.forEach(function (s) {
         document.body.innerHTML += s;
      });
      return;
    }
    console.log(data+'');
  });
```

WORK SHARING

```
var websocket = require('websocket-stream')
var ws = websocket('ws://localhost:9110/')
var StrStream = require('to-string-stream');
var strStream = new StrStream();
var JSONStream = require('JSONStream')
var streamplates = require('../../..');
var fs = require('fs');
var partial = fs.readFileSync('./partial.tmpl');
var rtcDataStream = require('rtc-data-stream')
var quickconnect = require('rtc-quickconnect')
var es = require('event-stream')

var parse = JSONStream.parse('doc', function (doc) {
    return {
      name: doc.name || '',
      description: doc.description ? doc.description || '' : '',
      author: doc.author ? doc.author.name || '' : '',
      url: doc.repository ? doc.repository.url || '' : ''
    }
})

var tmpl = streamplates.partial(partial);

var list = [];
var plateline = ws.pipe(strStream).pipe(parse).pipe(tmpl)
```

WORK SHARING

```
plateline.on('data', function (data) {
    if (process.title === 'browser') {
      list.unshift(data + '');
      list.length = 3;
      document.body.innerHTML = '';
      list.forEach(function (s) {
         document.body.innerHTML += s;
      });
    }
  });

var me = quickconnect('http://rtc.io/switchboard', {room: 'tmpl'})
  .createDataChannel('templates')
  .on('channel:opened:templates', function(id, dc) {
    var stream = rtcDataStream(dc);

    list.forEach(function (s) {
      stream.write(s);
    })

    plateline.pipe(stream);

  });
```

WORK SHARING

```
var rtcDataStream = require('rtc-data-stream')
var quickconnect = require('rtc-quickconnect')
var StrStream = require('to-string-stream');
var strStream = new StrStream();

var list = [];

var me = quickconnect('http://rtc.io/switchboard', {room: 'tmpl'})
  .createDataChannel('templates')
  .on('channel:opened:templates', function(id, dc) {
    var stream = rtcDataStream(dc);
    stream.pipe(strStream).on('data', function(data) {

      list.unshift(data + '');
      list.length = 3;
      document.body.innerHTML = '';
      list.forEach(function (s) {
        document.body.innerHTML += s;
      });

    })
  });
```

Components

- Create Write Stream UI components
  - The model is the data in the stream
  - View Controllers are the write streams

COMPONENTS

- Write directly to DOM
- Write to a representation that data-binds to DOM, representation can use dirty checking to optimise writes
- If using a UI framework it could write to whatever system is uses for models
    - e.g. write to the Angular Scope, let Angular do the dirty checking for you

UI STREAM

Classical

```
var Writable = require('stream').Writable;
var util = require('util');


util.inherits(MyCmpStream, Writable)
function MyCmpStream () {

  Writable.call(this);

}
MyCmpStream.prototype._write = function(chunk, enc, cb) {
   document.body.innerHTML += chunk;
   cb();
}

module.exports = MyCmpStream;
```

UI STREAM

Classical

```
var MyCmp = require('./classical-style');
var rs = require('random-stream');
var cmp = new MyCmp;

rs().pipe(cmp);
```

```
browserify index.js > built/bundle-classic.js
```

UI STREAM

event-stream

```
var es = require('event-stream');
module.exports = function () {
  return es.through(function (chunk) {
   document.body.innerHTML += chunk;
  })
}
```

UI STREAM

event-stream

```
var MyCmp = require('./es-style');
var rs = require('random-stream');
var cmp = MyCmp();

rs().pipe(cmp);
```

```
browserify index-es.js > built/bundle-es.js

stat -f '%z' built/bundle-classic.js # 123201
stat -f '%z' built/bundle-es.js # 127326
```

PRICE STREAM

```
var Pusher = require('pusher-client');
var http = require('http');

var pusher = new Pusher('de504dc5763aeef9ff52'),
  trades = pusher.subscribe('live_trades'),
  prices = require('./lib/priceStream')(),
  sse = require('sse-stream')('/');

sse.install(http.createServer().listen(1337))

trades.bind('trade', prices.write.bind(prices));

prices.pipe(process.stdout);

sse.on('connection', function (socket) {
  prices.pipe(socket);
});
```

CHART STREAM

