A Firebug Tutorial

This is an overview of the plugin, not a detailed explanation of all of it’s amazing features. This should be enough to get you started.

Installing Firebug

Firebug works with Firefox. There is a product called Firebug lite which you can include in a file via a javascript call in your file, for use in non-Firefox browsers, but I am not covering Firebug lite in this tutorial/overview.

To install Firebug visit the Firebug download page. Click on the huge orange button half-way down the page on the right hand side. You can also download it from Mozilla’s FireFox Add-ons site. Install it. Restart FireFox, and you’re good to go.

If you’ve already installed it, get the latest version! To update your extension, in Firefox select Tools > Add-ons. Then click on the button in the left hand bottom corner that reads “Find Updates”.

Opening and Closing Firebug

Keyboard and Mouse Shortcuts for Firebug can be found at the Firebug Website. The three sets I use most often include:

- **Open Firebug**: F12 or clicking on the ✓ at the right of the right of the browser status bar.
- **Close Firebug**: 12 or clicking on the ✓ at the right of the right of the browser status bar or clicking on the ✖ in the upper right hand corner of the firebug window.
- **Undock Firebug into its own window:** click on the ▲ in the top right corner of the firebug window or Ctrl+F12 / ⌘+F12.

Firebug settings

- **Set Firebug to always undock in a separate window:** open the firebug console. right click on the firebug in the upper left, select “options”, then select “Always Open in New Window”
- **Increase/Decrease font size:** open the firebug console. right click on the firebug in the upper left, select “Text Size”. The font changes are really slight, but it does work.
- **Set Firebug to only work for specified sites:** Start by disabling Firebug by right clicking on the ✓ and selecting “disable Firebug”. Then right click on the greyed out icon and add the domains you want to enable Firebug on.

Firebug Window Overview

- **Console Tab:** Contains command line javascript, shows javascript error message log, can enter JavaSript commands after the >>> at the bottom
- **HTML Tab:** Shows HTML as an indented hierachy of DOM nodes, which you can open and close to see or hide child nodes.
- **CSS Tab:** CSS inspector, view all loaded style sheets, modify styles on the fly. See list of styles sheets and select one to view from this pane by clicking on the drop down list on the top of Firebug window, to the right of "Edit".
- **Script Tab:** Shows the javascript files and the calling document. See list of included JavaScript files and select one to view from this pane by clicking on the drop down list on the top of Firebug window, to the right of "Inspect". Set breakpoints and conditions underwhich break points appear.
- **DOM Tab:** Shows all the page objects and properties of the window. As variables are properties of the window object, Firebug displays all JavaScript variables and their values.
- **Net Tab:** Shows all the downloads, how long each resource took to download, the HTTP request headers and server response sent for each resource. The XHR tab is useful for AJAX debugging.

Edit your document on-the-fly.

You can change text nodes by selecting the text node with the "inspect" function and then clicking on the text node in the HTML panel.

Firebug is both an inspector and an editor. All objects in the HTML, CSS and JavaScript files can be edited with a single or double click. As you type, the changes are immediately applied in the browser window providing instant feedback. The DOM inspector allows for full in-place editing of your document structure, not just text nodes. Simply click on the "edit" tab next to the "inspect tab", and the left panel becomes a black and white text editor. In the CSS panel, Firebug autocompletes as you type. In the DOM inspector, Firebug autocompletes property names when you hit the Tab key.

Inspecting and troubleshooting CSS using Firebug

DOM inspector's CSS tab reveals all applicable CSS rules for elements, including properties inherited from ancestor elements, lets you toggle on/off and edit individual style declarations and add new CSS rules.

Firebug hides rules that Firefox ignores: For example, Firebug will hide hacks targeting other browsers and CSS3 selectors that it doesn't support. So, it will both hide intentional CSS filters, such as _height:25px; (the underscore is an IE6 hack) and p:first-of-type {color: #ff0000;} (a CSS3 :first-of-type pseudo-class selector currently supported only by Safari 3). This means, however, that if the reason your page isn't looking the way you anticipated is because of a typo, you will need to use the developer toolbar to show all CSS and find your error.

Firebug shows all selectors impacting the selected element: When you inspect an element or node, Firebug displays the entire cascade of all the CSS attributes impacting that element, including attributes that were overwritten in the cascade. The right panel displays the selectors and attributes impacting the element in order of the cascade.

Firebug allows you to turn off styles impacting an element within the CSS: When you turn off an attribute, if that attribute value was overriding a different value in the cascade, the formerly crossed out value will become active so you can test what the page would look like if the attribute value were removed. To "turn off" an attribute, click to the left of the attribute in the right hand panel. A red "do not" icon 🚫 will appear, and the attribute will be grayed out or disappear, and the strike-through of the new attribute value effecting the element from the cascade will be removed. You can toggle the attribute value back on by clicking on the ⊘. However, if the attribute has "disappeared" since it's been overwritten, you have to re-inspect the element to see the missing attribute and toggle it back on.

Firebug allows you to edit attributes and attribute values: To change an attribute or the value of an attribute, click on the term or value, and edit it. Pretty simple! The effect of the change will be immediately visible in the browser window.

One of the best components of this feature is figuring out exact positioning, padding and margins. Firebug provides wonderful support for changing numeric values. Simply click on the numeric value you want to change and change it with the numbers on your keyboard, or click on the up or down arrow to increment or decrement the value by one.

Firebug allows you to add new attribute / value pairs to existing selectors: To add an attribute to a selector, double click on the selector. An input box with intelli-sense will appear. Hitting tab will bring you to a value text input box.

Firebug allows you to easily inspect ancestor elements: To the right of the keyterm "inspect", Firebug displays all the ancestors of the currently selected element. To inspect an ancestor, simply click on it in the list. The left and right panels will both change to display the properties of the newly selected element.

The Box Model: Evaluating Height, Width, Padding, Border and Margin

When you inspect an element, the left panel displays the HTML, and the right panel shows the CSS. Next to the CSS label at the top of the right panel is a tab labeled "Layout". This Layout tab illustrates the CSS box model as it applies to the selected element. To view the height or width of any other element on the page, simply select "inspect" while this window is open and hover your mouse over the inline or block level element you wish to inspect.

Evaluating download speed

The Net tab graphs the request times for all http requests that make up a page. Network monitoring must be turned on for the Net Tab to work, but it's on by default (the options drop down on the right hand side of the firebug panel allows you to toggle this feature off and on). This is a handy way to test (and prove) that putting your javascript files at the bottom of your files, especially for javascript heavy pages, improves perceived download time. Total download time will be the same, but since browsers stop loading pages when they hit scripts until the scripts have been downloaded, the page will appear to download faster, which is a better user experience (see list item 5).

By left clicking on the plus sign to the left of the file name, Firebug displays the **HTTP headers** of each file that was downloaded.

In the 1.0..5 version of Firebug you can see how long it takes for HTML files only to be downloaded, CSS files only, images only, etc., or all files. YSlow, Yahoo!s performance tool, should be available as a plug in for Firefox by the end of the summer. In the meantime, Firebug's Net tab is the next best thing.

JavaScript

The DOM inspector provides information about all the properties of all the objects in your document. The coolest feature of Firebug is that while making dynamic updates to your page will not alter the content visible in the "view source" feature of the browser, it will update the content in the Firebug panels. So, dynamically created content is visible not only in the browser (as it should be), but the generated code is inspectible (is that a word?) in Firebug.

The JavaScript profiler reports on the execution times of your JavaScript functions, so you can pin down performance JavaScript components that are slowing down your pages or javascript applications. To view the profiler, the submenu should be on “console”, and click on “Profile” in the top menu (the order of the tabs is “Inspect |Clear | Profile”). Firebug lists all the functions that were called and the time spent for each function call. You can target what function you want Firebug to profile by adding console.profile([title]) to the start of the section your want profiled, and console.profileEnd() at the end.

The command line at the bottom of the console tab, which starts with “>>>” accepts commands in JavaScript. The results or your javascript command, if there are any, are then displayed in the console. There is a Firebug Command Line API that is worth taking a look at.

