Four Chrome DevTools Features Making Web Dev More Efficient

By Andi Dysart
Vanamco

http://ghostlab.vanamco.com

Intro

The sharing of information amongst the development community has always been a great conversation to be involved in when building an app like Ghostlab. In a fast paced and innovative industry like web development, tools change rapidly and many devs solve problems by discussing them in the community and sharing knowledge. So, we thought we’d share our knowledge of a four Chrome DevTools that may make work lives of web devs and designers a little bit easier. We’re going to revisit the everchanging features surrounding Markup and Style, Networking and the Console and JavaScript and performance.

TL;DR

DevTools grew rapidly, and with it came some exciting new features you'll want to be aware of:

Screencasting: use your Android device to view and interact with web pages via DevTools.

Workspaces: utilise the Sources panel code editor to edit code in your project and have that persist to disk.

Source Map debugging: writing LESS/Sass or even the CoffeeScript? Understand the mapping by outputting Source Maps during compilation.

*Flame Chart: *view javascript processing over time with an interactive
visualisation

Mobile Emulation: test your pages' responses to certain devices characteristics by emulating screen dimensions, user agents and other device related
attributes.

Canvas Debugging: see your Canvas change with each instruction by stepping through the commands HTML5 canvas receives.

So, what's new?

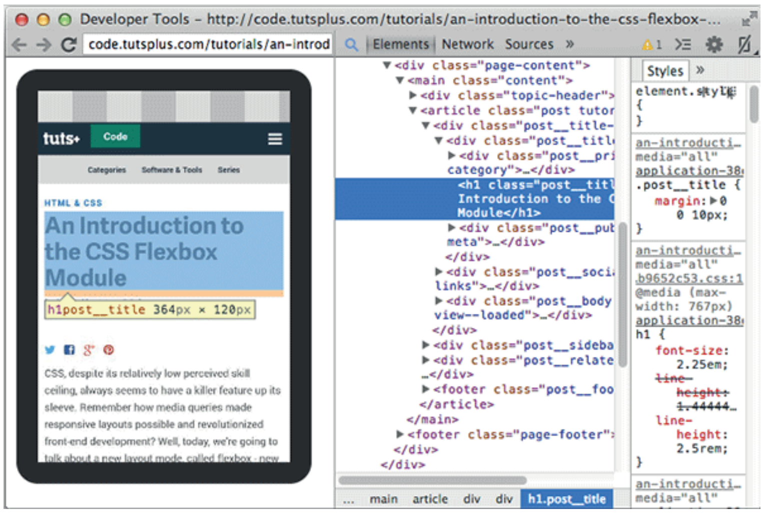
DevTools continues to improve development and debugging workflow issues. From source map debugging right through to remote debugging, there's hopefully at least one thing in this post you may begin to use in your work. The following features can be **tried with Chrome Canary**, go ahead and give it a try.

Screencasting

Screen Cast with Chrome DevTools for Mobile: **Watch Here**

What does it do?

Using Chrome for Android, you'll now possess the ability to interact with a webpage, from your desktop machine. Using your mouse, you can now interact with clicks, swipes and scrolling.


Developer Tools - http://code.tutsplus.com/tutorials/an-introduction-to-the-css-flexbox-...
code.tutsplus.com/tutorials/an-introd
Elements
Network
Sources
Styles
tuts+
Code
Categories
Software & Tools
Series
HTML & CSS
An Introduction to the CSS Flexbox Module
h1post__title 364px × 120px
... main article div div h1.post__title

CSS, despite its relatively low perceived skill ceiling, always seems to have a killer feature up its sleeve. Remember how media queries made responsive layouts possible and revolutionized front-end development? Well, today, we're going to

```
▼<div class="page-content">
  ▼<main class="content">
    ▶<div class="topic-header">
    ▼<article class="post tutor
      ▼<div class="post__title-
        ▼<div class="post__titl
          ▶<div class="post__pri
          category">…</div>
            <h1 class="post__titl
            Introduction to the C
            Module</h1>
          ▶<div class="post__pub
          meta">…</div>
          </div>
        ▶<div class="post__soci
        links">…</div>
        ▶<div class="post__body
        view--loaded">…</div>
        </div>
      ▶<div class="post__sideba
      ▶<div class="post__relate
      …</div>
      ▶<footer class="post__foo
      </article>
    </main>
  ▶<footer class="page-footer">
  </div>
</div>
```

