

The Future of Source Maps

June 3rd, 2024

This talk

1. History of source maps
2. Anatomy of a source map
3. What is (and isn't encoded)
4. How do debuggers work (regex, babel)
5. New proposals
 - a. Scopes proposal
 - b. Debug IDs
 - c. Range mappings
6. Get involved!

About me

- Jon Kuperman
- Engineer at Bloomberg
- TC39
- Co-convenor of the source map task group



Sorry, this is a million years old!

When were source maps created?



The official Google Code blog

Get the latest updates on Google APIs and developer tools.

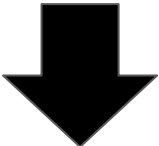
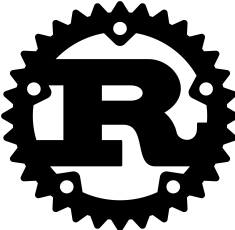
Thursday, November 05, 2009

Introducing Closure Tools



Millions of Google users worldwide use JavaScript-intensive applications such as [Gmail](#), [Google Docs](#), and [Google Maps](#). Like developers everywhere, Googlers want great web apps to be easier to create, so we've built many tools to help us develop these (and many other) apps. We're happy to announce the open sourcing of these tools, and proud to make them available to the web development community.

Why do we need source maps?



What are source maps?

- JSON objects
- Link between generated code and source code
- Created by “**generators**” (esbuild, webpack, SWC)
- Used by “**consumers**” (Chrome, Firefox, Replay.io)
- Also used by “**error monitoring tools**” (Sentry)

Anatomy of a source map

```
bundle.js.map

{
  "version": 3,
  "sources": ["index.js"],
  "sourcesContent": [
    "function greet(name) {\n  let message = \"Hello, \" + name;\n  console.log(message);\n}\ngreet(\"World\");"
  ],
  "mappings":
  "AAAA,SAASA,IAAK,GAAG;AACH,MAAMC,GAAI,GAAG,SAASC,GAAI;AACxBC,YAAY,CAACC,GAAG,CAAC;AAGtB,CAAC,"
}
```

How do generators work?

file1.js

```
export function add(first, second) {  
  return first + second;  
}
```

index.js

```
import { add } from "./file1.js";
```

```
add();
```

```
debugger;
```

bundle.js

```
((()=>{function r(d,e){return d+e}r();debugger;})();  
//# sourceMappingURL=bundle.js.map
```

bundle.js.map

```
{
  "version": 3,
  "sources": ["file1.js", "index.js"],
  "sourcesContent": ["export function add(first, second) {\n  return first +\n  second;\n}\n", "import { add } from \"../file1.js\";\n\nadd();\ndebugger;\n"],
  "mappings": "MAAO,SAASA,EAAIC,EAAOC,EAAQ,CACjC,OAAOD,EAAQC,CACjB,CCAAC,EAAI,EACJ",
  "names": ["add", "first", "second", "add"]
}
```

Mappings

VLQ encoding is used in source maps to efficiently encode integers. Each segment in the mappings string is VLQ-encoded. Here's a quick rundown of how it works:

- For each **token** in the generated file
- Capture its **line** and **column** (relative to the last generated token)
- Capture the source files **index** in the sources array
- Capture the **line** and **column** for the matching source file (relative to the last source token)

```

1 import { h, Fragment, render } from 'preact'
2 import { CounterClass, CounterFunction } from './counter'
3
4 render(() => {
5   <div>
6     <CounterClass label_="Counter 1" initialValue_={100} />
7     <CounterFunction label_="Counter 2" initialValue_={200} />
8   </div>,
9   document.getElementById('root')!,
10 },
11 )

```

```

1 // index.tsx
2 import { h as u, Fragment as l, render as c } from "preact";
3
4 // counter.tsx
5 import { h as t, Component as i } from "preact";
6 import { useState as a } from "preact/hooks";
7 var n = class extends i {
8   constructor(e) {
9     super(e);
10    this.n = () => this.setState({ t: this.state.t + 1 });
11    this.r = () => this.setState({ t: this.state.t - 1 });
12    this.state.t = e.e;
13  }
14  render() {
15    return t("div", {
16      class: "counter"
17    }, t("h1", null, this.props.label), t("p", null, t("button", {
18      onClick: this.r
19    }, "-"), " ", this.state.t, " ", t("button", {
20      onClick: this.n
21    }, "+")));
22  }
23 }, s = (r) => {
24   (o, e) => a(r.e);
25   return t("div", {
26     class: "counter"
27   }, t("h1", null, r.o), t("p", null, t("button", {
28     onClick: () => e(o - 1)
29   }, "-"), " ", 0, " ", t("button", {
30     onClick: () => e(o + 1)
31   }, "+")));
32 }
33
34 // index.tsx

```

AAAA

- A => 0 - The 0th column offset from the last generated token (aka this is the first mapping!)
- A => 0 - The 0th file in the `sources` array
- A => 0 - The 0th line offset from the last processed source token
- A => 0 - The 0th column offset from the last processed source token

How do debuggers work?



