



WASHINGTON DC
OCTOBER 15-18, 2012

Tooling for the JavaScript era

Andy Clement, Staff Engineer

Martin Lippert, Staff Engineer

Andrew Eisenberg, Senior Member of Technical Staff

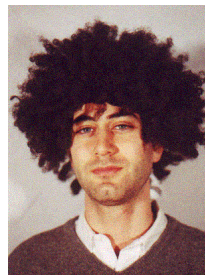
Speaker Introduction



- Andy Clement
 - Staff Engineer, R&D
 - Lead, language lab



- Martin Lippert
 - Staff Engineer, R&D
 - Lead, development tools



- Andrew Eisenberg
 - Senior Member of Technical Staff, R&D
 - Lead, Groovy-Eclipse

Disclaimer

- This session may contain product features that are currently under development.
- This session/overview of the new technology represents no commitment from SpringSource/VMware to deliver these features in any generally available product.
- Features are subject to change, and must not be included in contracts, purchase orders, or sales agreements of any kind.
- Technical feasibility and market demand will affect final delivery.
- Pricing and packaging for any new technologies or features discussed or presented have not been determined

Agenda

- The basic idea
- Observations and assumptions
- Our prototypes
- Where are we heading?

The basic idea

- Where are IDE and development tools in general heading towards?
 - Big integrated desktop IDE?
 - Lightweight editors?
 - Browser-based IDEs?
 - Does Cloud have an impact?
- Why can Google search the web in 10ms, but it takes 1000ms or more for my IDE to lookup a type hierarchy

Exploring...

- Lets do things differently:
 - Browser-based
 - Lightweight instead of fully integrated IDEs
 - Maybe cloud hosted
 - Innovate...
- And see what comes out of that...

Exploration

- Non-coding work:
 - Talk to people
 - Ask questions
 - Scribble
- Coding work:
 - Prototyping
 - Shipping early

JavaScript

- Increasing in popularity
- Not only web UI gadgets anymore
 - Serious large-scale apps in JavaScript
 - Server-side JS (node.js)
- Other dialects interesting: CoffeeScript, TypeScript
 - JS is first priority
- Fits nicely into our scope for browser-based tooling

Results and assumptions #1

- Lightweight beats heavyweight
 - Simple editors still the most popular JS tool
 - Don't want the uber tool
- Speed is essential (startup, coding, typing)
 - No acceptance for long startups, delay in typing

Results and assumptions #2

- Real code comprehension missed a lot
 - People would love to get good content-assist and code-completion
 - Aware of module definitions (AMD, RequireJS, ...)
 - Aware of frameworks
 - Fast/accurate navigation
 - Early error indication (more than just JSLint)

Results and assumptions #3

- Debugging is great, but good integration with editing is missing
 - Workarounds exist (for Chrome Dev Tools, for example)
 - Better integration would be good
- Connecting with existing popular tools

Results and assumptions #4

- A cloud-hosted workspace?
 - Need to work offline
 - Need to use other tools on my machine on the files
- A cloud-hosted tool?
 - Collaborative editing
 - Social coding
 - Zero installation – always up-to-date
 - Technically using cloud (aka unlimited) resources

Let's prototype

Prototyping

- Build some basic tools with key features that meet user need
- Make them available
- Collect feedback, adjust direction as necessary

Prototype #1

- Build:
 - Editing tool with server side ‘cloud’ workspace
 - browser based editing experience
 - Good content assist
 - Reuse tech where appropriate
 - No preference on backend technology

Prototype #1: Eclipse Orion

“Browser-based open tool integration platform”

- Eclipse Project
- Client/Server tool
- Orion is a tools platform
- Not an IDE in a browser tab

Prototype #1: Features

- An Eclipse Orion deployed internally in VMware
- With extra capabilities:
 - Better content assist than real orion
 - Basic command line console included for running some server side commands
 - e.g. vmc push to Cloud Foundry
- *What happened?*

What happened?

