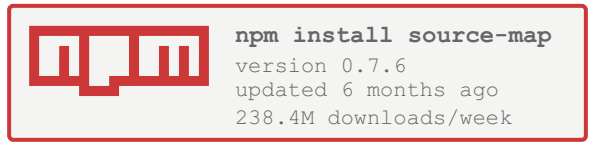
Source Map

Build Status



This is a library to generate and consume the source map format described here.

Use with Node

```
$ npm install source-map
```

Use on the Web

```
<script src="https://
raw.githubusercontent.com/mozilla/
source-map/master/dist/source-
map.min.js" defer></script>
```

Table of Contents

Examples

Consuming a source map

```
var rawSourceMap = {
  version: 3,
  file: 'min.js',
  names: ['bar', 'baz', 'n'],
  sources: ['one.js', 'two.js'],
  sourceRoot: 'http://example.com/www/
        js/',
```

```
  mappings:
        'CAAC,IAAI,IAAM,SAAUA,GAClB,OAAOC,IAAID;CCD
};

var smc = new
        SourceMapConsumer(rawSourceMap);

console.log(smc.sources);
// [ 'http://example.com/www/js/one.js',
//   'http://example.com/www/js/
        two.js' ]

console.log(smc.originalPositionFor({
  line: 2,
  column: 28
}));
// { source: 'http://example.com/www/js/
        two.js',
//   line: 2,
//   column: 10,
//   name: 'n' }

console.log(smc.generatedPositionFor({
  source: 'http://example.com/www/js/
        two.js',
  line: 2,
  column: 10
}));
// { line: 2, column: 28 }

smc.eachMapping(function (m) {
  // ...
});
```


SourceNode.prototype.toStringWithSourceMap([startOfSourceMa

Returns the string representation of this tree of source nodes, plus a SourceMapGenerator which contains all the mappings between the generated and original sources.

The arguments are the same as those to `new SourceMapGenerator`.

```
var node = new SourceNode(1, 2, "a.js",
          [
  new SourceNode(3, 4, "b.js", "uno"),
  "dos",
  [
    "tres",
    new SourceNode(5, 6, "c.js",
        "quatro")
  ]
]);

node.toStringWithSourceMap({ file: "my-
        output-file.js" })
// { code: 'unodostresquatro',
//   map: [object SourceMapGenerator] }
```

SourceNode.prototype.replaceRight(pattern, replacement)

Call `String.prototype.replace` on the very right-most source snippet. Useful for trimming white space from the end of a source node, etc.

- `pattern`: The pattern to replace.
- `replacement`: The thing to replace the pattern with.

```
// Trim trailing white space.
node.replaceRight(/\s*$/, "");
```

SourceNode.prototype.toString()

Return the string representation of this source node. Walks over the tree and concatenates all the various snippets together to one string.

```
var node = new SourceNode(1, 2, "a.js",
         [
  new SourceNode(3, 4, "b.js", "uno"),
  "dos",
  [
    "tres",
    new SourceNode(5, 6, "c.js",
        "quatro")
  ]
]);

node.toString()
// 'unodostresquatro'
```


Generating a source map

In depth guide: **Compiling to JavaScript, and Debugging with Source Maps**

With SourceNode (high level API)

```
function compile(ast) {
  switch (ast.type) {
  case 'BinaryExpression':
    return new SourceNode(
      ast.location.line,
      ast.location.column,
      ast.location.source,
      [compile(ast.left), " + ",
        compile(ast.right)]
    );
  case 'Literal':
    return new SourceNode(
      ast.location.line,
      ast.location.column,
      ast.location.source,
      String(ast.value)
    );
  // ...
  default:
    throw new Error("Bad AST");
  }
}

var ast = parse("40 + 2", "add.js");
console.log(compile(ast).toStringWithSourceMap({
```

```
  file: 'add.js'
}));
// { code: '40 + 2',
//   map: [object SourceMapGenerator] }
```