A code editor window titled "file1.js" with three window control buttons (red, yellow, green) in the top-left corner. The code inside is a JavaScript function named "add" that takes two arguments, "first" and "second". The function body contains a "debugger;" statement followed by a "return first + second;" statement. The code is color-coded: "export function" is pink, "add" is green, "first" and "second" are white, "return" is pink, and "+" is white. The closing curly brace is white.

```
file1.js  
  
export function add(first, second) {  
  debugger;  
  return first + second;  
}
```

How do debuggers work?



file2.js

```
export function multiply(first, second) {  
  return first * second;  
}
```


How do debuggers work?



index.js

```
import { add } from "./file1.js";  
import { multiply } from "./file2.js";  
  
add(2, 2);  
multiply(3, 3);
```

How do debuggers work?

```
build.js

const esbuild = require("esbuild");



esbuild
  .build({
    entryPoints: ["index.js"],
    bundle: true,
    minify: true,
    outfile: "bundle.js",
  })
  .catch(() => process.exit(1));
```




How do debuggers work?



bundle.js

```
(()=>{function o(r,t){debugger;return r+t}function u(r,t){return r*t}o(2,2);u(3,3);})();
```

Paused in debugger  


Elements Console Sources Network Performance Memory Application Security >>   

Page >>  bundle.js 

▼ top
 ▼ file://
 ▼ Users/jonat
 index.htm
 bundle.js

```
1  (()=>{  
  -    function o(r, t) { r = 2, t = 2  
  -    | debugger ;return r + t  
  -    }  
  -    function u(r, t) {  
  -    |   return r * t  
  -    }  
  -    o(2, 2);  
  -    u(3, 3);  
  -  }  
  -  )();  
2
```

 Debugger paused

▶ Watch
▼ Breakpoints
 Pause on uncaught exceptions
 Pause on caught exceptions
▼ Scope
▼ Local
 ▶ this: Window
 r: 2
 t: 2
▶ Global

 Line 1, Column 23

Coverage: n/a

Window

How do debuggers work?



```
build.js

const esbuild = require("esbuild");

esbuild
  .build({
    entryPoints: ["index.js"],
    bundle: true,
    minify: true,
    outfile: "bundle.js",
    sourcemap: true,
  })
  .catch(() => process.exit(1));
```

How do debuggers work?





bundle.js

```
(()=>{function o(r,t){debugger;return r+t}function u(r,t){return r*t}o(2,2);u(3,3);})();  
//# sourceMappingURL=bundle.js.map
```

How do debuggers work?

```
bundle.js.map

{
  "version": 3,
  "sources": ["file1.js", "file2.js", "index.js"],
  "sourcesContent": ["export function add(first, second) {\n  debugger;\n  return first + sec
return first * second;\n}\n", "import { add } from \"/file1.js\";\nimport { multiply } from
  "mappings":
  "MAAO,SAASA,EAAIC,EAAOC,EAAQ,CACjC,SACA,OAAOD,EAAQC,CACjB,CCHO,SAASC,EAASC,EAAOC,EAAQ,CACtC,0
  "names": ["add", "first", "second", "multiply", "first", "second", "add", "multiply"]
}
```

Paused in debugger  

Elements Console Sources Network Performance Memory Application Security Lighthouse

Page Workspace >> bundle.js file1.js index.js



top
file://
Users/jonathankuperman/wc
index.html
bundle.js
file1.js
file2.js
index.js

```
1 export function add(first, second) {  
2   debugger;  
3   return first + second;  
4 }  
5
```

Debugger paused

Watch
Breakpoints
 Pause on uncaught exceptions
 Pause on caught exceptions
Scope
Local
 ▶ this: Window
 first: 2
 second: 2
Global
Call Stack
 ▶ add file1.js:2
 (anonymous) index.js:4
 (anonymous) index.js:5
XHR/fetch Breakpoints
DOM Breakpoints
Global Listeners
Event Listener Breakpoints
CSP Violation Breakpoints

