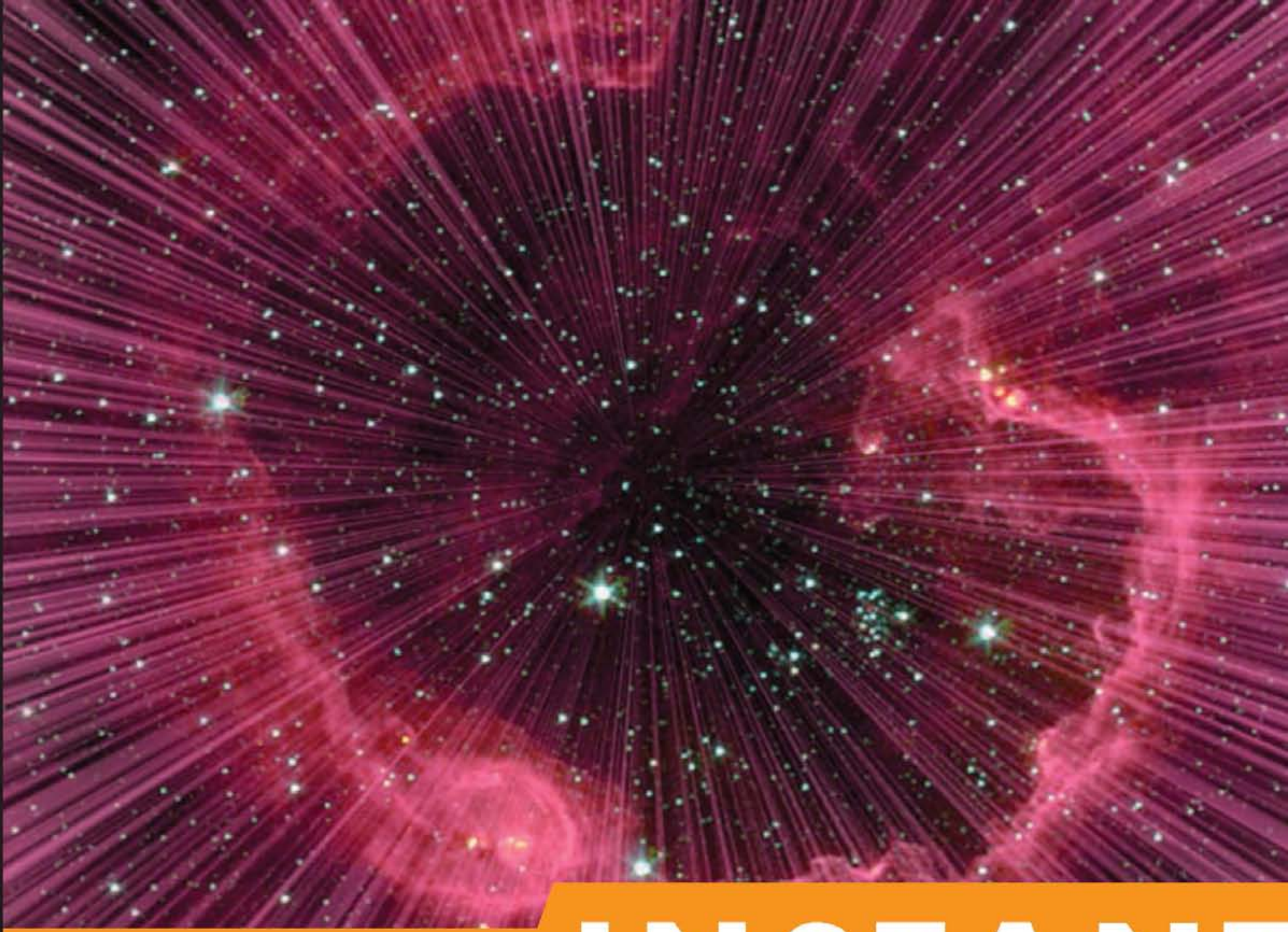


INSTANT

Short | Fast | Focused

Firebug Starter

Monitor, edit, and debug any web page in real time with this handy practical guide

Chandan Luthra

[PACKT] PUBLISHING

Instant Firebug Starter

Monitor, edit, and debug any web page in real time with this handy practical guide

Chandan Luthra

[PACKT] PUBLISHING

BIRMINGHAM - MUMBAI

Instant Firebug Starter

First published: January 2013

Production Reference: 1180113

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK..

ISBN 978-1-78216-116-5

`www.packtpub.com`

Credits

Author	**Project Coordinator**
Chandan Luthra	Abhishek Kori
Reviewers	**Proofreader**
Jan Odvárko	Maria Gould
Michael Ratcliffe	
	Production Coordinator
Acquisition Editor	Melwyn D'sa
Martin Bell	
	Cover Work
Commissioning Editor	Melwyn D'sa
Ameya Sawant	
	Cover Image
Technical Editor	Manu Joseph
Sharvari Baet	

About the Author

Chandan Luthra is an agile and pragmatic programmer, and an active participant at the local open source software events, where he evangelizes about Firebug, Groovy, Grails, and JQuery. He is a Linux and open source enthusiast. He also involves himself in writing blogs, articles, and is an active member on various tech-related mailing lists. He has developed web apps for various industries, including entertainment, finance, media and publishing, as well as others.

He loves to share his knowledge and good coding practices with other team members in order to perfect their development skills. In his free time, he loves to contribute to open source technologies.

He also loves to code in JQuery with Firebug, which makes development very easy for him. He is a fond lover of Firebug and has been using it since 2007 and co-authored a book on Firebug 1.5 with Packt Publishing in the year 2010.

I would like to thank my beloved wife, Rashmi, for her love and support during my flung adventures into esoteric nonsense. I would also like to thank Chetan Khurana for helping me out with valuable ideas and suggestions. Finally, I wish to thank to the Packt team and reviewers for giving a perfect shape to this book.

About the Reviewers

Jan Odvárko has been an independent software consultant for over 15 years. He has led and worked on projects as diverse as real-time data visualization, clinical data management, web performance tools, and visual tools for web development such as Firebug.

He has a deep passion for user-interface design and interaction, and has taught courses on a wide range of computer science topics. He is the author of numerous articles and a speaker at industry events. His experience includes roles such as software architect, project manager, technical lead, consultant, software business founder, and contributor to many open source projects.

He is currently leading the Firebug project. Prior to that he was co-founder and Chief Software Architect at AllPeers.

He received a degree in Computer Science from University in Ceske Budejovice and he is fluent in Czech and English. You can read more about Jan on his website at `http://www.softwareishard.com/blog/about/`.

He is currently living with his son Tomáš Odvárko in Czech Republic.

Michael Ratcliffe is a Software Developer with Mozilla, a non-profit organization whose focus is to make the Internet a better place whilst keeping it free and available to everybody. He specializes in JavaScript, but also enjoys hacking away on various projects in various other languages. He is an active participant at local open source software events and is a Linux and open source enthusiast. He is also an active member of various tech-related mailing lists.

He loves to share his knowledge and good coding practices in order to help other hackers improve their development skills. In his free time, he loves to contribute to open source technologies. He is a veteran Firebug user and has been using it almost since its conception.

He works for Mozilla as a member of the Firefox Developer Tools Team. His team's goal is to provide the greatest set of web developer tools ever. Along with Firebug, Firefox's built-in developer tools provide an amazing aid for any web developer but he believes that they can always be made better.

He was the technical editor for Packt Publishing's book, *Firebug 1.5*.

Follow him on Twitter `@ratcliffe_mike` or read his blog at `http://www.flailingmonkey.com`.

I would like to thank my beautiful wife Sabine for putting up with my continual desire to fill my brain with computer stuff.

Table of Contents

Instant Firebug Starter

Welcome to the *Instant Firebug Starter*. This book has been especially created to provide you with all the information that you need to get started with Firebug. You will learn the basics of Firebug, get started with editing, debugging, monitoring web pages, and discover some tips and tricks for using Firebug.

This document contains the following sections:

So what is Firebug? covers what Firebug actually is, what you can do with it, why it's so great, and why most web app and UI developers are using it across the globe.

Installation covers how to download and install Firebug with minimum fuss and then set it up so that you can use it as soon as possible.

Quick start gives you an overview of all the tabs in Firebug and how to perform one of the core tasks of Firebug; inspecting and editing HTML. Follow the steps to inspect and edit elements of your own web pages, which will be the basis of most of your work in Firebug.

Top 17 features you need to know about explains how to inspect and edit HTML, edit CSS on the fly, profile and debug JavaScript, inspect DOM, and use a few popular Firebug extensions. By the end of this section, you will be able to use Firebug for inspecting and editing HTML on the fly with an example which helps you in the web application development process. You can also edit CSS and visualize CSS metrics to tweak the look and feel of web pages immediately in Firefox itself. It will then be followed by profiling and debugging JavaScript with a small *Hello world* example of JavaScript. You will also be able to explore DOM, optimize the performance of a small sample application hosted on web server, and analyze AJAX calls of different sites with the help of the Net tab of Firebug.

People and places you should get to know tells us every open source project is centered around a community. This section provides you with many useful links to the project page and forums, as well as a number of helpful articles, tutorials, blogs, and Twitter feeds of Firebug super-contributors.

So, what is Firebug?

Firebug is one of the most popular and powerful web development tools. It is a free and open source tool, available as a Mozilla Firefox extension, which allows the debugging, editing, and monitoring of any website's CSS, HTML, DOM, and JavaScript. It allows the performance analysis of a website and has a JavaScript console for logging errors, watching values, and analyzing stack traces.

Firebug has many other tools to enhance the productivity of today's web developer, designer, and HTML developer. Today's websites are a product of several distinct technologies and we developers must be proficient in all of them – HTML, CSS, JavaScript, DOM and AJAX, among others. Web browsers have always focused on the needs of the end users; as a result, web developers have long been deprived of a good tool on the client/browser side to help them and debug their code.

Firebug fills this gap very nicely. It provides all the tools that today's web developer needs in order to be productive and efficient with code that runs in the browser.

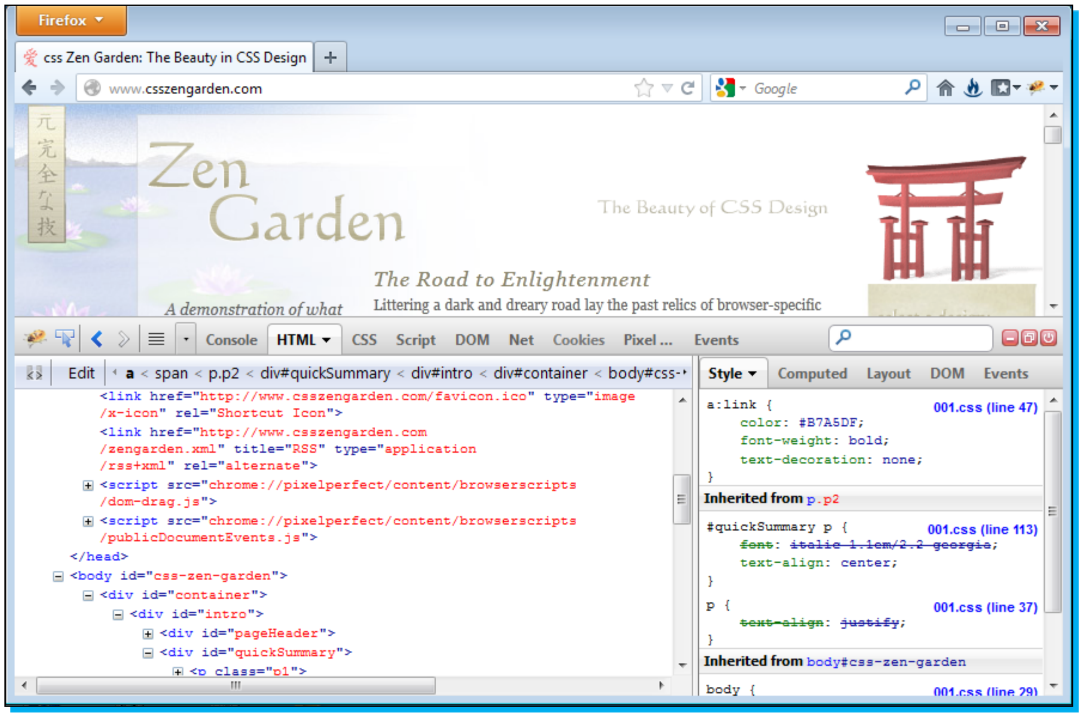
Firebug capabilities

Firebug has a host of features that allow us to do the following (and much more!):

- Inspect HTML and modify style, cookies, and layout on the fly
- Use the most advanced JavaScript debugger available for any browser
- Accurately analyze network usage and performance
- Explore the DOM and analyze AJAX calls
- Extend Firebug and add features to make Firebug even more powerful

Firebug is getting more popular day by day with web developers worldwide. The fact is that it is not only limited to Firefox, but you can also use the Firebug Lite version for non-Firefox browsers too. Firebug Lite made developers more comfortable and gained their trust by providing compatibility with all major browsers, such as IE6 and higher, Firefox, Opera, Safari, and Chrome. It also provides the same look and feel of Firebug.