With SourceMapGenerator (low level API)

```
var map = new SourceMapGenerator({
  file: "source-mapped.js"
});

map.addMapping({
  generated: {
    line: 10,
    column: 35
  },
  source: "foo.js",
  original: {
    line: 33,
    column: 2
  },
  name: "christopher"
});

console.log(map.toString());
// '{"version":3,"file":"source-mapped.js","sources'
        ["foo.js"],"names":
        ["christopher"],"mappings":";;;;;;;;;mCAgCE
```



```
var node = new SourceNode(null, null,
        null, [a, b, c]);
node.walkSourceContents(function
        (source, contents) {
        console.log("WALK:", source,
        ":", contents); })
// WALK: a.js : original a
// WALK: b.js : original b
// WALK: c.js : original c
```

SourceNode.prototype.join(sep)

Like `Array.prototype.join` except for SourceNodes. Inserts the separator between each of this source node's children.

- `sep`: The separator.

```
var lhs = new SourceNode(1, 2, "a.rs",
        "my_copy");
var operand = new SourceNode(3, 4,
        "a.rs", "=");
var rhs = new SourceNode(5, 6, "a.rs",
        "orig.clone()");

var node = new SourceNode(null, null,
        null, [ lhs, operand, rhs ]);
var joinedNode = node.join(" ");
```

```
node.walk(function (code, loc) {
        console.log("WALK:", code,
        loc); })
// WALK: uno { source: 'b.js', line: 3,
        column: 4, name: null }
// WALK: dos { source: 'a.js', line: 1,
        column: 2, name: null }
// WALK: tres { source: 'a.js', line: 1,
        column: 2, name: null }
// WALK: quatro { source: 'c.js', line:
        5, column: 6, name: null }
```

SourceNode.prototype.walkSourceContents(fn)

Walk over the tree of SourceNodes. The walking function is called for each source file content and is passed the filename and source content.

- `fn`: The traversal function.

```
var a = new SourceNode(1, 2, "a.js",
        "generated from a");
a.setSourceContent("a.js", "original
        a");
var b = new SourceNode(1, 2, "b.js",
        "generated from b");
b.setSourceContent("b.js", "original
        b");
var c = new SourceNode(1, 2, "c.js",
        "generated from c");
c.setSourceContent("c.js", "original
        c");
```


API

Get a reference to the module:

```
// Node.js
var sourceMap = require('source-map');

// Browser builds
var sourceMap = window.sourceMap;

// Inside Firefox
const sourceMap = require("devtools/
        toolkit/sourcemap/source-
        map.js");
```

SourceMapConsumer

A SourceMapConsumer instance represents a parsed source map which we can query for information about the original file positions by giving it a file position in the generated source.

new SourceMapConsumer(rawSourceMap)

The only parameter is the raw source map (either as a string which can be `JSON.parse`'d, or an object). According to the spec, source maps have the following attributes:

- `version`: Which version of the source map spec this map is following.
- `sources`: An array of URLs to the original source files.

- `names`: An array of identifiers which can be referenced by individual mappings.
- `sourceRoot`: Optional. The URL root from which all sources are relative.
- `sourcesContent`: Optional. An array of contents of the original source files.
- `mappings`: A string of base64 VLQs which contain the actual mappings.
- `file`: Optional. The generated filename this source map is associated with.

```
var consumer = new
        sourceMap.SourceMapConsumer(rawSourceMapJso
```

SourceMapConsumer.prototype.computeColumnSpans()

Compute the last column for each generated mapping. The last column is inclusive.

```
// Before:
consumer.allGeneratedPositionsFor({
        line: 2, source: "foo.coffee" })
// [ { line: 2,
//     column: 1 },
//   { line: 2,
//     column: 10 },
//   { line: 2,
//     column: 20 } ]

consumer.computeColumnSpans();

// After:
```


SourceNode.prototype.setSourceContent(sourceFile, sourceContent)

Set the source content for a source file. This will be added to the `SourceMap` in the `sourcesContent` field.