{ } Line 2, Column 3 (From bundle.js) Coverage: n/a

Paused in debugger  

Elements Console Sources Network Performance Memory Application Security Lighthouse

Page Workspace >> bundle.js file1.js index.js

top
file://
Users/jonathankuperman/wc
index.html
bundle.js
file1.js
file2.js
index.js

```
1 export function add(first, second) {  
2   debugger;  
3   return first + second;  
4 }  
5
```

Debugger paused

Watch
Breakpoints
 Pause on uncaught exceptions
 Pause on caught exceptions

Scope
Local
 this: Window
 first: 2
 second: 2
Global Window

Call Stack
add file1.js:2
(anonymous) index.js:4
(anonymous) index.js:5

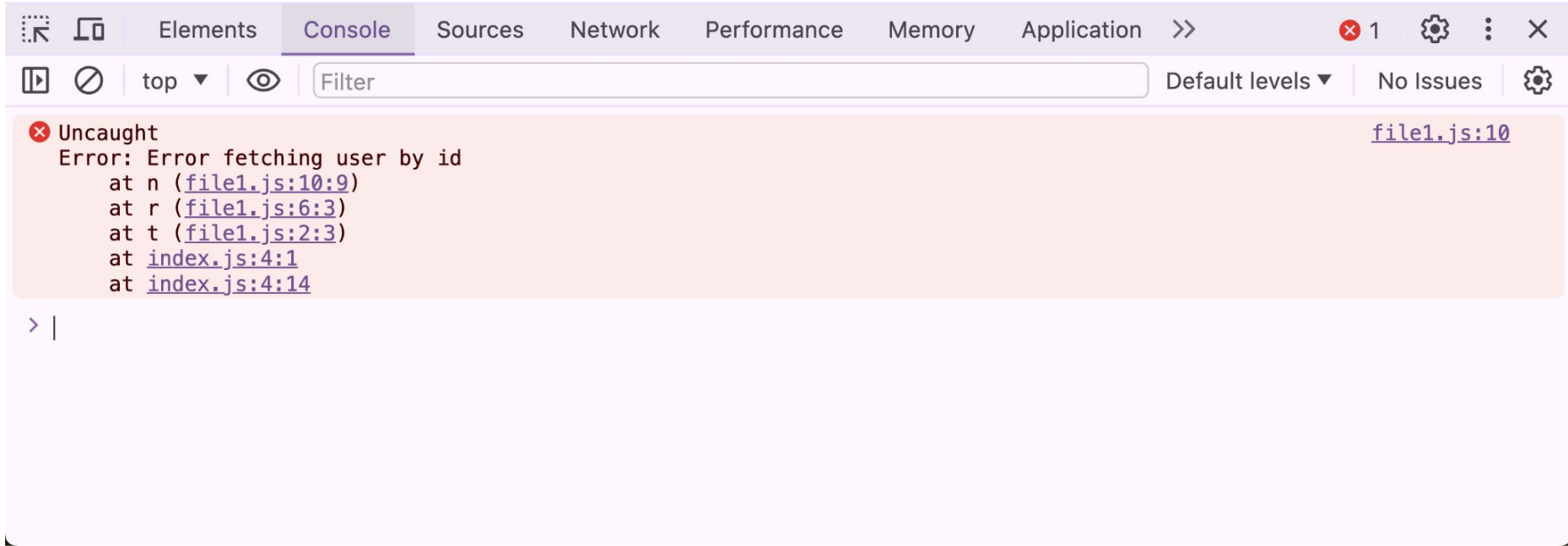
XHR/fetch Breakpoints
DOM Breakpoints
Global Listeners
Event Listener Breakpoints
CSP Violation Breakpoints

{ } Line 2, Column 3 (From bundle.js) Coverage: n/a

Consumers try to offer great experiences

- Chrome Devtools uses Regular Expressions
- Firefox runs Babel on the source files
- Scopes are hard to implement!

Stack traces are even harder...



The screenshot shows a browser's developer console with the 'Console' tab selected. The console displays an uncaught error with a stack trace. The error message is 'Error: Error fetching user by id'. The stack trace lists the following frames:

- at n ([file1.js:10:9](#))
- at r ([file1.js:6:3](#))
- at t ([file1.js:2:3](#))
- at [index.js:4:1](#)
- at [index.js:4:14](#)

The console also shows a red 'x' icon and the text 'Uncaught' next to the error message. The stack trace is displayed in a light orange background. The console also shows a 'Filter' input field and a 'Default levels' dropdown menu. The console is currently empty except for the error message and stack trace.