- Very little interest
 - Developers are busy people
 - The benefits of basic content assist did not outweigh cost of giving up their environment
 - Developers happier with code on their machine
 - Even just to 'try it out' they had to migrate some code over to the cloud workspace

Prototype #2

- Learn from our own experiences extending Orion
 - What do **we** need?
- Create a tool that would support local or remote workspace
 - Continues to be a web app, just with a local server
 - Can optionally have the server deployed remotely
 - Keep a low adoption barrier

Prototype #2

- Could use Orion, but:
 - UI for Orion not quite as snappy/fast as we wanted
 - Server is a bit heavy
 - Orion offered more facilities than we wanted
 - Another Git UI
 - Multi user setup

Prototype #2

- Reuse Eclipse Orion Code Editor
- Implement alternative lightweight backend
- Focus on:
 - Speed (startup and usage)
 - Code awareness:
 - Static code analysis
 - Content assist
 - Module system comprehension

Scripted Architecture: client side

- Eclipse Orion
 - Just the editor: familiar to Eclipse users
- Dojo for now
- Inferencing engine relying on
 - Recoverable JS parsing: esprima
 - Dependency analysis code

Scripted Architecture: server side

- Small Node.js server
 - Serving the client html/js
 - Serving requests from the client
 - ‘give me the contents of file X’
 - ‘search for this string’
 - ‘tell me the dependencies of this JS file’
- Restarted on each editor launch
 - Likely to eventually be a long running process

Scripted

- Much more positive feedback internally
- Decided to open source to access a wider audience
- Now on github:

<https://github.com/scripted-editor/scripted>

Scripted: Features

- Fast – startup and during use
- Code awareness
 - JSLint early error indication
 - Module system awareness, transitive dependency analysis
 - JSDoc comprehension
- Basic editor configuration
- Basic navigator
- Side-panel

Scripted Demo

Scripted – near term goals

- Even more code awareness
 - Even better content assist
 - More module systems
 - Maybe always ON content assist...
- Plugin model
 - Extend it with JavaScript

Scripted – near term goals

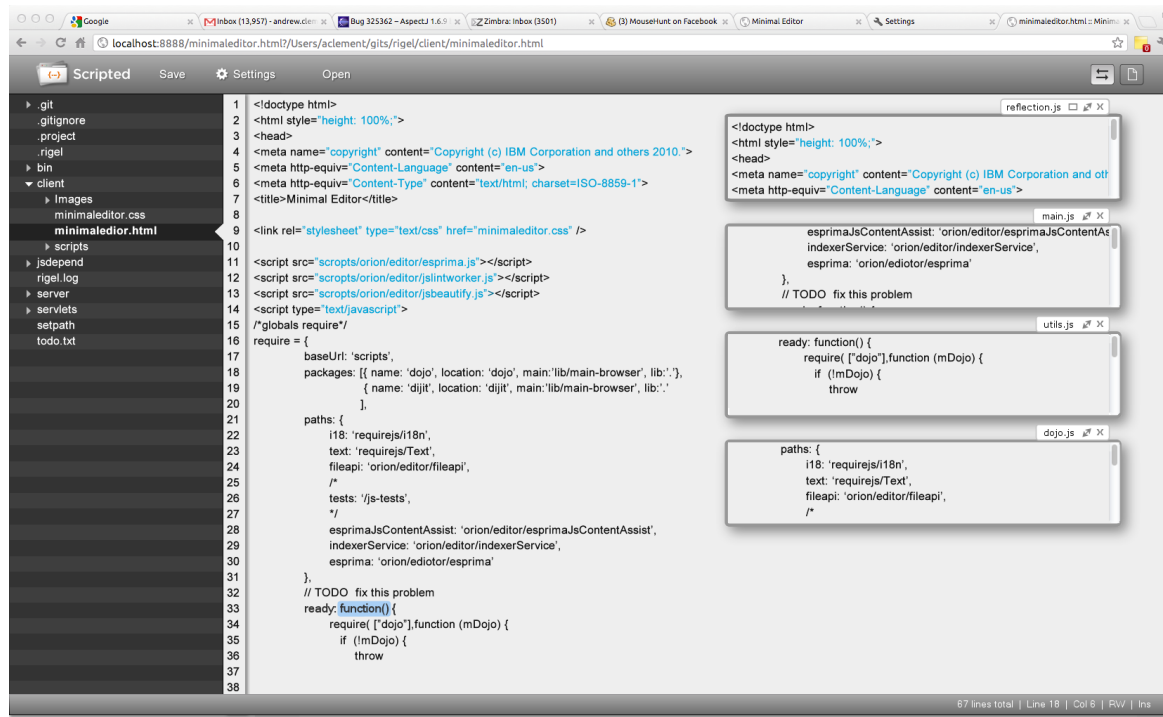
- More side-panel contents
 - More panes (documentation, search results)
 - Automated management of panes by the editor
- UI overhaul
 - we're smarter than we were when we started !