- `sourceFile`: The filename of the source file
- `sourceContent`: The content of the source file

```
node.setSourceContent("module-one.scm",

        fs.readFileSync("path/to/module-
        one.scm"))
```

SourceNode.prototype.walk(fn)

Walk over the tree of JS snippets in this node and its children. The walking function is called once for each snippet of JS and is passed that snippet and the its original associated source's line/column location.

- `fn`: The traversal function.

```
var node = new SourceNode(1, 2, "a.js",
          [
  new SourceNode(3, 4, "b.js", "uno"),
  "dos",
  [
    "tres",
    new SourceNode(5, 6, "c.js",
        "quatro")
  ]
]);
```

```
var consumer = new
        SourceMapConsumer(fs.readFileSync("path/
        to/my-file.js.map", "utf8"));
var node =
        SourceNode.fromStringWithSourceMap(fs.readF
        to/my-file.js"),

        consumer);
```

SourceNode.prototype.add(chunk)

Add a chunk of generated JS to this source node.

- `chunk`: A string snippet of generated JS code, another instance of `SourceNode`, or an array where each member is one of those things.

```
node.add(" + ");
node.add(otherNode);
node.add([leftHandOperandNode, " + ",
        rightHandOperandNode]);
```

SourceNode.prototype.prepend(chunk)

Prepend a chunk of generated JS to this source node.

- `chunk`: A string snippet of generated JS code, another instance of `SourceNode`, or an array where each member is one of those things.

```
node.prepend("/** Build Id:
        f783haef86324gf **/\n\n");
```



```
consumer.allGeneratedPositionsFor({
        line: 2, source: "foo.coffee" })
// [ { line: 2,
//     column: 1,
//     lastColumn: 9 },
//   { line: 2,
//     column: 10,
//     lastColumn: 19 },
//   { line: 2,
//     column: 20,
//     lastColumn: Infinity } ]
```

SourceMapConsumer.prototype.originalPositionFor(generatedPosi

Returns the original source, line, and column information for the generated source's line and column positions provided. The only argument is an object with the following properties:

- `line`: The line number in the generated source. Line numbers in this library are 1-based (note that the underlying source map specification uses 0-based line numbers – this library handles the translation).
- `column`: The column number in the generated source. Column numbers in this library are 0-based.
- `bias`: Either `SourceMapConsumer.GREATEST_LOWER_BOUND` or `SourceMapConsumer.LEAST_UPPER_BOUND`. Specifies whether to return the closest element that is smaller than or greater than the one we are searching for, respectively, if the exact element cannot be found. Defaults to `SourceMapConsumer.GREATEST_LOWER_BOUND`.

and an object is returned with the following properties:

- `source`: The original source file, or null if this information is not available.
- `line`: The line number in the original source, or null if this information is not available. The line number is 1-based.
- `column`: The column number in the original source, or null if this information is not available. The column number is 0-based.
- `name`: The original identifier, or null if this information is not available.

```
consumer.originalPositionFor({ line: 2,
        column: 10 })
// { source: 'foo.coffee',
//   line: 2,
//   column: 2,
//   name: null }

consumer.originalPositionFor({ line:
        99999999999999999, column:
        999999999999999 })
// { source: null,
//   line: null,
//   column: null,
//   name: null }
```

SourceMapConsumer.prototype.generatedPositionFor(originalPosition)

Returns the generated line and column information for the original source, line, and column positions provided. The only argument is an object with the following properties:

- `source`: The filename of the original source.