Some features are browser dependent and they won't be available in Firebug Lite. We'll discuss more about the features and limitations of the Lite version later in the book. Let's see what a Firefox window installed with Firebug looks like:


Firefox
css Zen Garden: The Beauty in CSS Design
www.csszengarden.com
Google
Zen Garden
The Beauty of CSS Design
The Road to Enlightenment
A demonstration of what
Littering a dark and dreary road lay the past relics of browser-specific
Console
HTML
CSS
Script
DOM
Net
Cookies
Pixel ...
Events
Edit
a < span < p.p2 < div#quickSummary < div#intro < div#container < body#css-
Style
Computed
Layout
DOM
Events

```
    <link href="http://www.csszengarden.com/favicon.ico" type="image
    /x-icon" rel="Shortcut Icon">
    <link href="http://www.csszengarden.com
    /zengarden.xml" title="RSS" type="application
    /rss+xml" rel="alternate">
  + <script src="chrome://pixelperfect/content/browserscripts
    /dom-drag.js">
  + <script src="chrome://pixelperfect/content/browserscripts
    /publicDocumentEvents.js">
  </head>
- <body id="css-zen-garden">
  - <div id="container">
    - <div id="intro">
      + <div id="pageHeader">
      - <div id="quickSummary">
        + <p class="p1">
```

```
a:link {                          001.css (line 47)
    color: #B7A5DF;
    font-weight: bold;
    text-decoration: none;
}
Inherited from p.p2
#quickSummary p {                 001.css (line 113)
    ~~font~~: ~~italic 1.1em/2.2 georgia~~;
    text-align: center;
}
p {                               001.css (line 37)
    ~~text-align~~: ~~justify~~;
}
Inherited from body#css-zen-garden
body {                            001.css (line 29)
```

Firebug is useful for all web developers working on any web technology. In today's world, web developers have to deal with different kinds of UIs and JavaScript bugs, and many of them could take lot of time to fix them as you need to re-deploy the web application to view the changes on browser (this depends upon the technology). Firebug provides you with the control and power to tweak HTML and CSS and view the changed UI on the fly by using its featured panels. Panels are like the golf clubs to deal with any kind of UI issue or JavaScript issue. In the preceding screenshot, you can see different panels that are used to fight with bugs. The **Console**, **HTML**, **CSS**, **Script**, **DOM**, **Net**, and **Cookies** panel are defaults that come bundled with Firebug 1.10.

Installation

Firebug is developed as a Firefox add-on and can be installed on Firefox, similar to all other add-ons. In order to use some of Firebug's features in non-Firefox browsers, a JavaScript version called Firebug Lite is available.

Installing Firebug on Firefox

In five easy steps, you can install Firebug on Mozilla Firefox and set it up on your system.

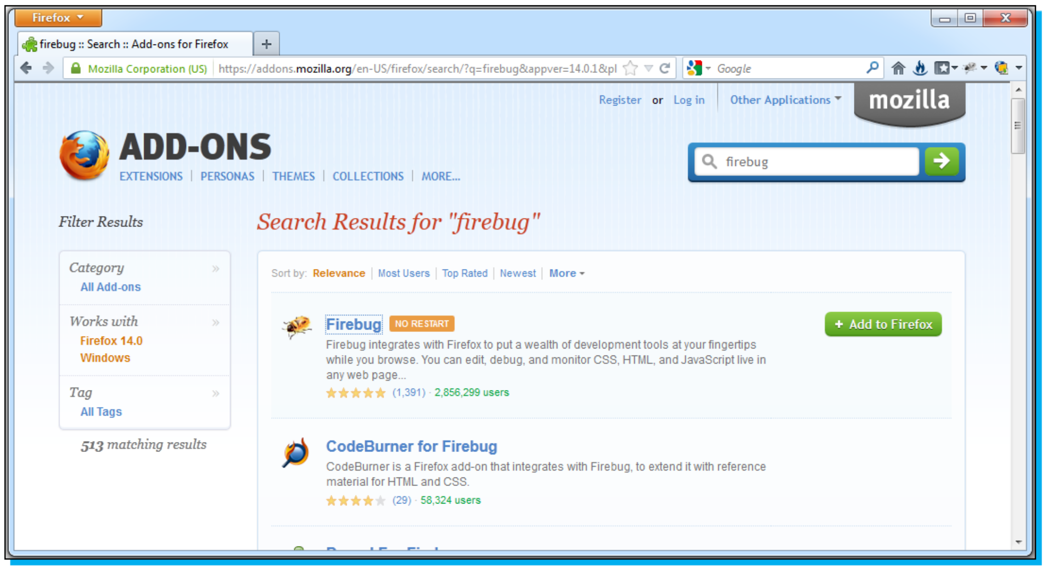
Step 1 – what do I need?

Before you install Firebug, you will need to check that you have all of the required elements, which are as follows:

- Disk space: 300MB free (min). You will require more free space to install Firebug extensions for other monitoring purposes.
- Memory: 512MB (min), 2GB (recommended when using Firefox with Firebug debugger).
- Firebug requires Mozilla Firefox web browser to be installed on the system as Firebug is an add-on of Mozilla Firefox.

Step 2 – downloading Firebug

The easiest way to download Firebug is from Mozilla's add-on site `https://addons.mozilla.org/en-US/firefox`, or you can also download it from Firebug's official site `https://getfirebug.com/`, which is updated sooner than Mozilla's add-on site.


Firefox
firebug :: Search :: Add-ons for Firefox
Mozilla Corporation (US)
https://addons.mozilla.org/en-US/firefox/search/?q=firebug&appver=14.0.1&pl
Google
Register or Log in
Other Applications
mozilla
ADD-ONS
EXTENSIONS | PERSONAS | THEMES | COLLECTIONS | MORE...
firebug
Filter Results
Category
All Add-ons
Works with
Firefox 14.0
Windows
Tag
All Tags
513 matching results
Sort by: Relevance | Most Users | Top Rated | Newest | More
NO RESTART
+ Add to Firefox

Search Results for "firebug"

Firebug
Firebug integrates with Firefox to put a wealth of development tools at your fingertips while you browse. You can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page...
★★★★★ (1,391) · 2,856,299 users

CodeBurner for Firebug
CodeBurner is a Firefox add-on that integrates with Firebug, to extend it with reference material for HTML and CSS.
★★★★ (29) · 58,324 users

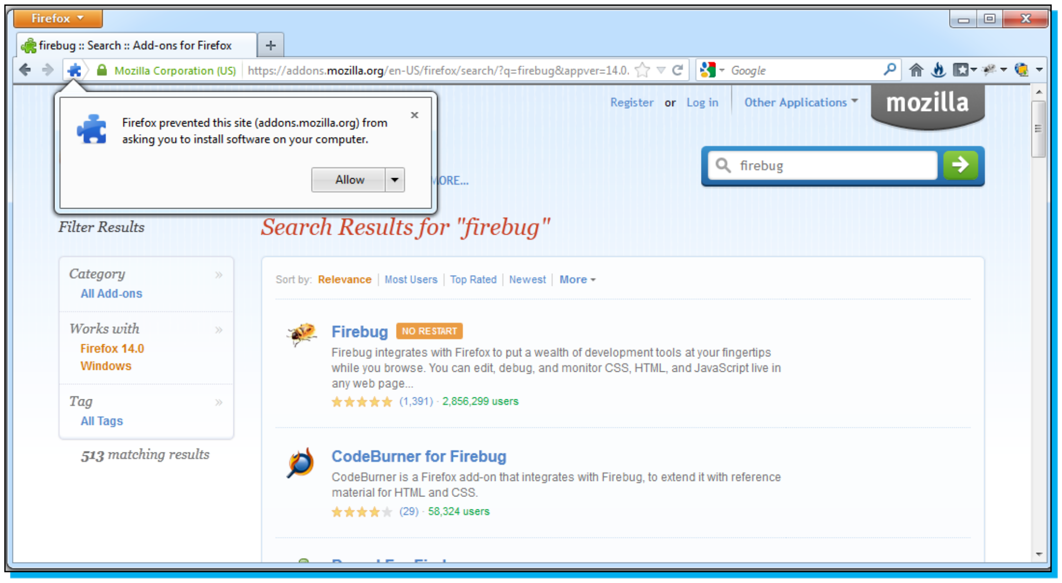
Step 3 – adding Firebug to Firefox

In the search box of the add-on site, type `Firebug` and press *Enter* or click on the green arrow.

In the search results, click on the **Add to Firefox** button. The preceding screenshot shows the search result page.

Step 4 – confirm Firebug installation

As shown in the following image, a pop-up will appear asking whether you want to continue with the installation. You need to click on **Allow**.



Firefox prevented this site (addons.mozilla.org) from asking you to install software on your computer.

Allow

Search Results for "firebug"

Filter Results

Category: All Add-ons
Works with: Firefox 14.0, Windows
Tag: All Tags
513 matching results

Sort by: Relevance | Most Users | Top Rated | Newest | More

Firebug NO RESTART
Firebug integrates with Firefox to put a wealth of development tools at your fingertips while you browse. You can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page...
(1,391) · 2,856,299 users

CodeBurner for Firebug
CodeBurner is a Firefox add-on that integrates with Firebug, to extend it with reference material for HTML and CSS.
(29) · 58,324 users

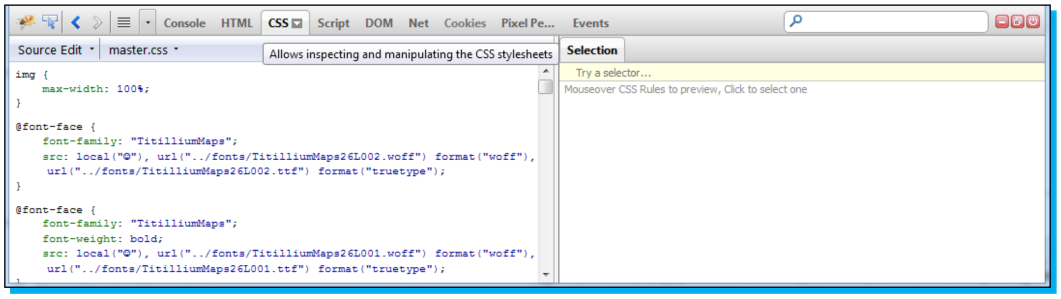
Firefox
firebug :: Search :: Add-ons for Firefox
Mozilla Corporation (US) https://addons.mozilla.org/en-US/firefox/search/?q=firebug&appver=14.0.
Google
Register or Log in
Other Applications
mozilla
firebug
MORE...

