Source Map Revision 3 Proposal

Better bidirectional mapping.

John Lenz, Google
Nick Fitzgerald, Mozilla
February 11, 2011

Table of Contents

Document Revisions

April 12, 2011	John Lenz	Initial Revision
April 15, 2011	John Lenz	Updates to reflect prototype

July 20, 2011	John Lenz	Removed “lineCount” field, remove “Combined Map” section
August 18, 2011	John Lenz	Draft
May 2, 2012	John Lenz	HTTP header and CC-BY-SA license
July 30, 2012	John Lenz	Modified recommended HTTP header name.
August 20, 2012	John Lenz	Add CSS linkage recommendation
October 24, 2012	John Lenz	Add clarifying section on source locations.
February 19, 2013	John Lenz	Add “sourcesContent” line to support self contained source maps. Added note regarding using data uri to load source maps.
May 16, 2013	John Lenz	Updated linking convention to use # instead of @. @ conflicts with internet explorer’s conditional code
November 18, 2013	John Lenz	Noted that “file” is an optional field. Minor typographical corrections

License

Discussion

To discuss or propose changes to this specification, please use the dev-js-sourcemap mailing list: https://lists.mozilla.org/listinfo/dev-js-sourcemap

Background

The original source map format (v1) was created by Joseph Schorr for use by Closure Inspector to enable source level debugging of optimized JavaScript code (although the format itself is

language agnostic). However, as the size of the projects using the source maps expanded the verbosity of the format started to be become a problem. The v2 was created trading some simplicity and flexibility to reduce to overall size of the source map. Even with the changes made with the v2 version of the format, the source map file size was limiting its usefulness. The v3 format is based on suggestions made by podivilov.

Related documents:
Revision 2 proposal

Terminology

Term	Definition
Generated Code	The code which is generated by the compiler.
Original Source	The source code which has not been passed through the compiler.
Base 64 VLQ	The VLQ is a Base64 value, where the most significant bit (the 6th bit) is used as the continuation bit, and the "digits" are encoded into the string least significant first, and where the least significant bit of the first digit is used as the sign bit. Note: The values that can be represent by the VLQ Base64 encoded are limited to 32 bit quantities until some use case for larger values is presented.
Source Mapping URL	The URL referencing the location of a source map from the generated code.

Revision 3 Format

General Goals

- Reduce the overall size to improve parse time, memory consumption, and download time.
- Support source level debugging allowing bidirectional mapping
- Support server side stack trace deobfuscation

Proposed Format

1. `{`
2. `"version" : 3,`

```
3. "file": "out.js",
4. "sourceRoot": "",
5. "sources": ["foo.js", "bar.js"],
6. "sourcesContent": [null, null],
7. "names": ["src", "maps", "are", "fun"],
8. "mappings": "A,AAAB;;ABCDE;"
9. }
```

Line 1: The entire file is a single JSON object
Line 2: File version (always the first entry in the object) and must be a positive integer.
Line 3: An optional name of the generated code that this source map is associated with.
Line 4: An optional source root, useful for relocating source files on a server or removing repeated values in the “sources” entry. This value is prepended to the individual entries in the “source” field.
Line 5: A list of original sources used by the “mappings” entry.
Line 6: An optional list of source content, useful when the “source” can’t be hosted. The contents are listed in the same order as the sources in line 5. “null” may be used if some original sources should be retrieved by name.
Line 7: A list of symbol names used by the “mappings” entry.
Line 8: A string with the encoded mapping data.

The “mappings” data is broken down as follows:

- each group representing a line in the generated file is separated by a ”;”
- each segment is separated by a “,”
- each segment is made up of 1,4 or 5 variable length fields.

The fields in each segment are:

1. The zero-based starting column of the line in the generated code that the segment represents. If this is the first field of the first segment, or the first segment following a new generated line (“;”), then this field holds the whole base 64 VLQ. Otherwise, this field contains a base 64 VLQ that is relative to the previous occurrence of this field. *Note that this is different than the fields below because the previous value is reset after every generated line.*
2. If present, an zero-based index into the “sources” list. This field is a base 64 VLQ relative to the previous occurrence of this field, unless this is the first occurrence of this field, in which case the whole value is represented.
3. If present, the zero-based starting line in the original source represented. This field is a base 64 VLQ relative to the previous occurrence of this field, unless this is the first occurrence of this field, in which case the whole value is represented. Always present if there is a source field.
4. If present, the zero-based starting column of the line in the source represented. This field is a base 64 VLQ relative to the previous occurrence of this field, unless this is the first occurrence of this field, in which case the whole value is represented. Always present if there is a source field.

5. If present, the zero-based index into the “names” list associated with this segment. This field is a base 64 VLQ relative to the previous occurrence of this field, unless this is the first occurrence of this field, in which case the whole value is represented.

Note: This encoding reduces the source map size 50% relative to the V2 format in tests performed using Google Calendar.

Resolving Sources

If the sources are not absolute URLs after prepending of the “sourceRoot”, the sources are resolved relative to the SourceMap (like resolving script src in a html document).

Encoding

For simplicity, the character set encoding is always UTF-8.

Compression

The file is allowed to be GZIP compressed. It is not expected that in-browser consumers of the the source map will support GZIP compression directly but that they will consume an uncompressed map that may be GZIP’d for transport.