- `source`: The original source's filename; null if no filename is provided.
- `chunk`: Optional. Is immediately passed to `SourceNode.prototype.add`, see below.
- `name`: Optional. The original identifier.

```
var node = new SourceNode(1, 2,
        "a.cpp", [
  new SourceNode(3, 4, "b.cpp", "extern
        int status;\n"),
  new SourceNode(5, 6, "c.cpp",
        "std::string* make_string(size_t
        n);\n"),
  new SourceNode(7, 8, "d.cpp", "int
        main(int argc, char** argv) {}
        \n"),
]);
```

SourceNode.fromStringWithSourceMap(code, sourceMapConsumer[, relativePath])

Creates a SourceNode from generated code and a SourceMapConsumer.

- `code`: The generated code
- `sourceMapConsumer` The SourceMap for the generated code
- `relativePath` The optional path that relative sources in `sourceMapConsumer` should be relative to.

SourceMapGenerator.prototype.toString()

Renders the source map being generated to a string.

```
generator.toString()
// '{"version":3,"sources":["module-
        one.scm"],"names":
        [],"mappings":"...snip...","file":"my-
        generated-javascript-
        file.js","sourceRoot":"http://
        example.com/app/js/"}'
```

SourceNode

SourceNodes provide a way to abstract over interpolating and/or concatenating snippets of generated JavaScript source code, while maintaining the line and column information associated between those snippets and the original source code. This is useful as the final intermediate representation a compiler might use before outputting the generated JS and source map.

new SourceNode([line, column, source[, chunk[, name]]])

- `line`: The original line number associated with this source node, or null if it isn't associated with an original line. The line number is 1-based.
- `column`: The original column number associated with this source node, or null if it isn't associated with an original column. The column number is 0-based.

- `line`: The line number in the original source. The line number is 1-based.
- `column`: The column number in the original source. The column number is 0-based.

and an object is returned with the following properties:

- `line`: The line number in the generated source, or null. The line number is 1-based.
- `column`: The column number in the generated source, or null. The column number is 0-based.

```
consumer.generatedPositionFor({ source:
        "example.js", line: 2, column:
        10 })
// { line: 1,
//   column: 56 }
```

SourceMapConsumer.prototype.allGeneratedPositionsFor(original

Returns all generated line and column information for the original source, line, and column provided. If no column is provided, returns all mappings corresponding to a either the line we are searching for or the next closest line that has any mappings. Otherwise, returns all mappings corresponding to the given line and either the column we are searching for or the next closest column that has any offsets.

The only argument is an object with the following properties:

- `source`: The filename of the original source.
- `line`: The line number in the original source. The line number is 1-based.

- `column`: Optional. The column number in the original source. The column number is 0-based.

and an array of objects is returned, each with the following properties:

- `line`: The line number in the generated source, or null. The line number is 1-based.
- `column`: The column number in the generated source, or null. The column number is 0-based.

```
consumer.allGeneratedpositionsfor({
        line: 2, source: "foo.coffee" })
// [ { line: 2,
//     column: 1 },
//   { line: 2,
//     column: 10 },
//   { line: 2,
//     column: 20 } ]
```

SourceMapConsumer.prototype.hasContentsOfAllSources()

Return true if we have the embedded source content for every source listed in the source map, false otherwise.

In other words, if this method returns `true`, then `consumer.sourceContentFor(s)` will succeed for every source `s` in `consumer.sources`.

```
// ...
if (consumer.hasContentsOfAllSources()) {
  consumerReadyCallback(consumer);
} else {
  fetchSources(consumer,
       consumerReadyCallback);
```


- `sourceContent` the content of the source file.

```
generator.setSourceContent("module-
        one.scm",

        fs.readFileSync("path/to/module-
        one.scm"))
```

SourceMapGenerator.prototype.applySourceMap(sourceMapCons sourceFile[, sourceMapPath]])

Applies a SourceMap for a source file to the SourceMap. Each mapping to the supplied source file is rewritten using the supplied SourceMap. Note: The resolution for the resulting mappings is the minimum of this map and the supplied map.

- `sourceMapConsumer`: The SourceMap to be applied.
- `sourceFile`: Optional. The filename of the source file. If omitted, sourceMapConsumer.file will be used, if it exists. Otherwise an error will be thrown.
- `sourceMapPath`: Optional. The dirname of the path to the SourceMap to be applied. If relative, it is relative to the SourceMap.