Bloomberg's Pasta Source Maps

x_com_bloomberg_sourcesFunctionMappings

```
// sample.js
const penne      = () => { throw Error(); }
const spaghetti = () => penne();
const orzo       = () => spaghetti();
orzo();
```



```
// **original** output
```

Error

```
  at penne (sample.js:2:33)
  at spaghetti (sample.js:3:25)
  at orzo (sample.js:4:25)
```

```
// **compiled** output
```

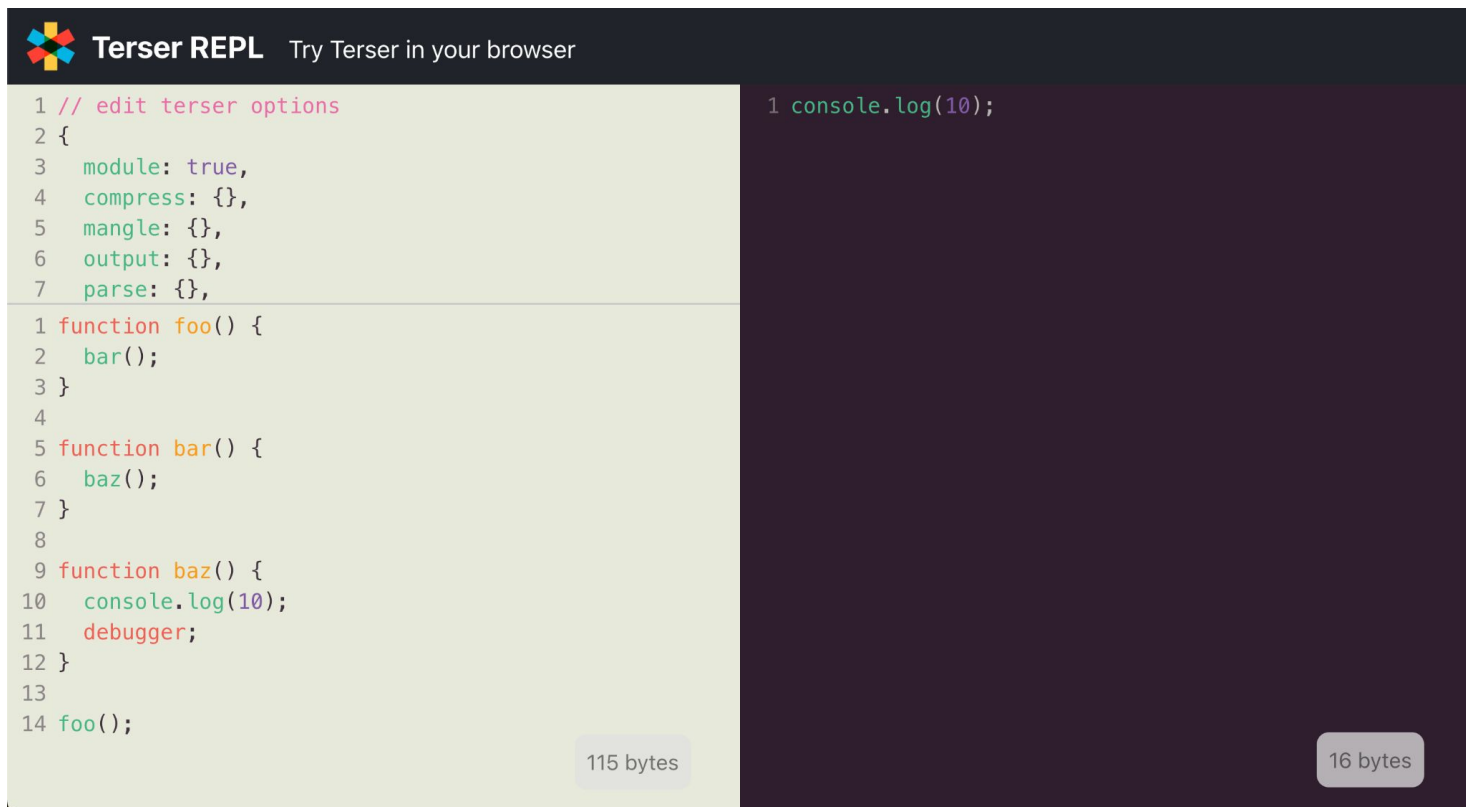
Error

```
  at r (out.js:1:82)
  at o (out.js:1:97)
  at n (out.js:1:107)
```

vs



It's not just names



The image shows a screenshot of the Terser REPL interface. The interface is split into two main panels. The left panel has a light green background and contains the original JavaScript code. The right panel has a dark purple background and contains the minified code. At the bottom of each panel, a small grey box indicates the size of the code in bytes.