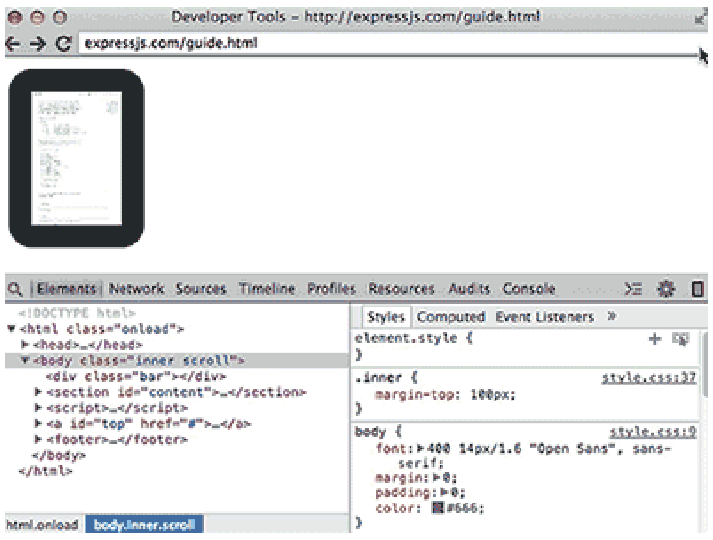
```
element.style
{
}
an-introducti…
media="all"
application-38
.post__title {
  margin:▶0
    0 10px;
}
an-introducti…
media="all"
.b9652c53.css:1
@media (max-
width: 767px)
application-38
h1 {
  font-size:
    2.25em;
  ~~line-
    height:
    1.44444…~~
  line-
    height:
    2.5rem;
}
an-introducti…
media="all"
```

1. Click the hotdog icon in the top right corner of Chrome.
2. Go to Tools > Inspect Devices. You can also enter the shortcut chrome://inspect/#devices into your address bar.
3. Ensure your devices is recognised in the list.
4. After the device is recognised, enter a URL you wish to remotely debug in the "Open Tab with URL" input box. Select open.
5. The page should be now loading up in Chrome for Android.
6. Hit Inspect back on the chrome://inspect page and note a small Screen icon in the top right corner of the DevTools. Clicking this will enable Screencasting.

The screencast is updated in real-time. In scrolling on the page, you will notice your device scrolls also. In the following screenshot, the Inspect Element feature behaves just as you would expect. For examples, feature such as typing and gestures enable you to move significant parts of your device in testing workflow over to the DevTools.

By Andi Dysart
Vanamco

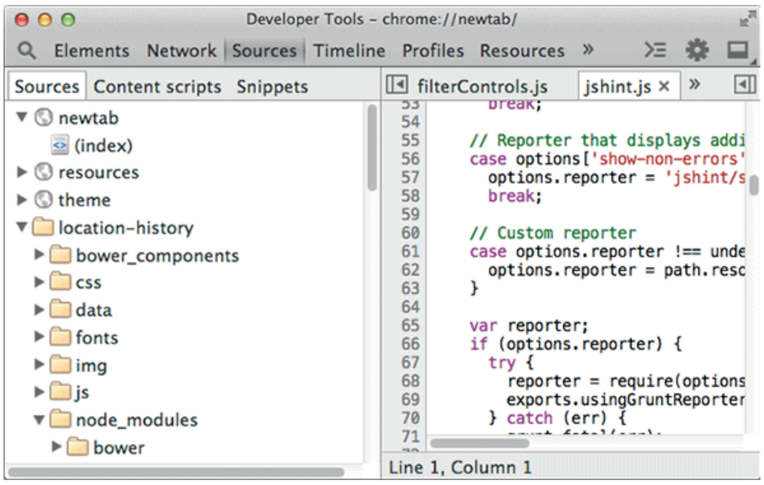
Click here to check out the official documentation for **Screencasting on your device's screen.**



```
<!DOCTYPE html>
▼<html class="onload">
  ▶<head>…</head>
  ▼<body class="inner scroll">
      <div class="bar"></div>
    ▶<section id="content">…</section>
    ▶<script>…</script>
    ▶<a id="top" href="#">…</a>
    ▶<footer>…</footer>
    </body>
</html>
```

```
element.style {
}
.inner {                                   style.css:37
  margin-top: 100px;
}
body {                                     style.css:9
  font:▶400 14px/1.6 "Open Sans", sans-
      serif;
  margin:▶0;
  padding:▶0;
  color: ■#666;
}
```

Workspaces



```
    break;

  // Reporter that displays addi
  case options['show-non-errors'
    options.reporter = 'jshint/s
    break;

  // Custom reporter
  case options.reporter !== unde
    options.reporter = path.reso
  }

  var reporter;
  if (options.reporter) {
    try {
      reporter = require(options
      exports.usingGruntReporter
    } catch (err) {
```

What does it do?