Scripted – longer term goals

- Debugging
 - Not reinventing CDT
 - Helping developers in callback hell
 - Step into server from client
- Selected tool integration
 - But not replacing command line tools, e.g. Git
- Even further out
 - Other languages (Java)

Some images from the drawing board...

Original overlays instead of side panel



Discarded

- Covering the user code just too irritating

Other side panel entries

The screenshot shows a web application interface with three main panels:

- Left Panel (File Explorer):** Displays a search bar and a list of files/folders. The list includes: `client`, `server`, `library`, `ui`, `css`, `adapter`, `clientutils.js`, `minimaleditor.js`, `minimaleditor.html`, `rigel.js`, and `utils.js`. The `client` folder is selected.
- Center Panel (Code Editor):** Displays the source code of `reflection.js`. The code includes comments and logic for handling binary files and parsing URLs. Line 31 is highlighted in yellow.
- Right Panel (Documentation):** Contains two sections:
 - url.parse(uriStr, [parseQueryString], [slashesDenoteHost])**: A function that takes a URL string and returns an object. It includes a note about passing `true` as the second argument to parse the query string.
 - url.parse is used in the following files:**: A table showing references to the `url.parse` function in various files.

File	References
minimaleditor.html	3 references
introspection.html	2 references
reflection.js	14 references
utils.js	7 references
introspection.js	1 reference
rigelserver.js	3 references
rigelserverutils.js	2 references

Will be implementing

- Documentation pane
- Search results pane
- Panes pinnable
- More panes to follow

The screenshot shows the Visual Studio Code interface with the following components:

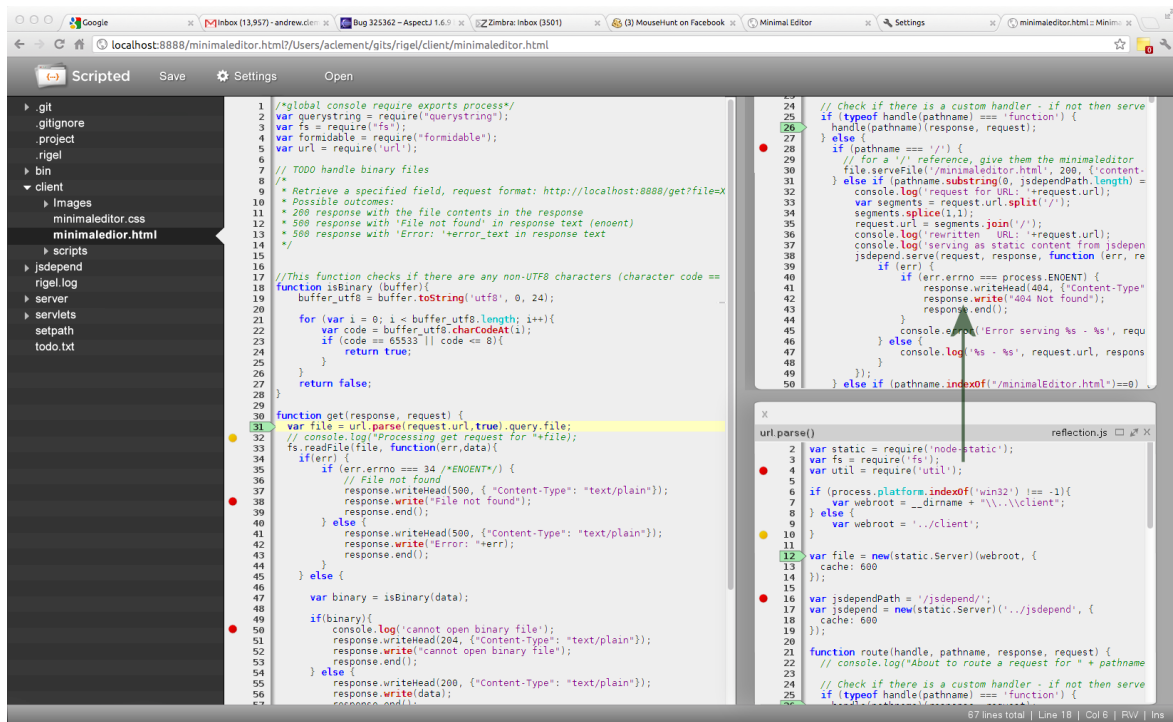
- Top Bar:** Displays the current file path as `/Users/aclement/gits/rigel/client/miniaeditor.html`.
- Left Sidebar:** Contains a search bar and a "Filters" section with categories like Attributes, Modified in the past 24 hours, Most active, Least active, and Most authors. Below this are tabs for client, server, library, ui, css, and adapter.
- Main Editor Area:** The primary workspace showing the `reflection.js` file. It contains JavaScript code for handling HTTP requests, including parsing query strings and checking for non-UTF8 characters. Line numbers 1 through 51 are visible on the left margin of the code editor.
- Right Panel:** Divided into two sections.
 - The top section, titled `url parse(uriString, [parseQueryString], [slashesDenoteHost])`, explains the function's purpose: "Take a URL string, and return an object. Pass true as the second argument to also parse the query string using the querystring module. Defaults to false." It also notes: "Pass true as the third argument to treat //foo/bar as {host: 'foo', pathname: '/bar'} rather than {pathname: '//foo/bar'}." Defaults to false.
 - The bottom section, titled `url parse is used in the following files:`, displays a table of references:

File	Symbol	References
miniaeditor.html	rige/client/	3 references
introspection.html	rige/client/	2 references
reflection.js	rige/client/scripts/	14 references
utils.js	rige/client/scripts/	7 references
introspection.js	rige/client/scripts/	1 reference
rigelserver.js	rige/server/scripts/	3 references
rigelservutils.js	rige/server/scripts/	2 references

- The bottom right corner of the editor shows the status bar with the text: `67 lines total | Line 18 | Col 8 | RV | In`.

- Filters try to ensure navigator content relevant
- Tag based filtering with auto-tagging

Scrolling side panel

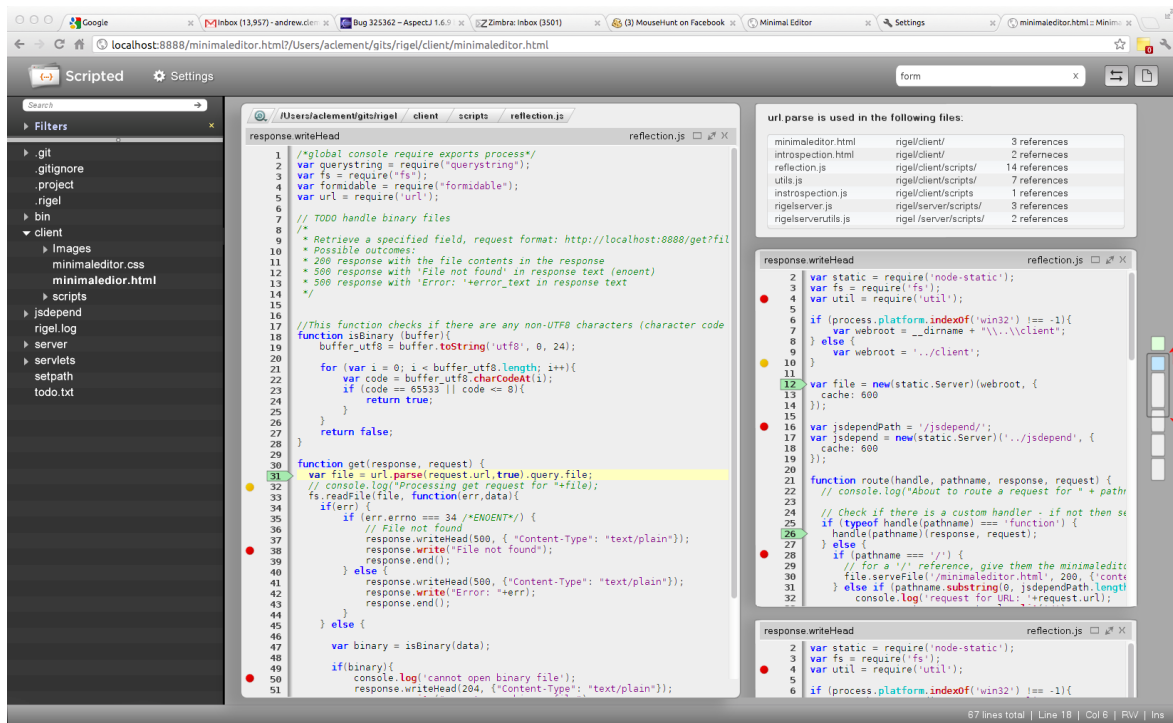


```
1 /*global console require exports process*/
2 var querystring = require('querystring');
3 var fs = require('fs');
4 var formidable = require('formidable');
5 var url = require('url');
6
7 // 1000 handle binary files
8 /*
9  * Retrieve a specified field, request format: http://localhost:8888/get?file=x
10  * Possible outcomes:
11  * 200 response with the file contents in the response
12  * 500 response with 'File not found' in response text (enoent)
13  * 500 response with 'Error: 'error_text in response text
14  */
15
16 //This function checks if there are any non-UTF8 characters (character code ==
17 function isBinary(buffer){
18   buffer_utf8 = buffer.toString('utf8', 0, 24);
19
20   for (var i = 0; i < buffer_utf8.length; i++){
21     var code = buffer_utf8.charCodeAt(i);
22     if (code == 65535 || code <= 0){
23       return true;
24     }
25   }
26   return false;
27 }
28
29
30 function get(response, request) {
31   var file = url.parse(request.url, true).query.file;
32   // console.log('Processing get request for '+file);
33   fs.readFile(file, function(err, data){
34     if(err) {
35       if (err.errno == 34 /*ENOENT*/) {
36         // File not found
37         response.writeHead(500, {'Content-Type': 'text/plain'});
38         response.write('File not found');
39         response.end();
40       } else {
41         response.writeHead(500, {'Content-Type': 'text/plain'});
42         response.write('Error: '+err);
43         response.end();
44       }
45     } else {
46       var binary = isBinary(data);
47
48       if(binary){
49         console.log('cannot open binary file');
50         response.writeHead(204, {'Content-Type': 'text/plain'});
51         response.write('cannot open binary file');
52         response.end();
53       } else {
54         response.writeHead(200, {'Content-Type': 'text/plain'});
55         response.write(data);
56         response.end();
57       }
58     }
59   });
60 }
61
62 // Check if there is a custom handler - if not then serve
63 if (typeof handle(pathname) == 'function') {
64   handle(pathname)(response, request);
65 } else {
66   if (pathname == '/') {
67     // for a '/' reference, give them the minialedior
68     file.serveFile('/minialedior.html', 200, {'content-
69   } else if (pathname.substr(0, jsdependPath.length) =
70     console.log('request for URL: '+request.url);
71     var segments = request.url.split('/');
72     segments.splice(1, 1);
73     request.url = segments.join('/');
74     console.log('rewritten URL: '+request.url);
75     console.log('serving as static content from jsdepend
76     jsdepend.serve(request, response, function (err, re
77       if (err) {
78         if (err.errno == process.ENOENT) {
79           response.writeHead(404, {'Content-Type':
80             response.write('404 Not found');
81             response.end();
82           }
83         } else {
84           console.log('Error serving %s - %s', requ
85         } else {
86           console.log('%s - %s', request.url, respons
87         }
88       }
89     } else if (pathname.indexOf('/minialedior.html')==0)
90   }
91 }
```