Terser REPL Try Terser in your browser

```
1 // edit terser options
2 {
3   module: true,
4   compress: {},
5   mangle: {},
6   output: {},
7   parse: {},
8 }
9
10 function foo() {
11   bar();
12 }
13
14 function bar() {
15   baz();
16 }
17
18 function baz() {
19   console.log(10);
20   debugger;
21 }
22
23 foo();
```

115 bytes

```
1 console.log(10);
```

16 bytes

Difficult to move forward

Add the `x_google_ignoreList` source map extension

unigazer started this conversation in Ideas



unigazer on Apr 2, 2023

Goals

1. This will add clarity to the Chrome console.
2. It will be easier to debug the application.
3. If this gets added, the issue within the console will be resolved.

Non-Goals

Thanks for tuning in to Google I/O! [Watch content on-demand.](#)

Chrome for Developers

Was this helpful?

The ignoreList source map extension

Improve debugging experience in Chrome DevTools with the `ignoreList` source map extension.

Add `x_google_ignoreList` (Ignore-listing code) support to sourcemaps #4225 New issue



Oxdevalias opened this issue on Dec 1, 2023 · 11 comments



Oxdevalias commented on Dec 1, 2023 · edited

Introduce the `x_google_ignoreList` extension in the sourcemaps generated by this project. This will facilitate a more streamlined debugging experience in Chrome (and other supporting browsers) by automatically filtering out framework and dependency code.

Benefit

Assignees

No one assigned

Labels

feature request

The Future of Source Maps

- Formed a task group underneath the TC39 umbrella
- Improve the specification
- Embed scope information
- Embeds function and variable names
- Add Debug IDs
- Add Range Mappings

Scopes Proposal

- **Inline Functions:** Reconstruct and step through inlined functions.
- **Variable Mapping:** Map renamed/erased variables back to original names.
- **Scope Reconstruction:** Rebuild original and hidden scopes.
- **New Fields:**
 - **originalScopes:** Describes original code scopes.
 - **generatedRanges:** Describes generated code scopes and bindings.

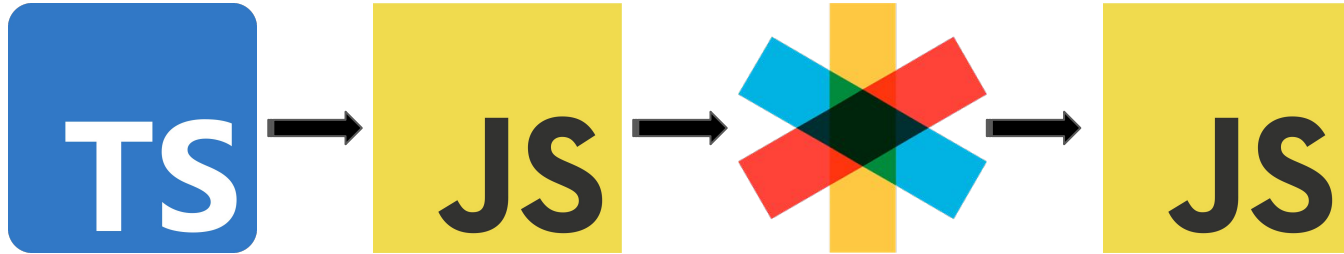
Debug IDs

- Multiple source maps
- Outdated source maps
- Stack traces

```
file1.js  
  
TypeError: Cannot read property 'length' of undefined  
    at app.min.js:2:4567  
    debugId: 85314830-023f-4cf1-a267-535f4e37bb17
```

Range Mappings

<https://github.com/tc39/source-map/blob/main/proposals/range-mappings.md>



Current participants

- Bloomberg
- Google
- JetBrains
- Meta
- Microsoft
- Mozilla
- Replay.io
- Sentry
- And more!

Come get involved!

- Already a TC39 member?
 - Join our Matrix chat! <https://matrix.to/#/#tc39-tg4:matrix.org>
 - Find our events on the TC39 calendar
 - Read out CONTRIBUTING guide
<https://github.com/tc39/source-map/blob/main/CONTRIBUTING.md>
- Not a TC39 member yet?
 - Join our matrix chat and message me (jkup) - I'll help you get involved!

Thank you!