Essentially, you're able to utilize the Sources panel to display the contents of your project source files, simply as they are on your system. The files are able to save-to-disk and is editable. Not only can you can edit just like using a text editor, but you can synchronise changes to them whether you edit styles in the Elements panel or in an external editor like DevTools.

By Andi Dysart
Vanamco

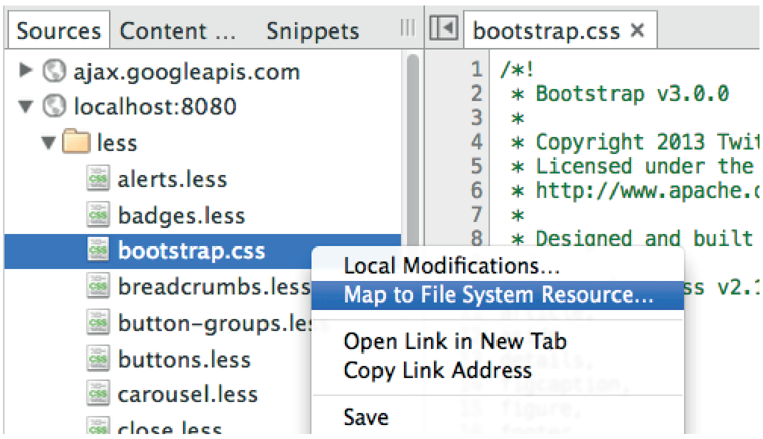
ghostlab
http://ghostlab.vanamco.com

Preparing Workspaces:

1. Add a folder to your workspace by right-clicking in the Sources pane and select ‘Add folder to Workspace’.
2. Grant the file system permission requested by Chrome.

You should now be able to edit your code within DevTools and save just as you would within a code editor. An additional benefit comes from editing code which is being shared by the webpage you are viewing. Serving pages up at http://localhost:3000/ means you can add the corresponding project as a Workspace Folder in DevTools. At this point, you’ll want to teach DevTools about the mapping between the Network resource and the correlating file system resource.

1. In the Sources panel, right-click on a file within a network resourced folder. Ensure it has an obvious mapping to a file within your file system).
2. Select Map to File System Resource.



```
/*!
 * Bootstrap v3.0.0
 *
 * Copyright 2013 Twit
 * Licensed under the
 * http://www.apache.c
 *
 * Designed and built
```

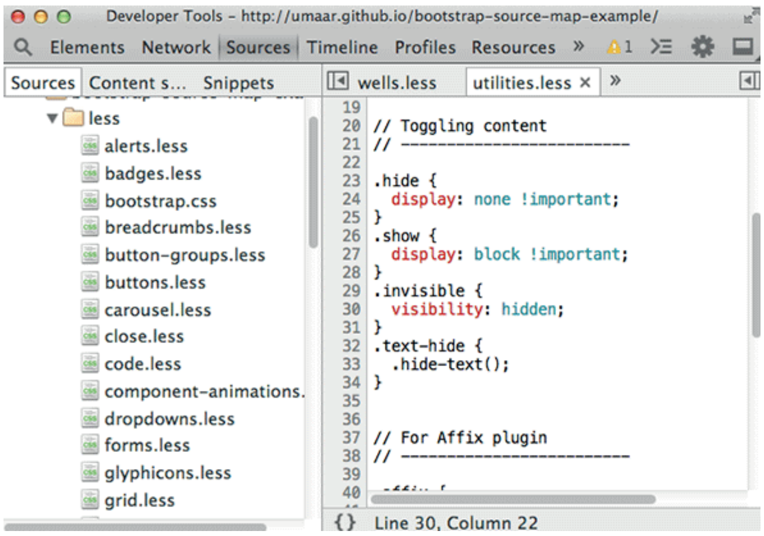
Your project source files will display instead of the network resource files once DevTools has been instructed of the mapping.

There are a few additional features to take note of also:

- Most files come with syntax highlighting
- Use Cmd+P to quickly open a file, if you see undesirable results such as files in node_modules, you can simple right-click on the containing folder and select 'Exclude Folder'.
- By going to Context menu > 'New File' you can create new files.
- DevTools will automatically attempt to reload changes from a file updated else where whenever it regains focus.