Step 5 – restart Firefox

This step is an optional step in the newer versions (since 1.10a2) of Firebug. If you are using an older version (before 1.10a2) of Firebug then it will ask you to restart the Firefox browser after completing the installation.

Installing Firebug on non-Firefox browsers

As you already know that Firebug is not limited to the Firefox browser you can use Firebug Lite version for non-Firefox browsers (IE6 and higher, Firefox, Opera, Safari, and Chrome) too. Firebug Lite is a product that can be easily included in your web page by including a JavaScript (or via a bookmarklet), just like any other JavaScript, to support all non-Firefox browsers.


Console
HTML
CSS
Script
DOM
Net
Cookies
Pixel Pe...
Events
Source Edit
master.css
Allows inspecting and manipulating the CSS stylesheets
Selection
Try a selector...
Mouseover CSS Rules to preview, Click to select one

```
img {
    max-width: 100%;
}

@font-face {
    font-family: "TitilliumMaps";
    src: local("☺"), url("../fonts/TitilliumMaps26L002.woff") format("woff"),
     url("../fonts/TitilliumMaps26L002.ttf") format("truetype");
}

@font-face {
    font-family: "TitilliumMaps";
    font-weight: bold;
    src: local("☺"), url("../fonts/TitilliumMaps26L001.woff") format("woff"),
     url("../fonts/TitilliumMaps26L001.ttf") format("truetype");
```

You can use Firebug Lite in three different ways, which will be described in the following section.

Live link

It's a link to a copy of the stable application (Firebug Lite) hosted on Firebug's servers. Include the following code within the `<head>` tag of your page:

```
<script type="text/javascript" src="https://getfirebug.com/firebug-
lite.js"></script>
```

Local link

If you want to use Firebug Lite as a local link then you need to download Firebug Lite from `https://getfirebug.com/releases/lite/latest/firebug-lite.tar.tgz` and then include the following code at the top of the `<head>` tag of your web page:

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

Path `/local/path/to/firebug-lite.js` is just an example. You need to provide the path where you have downloaded or kept Firebug Lite's JavaScript file.

Bookmarklet

With this way you don't need to modify your application's source code. You can run Firebug Lite by creating a bookmark with the value of the URL as the following JavaScript code:

```
javascript:(function(F,i,r,e,b,u,g,L,I,T,E){if(F.getElementById(b))
return;E=F[i+'NS']&&F.documentElement.namespaceURI;E=E?F[i+'NS']
(E,'script'):F[i]('script');E[r]('id',b);E[r]('src',I+g+T);E[r]
(b,u);(F[e]('head')[0]||F[e]('body')[0]).appendChild(E);E=new%20
Image;E[r]('src',I+L);})(document,'createElement','setAttribute','get
ElementsByTagName','FirebugLite','4','firebug-lite.js','releases/lite/
latest/skin/xp/sprite.png','https://getfirebug.com/','#startOpened');
```

For more information and updates on Firebug Lite, refer to `https://getfirebug.com/firebuglite`.

Based on your browser version, you can install the corresponding Firebug version:

- Firebug 1.7.3 with Firefox 3.6
- Firebug 1.7.3 with Firefox 4.0
- Firebug 1.8.2 (and also Firebug 1.7.3) with Firefox 5.0
- Firebug 1.8.2 (and also Firebug 1.9) with Firefox 6.0
- Firebug 1.8.2 (and also Firebug 1.9) with Firefox 7.0
- Firebug 1.8.3 (and also Firebug 1.9) with Firefox 8.0
- Firebug 1.8.4 (and also Firebug 1.9) with Firefox 9.0
- Firebug 1.9 with Firefox 10.0
- Firebug 1.9 with Firefox 11.0
- Firebug 1.9 with Firefox 12.0
- Firebug 1.10 (and also Firebug 1.9) with Firefox 13.0
- Firebug 1.10 with Firefox 14.0, Firefox 15.0, Firefox 16.0, and Firefox 17.0
- Firebug Lite with IE, Safari, Chrome, and Opera

And that's it!

By this point, you should have a working installation of Firebug or Firebug Lite, and are free to play around and discover more about it. Installing Firebug is as simple as installing any other add-on or extension of Firefox. Firebug Lite is nothing but a JavaScript, which can be included in any of your web pages to use some of the features of Firebug in non-Firebug browsers. In the next section, you will be given an overview of the most important features and panels in Firebug. You will know what power Firebug has and how useful it is.

Quick start – Firebug window overview and inspecting

Firebug is not about fixing the bugs and tweaking **CSS** (**cascade style sheet**), it consists of many tools that can be of great help to a web developer and designer. Firebug is similar golf club bag. Each club (panel) in Firebug is a powerful weapon for web developers. Like a golf player, a web developer has to choose a club (panel) for different situations. These weapons are:

- Console panel
- HTML panel
- CSS panel
- Script panel
- DOM panel
- Net panel
- Cookies panel

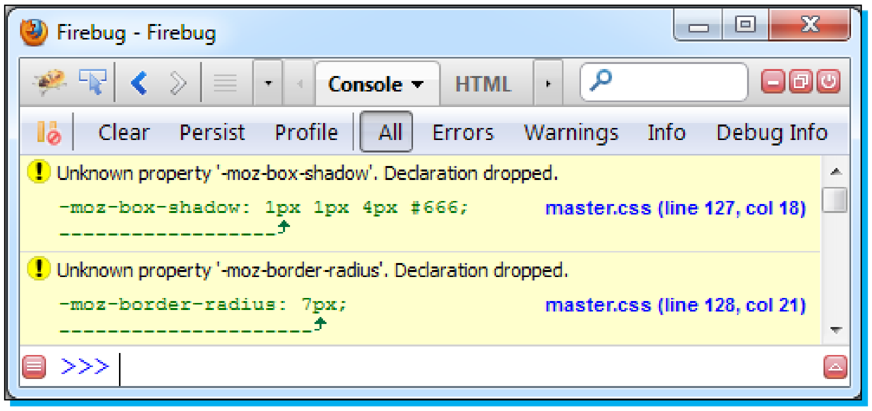
Console panel

The console panel offers a JavaScript command line. The JavaScript command line is a very powerful weapon of Firebug. This feature provides you with the power of executing arbitrary JavaScript expressions and commands on the fly at the command line without even reloading the document. You can validate and execute any JavaScript on the command line before integrating it on your web page.

When something goes wrong, Firebug will quickly let you know the details and information about:

- JavaScript errors and warnings
- CSS errors
- XML errors
- External errors
- Chrome errors and messages
- Network errors
- Strict warnings (performance penalty)
- Stack trace with errors
- XHR (`xmlHttpRequest`) info

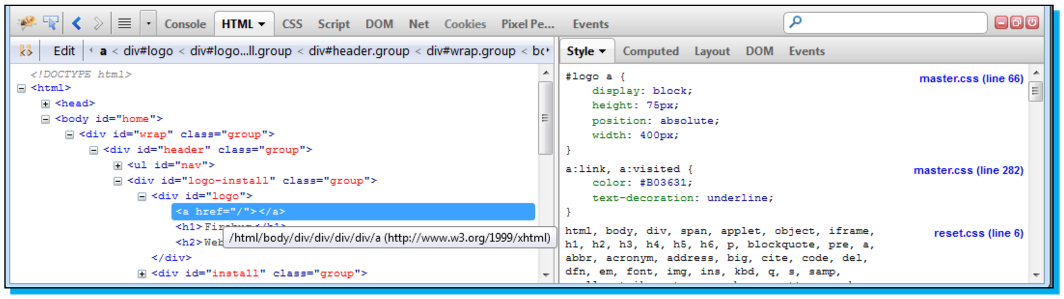
Some additional panels (for example, Pixel Perfect and Events) can be seen in the screenshots in this book. These panels came from the Firebug extensions, Pixel Perfect, and Eventbug.



```
Unknown property '-moz-box-shadow'. Declaration dropped.
  -moz-box-shadow: 1px 1px 4px #666;          master.css (line 127, col 18)
  ------------------^
Unknown property '-moz-border-radius'. Declaration dropped.
  -moz-border-radius: 7px;                    master.css (line 128, col 21)
  ---------------------^
```

HTML panel

The HTML panel displays the generated HTML of the currently opened web page. It allows you to edit HTML on the fly and play with your HTML **DOM** (**document object model**) in Firefox. It differs from the normal source code view, because it also displays all manipulations on the DOM tree. On the right-hand side it shows the CSS styles defined for the currently selected tag, the computed styles for it, layout information, and the DOM variables assigned to it under different tabs.

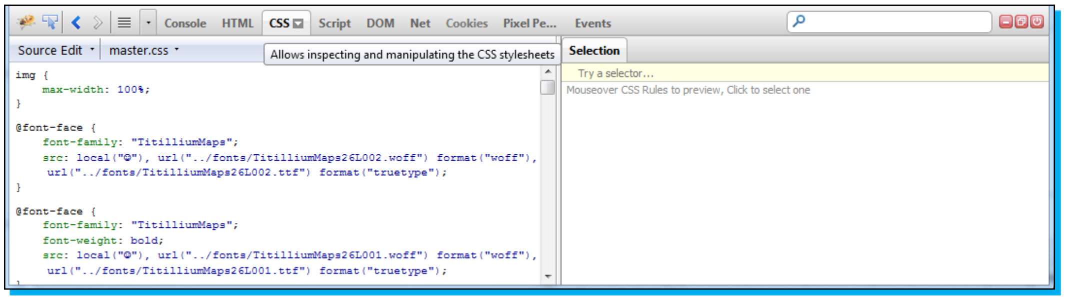

Console
HTML
CSS
Script
DOM
Net
Cookies
Pixel Pe...
Events
Edit
a < div#logo < div#logo...ll.group < div#header.group < div#wrap.group < bo
Style
Computed
Layout
DOM
Events
/html/body/div/div/div/div/a (http://www.w3.org/1999/xhtml)

```
<!DOCTYPE html>
<html>
  <head>
  <body id="home">
    <div id="wrap" class="group">
      <div id="header" class="group">
        <ul id="nav">
        <div id="logo-install" class="group">
          <div id="logo">
            <a href="/"></a>
            <h1>Fi
            <h2>Web
          </div>
          <div id="install" class="group">
```

```
#logo a {                                          master.css (line 66)
    display: block;
    height: 75px;
    position: absolute;
    width: 400px;
}
a:link, a:visited {                                master.css (line 282)
    color: #B03631;
    text-decoration: underline;
}
html, body, div, span, applet, object, iframe,     reset.css (line 6)
h1, h2, h3, h4, h5, h6, p, blockquote, pre, a,
abbr, acronym, address, big, cite, code, del,
dfn, em, font, img, ins, kbd, q, s, samp,
```

CSS panel

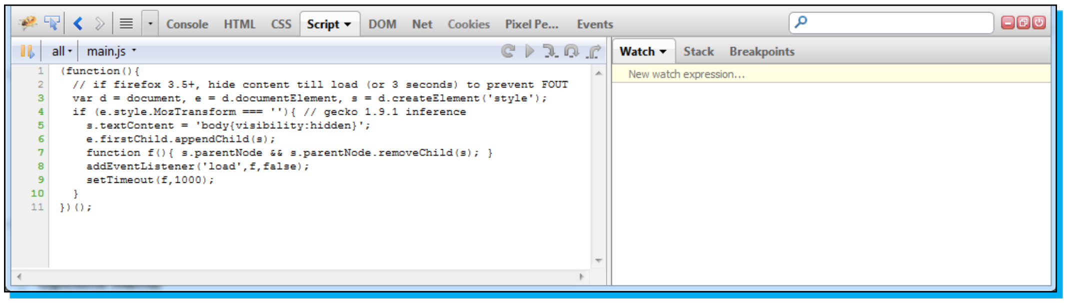
When you want to alter CSS rules, the CSS panel is the right place for this. It allows the user to tweak the CSS stylesheet in his/her taste. You can use this panel for viewing/editing CSS stylesheets on the fly and view the results live on the current document. This tab is mostly used by CSS developers to tweak the pixels, position, look and feel, or area of an HTML element. This tab is also useful for web developers when they want to view those elements whose CSS property **display** is set to **none**. Furthermore it offers an exclusive editing mode, in which you can edit the content of the CSS files directly via a text area.