AJAX

As mentioned above, Firebug captures dynamic content and other DOM changes made to your webpage. If you take as an example the sample file created in my tutorial Beginning AJAX using the YUI Library, if you check the view source after calling the function (click on the link) via the browser’s functionality, you’ll only get the link that calls the AJAX function. If you view the source via Firebug, Firebug reflects changes made to the DOM, and includes the “Hello World” content. This is one of the core strengths of Firebug: without Firebug, AJAX requests and responses are invisible. With it, you can view the sent and received text as well as the headers that went along with it. You can also monitor how long it took for each request/response with the Net tab.

You can see XHR response by clicking on the Net tab, and then the XHR tab in the sub navigation that opens up when you are in the Net view. While the All Net tab is like yslow, the XHR focuses on the AJAX only. If you click on the plus sign to the left of the response file, you can see both the headers and the content of the response.

Details of the Net tab’s XHR sub-tab

When a request is made to the server via a XMLHttpRequest object, Firebug logs the POST or GET request, the response headers and the raw text of the response. To view this data, click on the Net Tab’s XHR sub-tab. This will show a list of the calls and the time it took for the response. To the left of the request click on the + or simply click on the request (it is a link). Three tabs will appear in the case of a GET request, four for a POST:

- **Params:** Displays the name/value pairs of the request URL
- **Headers:** Displays requestion and response headers
- **Response:** Shows the actual data received from the server as it was received.
- **Post.** Displays the data sent to the server from a POST request (tab only shows for POST requests, not GET requests).

These four tabs are useful in development and debugging. Check the POST and Params tabs to ensure your request are being correctly posted. Check the Response to determine the format of the response and ensure that your JavaScripts are written to handle that formatting. If you don’t control the feed that you are fetching, you can copy and paste the response into a text editor, format it so it’s human legible, and work that way.

Debugging AJAX and other JavaScripts

To debug AJAX, it’s similar to how you debug JavaScript above, except for the console.log is even more helpful, since you are reloading components of the page and not the whole page (reloading the whole page resets the HTML, JS and CSS to what is defined in the static files). Firebug provides the console object that has several methods usable for logging. Properties of console object include `console.debug`, `console.info`, `console.warning`, and `console.error`. When one of these methods produces an output, Firebug links to the line causing the output so you can quickly find the responsible code.

One of the other ways to debug is to set breakpoints (yay! no more alerts!). The “Script” tab allows you to pause execution on any line. Clicking on the line number in the script tab sets a breakpoint. You can make that breakpoint conditional by setting a statement that if true will trigger the breakpoint. Simply right click to the left of the line number to set an expression (where you had left clicked to set the breakpoint) or right click on the line and select “Edit breakpoint condition”: “this breakpoint will stop only if this expression is true” is the message you should see. You can contine, step over, step into or step out of a line. The right panel has a “watch” tab where you can inspect the value of local variables (click on objects to view their properties). Hovering over an object in the script activates a tooltip that displays the object’s properties.

Problems with Firebug

Webpages never finish loading: likely because there is a javascript error in the page, and Firebug has put a breakpoint at the error and halted the script.

Firebug crashes my Firefox some of the heavier Web 2.0 sites that I visit, like Netflix and Yahoo! Mail. Firebug keeps track of all the errors. This can use up a lot of memory and crash the browser. Also, likely irrelevant, but SoThink SWF makes Firebug crash FF.

Solutions:

Disable Firebug for a particular domain: Unless you develop for a site that crashes your browser, you can simply turn off Firebug for a domain/subdomain.. To turn off Firebug, right click on the ✅ (or ❌ if there are errors) in the status bar and select “disable firebug for thisserversname.com”.

Disable Network Monitoring: Sometimes disabling Firebug for a particular domain is not an option . For example, I code a small part of a much larger internet application which throws errors and therefore never finishes loading, but disabling Firebug is not an option since I am working on that page.. To resolve this issue, click on the Net tab. When on that tab, to the far right you will see “options”. Select “Disable Network Monitoring”. Since I have selected that, I no longer crash my Firefox, and can work on the website I need to work on. (I did update my Firebug, and that might have helped the issue too. Not sure.)

Update Your Version of Firebug: Firebug is continuously being updated and bugs are being fixed. To update to the latest version of Firebug, in your FF browser window select Tools > Add-Ons > Find Updates, and FF will check for and install upon request, the most recent release of Firebug and all of your other FF extensions.

Notes:

- Firebug 1.05 and earlier do not work with Firefox 3.0, currently in beta
- Joe Hewitt developed Firebug and offers it for free. Consider giving him some love in the form of money.
- For more in depth view, view Joe Hewitt: Welcome to Firebug which I included below. The above tutorial is more of what you definitely need to know. The video covers the more advanced features. Take a look at it if my overview doesn’t provide enough info, especially in the area of JS debugging and XHR debugging.

[Tutorial] Como usar o Firebug?

por gccsistemas em 06 Jun 2009 10:27

Aqui vai o básico com o tempo vamos aperfeiçoando o tutorial

Primeiramente, para quem não sabe, o Firebug é um plugin do Firefox desenvolvido para auxiliar e depurar o desenvolvimento de aplicações web. O plugin possui algumas funções interessantes para quem programa em JavaScript pois ele é capaz de depurar seu código lhe retornando erros, sucessos, requisições ajax feitas, tempo gasto, manipulaçao de DOM, etc…

Efetuando download do firebug

Seguem duas formas para se fazer o download do plugin:
1a) Acesse o seguinte endereço em seu firefox: http://getfirebug.com
2a) Digite "firebug" na aba de endereço do firefox e será encaminhado a página do addon.

Ativando o Firebug

Com o Firebug já instalado, reinicie o Firefox. Vá em Ferramentas » Complementos » Ative o addon (plugin) Firebug.

firebug-1.gif

Habilitando o debug do Firebug

Acesse o site em que quer testar os recursos do firebug. Vá em Ferramentas » Firebug » Abrir Firebug. Em seguida, na janela que foi aberta, marque Console, Script, Rede e clique em Habilitar.

firebug-21.gif

Pronto, agora o Firebug foi instalado, habilitado e está pronto para ser usado. Perceba que na janela do Firebug, você pode observar depurações para: Console, Html, CSS, Script, Dom e Rede. Faça bom proveito dessa excelente ferramenta!

Fonte: http://www.joaopedrobarros.com.br/

Em breve estarei explicando a ideologia do firebug!

Firebug Tutorial – Logging, Profiling and CommandLine (Part I)

9 Sep, 2007 Firebug, Firefox

Firebug Tutorial

Section 1: Console Tab : Logging, Profiling and CommandLine (Part II)

Overview of Console Tab

This tab is mainly used for logging. It also can be used as CommandLine window (like immediate window in Microsoft Visual Studio) while you are debugging the Javascript. It can be used for monitoring the execution of Javascript code by using Profiling service.

The following topic will be covered in this section.

- Logging in Firebug (with String Substitution pattern)
- Grouping the logs or messages
- console.dir and console.dirxml
- Assertion (console.assert())
- Tracing (console.trace())
- Timing (Measuring the time of your code)
- Javascript Profiler (An introduction in this tutorial, the details will be covered in next tutorial.)

#1. Logging in Firebug

Firebug supports logging in Console tab. So, you don’t need to use alert(’something’) or document.write(’something’) anymore.

There are five types of logging in Firebug.