For more information, head over to the **DevTools Documentation over at Google.**

Source Map Debugging

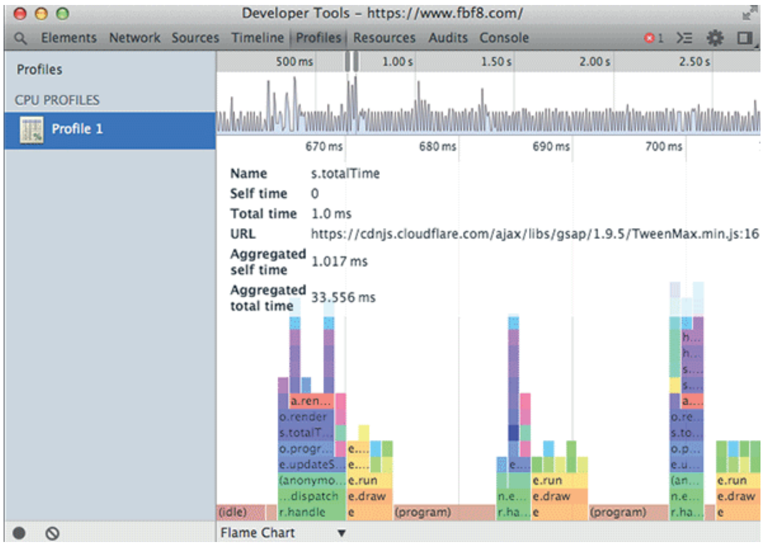


```
// Toggling content
// -------------------------

.hide {
  display: none !important;
}
.show {
  display: block !important;
}
.invisible {
  visibility: hidden;
}
.text-hide {
  .hide-text();
}


// For Affix plugin
// -------------------------
```

There are a few tools that produce source map files through their compilation process, such as Sass, LESS and CoffeeScript. By interpreting such files, DevTools is able to provide you live-edie abilities. Now, within the Source panel, you'll be able to edit your applicable source files (.less, .scss, .coffee). DevTools will automatically reload from a file on change and update the web page accordingly, only if you have a watch task on your preprocessed files (which re-compiles a new CSS/JS file upon change).

By Andi Dysart
Vanamco

Flame Charts


Developer Tools – https://www.fbf8.com/
Elements Network Sources Timeline Profiles Resources Audits Console
Profiles
CPU PROFILES
Profile 1
500 ms 1.00 s 1.50 s 2.00 s 2.50 s
670 ms 680 ms 690 ms 700 ms
a.ren...
o.render
s.totalT...
o.progr...
e.updateS...
(anonymo...
...dispatch
r.handle
e.run
e.draw
(idle)
(program)
Flame Chart

Name	s.totalTime
Self time	0
Total time	1.0 ms
URL	https://cdnjs.cloudflare.com/ajax/libs/gsap/1.9.5/TweenMax.min.js:16
Aggregated self time	1.017 ms
Aggregated total time	33.556 ms

What does it do?

By using Flame Chart, you are able to receive a visual overview of the Javascript activity over a time.
How?

1. Create and view a flame chart visualisation by accessing the Profiles tab and collect ing a JavaScript preview. Simply select the Chart view on the profile result.
2. By dragging and selecting a portion of interest, you can zoom in on spikes in the graph.
3. Selecting a range will result in the flame chart being updated, at which point you can drag the flames horizontally or vertically to pan around.

Understanding the Flame Chart

It is not significant to consider the height of all bars and it should be noted that it simply represents each functional call that occurred. It is in fact the width that is crucial, as the length is related to the time that function took to execute.