Console
HTML
CSS
Script
DOM
Net
Cookies
Pixel Pe...
Events
Source Edit
master.css
Allows inspecting and manipulating the CSS stylesheets
Selection
Try a selector...
Mouseover CSS Rules to preview, Click to select one

```
img {
    max-width: 100%;
}

@font-face {
    font-family: "TitilliumMaps";
    src: local("☺"), url("../fonts/TitilliumMaps26L002.woff") format("woff"),
     url("../fonts/TitilliumMaps26L002.ttf") format("truetype");
}

@font-face {
    font-family: "TitilliumMaps";
    font-weight: bold;
    src: local("☺"), url("../fonts/TitilliumMaps26L001.woff") format("woff"),
     url("../fonts/TitilliumMaps26L001.ttf") format("truetype");
```

Script panel

The Script panel is the next gem that Firebug provides for us. It allows you to debug JavaScript code on the browser. The script panel integrates a powerful debugging tool based on features such as different kinds of breakpoints, step-by-step execution of scripts, a display for the variable stack, watch expressions, and more.



```
(function(){
  // if firefox 3.5+, hide content till load (or 3 seconds) to prevent FOUT
  var d = document, e = d.documentElement, s = d.createElement('style');
  if (e.style.MozTransform === ''){ // gecko 1.9.1 inference
    s.textContent = 'body{visibility:hidden}';
    e.firstChild.appendChild(s);
    function f(){ s.parentNode && s.parentNode.removeChild(s); }
    addEventListener('load',f,false);
    setTimeout(f,1000);
  }
})();
```

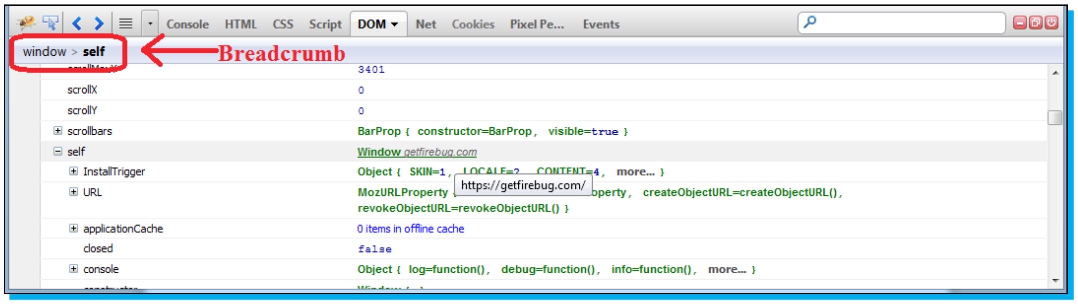
Things you can perform under the Script panel

A few tasks that you can perform under the Script panel and play with JavaScript are as follows:

- ✦ View the list of JavaScript files that are loaded with your document
- ✦ Debug the JavaScript code
- ✦ Insert and remove breakpoints
- ✦ Insert and remove conditional breakpoints
- ✦ View the stack trace
- ✦ View list of breakpoints
- ✦ Add new Watch expressions
- ✦ Keep an eye on the watches for viewing the values of variables
- ✦ Debug an Ajax call

DOM panel

HTML elements are also known as DOM elements. **Document Object Model** (**DOM**) represents the hierarchy of the HTML elements and functions. You can traverse in any direction within the DOM using JavaScript. The DOM elements have parent, child, and sibling relationships between them therefore for objects and arrays it offers an expandable tree view. Clicking on them (objects and arrays) limits the display of the DOM tree to the context of these objects and shows the current element path as a breadcrumb list on the DOM panel's toolbar.



```
                        3401
scrollX                 0
scrollY                 0
⊞ scrollbars            BarProp { constructor=BarProp, visible=true }
⊟ self                  Window getfirebug.com
  ⊞ InstallTrigger      Object { SKIN=1, LOCALE=2, CONTENT=4, more... }
  ⊞ URL                 MozURLProperty { https://getfirebug.com/ operty, createObjectURL=createObjectURL(),
                        revokeObjectURL=revokeObjectURL() }
  ⊞ applicationCache    0 items in offline cache
    closed              false
  ⊞ console             Object { log=function(), debug=function(), info=function(), more... }
```

The DOM panel in Firebug shows:

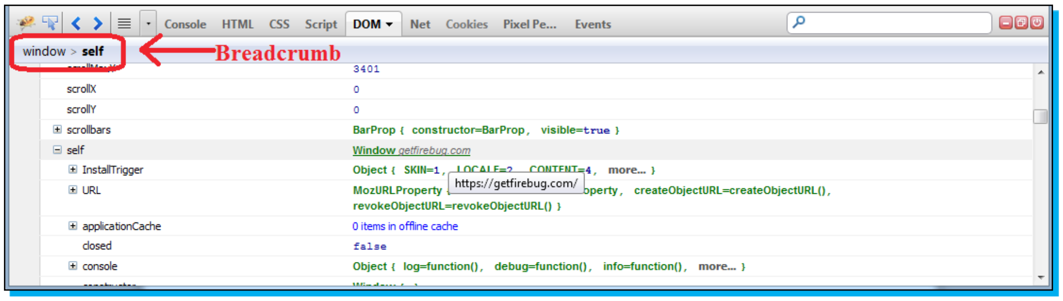
- DOM properties
- DOM functions
- DOM constants
- User-defined properties
- User-defined functions
- Inline event handlers
- Enumerable properties
- Own properties

Net panel

The Net panel allows you to monitor the network traffic initiated by your web page. It shows all collected and computed information in a tabular form to the user. Each row in the table represents one request or response round trip made by the web page. You can quickly measure the performance of your web page by analyzing the following information:

- Time taken to load the page
- The size of each file
- The loading time of each file
- The number of requests the browser sends to load the page

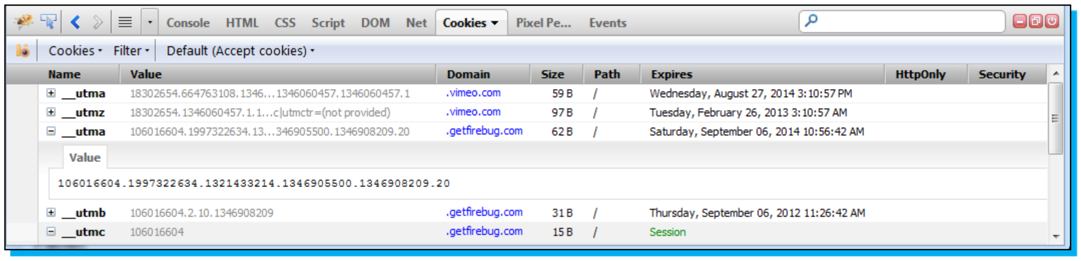
The Net panel also provides the facility to filter the type of request or response that you want to focus on. You can choose filters to view only **HTML**, **CSS**, **JS**, **Image**, **Flash**, **Media**, and even **XHR** (Ajax) requests.



```
                     3401
scrollX              0
scrollY              0
⊞ scrollbars         BarProp { constructor=BarProp, visible=true }
⊟ self               Window getfirebug.com
  ⊞ InstallTrigger   Object { SKIN=1, LOCALE=2, CONTENT=4, more... }
  ⊞ URL              MozURLProperty {                    operty, createObjectURL=createObjectURL(),
                     revokeObjectURL=revokeObjectURL() }
  ⊞ applicationCache 0 items in offline cache
    closed           false
  ⊞ console          Object { log=function(), debug=function(), info=function(), more... }
```

Cookies panel

The Cookies panel allows you to view, manage, and debug cookies within your browser via Firebug. Within the context of the current domain, this panel displays a list of all cookies associated with the current web page. Each entry in the list shows the basic information about a cookie (**Name**, **Value**, **Domain**, **Path**, **Expires**, and so on).



Name	Value	Domain	Size	Path	Expires	HttpOnly	Security
⊞ __utma	18302654.664763108.1346...1346060457.1346060457.1	.vimeo.com	59 B	/	Wednesday, August 27, 2014 3:10:57 PM		
⊞ __utmz	18302654.1346060457.1.1...c|utmctr=(not provided)	.vimeo.com	97 B	/	Tuesday, February 26, 2013 3:10:57 AM		
⊟ __utma	106016604.1997322634.13...346905500.1346908209.20	.getfirebug.com	62 B	/	Saturday, September 06, 2014 10:56:42 AM		
⊞ __utmb	106016604.2.10.1346908209	.getfirebug.com	31 B	/	Thursday, September 06, 2012 11:26:42 AM		
⊟ __utmc	106016604	.getfirebug.com	15 B	/	Session		

Value

```
106016604.1997322634.1321433214.1346905500.1346908209.20
```

Inspecting page components

Sometimes, something in your page doesn't quite look correct and you want to know why; you want to understand how a particular section of the page has been constructed. Firebug's *Inspect* functionality allows you to do that in the fastest possible manner using minimal clicks or mouse strokes. As you move around the web page, Firebug creates visual frames around various page components/sections and shows the HTML and CSS behind the page section. In fact, this is one of the most frequently used features while tweaking the look and feel of your web application. Inspecting page elements and doing tweaks (editing HTML attributes, CSS rules, and so on) are real time savers during the fine-tuning of a web application's look and feel.

In this section, you will learn how to perform one of the core tasks of Firebug; inspecting and editing HTML. The following steps will help you perform this task.

You can activate Firebug by pressing the *F12* key or by clicking on the (bug) icon on the top-right side of the browser's toolbar. Similarly, you can close or deactivate Firebug by pressing the *F12* key or by clicking on the (bug) again.

Step 1 – open Firebug

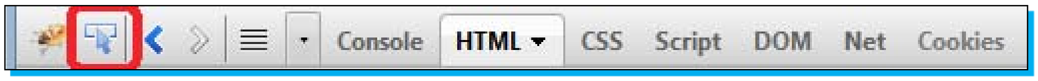
To start inspection of your web page, you first need to activate Firebug.