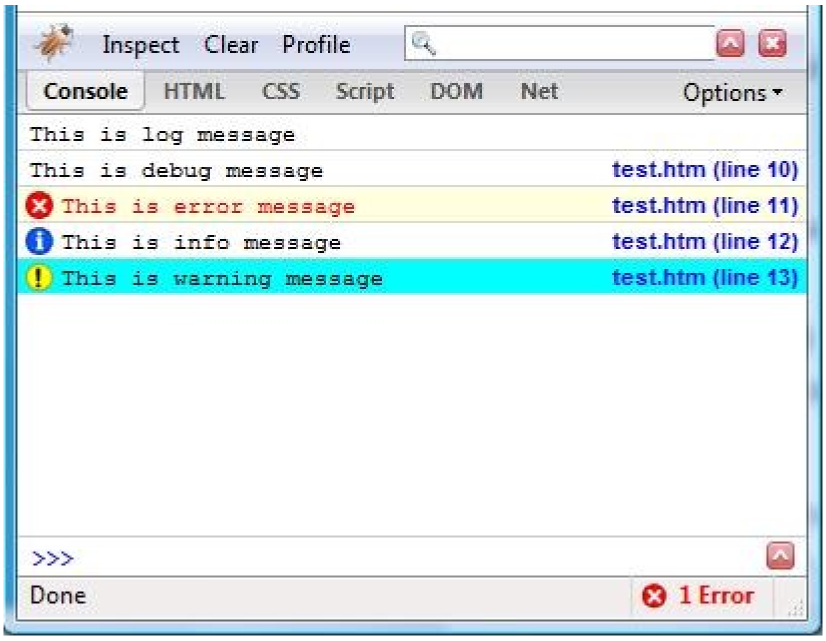
- console.log : Write a message without icon.
- console.debug : Writes a message to the console, including a hyperlink to the line where it was called
- console.error() : Writes a message to the console with the visual “error” icon and color coding and a hyperlink to the line where it was called.
- console.info() : Writes a message to the console with the visual “info” icon and color coding and a hyperlink to the line where it was called.
- console.warn() : Writes a message to the console with the visual “warning” icon and color coding and a hyperlink to the line where it was called.

Example Code:

- Open the htm file called “Plain HTML” or create one HTML file.
- Paste the following code with <body> tag.

```
<script language="javascript" type="text/javascript">
console.log('This is log message');
console.debug('This is debug message');
console.error('This is error message');
console.info('This is info message');
console.warn('This is warning message');
</script>
```

You will get the following output. If you click on hyperlink (”test.htm” in this case), it will take you to script tab and will highlight the line that wrote this message.



```
This is log message
This is debug message                test.htm (line 10)
This is error message                test.htm (line 11)
This is info message                 test.htm (line 12)
This is warning message              test.htm (line 13)

>>>
Done                                 1 Error
```

String Substitution Patterns

String substitution parterns can be used in console.log, console.info, console.debug, console.warn and console.error . You can use the same way that we used in C/C++.

%s	String
%d, %i	Integer (numeric formatting is not yet supported)
%f	Floating point number (numeric formatting is not yet supported)
%o	Object hyperlink

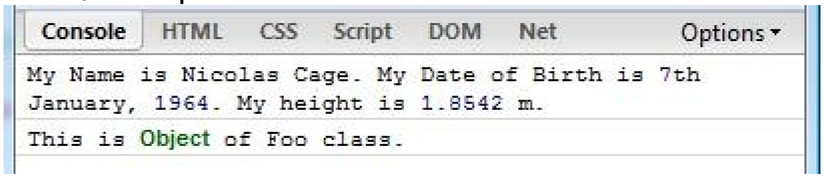
Example :

Note: I will use console.log in the example below even all console objects (console.log, console.info, console.debug, console.warn and console.error) support string substitution.

- Remove "script" tag that we pasted for the previous example.
- Paste the code below within <body> tag.

```
<script language="javascript" type="text/javascript">

//This is for normal string substitution " %s, %d, %i, %f".
console.log("My Name is <strong>%s</strong>. My Date of Birth is <strong>%dth %s, %i</strong>. My height is <strong>%f</strong> m.", "Nicolas Cage", 7, 'January', 1964, 1.8542);

function Foo(){
this.LeftHand = function(){
return "Left Hand";
}
this.RightHand = function(){
return "Right Hand";
}
}

//This is for object "%o".
var objFoo = new Foo();
console.log('This is <strong>%o</strong> of Foo class.', objFoo);
```

```

</script>
```



```
My Name is Nicolas Cage. My Date of Birth is 7th
January, 1964. My height is 1.8542 m.
This is Object of Foo class.
```

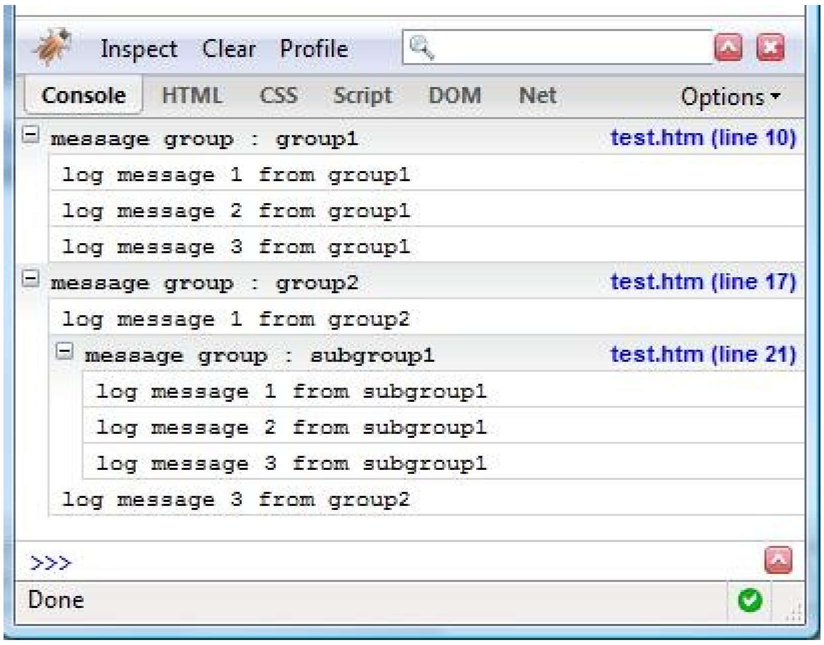
If you are using **%o** in your log, the object will be shown as a hyperlink in green color. This hyperlink is linked to the DOM tab. So, If you click “object” in second line, you will see the list of properties of that object (LeftHand and RightHand in this case.)

#2. Grouping

Firebug allows you to group the message or log in Console tab. If you have some many logs in your code, you can probably divide your log into small group or subgroup

Example ~

```
<script language="javascript" type="text/javascript">

var groupname = 'group1';
console.group("message group : %s " , groupname);
console.log("log message 1 from %s", groupname);
console.log("log message 2 from %s", groupname);
console.log("log message 3 from %s", groupname);
console.groupEnd();

groupname = 'group2';
console.group("message group : %s " , groupname);
console.log("log message 1 from %s", groupname);

var subgroupname = 'subgroup1';
console.group("message group : %s " , subgroupname);
console.log("log message 1 from %s", subgroupname);
console.log("log message 2 from %s", subgroupname);
console.log("log message 3 from %s", subgroupname);
console.groupEnd();

console.log("log message 3 from %s", groupname);
console.groupEnd();

</script>
```

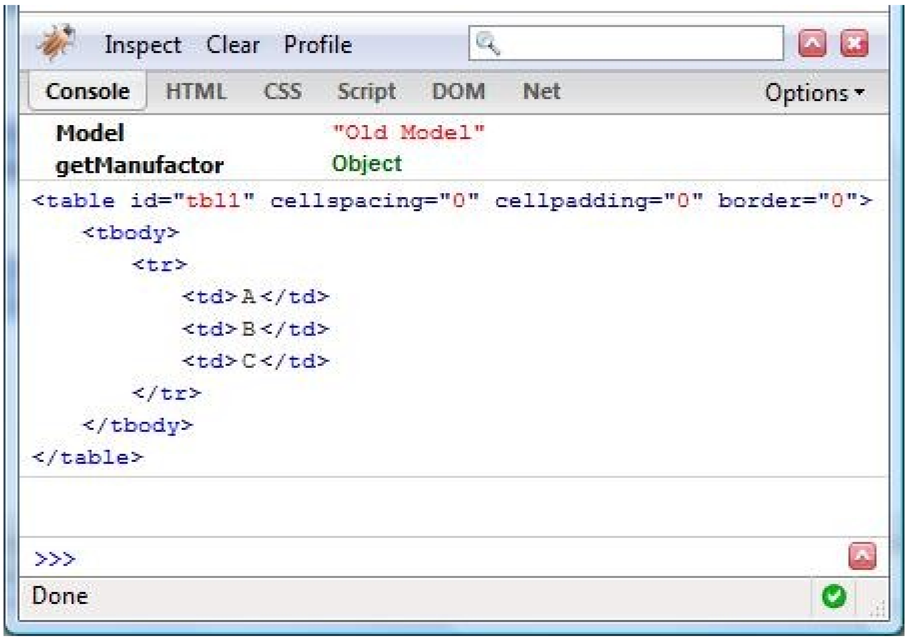


```
message group : group1                       test.htm (line 10)
    log message 1 from group1
    log message 2 from group1
    log message 3 from group1
message group : group2                       test.htm (line 17)
    log message 1 from group2
    message group : subgroup1                test.htm (line 21)
        log message 1 from subgroup1
        log message 2 from subgroup1
        log message 3 from subgroup1
    log message 3 from group2
>>>
Done
```