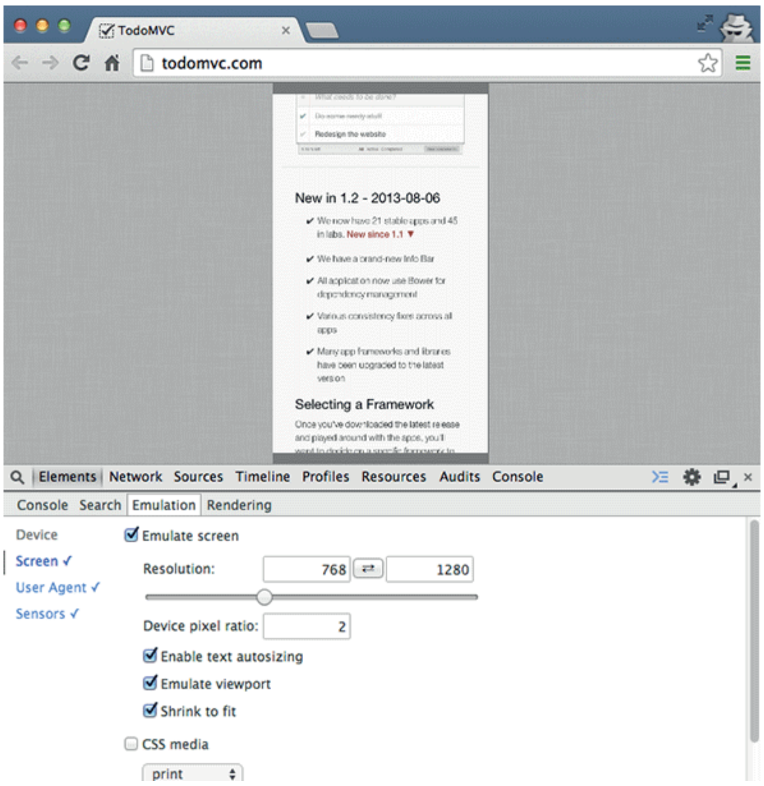
By Andi Dysart
Vanamco

Each bar can be governed over to bring up timing details and you should begin to address potential performance bottlenecks:

1. Find a wide bar in the flame chart.
2. Hover over it to reveal the details section.
3. Ensure the ‘Total Time’ is at a satisfactory amount.

A bar which has a high execution time can be slowing your web page down, click on it to be taken to the corresponding point in code.

Mobile Emulation



☑ Emulate screen

Resolution: 768 ⇄ 1280

Device pixel ratio: 2

☑ Enable text autosizing

☑ Emulate viewport

☑ Shrink to fit

☐ CSS media

print

Chrome is able to imitate certain characteristics of mobiles device by enabling mobile emulation. Controls devices like Nexus 5, Galaxy S4, iPhones and Kindle, amongst many more, by individually controlling or utilizing a set of fixed presets.

Immediately you can apply a device preset to see how the inspected webpage behaves under mobile conditions. Here's how:

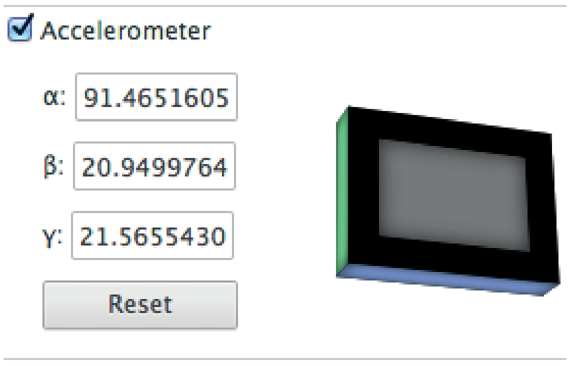
1. Hit Escape in the open DevTools drawer.
2. Navigate to the Emulation panel
3. Select *DEVICE*

Ensure the device setting have been applied correctly by refreshing your page. You will now notice the visible page area is much smaller, corresponding now to the aspect ration of the device present.

What Else Has Changed?

- User Agent: typing navigator.userAgent in the console will not output the emulate device user agent, for example: "Mozilla/5.0 (Linux; Android 4.2.1; en-us; Nexus 5 Build/JOP04D) Apple Web Kit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.166 Mobile Safari/535.19"
- Screen resolution: Screen dimensions are now reported as your would receive the information from the device itself. Ensure you understand the pixel ratio of the device you are working with, as many devices are not standardised.
- *devicePixelRatio: *window.devicePixelRatio report 2 which will enable a media query target at min-device-pixel-ratio: 2.
- Touch Events: "Clicks" you carry out are translated into the usual touchend and other events, via the standard touch start. Quick tip: listen to touch events using monitorEvents(window, "touchstart"); found in the Console panel.

There are also some emulation options available which can be enabled regardless of a device present. Using the Sensors pane in the Emulation panel contains an accelerometer which can be dragged around.