This parameter is needed when the two SourceMaps aren't in the same directory, and the SourceMap to be applied contains relative source paths. If so, those relative source paths need to be rewritten relative to the SourceMap.

If omitted, it is assumed that both SourceMaps are in the same directory, thus not needing any rewriting. (Supplying `'.'` has the same effect.)

SourceMapGenerator.fromSourceMap(sourceMapConsumer)

Creates a new `SourceMapGenerator` from an existing `SourceMapConsumer` instance.

- `sourceMapConsumer` The SourceMap.

```
var generator =
        sourceMap.SourceMapGenerator.fromSourceMap(
```

SourceMapGenerator.prototype.addMapping(mapping)

Add a single mapping from original source line and column to the generated source's line and column for this source map being created. The mapping object should have the following properties:

- `generated`: An object with the generated line and column positions.
- `original`: An object with the original line and column positions.
- `source`: The original source file (relative to the sourceRoot).
- `name`: An optional original token name for this mapping.

```
generator.addMapping({
  source: "module-one.scm",
  original: { line: 128, column: 0 },
  generated: { line: 3, column: 456 }
})
```

SourceMapGenerator.prototype.setSourceContent(sourceFile, sourceContent)

Set the source content for an original source file.

- `sourceFile` the URL of the original source file.


```
}
// ...
```

SourceMapConsumer.prototype.sourceContentFor(source[, returnNullOnMissing])

Returns the original source content for the source provided. The only argument is the URL of the original source file.

If the source content for the given source is not found, then an error is thrown. Optionally, pass `true` as the second param to have `null` returned instead.

```
consumer.sources
// [ "my-cool-lib.clj" ]

consumer.sourceContentFor("my-cool-
        lib.clj")
// "..."

consumer.sourceContentFor("this is not
        in the source map");
// Error: "this is not in the source
        map" is not in the source map

consumer.sourceContentFor("this is not
        in the source map", true);
// null
```

SourceMapConsumer.prototype.eachMapping(callback, context, order)

Iterate over each mapping between an original source/line/column and a generated line/column in this source map.

- `callback`: The function that is called with each mapping. Mappings have the form `{ source, generatedLine, generatedColumn, originalLine, originalColumn, name }`
- `context`: Optional. If specified, this object will be the value of `this` every time that `callback` is called.
- `order`: Either `SourceMapConsumer.GENERATED_ORDER` or `SourceMapConsumer.ORIGINAL_ORDER`. Specifies whether you want to iterate over the mappings sorted by the generated file's line/column order or the original's source/line/column order, respectively. Defaults to `SourceMapConsumer.GENERATED_ORDER`.

```
consumer.eachMapping(function (m) {
        console.log(m); })
// ...
// { source: 'illmatic.js',
//   generatedLine: 1,
//   generatedColumn: 0,
//   originalLine: 1,
//   originalColumn: 0,
//   name: null }
// { source: 'illmatic.js',
//   generatedLine: 2,
//   generatedColumn: 0,
```



```
//   originalLine: 2,
//   originalColumn: 0,
//   name: null }
// ...
```

SourceMapGenerator

An instance of the SourceMapGenerator represents a source map which is being built incrementally.

new SourceMapGenerator([startOfSourceMap])

You may pass an object with the following properties:

- `file`: The filename of the generated source that this source map is associated with.
- `sourceRoot`: A root for all relative URLs in this source map.
- `skipValidation`: Optional. When `true`, disables validation of mappings as they are added. This can improve performance but should be used with discretion, as a last resort. Even then, one should avoid using this flag when running tests, if possible.

```
var generator = new
        sourceMap.SourceMapGenerator({
  file: "my-generated-javascript-
        file.js",
  sourceRoot: "http://example.com/app/
        js/"
});
```