Still exploring

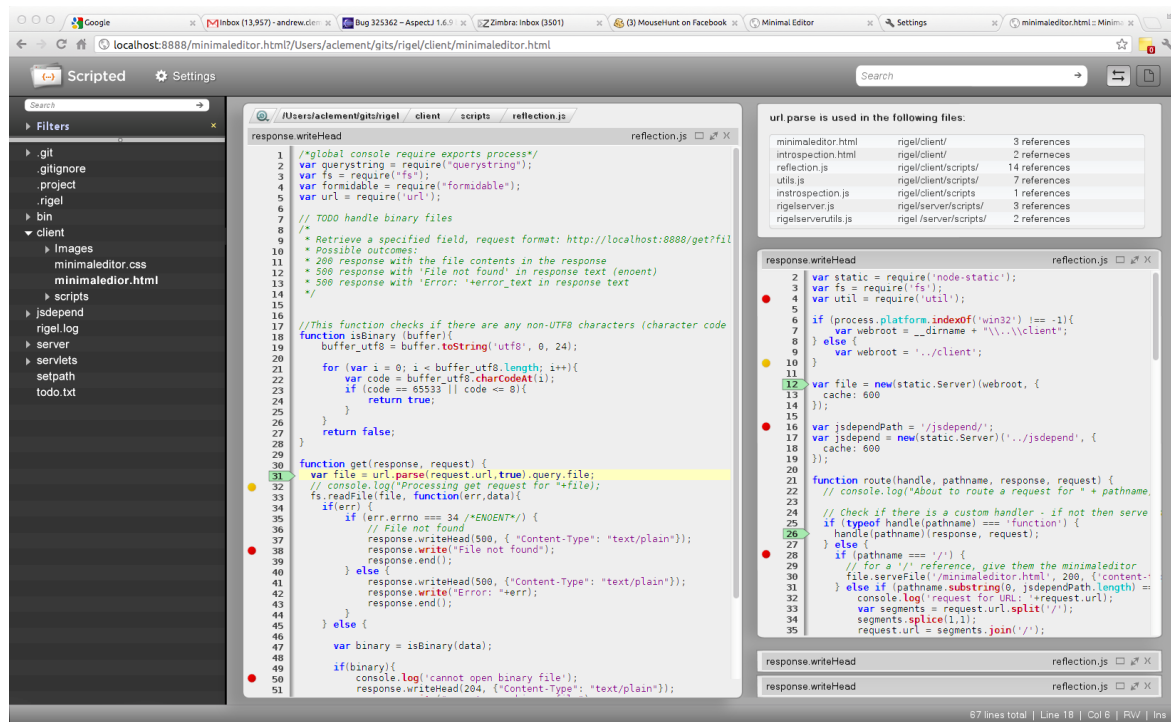
- Confusing 'scroll' story

'Overview' for side panel entries



- Unlikely to pursue
- Overview doesn't communicate enough information
- Still somewhat fiddly scrolling story