```
var smoothie = require('smoothie');
var es = require('event-stream');
var series = new smoothie.TimeSeries();

function createTimeline(canvas) {
  var chart = new smoothie.SmoothieChart();
  chart.addTimeSeries(series, {
    strokeStyle: 'rgba(0, 255, 0, 1)',
    fillStyle: 'rgba(0, 255, 0, 0.2)',
    lineWidth: 4 });
  chart.streamTo(canvas, 2000);
}

module.exports = function (canvas) {
   var stream = es.through(function (number) {
      series.append(new Date().getTime(), +(number+''));
   })
   stream.init = createTimeline.bind(null, canvas);

   return stream;
}
```

BITCOIN TICKER

```
var EvtSrcStream = require('evt-src-stream');
var evtsrc = new EvtSrcStream('http://localhost:1337/');
var chartStream = require('./chartStream');

var prices = chartStream(document.getElementById("canvas"));

evtsrc.pipe(prices);

window.addEventListener('load', prices.init);
```

TX STREAM

```
var http = require('http');
var websocket = require('websocket-stream');
var JSONStream = require('JSONStream');
var es = require('event-stream');
var sse = require('sse-stream')('/');
var ws = websocket('ws://ws.blockchain.info:8335/inv');
sse.install(http.createServer().listen(1337))
ws.write('{"op":"unconfirmed_sub"}')

amounts = ws.pipe(JSONStream.parse())
  .pipe(es.map(function (tx, cb) {
   cb(null, (tx.x.out[0].value/1e8)+'')
  }));

amounts.pipe(process.stdout);

sse.on('connection', function (socket) {
  amounts.pipe(socket);
});
```

PIPE ON THROUGH

FS ➡ TRANSFORM ➡ TRANSPORT ➡
TRANSFORM ➡ COMPONENT ➡ DOM

PIPE FROM THE UI

- The DOM was never built with streaming in mind
- It is event driven however, and streams are just a series of events
  - But there's lots of elements and lots of events, may not fit

UI ➡ NODE

- Create a "back flow" ReadStream
- Push to the stream when relevant UI events happen
  - A click, a text box input
  - Or using your UI frameworks controller
- Be sure to mark the data you push to the ReadStream so it can be differentiated
- Multiple ReadSteams may be relevant to use case, e.g. a ReadStream for uploads, one for form input, or one for a views interaction events. All of them pipe to the transport however, so still needs to be marked.

UI → DB
NODE ← DB

- Either
  - Implement the same back flow ReadStream but pipe to a Write Stream that persists data to DB server
  - Use a client-side DB that automatically replicates to DB server - write direct to the client DB
- Node process then listens to the DB servers changes stream

POUCHDB

- CouchDB repo implemented in JavaScript, runs in the browser
- Passively synchronises with Couch server - push and pull (replication)
- Just like CouchDB, it has a changes stream
- Concept: make the DB the mediator, and put Node services on the fringes, listening to the changes stream

TODO-POUCH-ANGULAR

- Take the todo-angular seed project
- Alter it's Todo factory service to persist to PouchDB
- Setup a couch server and tell pouch to replicate to it
- Create a Node micro service that listens to the changes stream

REPLICATOR

Todo Factory

```
angular.module('myApp.services', []).
  factory('Todos', function ($rootScope) {
   var db = new PouchDB('todos');

   PouchDB.replicate('todos', 'http://localhost:5984/todos', {live: true})

   this.collection = [];


   db.allDocs({include_docs:true}, function (err, recs) {
      recs.rows.forEach(function(rec) {
         if (!rec.doc.completed) {
            this.collection.push(rec.doc)
         }

      }, this)

      $rootScope.$apply();
   }.bind(this))
```

REPLICATOR

Todo Factory

```
this.addTask = function (todo, cb) {
  var newTodo = {
    title: todo,
    complete: false
  };

  db.post(newTodo).then(function (res) {
     newTodo._rev = res.rev;
     newTodo._id = res.id;
    this.collection.push(newTodo);
    cb(todo);
  }.bind(this)).catch(console.error.bind(console));

};
```

REPLICATOR

Todo Factory

```
    this.deleteTask = function (todo) {
        db.remove(todo)
     var i = this.collection.indexOf(todo);
         this.collection.splice(i, 1);
    };
```

REPLICATOR

Todo Factory