- Go to your web application's page in the browser
- Press the *F12* key or click on the (bug) icon on the top-right side of the Firefox browser's toolbar

Step 2 – activate inspection tool

In this step, you activate the inspection tool and search for the block/component/HTML element in your web page using the mouse:

- Click on the **Inspect** button on the Firebug console toolbar (the inspect button is like a mouse arrow pointer on the toolbar of Firebug), as shown in the following screenshot:

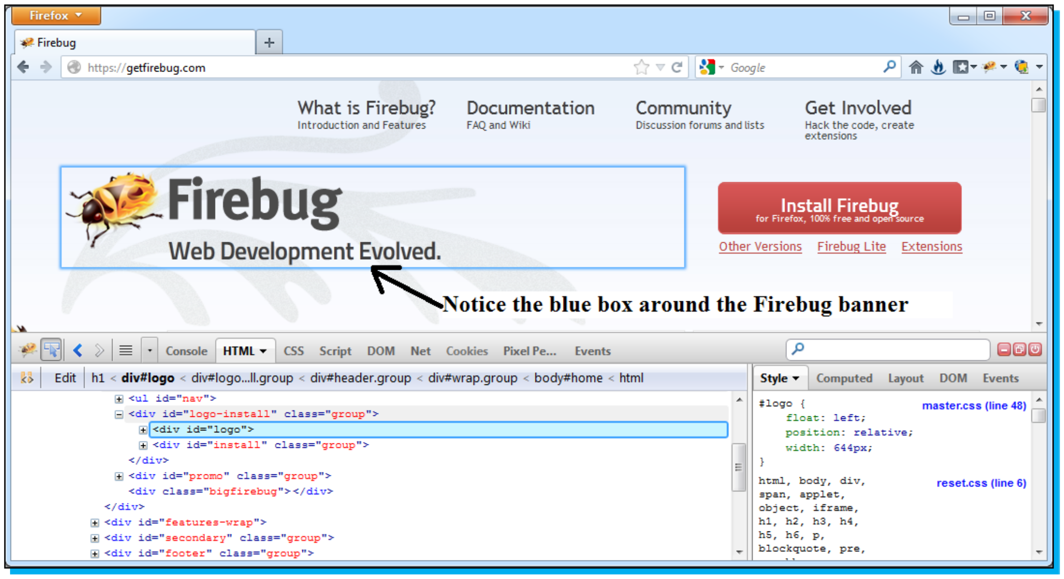


- Move your mouse cursor over the page component/section that you want to inspect

Step 3 – selecting the element

While moving the mouse pointer, a visual frame is created around the element and the HTML source of the element is shown highlighted in the Firebug window. Now, after activating the inspection tool, select the block/component/HTML element to inspect.

Click on the page component/section to select the element in the HTML panel.


Firefox
Firebug
https://getfirebug.com
What is Firebug? Introduction and Features
Documentation FAQ and Wiki
Community Discussion forums and lists
Get Involved Hack the code, create extensions
Firebug
Web Development Evolved.
Install Firebug for Firefox, 100% free and open source
Other Versions
Firebug Lite
Extensions
Notice the blue box around the Firebug banner
Console
HTML
CSS
Script
DOM
Net
Cookies
Pixel Pe...
Events
Edit
h1 < div#logo < div#logo...ll.group < div#header.group < div#wrap.group < body#home < html
Style
Computed
Layout
DOM
Events

```
<ul id="nav">
<div id="logo-install" class="group">
    <div id="logo">
    <div id="install" class="group">
</div>
<div id="promo" class="group">
<div class="bigfirebug"></div>
</div>
<div id="features-wrap">
<div id="secondary" class="group">
<div id="footer" class="group">
```

```
#logo {                          master.css (line 48)
    float: left;
    position: relative;
    width: 644px;
}
html, body, div,                 reset.css (line 6)
span, applet,
object, iframe,
h1, h2, h3, h4,
h5, h6, p,
blockquote, pre,
```

Quick inspect

You can even inspect an element with minimal fuss by right-clicking on the element in the Firefox window and selecting the **Inspect Element with Firebug** option from the context menu. This allows you to *inspect* an element without first activating the Firebug window. In this way you can inspect an element with minimal mouse-clicks when the Firebug window is not already activated.

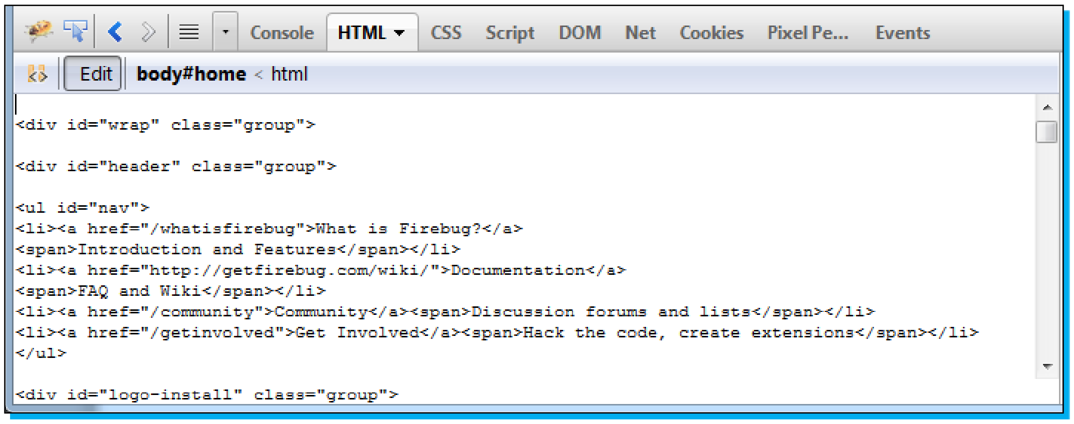
Editing page components

Firefox and most other browsers have a feature for viewing the source of the HTML document sent by the server. Firebug's HTML panel shows you what the HTML *currently* looks like. In addition to the HTML panel, there are three tabs on the right which let you view and modify properties on an individual element, including the CSS rules that are being applied to the element, the layout of the element in the document, and its DOM properties.

Firebug's HTML panel has more advanced features than the default **View Source** of Firefox. It shows the HTML DOM in a hierarchical structure or tree view with a highlight color. It allows you to expand or collapse the HTML DOM to navigate and provides easy visualization of the HTML page. It is a *viewer* as well as an *editor*, and it allows you to edit/delete the HTML elements or attributes on the fly and the changes are immediately applied to the page being currently viewed and rendered by the browser.

In order to edit the HTML source on a web page, do the following steps:

- Open Firebug.
- Click on the **HTML** tab. This will show the source of the document.
- Simply click on the **Edit** button under the HTML panel. On clicking this button, the HTML panel turns into an editable text area. You can now edit the HTML and see it taking effect instantly as shown in the following screenshot:



```
<div id="wrap" class="group">

<div id="header" class="group">

<ul id="nav">
<li><a href="/whatisfirebug">What is Firebug?</a>
<span>Introduction and Features</span></li>
<li><a href="http://getfirebug.com/wiki/">Documentation</a>
<span>FAQ and Wiki</span></li>
<li><a href="/community">Community</a><span>Discussion forums and lists</span></li>
<li><a href="/getinvolved">Get Involved</a><span>Hack the code, create extensions</span></li>
</ul>

<div id="logo-install" class="group">
```

Conclusion

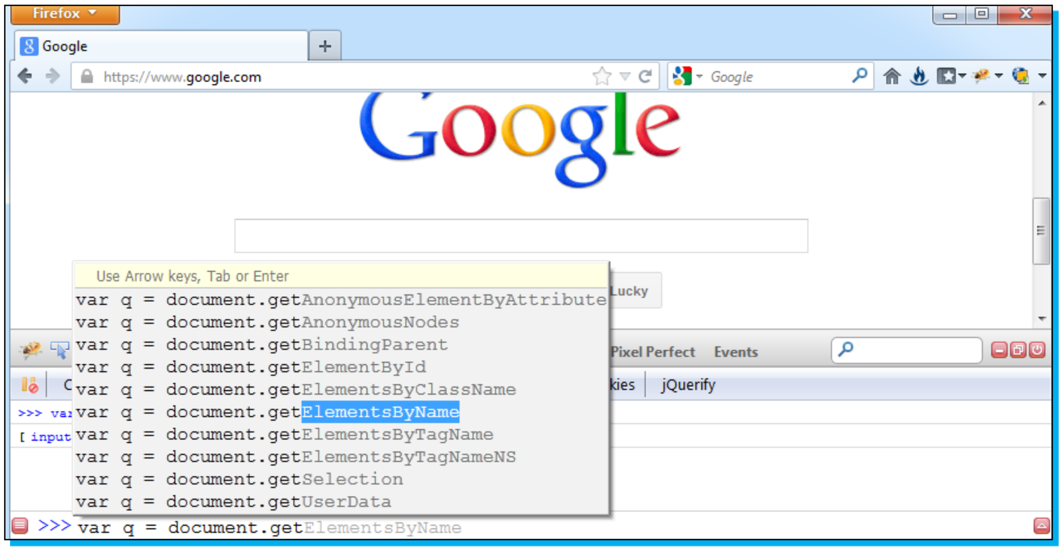
Firebug is armed with lot of weapons to deal with UI and JavaScript bugs. You have the JavaScript command line, the Console panel, HTML panel, CSS panel, Script panel, DOM panel, Net panel, and Cookies panel to tackle a variety of issues. Inspection is the most common task that you will perform almost every time you find yourself surrounded by hair pulling client side issues. Editing and viewing changes on the fly (without making changes on the server side code) is the specialty of Firebug.

Top 17 features you'll want to know about

As you start to use Firebug, you will realize that there are a wide variety of things that you can do with it. This section will teach you all about the most commonly performed tasks and most commonly used features in Firebug. So far you have learned how to inspect the elements on the web page, so let's take you to the next level, where you will see the awesomeness of Firebug.

Executing JavaScript on the fly

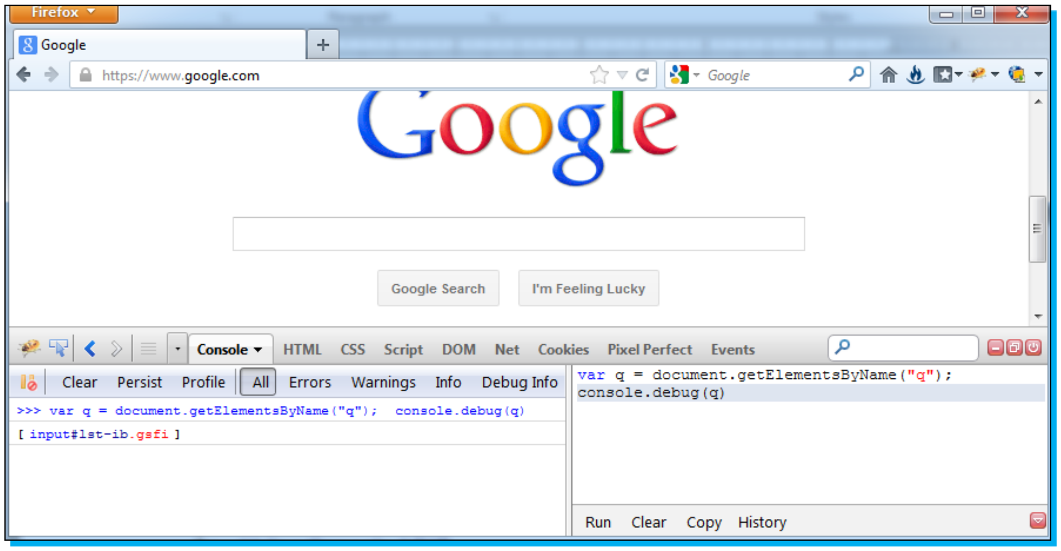
Firebug offers you a command line for executing JavaScript expressions. It is designed to support people writing code and inspecting objects. For this reason it offers several functionalities. Thereby, there are some additional features available inside the **Console** panel. It also offers a feature to autocomplete your JavaScript code. When you start typing a JavaScript command in the command line, then Firebug shows you a list of code suggestions as shown in the following screenshot:



```
var q = document.getAnonymousElementByAttribute
var q = document.getAnonymousNodes
var q = document.getBindingParent
var q = document.getElementById
var q = document.getElementsByClassName
var q = document.getElementsByName
var q = document.getElementsByTagName
var q = document.getElementsByTagNameNS
var q = document.getSelection
var q = document.getUserData
>>> var q = document.getElementsByName
```

Command editor

The command editor offers you a bigger text field for entering JavaScript commands consisting of several lines. It has an integrated menu for running the command(s), clearing the text field, and copying the contents to the clipboard.



```
>>> var q = document.getElementsByName("q");   console.debug(q)
[ input#lst-ib.gsfi ]
```

```
var q = document.getElementsByName("q");
console.debug(q)
```

Smart paste

If you are pasting code in the command line, which consists of several lines, the command editor is automatically opened to keep the line breaks.