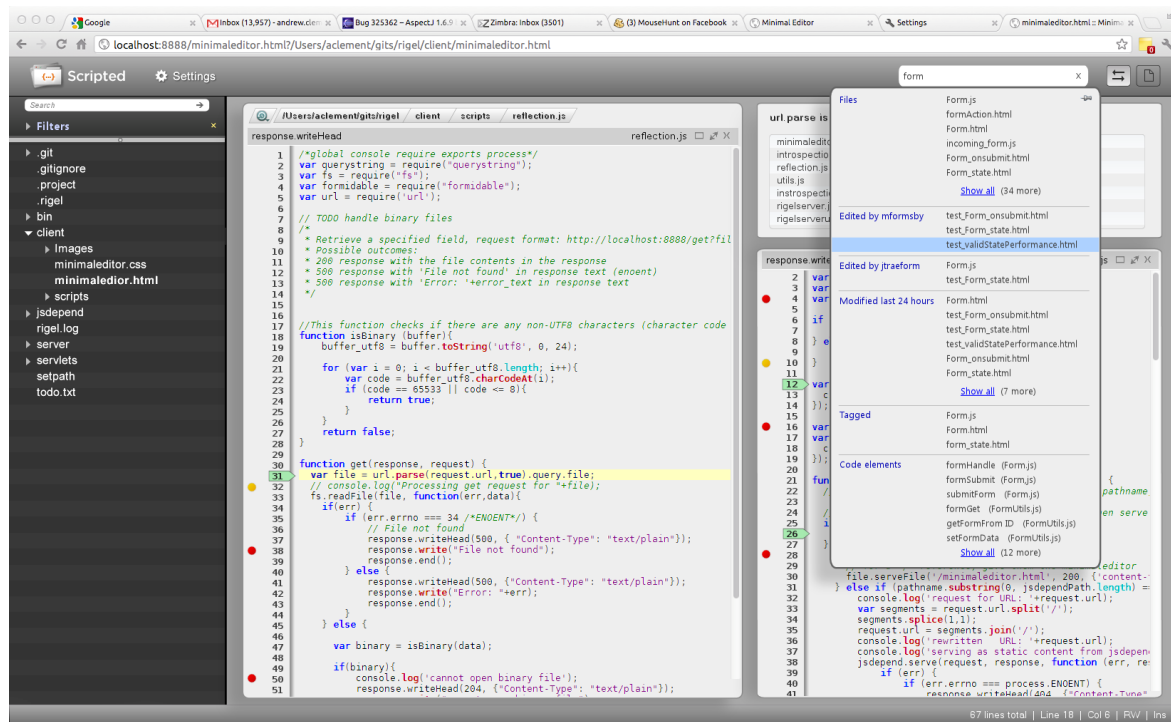
Accordion style side panel



Still exploring

- How lightweight can the collapsed entries be?
- Easier scroll story

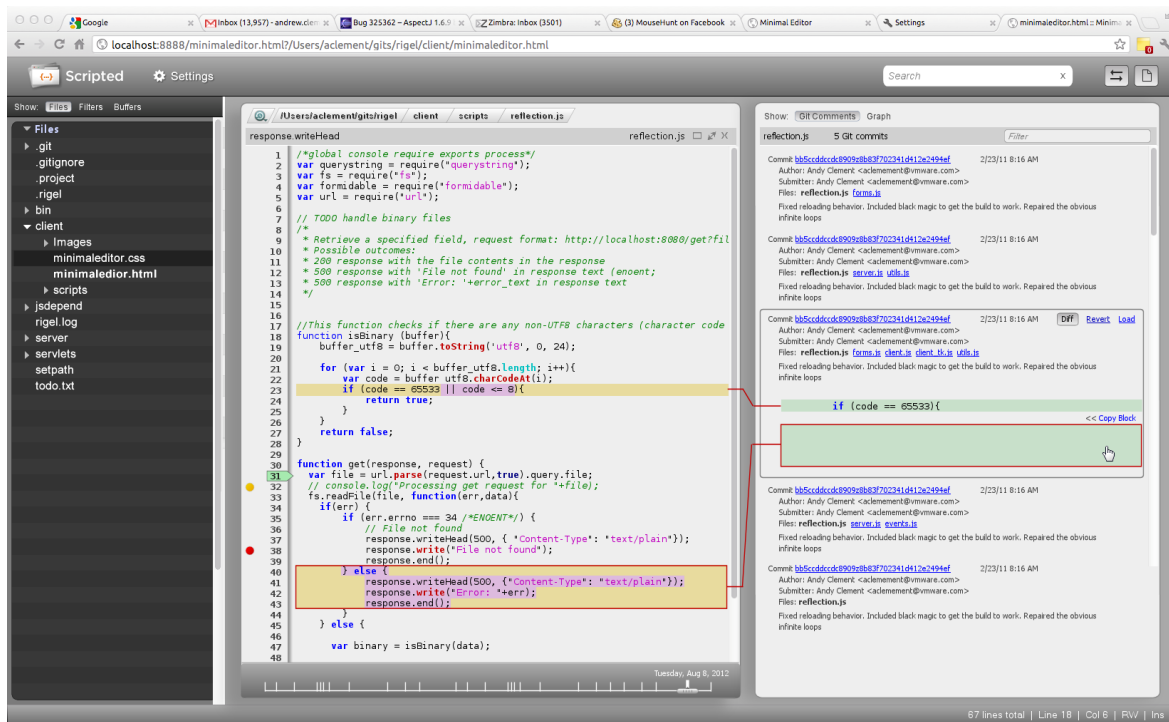
Smart global search box



Likely to implement

- One place to search for anything

Git integration



Likely to implement

- Initially just a browsing tool for comments/commit changes
- Timeline feature (at the bottom) still work in progress

Feedback welcome

- Having a reaction to some of those images?
- Let us know!

<https://groups.google.com/forum/#!forum/scripted-dev>

<https://issuetracker.springsource.com/browse/scripted>

Summary

- The basic idea
- Observations and assumptions
- The prototypes
 - The Scripted Code Editor
- Where are we heading?

Any questions?

- Google Group: scripted-dev
<https://groups.google.com/forum/#!forum/scripted-dev>
- Project page:
<https://github.com/scripted-editor/scripted>

Andy Clement

aclement@vmware.com

@andy_clement

Martin Lippert

mlippert@vmware.com

@martinlippert

Andrew Eisenberg

aeisenberg@vmware.com

@werdnagreb

The End