```
    this.clearCompleted = function (todos) {

      var remove = todos.filter(function(todo) {
        return todo.complete;
      }).map(function(todo) {
        todo._deleted = true;
        return todo;
      });

      db.bulkDocs(remove);

     this.collection = todos.filter(function(todo) {
      return (!todo.complete)
      });
    };

    return this;
  });
```

REPLICATOR

Change Stream Reactor

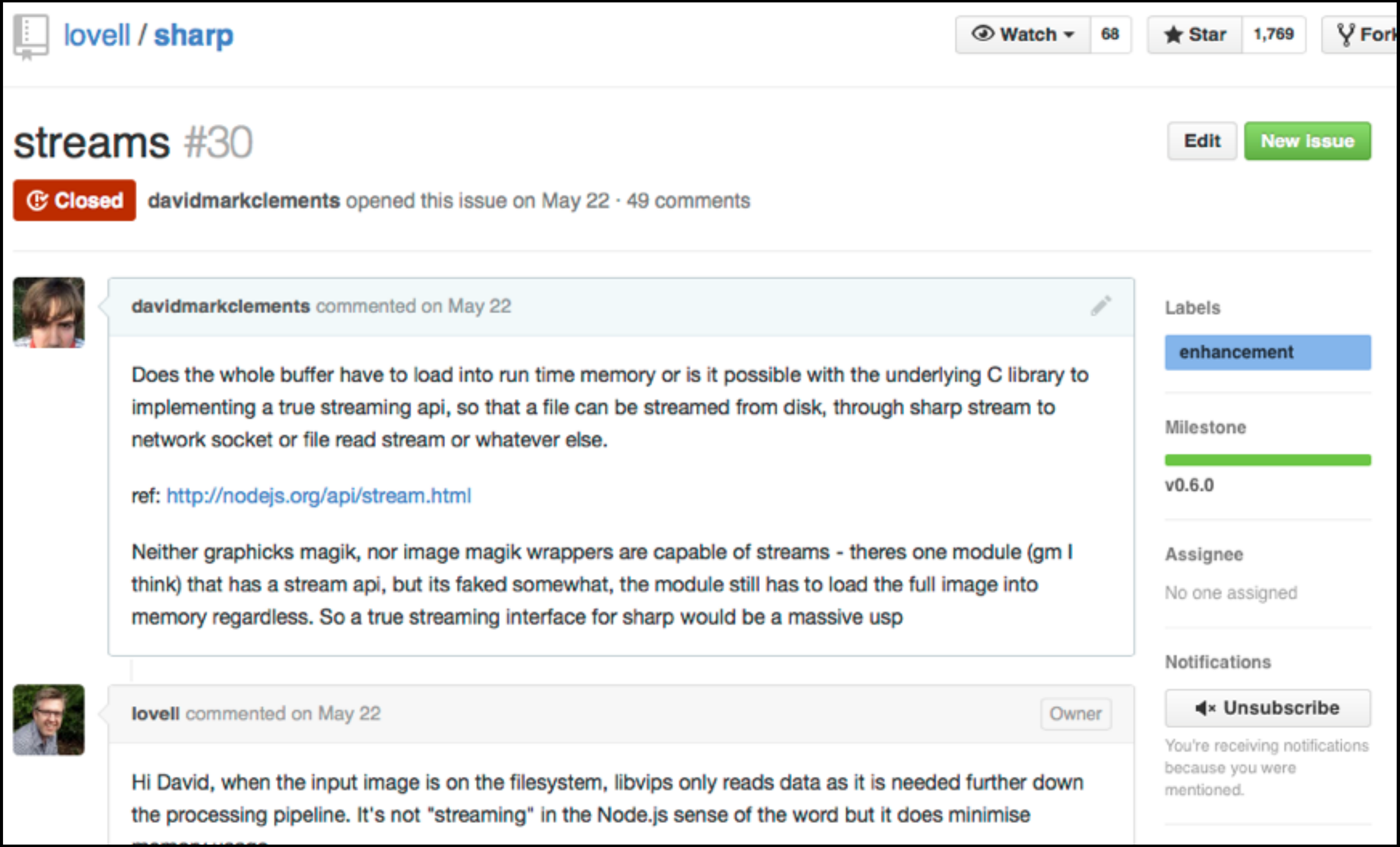
```
var request = require('request');
var es = require('event-stream');
var JSONStream = require('JSONStream');
var repo = 'http://localhost:5984/todos/';
var feed = '_changes?include_docs=true&feed=continuous';