Using Console API

Firebug adds a global variable named **console** to all web pages loaded in Firefox. This object contains many methods that allow us to write to the Firebug console to expose information that is flowing through your JavaScript.

For more details on the Console API visit `https://getfirebug.com/wiki/index.php/Console_API`.

console.log(object[, object, ...])

This method writes a message to the console. You may pass as many arguments as you want, and they will get joined together in a space-delimited line.

The first argument to log may be a string containing string substitution patterns. For example:

```
console.log("The %s jumped over %d tall buildings", animal, count);
```

The same example can be re-written without string substitution to achieve the same result:

```
console.log("The", animal, "jumped over", count, "tall buildings");
```

These two techniques can be combined. If we use string substitution but we need to provide more arguments than are in substitution patterns, the remaining arguments will be appended in a space-delimited line, as shown in the following code:

```
console.log("I am %s and I have:", myName, thing1, thing2, thing3);
```

If objects are logged, they will be written not as static text, but as interactive hyperlinks that can be clicked to inspect the object in Firebug's **HTML**, **CSS**, **Script**, or **DOM** panels. You may also use the `%o` pattern to substitute a hyperlink in a string.

Here is the complete set of patterns that we may use for string substitution:

String	Substitution Patterns
`%s`	String
`%d`, `%i`	Integer
`%f`	Floating point number
`%o`	Object hyperlink

console.debug(object[, object, ...])

This method writes a message to the console, including a hyperlink to the line where it was called.

console.info(object[, object, ...])

This method writes a message to the console with the visual *info* icon, color coding, and a hyperlink to the line where it was called.

console.warn(object[, object, ...])

This method writes a message to the console with the visual *warning* icon, color coding, and a hyperlink to the line where it was called.

console.error(object[, object, ...])

This method 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.assert(expression[, object, ...])

This method tests whether an expression is true. If not, it will write a message to the console and throw an exception.

console.dir(object)

This method prints an interactive listing of all properties of the object. This looks identical to the view that you would see in the **DOM** tab.

console.dirxml(node)

This method prints the XML source tree of an HTML or XML element. This looks identical to the view that you would see in the **HTML** tab. You can click on any node to inspect it in the **HTML** tab.

console.trace()

This method 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 on each function to take you to its source in the **Script** tab, and click on each argument value to inspect it in the **DOM** or **HTML** tabs.

console.group(object[, object, ...])

This method writes a message to the console and opens a nested block to indent all future messages sent to the console. Call `console.groupEnd()` to close the block.

console.groupCollapsed(object[, object, ...])

This method works just like `console.group()`, but the block is initially collapsed.

console.groupEnd()

This method closes the most recently opened block created by a call to `console.group()` or `console.groupCollapsed()`.

console.time(name)

This method creates a new timer under the given name. Call `console.timeEnd(name)` with the same name to stop the timer and print the time elapsed.

console.timeEnd(name)

This method stops a timer created by a call to `console.time(name)` and writes the time elapsed.

console.profile([title])

This turns on the JavaScript profiler. The optional argument title would contain the text to be printed in the header of the profile report.

console.profileEnd()

This method turns off the JavaScript profiler and prints its report.

> `console.profile()` and `console.profileEnd()` is explained later in the *JavaScript profiling* section.

console.count([title])

This method writes the number of times that the line of code where `count` call was executed. The optional argument `title` will print a message in addition to the number of the counts.

> The console is an object attached to the window object in the web page. The `Console` object is built-in Firefox and only the **Console** panel needs to be enabled to play with it. In Firebug Lite, the console is attached if Lite is installed in the page.

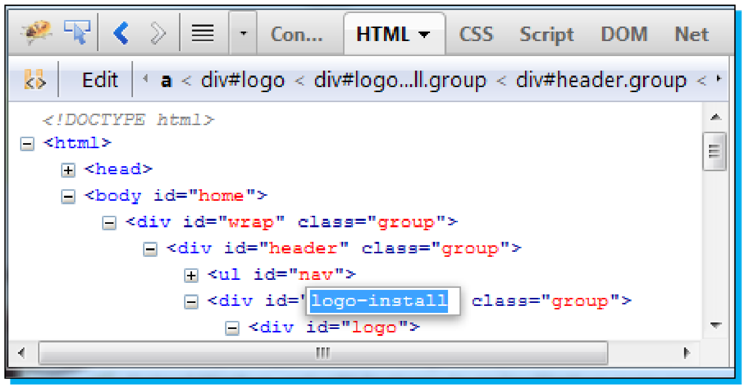
Modifying an HTML element's attribute on the fly

Firebug makes it really easy to make experimental HTML changes and see them take effect instantly. You can create, delete, modify HTML attributes, or create new HTML elements, or completely modify the source of the document.

This feature is most useful for scenarios where you have to make minor tweaks to any of the HTML attributes on the server side and then keep refreshing the browser page to see how the changes look. Firebug allows you to tweak and fine-tune your HTML attributes very easily, and the changes are applied immediately as you type. To modify an element's attributes on the fly carry out the following steps:

1. Inspect the element that you want to modify.
2. The **HTML** panel will show the source of that element in the document.

3. Simply click on the attribute value of the element that you want to modify. On clicking, it the attribute value turns into an editable textbox (seen in the following screenshot). You can modify the value and see it taking effect instantly.

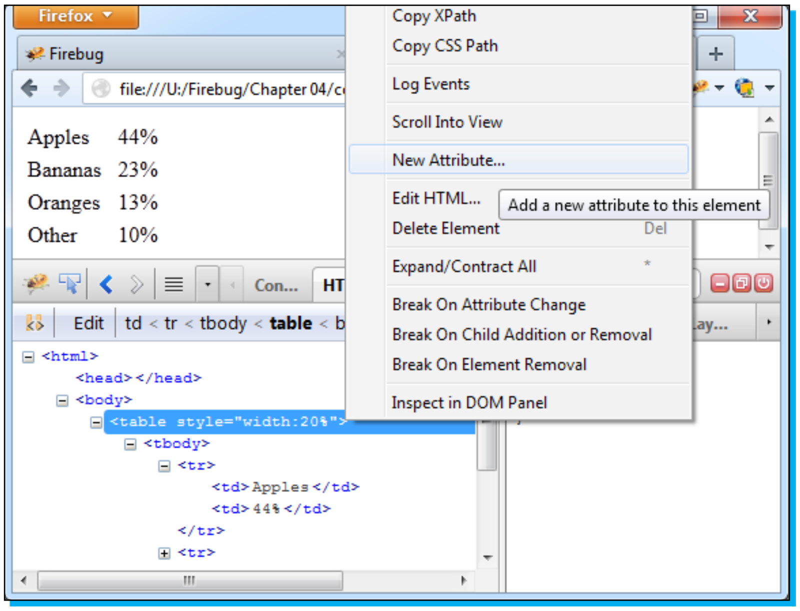


```
<!DOCTYPE html>
<html>
  <head>
  <body id="home">
    <div id="wrap" class="group">
      <div id="header" class="group">
        <ul id="nav">
        <div id="logo-install" class="group">
          <div id="logo">
```

Adding a new attribute to an existing HTML element

In this task you will be adding a new attribute to an existing HTML element in your web page via the **HTML** panel. To add a new HTML attribute to an element, do the following steps:

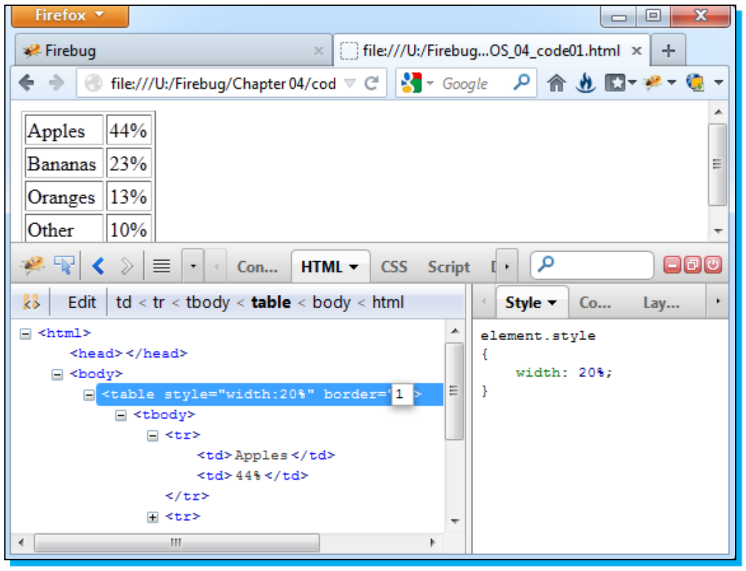
1. Open Firebug and locate the element (in the **HTML** panel) that you want to modify.
2. Right-click on the element and choose the **New Attribute...** option in the **Context** menu as shown in the following screenshot:



Apples	44%
Bananas	23%
Oranges	13%
Other	10%

```
<html>
  <head></head>
  <body>
    <table style="width:20%">
      <tbody>
        <tr>
          <td>Apples</td>
          <td>44%</td>
        </tr>
        <tr>
```

3. Enter the name of the attribute and press *Tab* and then enter the value for the attribute. The new attribute gets applied as you type, without you having to click outside the textbox or do anything else (see the following screenshot).


Firefox
Firebug
file:///U:/Firebug...OS_04_code01.html
file:///U:/Firebug/Chapter 04/cod
Google
Con...
HTML
CSS
Script
Edit
td < tr < tbody < **table** < body < html
Style
Co...
Lay...

Apples	44%
Bananas	23%
Oranges	13%
Other	10%

```
<html>
    <head></head>
    <body>
        <table style="width:20%" border="1">
            <tbody>
                <tr>
                    <td>Apples</td>
                    <td>44%</td>
                </tr>
                <tr>
```

```
element.style
{
    width: 20%;
}
```

Deleting an HTML element

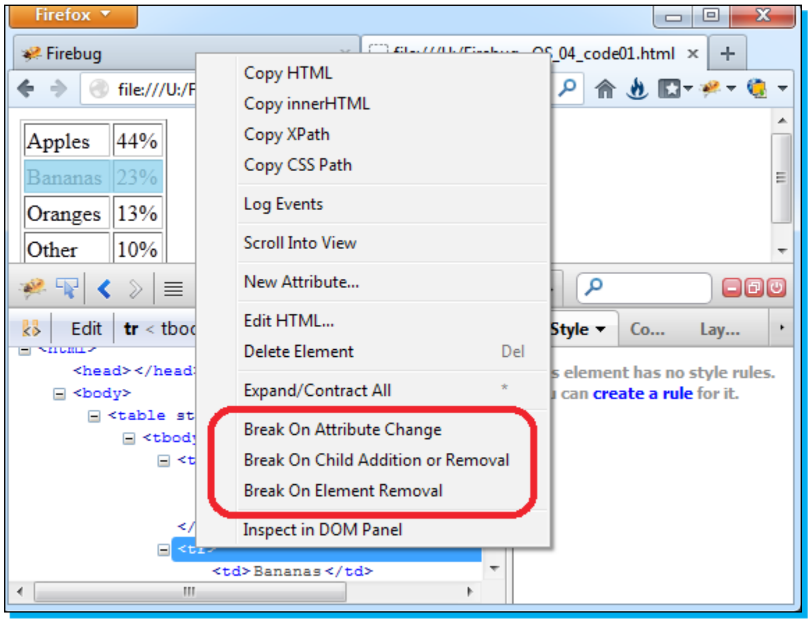
Deleting an element from the page means completely removing the HTML source of that element from the page. Firebug, then adjust the view accordingly on the fly. To delete an existing node in the HTML source tree, do the following steps:

1. Open Firebug and locate the element (in the **HTML** panel) that you want to delete.
2. Right-click on the node that you want to delete and choose **Delete Element** on the content menu or simply press the *Delete* button on the keyboard.