#3. console.dir and console.dirxml

- **console.dir** : It can be used for getting all properties and methods of a particular object. According the example below, we can get the Model (property) and getManufactor (method) of Car object by using console.dir(); You can also pass the object of HTML element (eg: console.dir(document.getElementById('tbl1'));) instead of objCar and let's see the result. (You will get all properties and methods of the HTML table called "tbl1″).
- **console.dirxml** : print the XML source tree of HTML element.

```
<table id="tbl1" cellpadding="0" cellspacing="0" border="0">
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</table>
<script language="javascript" type="text/javascript">
//Create a class
function Car(){
this.Model = "Old Model";

this.getManufactor = new function(){
return "Toyota";
}
}

//Create a object
var objCar = new Car();

//Firebug
console.dir(objCar);
console.dirxml(document.getElementById('tbl1'));

</script>
```



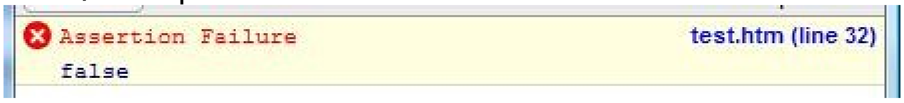
```
Model              "Old Model"
getManufactor      Object
<table id="tbl1" cellspacing="0" cellpadding="0" border="0">
    <tbody>
        <tr>
            <td>A</td>
            <td>B</td>
            <td>C</td>
        </tr>
    </tbody>
</table>
>>>
Done
```

#4. Assertion (console.assert())

You can use console.assert() to test whether an expression is true or not. If the expression is false, it will write a message to the console and throw an exception.

Example :

```
<script language="javascript" type="text/javascript">
function whatIsMyAge(year){
var currYear = 2007;
return currYear - year;
}

var yearOfBirth1 = 1982;
var age1 = 25;
console.assert(whatIsMyAge(yearOfBirth1) == age1);

var yearOfBirth2 = 1982;
var age2 = 11;
console.assert(whatIsMyAge(yearOfBirth2) == age2); //You should get the error here.
</script>
```



```
Assertion Failure                              test.htm (line 32)
  false
```

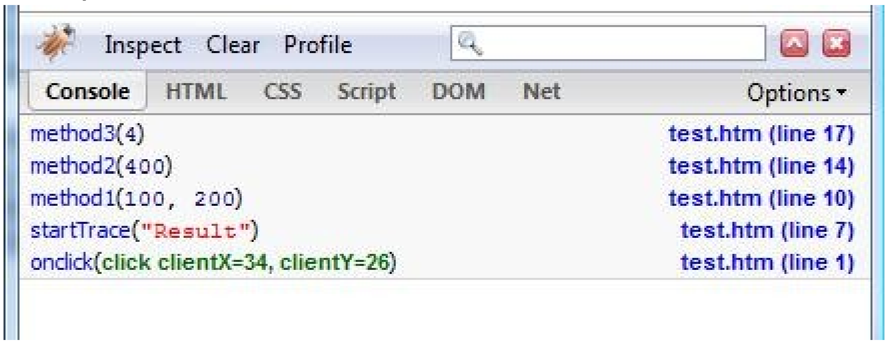
#5. Tracing (console.trace())

This function is very interesting. Before I tell you the way that I understand, let's take a look what console.trace does exactly in official website.

> **console.trace()**
>
> Prints an interactive stack trace of JavaScript execution at the point where it is called.
>
> The stack trace details the functions on the stack, as well as the values that were passed as arguments to each function. You can click each function to take you to its source in the Script tab, and click each argument value to inspect it in the DOM or HTML tabs.

This function will tell you about the route information from start point to end point. If you are not clear what I mean, let's take a look at the sample code and the result.

```
01.<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
02.<html xmlns="http://www.w3.org/1999/xhtml" >
03.<head>
04.<title>Firebug</title>
05.<script language="javascript" type="text/javascript">
06.function startTrace(str){
07.return method1(100,200);
08.}
09.function method1(arg1,arg2){
10.return method2(arg1 + arg2 + 100);
11.}
12.function method2(arg1){
13.var var1 = arg1 / 100;
14.return method3(var1);
15.}
16.function method3(arg1){
17.console.trace();
18.var total = arg1 * 100;
19.return total;
20.}
21.
22.</script>
23.</head>
24.<body>
25.<input type="button" value="Trace"
onclick="startTrace('Result');"/>
26.</body>
27.</html>
```



```
method3(4)                                   test.htm (line 17)
method2(400)                                 test.htm (line 14)
method1(100, 200)                            test.htm (line 10)
startTrace("Result")                         test.htm (line 7)
onclick(click clientX=34, clientY=26)        test.htm (line 1)
```

Suppose: we wanna know how "method3″ function is invoked. So, we put this code "console.trace()" in that method. then, we run the program and we got the result as picture above. If we read the result from bottom to top, we will see "onclick(click clientX=34, clientY=26)". That means the execution of Javascript started at on click event of button. then, we got "startTrace("Result")" in second line. That means startTrace function is invoked after firing onclick event and the parameter is "Result". If we keep on checking from bottom to top, we will figure out the completed route from onclick event to method3.

If you wanna test more, you can move this code "console.trace()" to method2(). then, firebug will give the new route from onclick event which is a started point to method2() which is the end point.

I think that it's pretty useful if you are debugging the other developer's source code and you have no idea why this function is invoked.

Let me know if you are not clear what I'm trying to explain about console.trace();.

#6. Timing (Measuring the time of your code)

You can use console.time(timeName) function to measure how long a particular code or function take. This feature is very helpful if you are trying to improve the performance of your Javascript code.

Example :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>Firebug</title>
<script language="javascript" type="text/javascript">
function measuretheTime(){
var timeName = 'measuringTime';
console.time(timeName);

for(var i=0;i&lt;1000;i++){
///do something
for(var j=0;j&lt;100;j++){
//do another thing.
}
}