Accelerometer ✓

α: 91.4651605

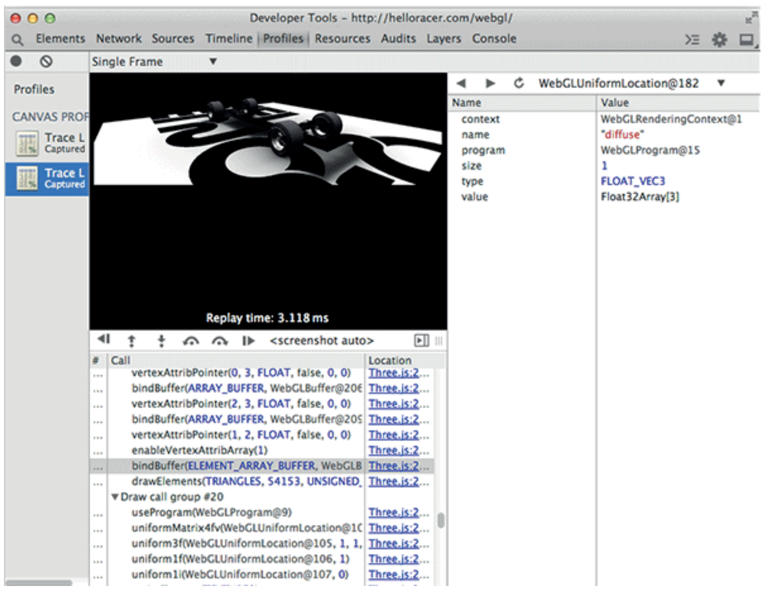
β: 20.9499764

γ: 21.5655430

This will be most useful when the inspected webpage make use of the navigator.geolocation methods when customising the report latitude and longitude which also is within the Sensors pane.

Once more, Google has a more detailed look at Emulation options in the **official Mobile Emulation Documentation.**

Canvas Debugging



Name	Value
context	WebGLRenderingContext@1
name	"diffuse"
program	WebGLProgram@15
size	1
type	FLOAT_VEC3
value	Float32Array[3]

#	Call	Location
...	vertexAttribPointer(0, 3, FLOAT, false, 0, 0)	Three.js:2...
...	bindBuffer(ARRAY_BUFFER, WebGLBuffer@206	Three.js:2...
...	vertexAttribPointer(2, 3, FLOAT, false, 0, 0)	Three.js:2...
...	bindBuffer(ARRAY_BUFFER, WebGLBuffer@209	Three.js:2...
...	vertexAttribPointer(1, 2, FLOAT, false, 0, 0)	Three.js:2...
...	enableVertexAttribArray(1)	Three.js:2...
...	bindBuffer(ELEMENT_ARRAY_BUFFER, WebGLB	Three.js:2...
...	drawElements(TRIANGLES, 54153, UNSIGNED_	Three.js:2...
	▼Draw call group #20	
...	useProgram(WebGLProgram@9)	Three.js:2...
...	uniformMatrix4fv(WebGLUniformLocation@10	Three.js:2...
...	uniform3f(WebGLUniformLocation@105, 1, 1,	Three.js:2...
...	uniform1f(WebGLUniformLocation@106, 1)	Three.js:2...
...	uniform1i(WebGLUniformLocation@107, 0)	Three.js:2...

Canvas inspection is a unique, experimental feature within DevTools. Essentially, Canvas Inspection is a profile type, so you'll find it under the Profiles pane, which is where you'll be able to enable it. Simply, DevTools > Setting > Experiments.

But What does Canvas Profiler Do?

The canvas profile allows you to step through instructions sent to the canvas itself, while viewing the current canvas property states as it collects instructions, displaying a visual representation of the canvas at a particular point in time.

After you have recorded a profile, you may explore the top level draw call groups (or, items) and discover individual draw calls. To the right of each draw call, take note of the file:linenumber reference. By clicking on this, you will navigate to the Sources panel with the relevant line of code selected. You can use the toolbar to easily navigate through individual drawl calls or subsequent draw call groups. Also note, there is a dropdown that allows you to specify the canvas context to inspect the properties of it, in the event there is more than one.

Our friends at HTML5 have a fantastic, detailed guide, see: **Canvas Inspection using Chrome Dev Tools.**

By Andi Dysart
Vanamco

A note from Ghostlab support

As we have developed Ghostlab and want to continue to provide our customers with the highest level of support, we want to encourage you to get in contact with our support team to find out how to best utilize Ghostlab within DevTools. Our team have a broad range of knowledge and solutions around workflow. By understanding how you work, we can make suggestions based on your needs and delivery. Find us on **Twitter** and **Facebook**, we'd love your thoughts, questions and ideas and what you think of these tools.

By Andi Dysart
Vanamco