How To Node Core

NodeConf 2013

Pre-test: Raise your hand if…

- You do not have Node checked out on your computer
- You do not have Node built
- Node isn't working on your computer right now
- If so, `make -j4` now.

Who should contribute?

- Super star C++ haxx0rz?
- Low-level OS people?
- Close to metal?
- JavaScript ninja rockstar gurus?

NO!* You!

- You will hit bugs.
- You are a Node programmer.
- You can fix those bugs.
- You can achieve immortality in code.

*well, yes, those people. but ALSO you.

OSS

- Open Source Software
- Ownership of Software Stack
  - If you use it, know it.
  - If it's busted, fix it.
  - Be involved.

TODO

- Writing good tests
- Structure (brief 50¢ tour)
- Let's Feature!
- Homework

tests

tests

- Node's tests are simple JavaScript programs that either throw, or don't.

- Not throwing = test pass
- Tests (mostly) go in tests/simple/test-*.js
- No test = Rejected pull req

Running tests

- unix: make test
  Runs simple tests, and jslint
- unix: make test-all
  Runs ALL the tests, even slow awful pummel tests
- ./node test/simple/test-x.js
  To run a single test

Bad Test Names

- test-GH4313.js
- test-rfc3523.js
- test-breaks-on-linux.js
- test-thisisnotveryreadable.js

Good Test Names

- test-https-keepalive.js
- test-url-double-encode.js
- test-tls-large-request.js
- test-http-simple.js
- test-fs-writefile.js

Good Test Output

- Should be almost nothing.
- I personally like doing `console.log('ok')` at the very end, just so I know it ran fully.
- More than that is too much!

Good Test Speed

- Each test should take less than 100ms to run in the normal case.
- Rule of thumb:
  If you can read the name while `make test` runs, it's taking too long.

Good Test Code

- Short and simple
- Entirely self-contained

- Do not use tests/fixtures/ unless ABSOLUTELY necessary
- assert something in process.on('exit')

Always include the test header

```
//
// copyright mumbo jumbo
//
var common = require('../common');
var assert = require('assert');
```

Be parallel-friendly

- Servers listen on common.PORT
- Hard-coded ports don't play nicely with CI servers!

Exit Gracefully

- Close servers, unref timers
- Don't process.exit() explicitly.
- Exiting early can hide serious problems!

Fixture-free

```
switch (process.argv[2]) {
  case undefined: return parent();
  case 'child': return child();
  default: throw new Error('wtf');
}
```

Fixture-free

```
function parent() {
  var c = require('child_process')
  var child = c.spawn(
    process.execPath,
    [__filename, 'child']
  );

  // .. etc
```

Be Relevant!

- Tests FAIL without your patch
- Tests PASS with your patch
- otherwise… what's the point?

Be Relevant!

- Test the failure cases as well as success cases.
- If something SHOULD fail, make sure it DOES fail.

questions?

Exercise

- Write test-net-hello-world.js
- Assert that
  require('net').hello()
  returns the string 'world'
- Verify that test fails

structure

src/*.{cc,h}

- The C++ code, a lot like .node addons.
- This code is unforgiving. assert(0) on any weird stuff!

process.binding

- Private API function for loading binding layer
- NODE_MODULE(node_foo, fn) results in: process.binding('foo')

lib/*.js

- JavaScript Node modules that are bundled with the node binary.
- "Native modules"
- Public API
- Just like normal Node modules

lib/*.js

- Wrap process.binding() APIs in a friendly JavaScript coating
- Validate input
- Throw on bad args, emit('error') or cb(er) for run-time failures

src/node.js

- Boot-straps the module system
- Figures out what main module to load
- Sets up process object

questions?

tcp

tcp

- src/tcp_wrap.cc: process.binding('tcp_wrap')
- lib/net.js: require('net')
- Today, that's as deep as we'll go.
- (We'll cover pipes and ttys in the level 2 class.)

net.Socket

- JavaScript abstraction of a TCP connection
- Duplex stream, with methods to do TCP type stuff
- Defined in lib/net.js

net.Server

- TCP server abstraction
- connection cb => gets a connected net.Socket object
- call listen(cb) to bind to a port/IP or fs socket path
- listen cb => actually listening (async)

net.connect()

- TCP client abstraction
- Returns a net.Socket object
- Not yet connected, but in the process of connecting

questions?

Exercise

- Add a "hello" method to process.binding('tcp_wrap')
- Export this method from require('net')
- Verify test passing test-net-hello-world

http

How HTTP Works

- TCP server waits for connections
- TCP client connects, writes HTTP formatted request
- Server replies with HTTP formatted response

http_parser

- process.binding('http_parser')
- C utility for parsing HTTP
- interprets a stream of bytes as HTTP data, headers, chunked encoding, etc.

http.Server

- Inherits from net.Server
- 'request' event gets request/response objects
- Pipes incoming Sockets through http_parser
- wraps up response headers/body/trailers for transport

http.request

- HTTP client abstraction
- Returns a request object, emits 'response' with a response

HTTP Client

- Request = ClientRequest
- Response = ClientResponse

HTTP Server

- Request = ServerRequest
- Response = ServerResponse

IncomingMessage

- Incoming messages are messages that the OTHER party is sending to YOU. (They're COMING IN.)
  - ServerRequest
  - ClientResponse

Outgoing Message

- Outgoing messages are messages that the YOU are sending to the OTHER party. (They're GOING OUT.)
  - ServerResponse
  - ClientRequest

questions?

let's feature!

Protips:

- Say hi to your neighbors.
- This is your team.
- Write the test first.

.json(obj)

- Client:
  `request.json({some:'obj'})`
- Server:
  `response.json({some:'obj'})`

.json(obj)

- JSON.stringify's the object
- Sets headers:
  content-type:application/json
  content-length:<number>
- Calls this.end()
- emit('error') if can't be JSON encoded

Extra Credit

- Add optional arguments for JSON.stringify pretty-printing, and write() callback!
  - .json(obj, [indent], [cb])

pop quiz

What prototype?

A) IncomingMessage

B) OutgoingMessage

C) ServerResponse

D) ServerRequest

E) ClientRequest

F) ClientResponse

G) None of the above

What prototype?

A) IncomingMessage

B) OutgoingMessage

C) ServerResponse

D) ServerRequest

E) ClientRequest

F) ClientResponse

G) None of the above

ok for reals
do it now

- We'll give you about 15 minutes to add this function
- Raise your hand if you get stuck.
- Protip: _http_outgoing.js

homework

Homework Assignment #1

- Required reading: CONTRIBUTING.md
- Sign the CLA http://nodejs.org/cla.html
- Write good commit messages
- Be friendly on GitHub

Of course…

- It's silly to add a response.json() method to node-core
- This is an example only, it clearly belongs in userland
- "Real" bug fixes are usually much trickier

Homework Assignment #2

- Go to https://github.com/joyent/node/issues
- Find a bug
- Write a test to confirm it
- Send a patch to fix it

Homework Assignment #3

- Write this as a userland module, with a package.json
- (optional)