request(repo+feed)
  .pipe(JSONStream.parse('doc'))
  .pipe(es.map(function(doc, cb) {
    if (doc._deleted) { return cb() }
    cb(null, doc.title + '\n')
  }))
  .pipe(process.stdout);
```

Image Manipulation

- Diverse device proliferation and mobile latency demands that optimal sites deliver images tailored to device characteristics
- This is always a resource intensive process, both in CPU cycles and developer effort
- Perfect case for streams
- But until very recently, image manipulation libraries with true stream support didn't exist

SHARP.JS



lovell / sharp

Watch 68 · Star 1,769 · Fork

Edit · New issue

streams #30

Closed · davidmarkclements opened this issue on May 22 · 49 comments

davidmarkclements commented on May 22

Does the whole buffer have to load into run time memory or is it possible with the underlying C library to implementing a true streaming api, so that a file can be streamed from disk, through sharp stream to network socket or file read stream or whatever else.

ref: http://nodejs.org/api/stream.html

Neither graphicks magik, nor image magik wrappers are capable of streams - theres one module (gm I think) that has a stream api, but its faked somewhat, the module still has to load the full image into memory regardless. So a true streaming interface for sharp would be a massive usp

lovell commented on May 22 · Owner

Hi David, when the input image is on the filesystem, libvips only reads data as it is needed further down the processing pipeline. It's not "streaming" in the Node.js sense of the word but it does minimise

Labels: enhancement

Milestone: v0.6.0

Assignee: No one assigned

Notifications: Unsubscribe

You're receiving notifications because you were mentioned.

SHARP.JS

```
var transform = sharp()
  .resize(300, 200)
  .crop(sharp.gravity.north)
  .rotate(45)
  .quality(80);

readable.pipe(transform).pipe(writable);


sharp('./img.jpg').resize(200).webp().pipe(writeable)
```

RESIZE STREAM

```
var fs = require('fs');
var sharp = require('sharp');
var express = require('express');

var app = express();

app.get('/:img/:width?/:height?', function (req, res) {
  var img = req.params.img;
  var h = ~~(req.params.height);
  var w = ~~(req.params.width);
  var imgStrm = fs.createReadStream('./img/' + img);
  imgStrm.pipe(sharp().resize(w, h)).pipe(res)
});

app.listen(8080)
```

RESIZE STREAM

```
var fs = require('fs');
var path = require('path');

var sharp = require('sharp');
var express = require('express');

var app = express();

app.use(rewrite);
app.use(express.static('./cache'))
app.use(unwrite);
```

RESIZE STREAM

```
app.get('/:img/:width?/:height?', function (req, res) {
  var img = req.params.img;
  var h = ~~(req.params.height);
  var w = ~~(req.params.width);
  var imgStrm = fs.createReadStream('./img/' + img);

  imgStrm = imgStrm.pipe(sharp().resize(w, h))

  imgStrm.pipe(res);
  imgStrm.pipe(fs.createWriteStream(cache(img, w, h)))

});

app.listen(8080);
```

RESIZE STREAM

```
function unwrite(req, res, next) {
  req.url = req._trueUrl;
  delete req._trueUrl;
  next();
}
function rewrite(req, res, next) {
  req._trueUrl = req.url;
  req.url = '/' + req.url.substr(1).replace(/\//g,'-')
  req.url += path.extname(req.url).split('-')[0]
  next();
}
function cache(img, w, h) {
  var cachePath = './cache/' + img
  if (w) { cachePath += '-' + w; }
  if (h) { cachePath += '-' + h; }
  if (w || h) { cachePath += path.extname(img) }
  return cachePath;
}
```

Final Thoughts

- The browser is not the only UI
  - But the same concepts discussed apply
- Use server routes to create ui view routes
  - Grab the router object
  - Stream specialised ui routes to the browser
  - Create data service routes (REST) on the server
  - Now you have UI routes + backend rendering

@DAVIDMARKCLEM

GITHUB.COM/DAVIDMARKCLEMENTS

GOTO 1