console.timeEnd(timeName);
}
</script>
</head>
<body>
<input type="button" value="Trace" onclick="measuretheTime();"/>
</body>
</html>
```

Result : measuringTime: 16ms

#7. Javascript Profiler

You can start the profiler thought code (console.profile('profileName')) or by clicking "Profile" button from "Console" tag. It can be used for improving the performance of Javascript. It is similiar to the console.time() function but profiler can give your more advanced and detailed information.

I will tell you about this more details in next tutorial (Part2) . I hope you all are clear about this tutorial. If you have any comment or suggestion, please drop a comment.. Thanks. C ya tomorrow.

Quick & Easy CSS Development with Firebug

In the past, I found myself spending countless hours tweaking my CSS and making everything work in Internet Explorer just as it would in Firefox. Everything changed when I found Firebug. In this tutorial, I am going to discuss how to use Firebug to make CSS development faster, and share some tips for a consistent look between browsers.

Quick Nav:

- Installing and Getting around Firebug
- Live editing of CSS using Firebug
- Firebug for Internet Explorer

Materials Needed:

- Firebug (for Mozilla Firefox)
- Web Editor (Dreamweaver, Aptana, Notepad++, etc)

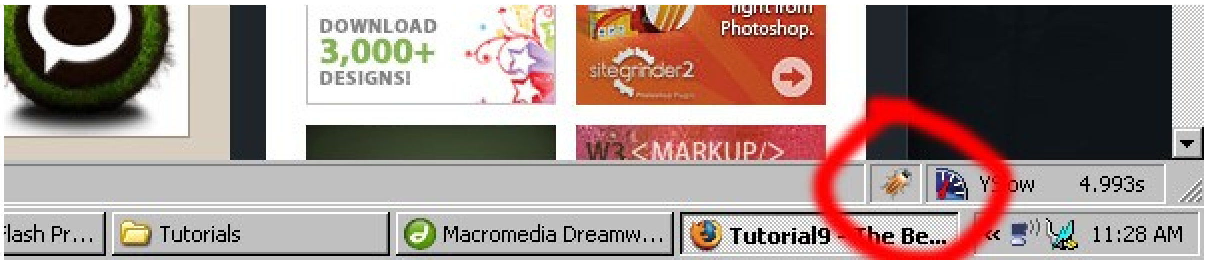
Part 1: Installing and Getting Around Firebug

If you haven't installed Firebug yet, Please go to the Mozilla plug-in directory, download and install: The link can be found here:

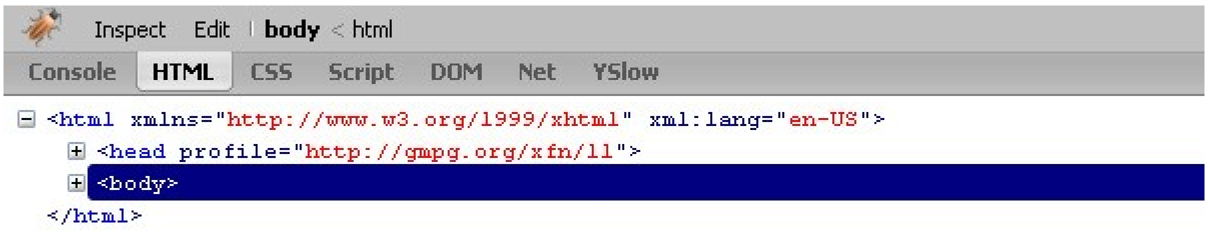
https://addons.mozilla.org/en-US/firefox/addon/1843

Now, the Firebug is installed its time to understand what we're looking at, and how we can use it. First, lets navigate to Tutorial9.net so we are all working with the same example.

Next click on the **little bug in the bottom right corner of your browser (see photo below).**

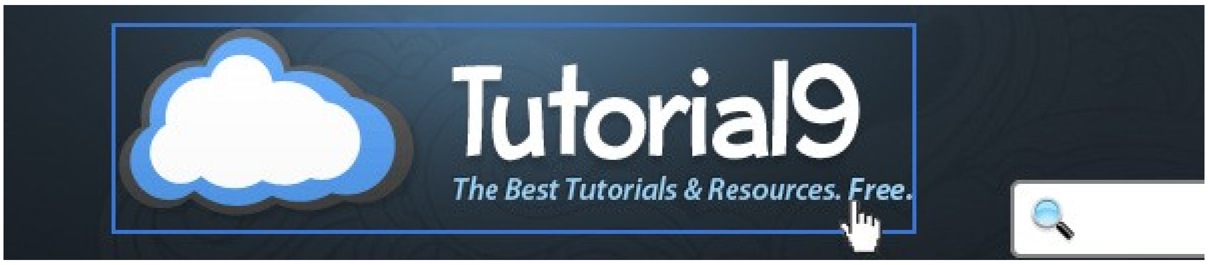


You'll see the firebug window expand. And you'll see the menu below. In the top menu you will see **Inspect** and **Edit. Click on Inspect.** If you mouse over the website, you will see a blue box outline around whichever html object you hover over.



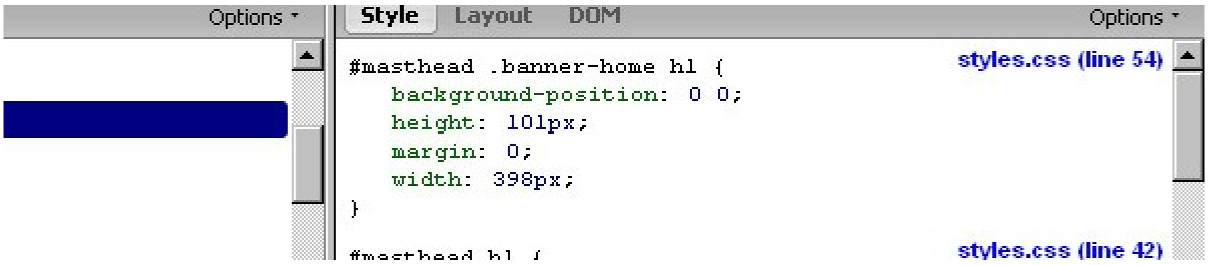
```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-US">
  <head profile="http://gmpg.org/xfn/11">
  <body>
</html>
```

Now, when you click an element you will see that the right side of the firebug window show the CSS elements and attributes of that item. **For our example we're going to click inspect and select the tutorial9.net logo.**



Once you select the element, look in the bottom right hand corner of your firebug window. Here you will see the CSS elements and attributes of those elements that relate to the object you selected in the web site. To the right of each element you will see the Location of the CSS element in the document as well as the line it appears on.

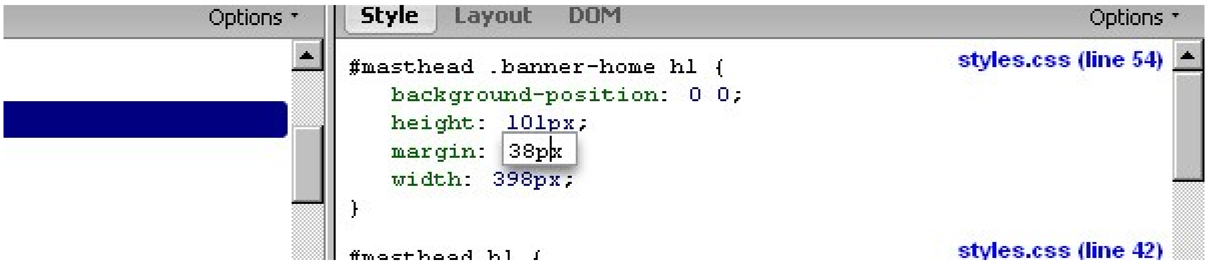
So using our example, The Tutorial9.net logo inherits its CSS styles from ID masthead (#masthead) in Styles.CSS on line 54.



```
#masthead .banner-home h1 {                    styles.css (line 54)
   background-position: 0 0;
   height: 101px;
   margin: 0;
   width: 398px;
}
#masthead h1 {                                 styles.css (line 42)
```

So lets Play around a little but:

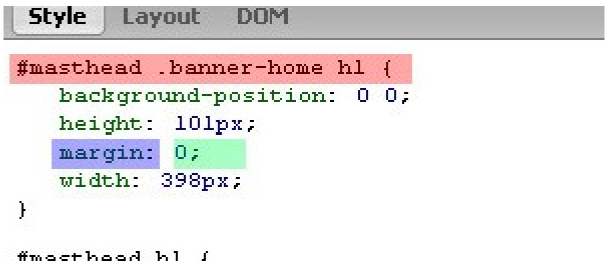
Move your mouse over the value of the margin property, and change 0 to 38px, Notice when you change the value, the appearance in the live document instantly change as well.



```
#masthead .banner-home h1 {                    styles.css (line 54)
   background-position: 0 0;
   height: 101px;
   margin: 38px
   width: 398px;
}
#masthead h1 {                                 styles.css (line 42)
```

Element, Attributes and Properties????

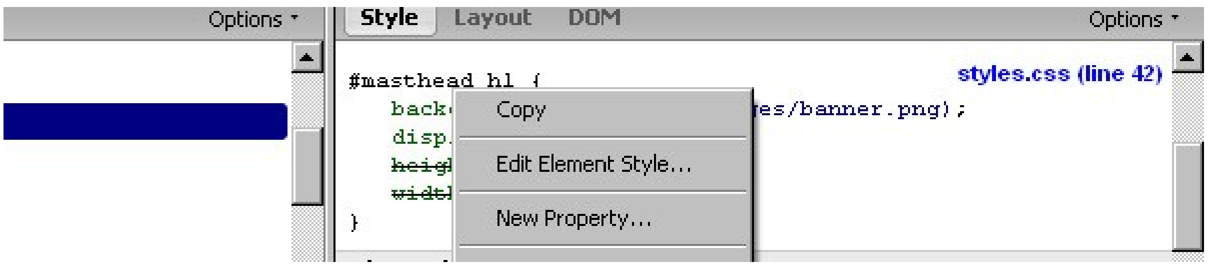
The Red – CSS Element (Class, ID, or Tag)
The Blue – Element Property (Margin, Width, Height, Etc)
The Teal – Property Value



```
#masthead .banner-home h1 {
   background-position: 0 0;
   height: 101px;
   margin: 0;
   width: 398px;
}
#masthead h1 {
```

Now, if you play with the properties you can preview your CSS in a live browser environment, saving you from jumping back and forth between your doucment editor, browser and ftp server.

Moving on, If you mouse over an element title and right click you have the ability to **add a new property**, to adjust the CSS element as well.