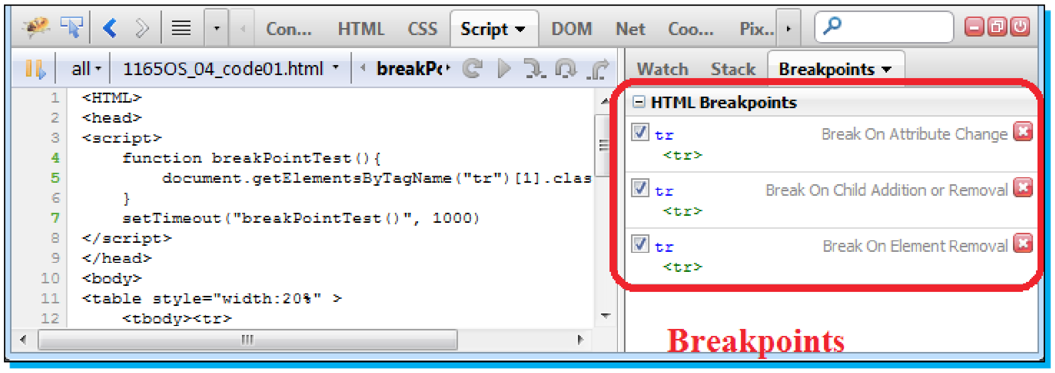
Setting breakpoints on an HTML element

You might have used breakpoints in your application's source code to debug it. Firebug also provides a unique feature to set breakpoints on HTML elements. These breakpoints are triggered whenever an attribute is changed, a child element is added/removed or the element itself is removed. The following steps explain how to set breakpoints on HTML elements:

1. Inspect the element on which you want the breakpoint to be inserted. Within the **HTML** panel, right-click on the element to open up the context menu. Choose options to **Break On Attribute Change**, or **Break On Child Addition or Removal**, or **Break On Element Removal** from the context menu. You can choose more than one option at a time, as shown in the following screenshot:

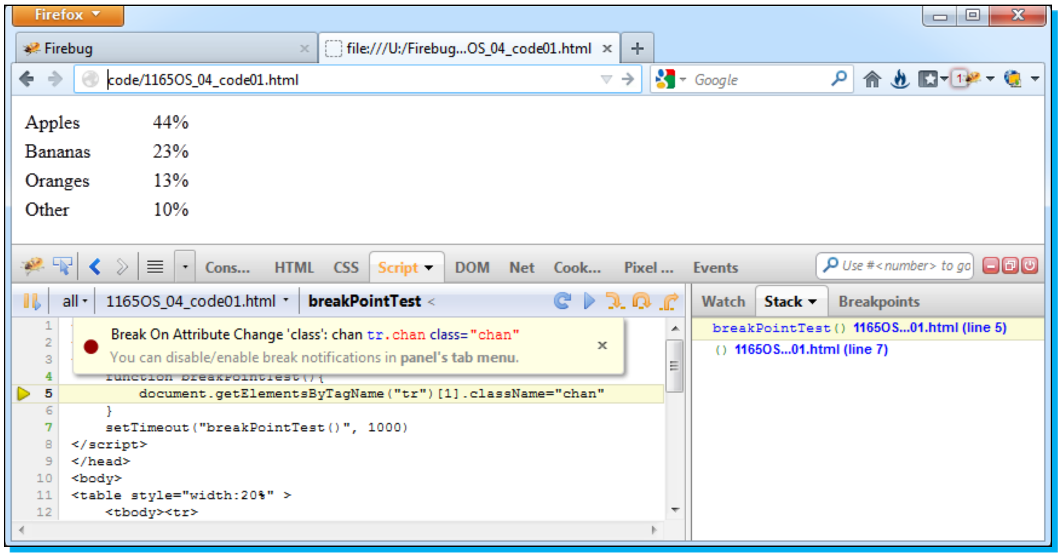


2. On selecting, Firebug will keep an eye on the behavior of node, as shown in the following screenshot:



```
<HTML>
<head>
<script>
    function breakPointTest(){
        document.getElementsByTagName("tr")[1].clas
    }
    setTimeout("breakPointTest()", 1000)
</script>
</head>
<body>
<table style="width:20%" >
    <tbody><tr>
```

3. Now, whenever the HTML (within the marked) node changes, Firebug will trigger the breakpoint and focus on the line in the JavaScript code whose execution is changing the DOM/HTML of the marked element.


Firefox
Firebug
file:///U:/Firebug...OS_04_code01.html
code/1165OS_04_code01.html
Google
Cons... HTML CSS Script DOM Net Cook... Pixel ... Events
Use #<number> to go
all
1165OS_04_code01.html
breakPointTest
Watch Stack Breakpoints

Apples	44%
Bananas	23%
Oranges	13%
Other	10%

Break On Attribute Change 'class': chan tr.chan class="chan"
You can disable/enable break notifications in panel's tab menu.

```
    function breakPointTest(){
        document.getElementsByTagName("tr")[1].className="chan"
    }
    setTimeout("breakPointTest()", 1000)
</script>
</head>
<body>
<table style="width:20%" >
    <tbody><tr>
```

breakPointTest() 1165OS...01.html (line 5)
() 1165OS...01.html (line 7)

By pressing *F10* (Step Over) or *F11* (Step Into), you can debug the JavaScript in the **Script** panel.

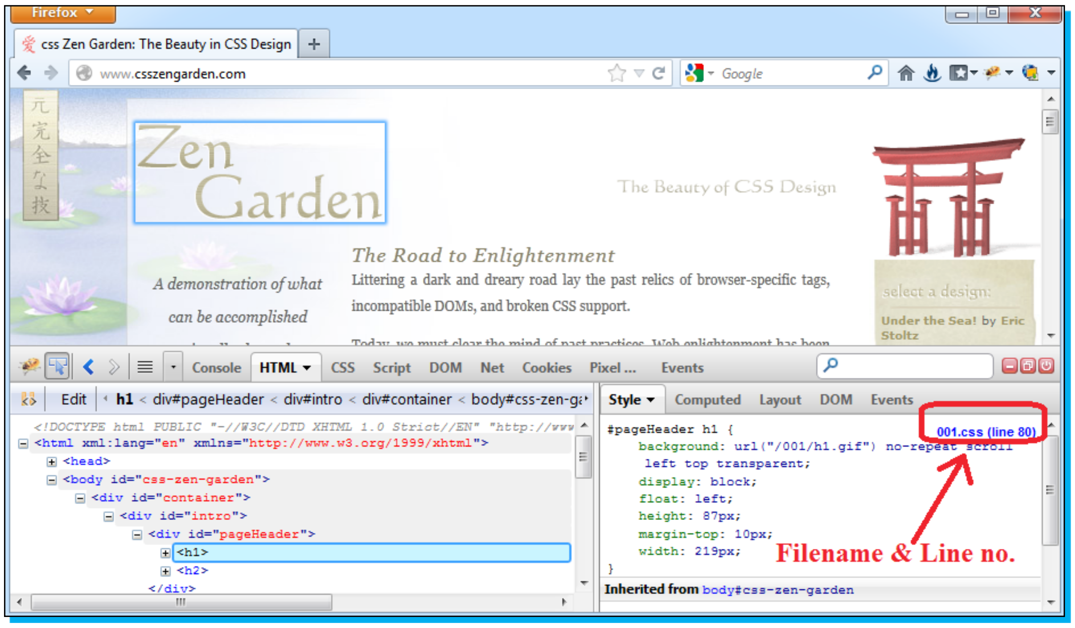
[JavaScript debugging is discussed later in the *JavaScript debugging* section.]

Inspecting cascading rules

With non-Firefox browsers, you left pulling your hair out when the color of a paragraph comes out red, instead of blue. Thank goodness for Firebug because with it you can easily inspect the problematic HTML element and before you can blink it will reach the CSS rule that is causing the problem.

Inspecting a CSS element is very simple. It is similar to inspecting an HTML element as described previously. For inspecting, we just need to open Firebug in inspect mode.

Directly open Firebug in inspect mode by pressing *Ctrl* + *Shift* + *C*.


css Zen Garden: The Beauty in CSS Design
www.csszengarden.com
Console HTML CSS Script DOM Net Cookies Pixel ... Events
Edit h1 < div#pageHeader < div#intro < div#container < body#css-zen-ga
Style Computed Layout DOM Events
001.css (line 80)
Filename & Line no.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <body id="css-zen-garden">
    <div id="container">
      <div id="intro">
        <div id="pageHeader">
          <h1>
          <h2>
        </div>
```

```
#pageHeader h1 {
    background: url("/001/h1.gif") no-repeat scroll
     left top transparent;
    display: block;
    float: left;
    height: 87px;
    margin-top: 10px;
    width: 219px;
}
Inherited from body#css-zen-garden
```

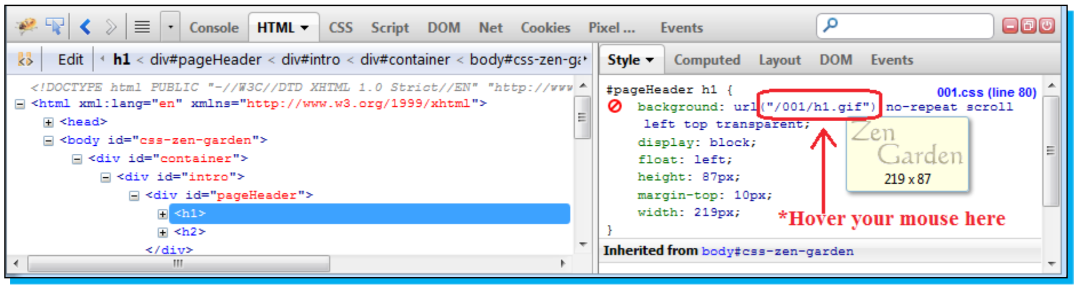
Follow the steps to inspect the CSS of HTML elements:

1. Open any site (in our case it's `http://www.csszengarden.com`) and press *Ctrl* + *Shift* + *C* (default shortcut) to open Firebug in inspect mode.
2. Move the mouse pointer over the HTML element on the page that you want to inspect as shown in the preceding screenshot (in our case it is the image on which **Zen Garden** is engraved). With the movement of the pointer of the mouse you can see a blue box beneath the pointer and whatever is wrapped in that box will be instantly revealed within Firebug, which shows HTML on the left panel and CSS on the right panel.
3. When you reach the (problematic) HTML element, click on that element. As soon as you do that, the box vanishes, and the HTML and CSS rules of that element will be shown.

Firebug shows the link, with the line number indicated, to the CSS file from which the CSS property is getting applied. When we click on the link, Firebug will switch to the **CSS** panel and take us to that file, and specifically to the line number indicated in the link.

Previewing colors and images

Firebug also allows you to preview images and colors defined in CSS. By just hovering your mouse pointer to the value of a CSS rule (let's discuss the `background` property), Firebug shows a handy tool tip that previews the image with its width and height as shown in the following screenshot:



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <body id="css-zen-garden">
    <div id="container">
      <div id="intro">
        <div id="pageHeader">
          <h1>
          <h2>
        </div>
```

```
#pageHeader h1 {                                    001.css (line 80)
    background: url("/001/h1.gif") no-repeat scroll
     left top transparent;
    display: block;
    float: left;
    height: 87px;
    margin-top: 10px;
    width: 219px;
}
Inherited from body#css-zen-garden
```

In a similar fashion, when you hover the mouse pointer over CSS's `color` property, Firebug shows a tool tip filled with the color of that property's value.