Extensions

Additional fields may be added to the top level source map provided the fields begin with the “x_” naming convention. It is expected that the extensions would be classified by the organization providing the extension, such as “x_google_linecount”. Field names outside the “x_” namespace are reserved for future revisions. It is **recommended** that fields be namespaced by domain, i.e. x_com_google_gwt_linecount.

Known Extensions

“x_google_linecount” - the number of line represented by this source map.

Notes

Using file offsets were considered but rejected in favor of using line/column data to avoid becoming misaligned with the original due to platform specific line endings.

Index map: supporting post processing

To support concatenating generated code and other common post processing, an alternate representation of a map is supported:

```
{
version : 3,
file: “app.js”,
sections: [
 { offset: {line:0, column:0}, url: “url_for_part1.map” }
 { offset: {line:100, column:10}, map:
   {
```

```
8.       version : 3,
9.       file: "section.js",
10.      sources: ["foo.js", "bar.js"],
11.      names: ["src", "maps", "are", "fun"],
12.      mappings: "AAAA,E;;ABCDE;"
13.    }
14.  }
15. ],
16. }
```

The index map follow the form of the standard map
Line 1: The entire file is an JSON object.
Line 2: The version field. See the description of the standard map.
Line 3: The name field. See the description of the standard map.
Line 4: The sections field.

The "sections" field is an array of JSON objects that itself has two fields "offset" and a source map reference. "offset" is an object with two fields, "line" and "column", that represent the offset into generated code that the referenced source map represents.
The other field must be either "url" or "map". A "url" entry must be a URL where a source map can be found for this section and the url is resolved in the same way as the "sources" fields in the standard map. A "map" entry must be an embedded complete source map object. An embedded map does not inherit any values from the containing index map.

The sections must be sorted by starting position and the represented sections may not overlap.

Conventions

Source Map Naming

Optionally, a source map will have the same name as the generated file but with a ".map" extension. For example, for "page.js" a source map named "page.js.map" would be generated.

Linking generated code to source maps

While the source map format is intended to be language and platform agnostic, it is useful to have a some conventions for the expected use-case of web server hosted javascript.

There are two suggested ways to link source maps to the output. The first requires server support to add a HTTP header and the second requires an annotation in the source.

The HTTP header should supply the source map URL reference as:

```
SourceMap: <url>
```

Note: previous revisions of this document recommended a header name of “X-SourceMap”. This is now deprecated; “SourceMap” is now expected.

The generated code may include a line at the end of the source, with the following form:

```
//# sourceMappingURL=<url>
```

Note: The prefix for this annotation was initially “//@” however this conflicts with Internet Explorer’s Conditional Compilation and was changed to “//#”. It is reasonable for tools to also accept “//@” but “//#” is preferred.

This recommendation works well for JavaScript, it is expected that other source files will have other conventions:

CSS	/*# sourceMappingURL=<url> */

Note: <url> is a URL as defined in RFC3986; in particular, characters outside the set permitted to appear in URIs must be percent-encoded.
Note: <url> maybe a data URI. Using a data URI along with “sourcesContent” allow for a completely self-contained source-map.

Regardless of the method used to retrieve the source mapping URL the same process is used to resolve it, which is as follows:
When the source mapping URL is not absolute, then it is relative to the generated code’s “source origin”. The source origin is determined by one of the following cases:

- If the generated source is not associated with a script element that has a “src” attribute and there exists a `//# sourceURL` comment in the generated code, that comment should be used to determine the source origin. Note: Previously, this was “//@ sourceURL”, as with “//@ sourceMappingURL”, it is reasonable to accept both but //# is preferred.
- If the generated code is associated with a script element and the script element has a “src” attribute, the “src” attribute of the script element will be the source origin.
- If the generated code is associated with a script element and the script element does not have a “src” attribute, then the source origin will be the page’s origin.
- If the generated code is being evaluated as a string with the `eval()` function or via `new Function()`, then the source origin will be the page’s origin.

Linking eval'd code to named generate code

There is an existing convention that should be supported for the use of source maps with eval'd code, it has the following form:

```
//@ sourceURL=foo.js
```

It is described here:
http://blog.getfirebug.com/2009/08/11/give-your-eval-a-name-with-sourceurl/

Language Neutral Stack Mapping Notes

Stack tracing mapping without knowledge of the source language is not covered by this document.

Multi-level Mapping Notes

It is getting more common to have tools generate source from some DSL (templates) or to compile CoffeeScript -> JavaScript -> minified JavaScript, resulting in multiple translations before the final source map is created. This problem can be handled in one of two ways. The easy but lossy way is to ignore the intermediate steps in the process for the purposes of debugging, the source location information from the translation is either ignored (the intermediate translation is considered the "Original Source") or the source location information is carried through (the intermediate translation hidden). The more complete way is to support multiple levels of mapping: if the Original Source also has a source map reference, the user is given the choice of using the that as well.

However, It is unclear what a "source map reference" looks like in anything other than JavaScript. More specifically, what a source map reference looks like in a language that doesn't support JavaScript style single line comments. An HTTP header would address this, but is not yet agreed upon.

JSON over HTTP Transport

XSSI attacks could potentially make source maps available to attackers by doing a direct script src to a source map after overriding the Array constructor. This can be effectively prevented by preprending a JavaScript syntax error to the start of the response.
Thus when delivering source maps over HTTP, servers may prepend a line starting with the string ")]}'" to the sourcemap. If the response starts with this string clients must ignore the first line.