```
#masthead h1 {
    back                es/banner.png);
    disp
    ~~heig~~
    ~~widtl~~
}
```

Part 2: Edit YOUR web page's CSS with firebug

Now that we understand firebug, and what it does, its time to dig into how we can use it to edit CSS in your document.

Open your webpage in firefox. Select the first element you wish to modify. Add/Remove/Modify properties until the element is positioned the way you want. Once you get the look you want, **select the properties**, **select copy, open your document editor navigate to the appropriate line paste the new properties, save and reopen in your browser.** You will see the new changes.

In the video below you can see how to I use firebug to adjust various objects in my live document.

Tutorial on Using Firebug from Ryan Hickman on Vimeo.

Part 3: Firebug and Internet Explorer

Of course, like many — I am not an advocate of Internet Explorer — sad thing is there are still many internet explorer users out there. *As a developer, I've been in a few instantces where I've prepared a website for a client who works for a government agency, and the agency has a contract to only use internet explorer at their offices – So I found that I HAD to build for ie6. I also learned how many people (which there are a lot) that still use IE and how many of those IE users still use IE6.*

Of course the Firefox plugin doesn't work for Internet Explorer, so you have to use 'FireBug Lite'. Which a very lightweight JS script that you embed in your code while developing, and remove when your all set. Here's how, In your header insert the following code:

```
 <script type='text/javascript'
src='http://getfirebug.com/releases/lite/1.2/firebug-lite-
compressed.js'></script>
```

If you need to adjust the size of the firebug window you can by inserting the following script, below the one above:

```
<script type="text/javascript">
        firebug.env.height = 500;
</script>
```

For those of you who are developing on your local machine, you can download firebug lite here. Note: you must replace your script with the follow:

```
 <script language="javascript" type="text/javascript"
src="/path/to/firebug/firebug-lite.js"></script>
```

Working with firebug in internet explorer is a bit different functionaility wise, as once you make change to your CSS, you have to manually run the CSS changes from the firebug consul window. Aside that, Its the same principals as developing for Firefox.

Part 4:The CSS Im using works in Firefox but Not IE

Okay — we're making progress, but it seems some CSS properties **Dont look the same in Firefox as they do in Internet Explorer.**

1. Conditional Styles

Since IE is very quirky and non-compliant with w3 Schools CSS standards, often you must create a custom style sheet, **that only loads when the user is browsing from that particular version of internet explorer**.To do so, simply add the following code to the <head> of your document.

```
<!--[if IE]>    <link rel="stylesheet" type="text/css" href="ie.css" /><!
[endif]-->
```

In between the tags, you can place CSS code or, like the above example link to an external stylesheet. Any code you place in between the tags will appear if the tags are true. The script below shows a DIV only if the user is using IE6.

```
<!--[if IE 6]>  <div id="ie6">
                We reccommend that you upgrade to at least Interent Explorer 7
        </div>
<![endif]-->
```

Here are a few other Conditions:

- IE (Any version of Internet Explorer)
- lt IE 7 (Versions less than version)
- lte IE 6(Versions less than or equal to version)
- IE 6 (Only version)
- gte IE 6 (Versions greater than or equal to version)
- gt IE 7 (Versions greater than version)

2. Opacity

Did you know you CAN do opacity in Internet Explorer with a Javascript. Many people don't know how. Its every simple. In Firefox you use the property **Opacity.** When in IE you use **filer: alpha(opacity = value);** as show in the code below:

```
.logo {
        opacity: 0.5;  //For Firebox
        filter: alpha(opacity = 50); //For Internet Explorer
}
```

```
<!--[if IE]>    <link rel="stylesheet" type="text/css" href="ie.css" /><!
[endif]-->
```

In between the tags, you can place CSS code or, like the above example link to an external stylesheet. Any code you place in between the tags will appear if the tags are true. The script below shows a DIV only if the user is using IE6.

```
<!--[if IE 6]>  <div id="ie6">
                We reccommend that you upgrade to at least Interent Explorer 7
        </div>
<![endif]-->
```

3. IE6 Scroll Renders Artifact

What happens is IE6 (which again sadly, people still use) is when scrolling vertically down the page, artifacts make the page look all messed up. What is happening is a DIV element that has nothing behind it on the page is rendered bit by bit (for efficiency says Microsoft) as the page scrolls. If the scrolling is anything other than 100% smooth the rendering fails leaving the sorts of glitches shown above. To fix, we have to insert a null object behind the DIV. Simple fix, see blow:

```
html {
        background : url(null) fixed no-repeat;
    }
```

4. Min-Width and Min-Height Issues

IE doesn’t understand these commands, so to make this work, you’ll need to use conditions to access customs styles to make it work in Internet Explorer. So Lets say we have a div called wrapper:

```
#wrapper{
        min-width: 750px;
        width:expression(
                document.body.clientWidth < 750? "750px": "auto"
        );
}
```

5. A:Hover, Button:Hover — Hover Issues

:hover enables you to have cool effects for HTML elements like button rollovers, etc. Most browsers have no problem with this, except IE (once again of course) which look at the stylesheets and each individual rule with Javascript.
If :hover rules can be tracked, and .htc can be used to change an elements behavior, then it should be possible to create a behavior that enables :hover for any element in IE.

In my building I’ve found the only effective work-around is using a small JS file. Which you insert like below:

```
  body {
        behavior: url("csshover3.htc");
}
```

You can download the csshover3.htc file here.
Unpacked (9k)

Packed (2.5k)

6. Margin + Centering your Div

When centering DIV tags using either the margin-left: auto; or margin-right: auto; settings, **will not work in Internet explorer**. Your would have to conditionally add the following code:

```
#divname {      text-align: center;}
```

Note: IF YOU MAKE THE DIV CENTER, YOU MUST RE-ADJUST THE INNER CONTENT

```
#p {    text-align: left;}
```

7. Margins Too Large

I hate this one the most. Setting the margin attribute for any CSS element in Internet Explorer will often appear with added width, which can seriously mess up detailed layouts. By using the {display: inline;} property/value on the tag containing your margin setting can fix this

```
<div id="special" style="display: inline; margin-left: 10px;"></div>
```

8. Lightbox's (modal boxes) don't hide Flash underneath

Everyone is using Lightbox script's nowadays. If ever your running into the problem of a flash element (swf or even a video) and you want the flash element to continue to play… heres how:

```
<param name="wmode" value="transparent"/>
```

If your using SWFObject:

```
        so.addParam("wmode","transparent");
```

Final Thought:

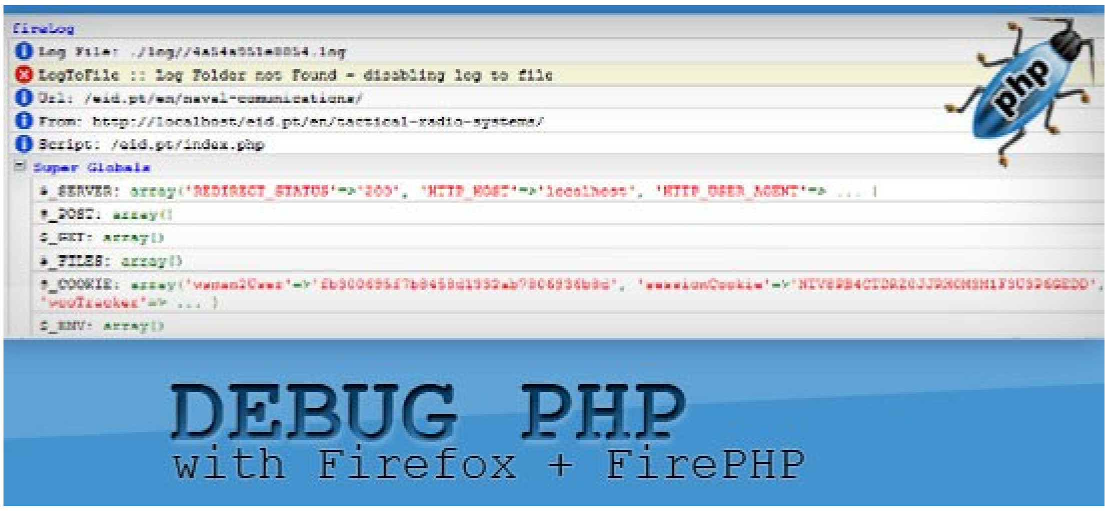
Dont test User-Agents to detect the users Browser, cause Google will look at that as Cloaking and ban your site, believe me, I know

Got any tips to add that I may have missed? Please share in the comments below!

How to Debug PHP Using Firefox with FirePHP

July 11th, 2009 by Nuno Franco da Costa | 29 Comments | Stumble It! Delicious

Typically, there are two main ways of debugging server-side code: you can utilize an Integrated Development Environment (IDE) with a built-in debugger or log and perform your debugging processes in a web browser.

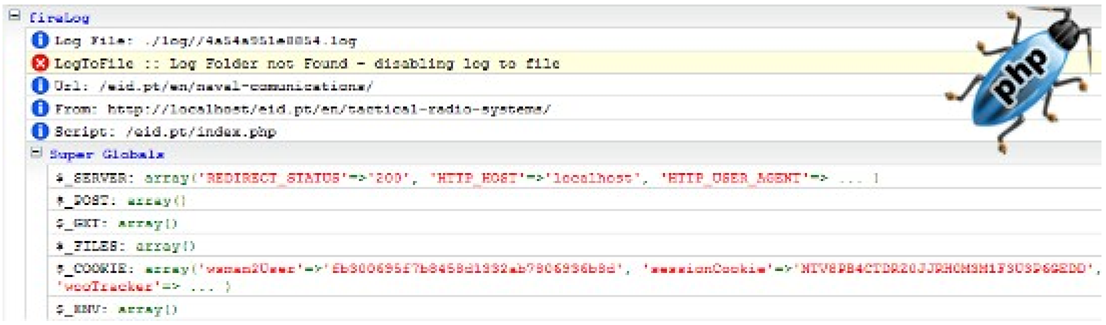


```
fireLog
Log File: ./log//4a54a551e8854.log
LogToFile :: Log Folder not Found - disabling log to file
Url: /eid.pt/en/naval-communications/
From: http://localhost/eid.pt/en/tactical-radio-systems/
Script: /eid.pt/index.php
Super Globals
  $_SERVER: array('REDIRECT_STATUS'=>'200', 'HTTP_HOST'=>'localhost', 'HTTP_USER_AGENT'=> ... )
  $_POST: array()
  $_GET: array()
  $_FILES: array()
  $_COOKIE: array('wanan2User'=>'fb9006951f7b8458d1332ab7906936b8d', 'sessionCookie'=>'NIV8PB4CTDR20JJRHOM9H1F9U9P6GEDD',
  'wooTracker'=> ... )
  $_ENV: array()
```

This article shares an **elegant, simple, and more maintainable way of debugging Ajax apps** via the web browser (more specifically for the **Mozilla Firefox** browser). You'll learn the basics of leveraging Firefox in conjunction with Firebug and FirePHP to implement FirePHP libraries on web apps and logging messages in the Firebug console.

A Brief Introduction

When Ajax techniques became popular, developers faced a new problem: how can we debug Ajax requests and responses efficiently for complex web applications? If using a debugger was hard enough with the RESTful model, triggering an Ajax-specific request is a pain and a bit more difficult; dumping logs and information pertaining to those Ajax operations had to be done using JSON or XML.



```
fireLog
Log File: ./log//4a54a551e8854.log
LogToFile :: Log Folder not Found - disabling log to file
Url: /eid.pt/en/naval-communications/
From: http://localhost/eid.pt/en/tactical-radio-systems/
Script: /eid.pt/index.php
Super Globals
  $_SERVER: array('REDIRECT_STATUS'=>'200', 'HTTP_HOST'=>'localhost', 'HTTP_USER_AGENT'=> ... )
  $_POST: array()
  $_GET: array()
  $_FILES: array()
  $_COOKIE: array('wanan2User'=>'fb9006951f7b8458d1332ab7906936b8d', 'sessionCookie'=>'NIV8PB4CTDR20JJRHOM9H1F9U9P6GEDD',
  'wooTracker'=> ... )
  $_ENV: array()
```

This is where *FirePHP helps*, allowing you to log your debugging messages to the Firebug console. FirePHP doesn't mess with your code (and it doesn't require you to modify anything to trap errors): the messages you print are sent to the browser in the HTTP response headers, which is great when you're using JSON or XML since it won't break their encoding.

This makes FirePHP ideal not only for debugging your Ajax requests, but also your entire PHP codebase.

So, what is FirePHP?

FirePHP is an add-on for an add-on: it extends the popular in-browser web development tool called Firebug with an API for PHP web application developers. FirePHP is *free* and easily attainable via the Mozilla Add-Ons section on the official Mozilla site. The official FirePHP site can be found via this web address: www.firephp.org. Christoph Dorn is the person responsible for creating FirePHP.

What Do I Need to Get Started?

As you might have guessed, you need three things to get up and running with FirePHP, they are:

1. Firefox
2. Firebug
3. FirePHP

If you don't have all of the above applications installed on your machine, click on their link to learn about how to download them for your particular system.

Installation of the three things above is a straightforward process. FirePHP can be a little tricky to install if you've just recently started learning about web development, but there's good documentation out there about it.

This article is about using FirePHP, so I'll let you handle the installation part (though feel free to ask in the comments – we'd be happy to help if you encounter issues).

A Couple of Tips

Once you've installed FirePHP, and included it in your web application, you are ready to debug and log data. But first, I'd like to share two helpful tips I've learned:

Tip #1: call ob_start()

Because the information is sent to Firebug in the HTTP Headers, you should activate the output buffering or you might get the "headers already sent error". It may sound complicated, but all you have to do is write ob_start() on the first line of your PHP script that you're debugging.

Tip #2: don't forget to disable FirePHP Logging when you go live

You have to disable FirePHP when the site goes live or you will risk exposing sensitive information to anyone that has Firebug/FirePHP installed (we'll talk about how to do this later down the article).

And then just a general tip for Firebug/FirePHP users: it's also a good idea to disable or suspend Firebug and FirePHP when you're just browsing the web because they can really slow down some websites and web applications (such as Gmail, for example).

Getting Started with FirePHP Logging

This is the fun part where we start logging messages and reviewing the basic logging functions.

One thing to note is that, just like PHP, which (at least for PHP4 and PHP5) is a "*pseudo* object-oriented" language, you can use FirePHP in a procedural or object-oriented (abbreviated **OO** from now on) manner.

I prefer the object-oriented techniques and I encourage you to use the same pattern as it is considered the most popular and most modern approach to building apps.

The *OO API* allows you to instantiate a Firebug object to use it or to call its static methods directly. I'm a lazy guy and because static methods are more terse and require less typing, that's what I use.

Instantiating the OO API Object

In your script, you can use the following code block to create the FirePHP object ($firephp).

```
require_once('FirePHPCore/FirePHP.class.PHP');
$firephp = FirePHP::getInstance(true);
$firephp -> [classmethod]
```

OO API with Static Methods

This is the format for calling static methods in your scripts.

```
require_once('FirePHPCore/fb.PHP');
FB::[nameofmethod]
```

The Procedural API

Here's how to use FirePHP's Procedural API:

```
require_once('FirePHPCore/fb.PHP');
fb($var)
fb($var, 'Label')
fb($var, FirePHP::[nameofmethod])
```

We will not get into any detail about the benefits and coding style of each API, I've included them here only so you see what options are available for you. In other words, I don't want to start a flame war about which procedure is better – it's up to you to decide and I've noted my preference.

Logging Messages and Information in the Firebug Console

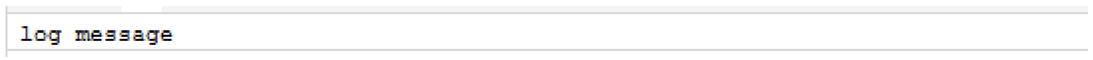
Let's talk about logging messages in the Firebug console (remember, this will only work if you've configured your app for FirePHP).

Examples of Basic Logging Calls

If you are ad-hoc debugging a bug, the following examples are what you'll be interested in utilizing.

```
Fb::log("log message")
```

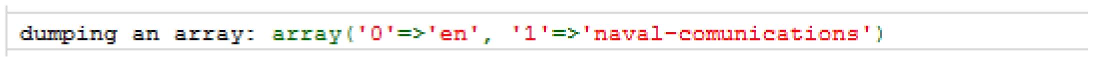
This will print a string that you pass to it onto the Firebug console. Using the above example results in the following message:



```
log message
```

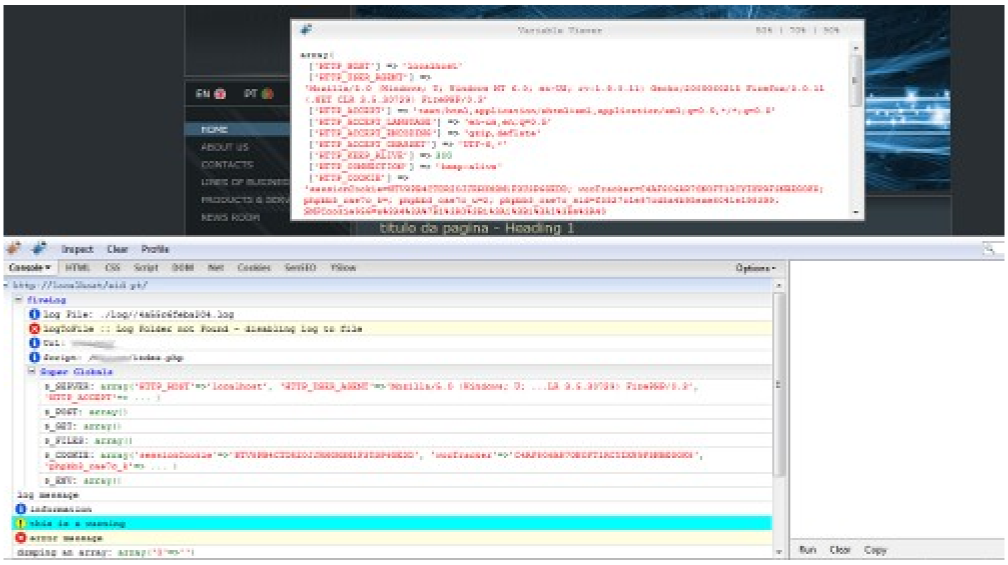
```
Fb::log($array, "dumping an array")
```

Passing an array (no more for loops or print_r() in your scripts) outputs the content of your array. The above example will result in the following message in the Firebug console:



```
dumping an array: array('0'=>'en', '1'=>'naval-comunications')
```

Tip: when you hover your mouse on logged variables in the Firebug console, an info window will appear in the web page containing all of its elements. It's a nifty feature, don't you agree?



```
array(
 ['HTTP_HOST'] => 'localhost'
 ['HTTP_USER_AGENT'] =>
'Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.11) Gecko/2009060215 Firefox/3.0.11
(.NET CLR 3.5.30729) FirePHP/0.3'
 ['HTTP_ACCEPT'] => 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
 ['HTTP_ACCEPT_LANGUAGE'] => 'en-us,en;q=0.5'
 ['HTTP_ACCEPT_ENCODING'] => 'gzip,deflate'
 ['HTTP_ACCEPT_CHARSET'] => 'UTF-8,*'
 ['HTTP_KEEP_ALIVE'] => 300
 ['HTTP_CONNECTION'] => 'keep-alive'
 ['HTTP_COOKIE'] =>
'[illegible]
```

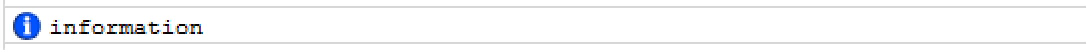
```
http://localhost/[illegible]/
fireLog
  log File: ./log/[illegible].log
  logToFile :: Log Folder not Found - disabling log to file
  url: [illegible]
  design: /[illegible]/index.php
  Super Globals
    $_SERVER: array('HTTP_HOST'=>'localhost', 'HTTP_USER_AGENT'=>'Mozilla/5.0 (Windows; U; ...LR 3.5.30729) FirePHP/0.3',
    'HTTP_ACCEPT'=> ... )
    $_POST: array()
    $_GET: array()
    $_FILES: array()
    $_COOKIE: array('sessionCookie'=>'[illegible]', 'userTracker'=>'[illegible]',
    'phpbb3_cae7o_k'=> ... )
    $_ENV: array()
log message
information
this is a warning
error message
dumping an array: array('0'=>'')
```

Logging an Information Message

Here is an example of logging information messages using the info method.

```
Fb::info("information")
```

This is the message it logs in your Firebug console:



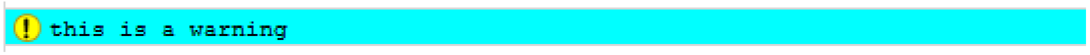
```
information
```

Logging a Warning Message

Here is an example of logging a warning message.

```
Fb::warn("this is a warning")
```

This is the message it logs in your Firebug console:



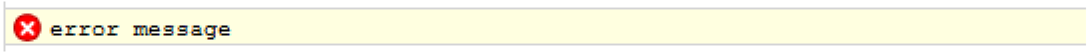
```
this is a warning
```

Logging an Error Message

Here is an example of logging a warning message using the info method.

```
Fb::error("error message")
```

Here’s what an error message looks like in the Firebug console:



```
error message
```

Enabling or Disabling FirePHP Logging

When your site goes live, you can (and should) disable FirePHP logging. Call the following line of code on the first lines of your script.

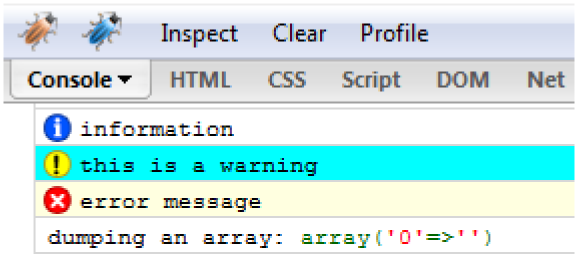
```
FB::setEnabled(false);
```

What’s great about this is that you can leave all of your FirePHP code in your scripts for later use – just pass either `true` or `false` when you want to turn logging on or off.

If your application uses a "config file" to keep track of global settings, it is advisable to set a configuration option to enable or disable it.

Conclusion

First, here's a screen capture showing all of our messages in Firebug all at once (I ran it sequentially).



```
information
this is a warning
error message
dumping an array: array('0'=>'')
```

In this article, we covered the very basics of using FirePHP to help you debug and gain information about your PHP/Ajax applications easier and through the web browser. I hope that this results in you becoming convinced that you should explore your debugging options outside of "old-school" techniques such as using echo or print on top of your web page layout to see what a variable or array contains. Using FirePHP is so easy and convenient, and gives you much more options and data for debugging purposes.

In a *future article*, I'll be covering **more complex features of FirePHP and using Firebug** to make this simple debugging tool a more *robust and fully-featured logging framework*.

So stick around, don't forget to subscribe to the **Six Revisions RSS feed** so you get a note when that article is posted. * *Article edits by Jacob Gube*

Related content

- Four Ways Ruby on Rails Can Help You
- Using XAMPP for Local WordPress Theme Development
- 25 Excellent Ajax Techniques and Examples
- *Related categories*: Ajax, Web Development, and Tools

About the Author

Nuno Franco da Costa is a web developer and sys admin. By day, he works at a design agency coordinating the development and sys admin teams where he developed a PHP MVC framework and a WEB 2 CMS. He loves to code and has a "*getting things done*" attitude. You can find over at his online presence **www.francodacosta.com**