Tweaking CSS on the fly

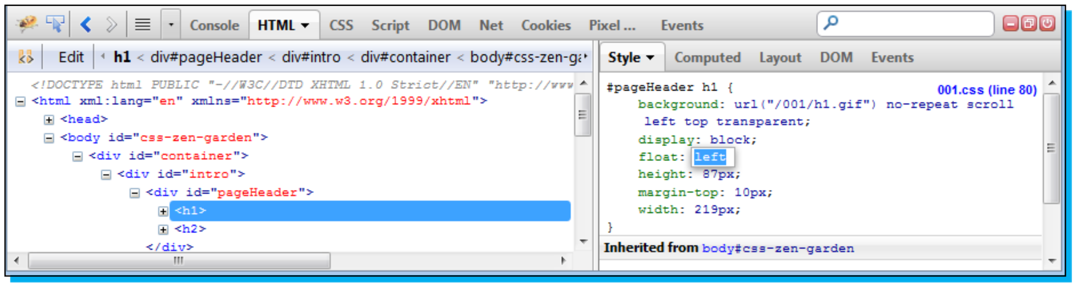
For editing CSS rules and properties, you used to have to edit in the CSS file associated with the document and reload the page to view changes. Now this is history. With the new generation tool, Firebug, you can edit CSS rules, tweak CSS properties on the fly, and view the live changes on the page instantly in real time. You don't need to reload the page each time changes are made to CSS files.

Firebug shows all the CSS rules that are impacting the selected HTML element and the CSS rules that an element inherits from its ancestor elements. If one of the CSS properties or styles is overridden, that rule and property is also shown by Firebug in strike (for example, `color : red`) fashion.

To tweak the CSS rule within Firebug, you need to follow these steps:

1. Inspect the (problematic) HTML element whose CSS rule is to be edited.

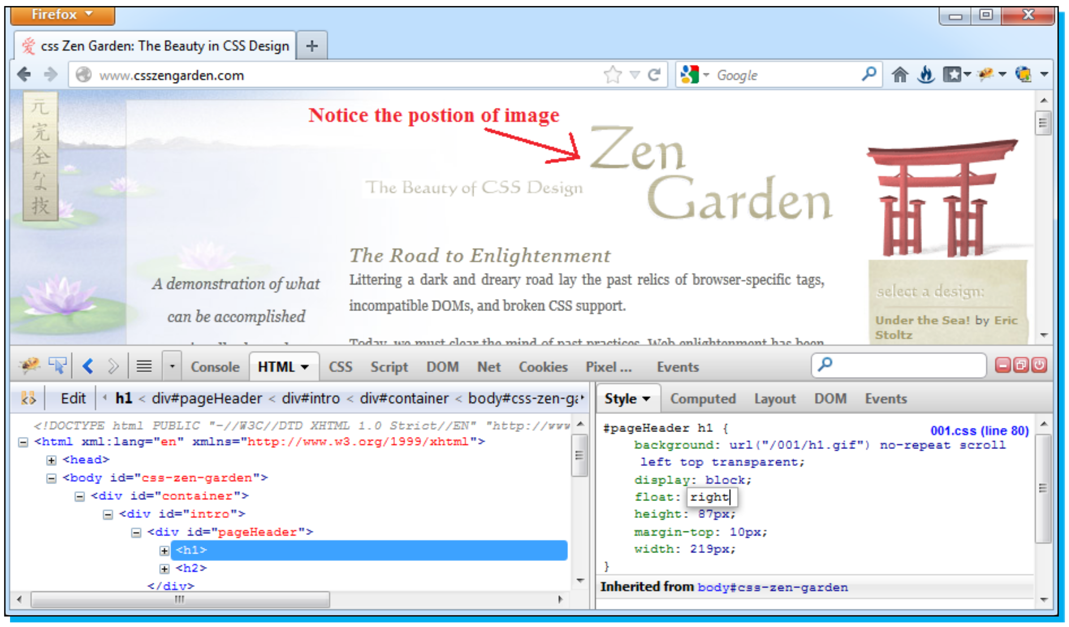
2. Click on the *value* of the CSS property under the **Style** panel of the **HTML** tab (in my case click on the value `left` of the CSS property `float`). As soon as we click on it, a little text box editor will appear as shown in the following screenshot:



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <body id="css-zen-garden">
    <div id="container">
      <div id="intro">
        <div id="pageHeader">
          <h1>
          <h2>
        </div>
```

```
#pageHeader h1 {                                    001.css (line 80)
    background: url("/001/h1.gif") no-repeat scroll
     left top transparent;
    display: block;
    float: left
    height: 87px;
    margin-top: 10px;
    width: 219px;
}
Inherited from body#css-zen-garden
```

3. Now, type `right` in place of `left`. The result can be seen instantly on the page; the HTML element `<h1>` moved to the right-hand side of the page without having to reload the page, as shown in the following screenshot:


Firefox
css Zen Garden: The Beauty in CSS Design
www.csszengarden.com
Google
Notice the postion of image
Zen Garden
The Beauty of CSS Design
The Road to Enlightenment
A demonstration of what can be accomplished
Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.
select a design:
Under the Sea! by Eric Stoltz
Console
HTML
CSS
Script
DOM
Net
Cookies
Pixel ...
Events
Edit
h1 < div#pageHeader < div#intro < div#container < body#css-zen-ga
Style
Computed
Layout
DOM
Events

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www
<html xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <body id="css-zen-garden">
    <div id="container">
      <div id="intro">
        <div id="pageHeader">
          <h1>
          <h2>
        </div>
```

```
#pageHeader h1 {                                    001.css (line 80)
    background: url("/001/h1.gif") no-repeat scroll
     left top transparent;
    display: block;
    float: right
    height: 87px;
    margin-top: 10px;
    width: 219px;
}
Inherited from body#css-zen-garden
```

[While editing the CSS properties, you can press the *ESC* key to cancel the editing.]

Enabling and disabling CSS rules

Firebug allows you to turn off styles impacting an element within the CSS. When you turn off a CSS property and if that property value was overriding a different value in the cascade, then the formerly crossed out value will become active and we can test the page with the removed property.

In order to *turn off* a CSS property, click on the left-hand side of the property in the Style panel where a red colored **do not** ⊘ icon will appear, and the property will be grayed out or disappear. The strikethrough of the new property value affecting the element from the cascade will be removed. You can now toggle the property 's value back to *On* by clicking on the *do not* insert icon here again. However, if the property has *disappeared* as it has been overwritten, we will have to re-inspect the element to see the missing property and then turn it on.

JavaScript profiling

Firebug has a built-in JavaScript profiler. The JavaScript profiler is used to find out how much time (average, min, max) a function/script consumed to execute on the browser. It is a very useful feature of Firebug for improving the performance of your code or if you want to find out why a particular method (code block) takes a long time to execute. The JavaScript profiler can give you a hell of a lot of detailed information about what's happening with your code.

The following are three ways to start the JavaScript profiler in Firebug:

- By clicking on the **Profile** button under the **Console** tab
- By using `console.profile("Profiler Title")` from JavaScript code
- By using `profile("Profiler Title")` from the command line

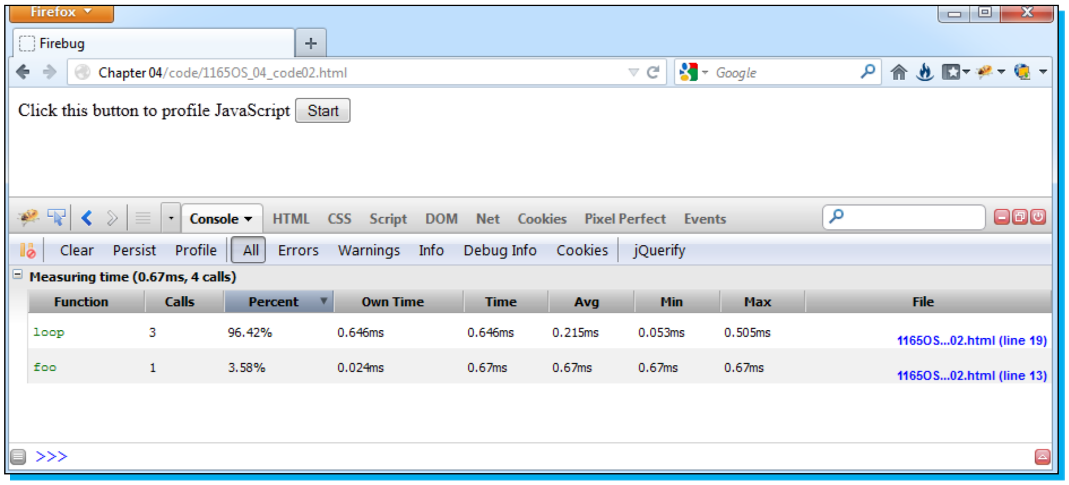
[The optional argument `Profiler.title` would contain the text to be printed in the header of the profile report.]

Let's discuss the profiler with the help of an example. Type the following HTML code, save it as an HTML file, and open it in the browser (Firefox). Press *F12* to open Firebug and click on the **Start** button.

```
<html>
<head>
<title>Firebug</title>
<script>

function bar(){
   console.profile('Measuring time');
   foo();
   console.profileEnd();
}
```

```
function foo(){
   loop(1000);
   loop(100000);
   loop(10000);
}

function loop(count){
   for(var i=0;i<count;i++){}
}

</script>
</head>
<body>
   Click this button to profile JavaScript
   <input type="button" value="Start" onclick="bar();"/>
</body>
</html>
```

As soon as you click on the **Start** button, Firebug profiles the JavaScript and generates the statistics for it. By analyzing the stats, you will get to know which method or code block is the problematic one. The following screenshot shows the statistics:



Measuring time (0.67ms, 4 calls)

Function	Calls	Percent	Own Time	Time	Avg	Min	Max	File
loop	3	96.42%	0.646ms	0.646ms	0.215ms	0.053ms	0.505ms	1165OS...02.html (line 19)
foo	1	3.58%	0.024ms	0.67ms	0.67ms	0.67ms	0.67ms	1165OS...02.html (line 13)

Columns and description of profiler

The statistics that you can see in the preceding screenshot display all the information related to the execution of the JavaScript in a tabular form. Let's review each table heading:

- **Function**: This column shows the name of each function.

- **Calls**: This shows the count of how many times a particular function has been invoked. (three times for `loop()`, in my case.)
- **Percent**: This shows the time consumed by each function in percentage.
- **Own Time**: This shows the duration of the own script in a particular function. For example, `foo()` has none of its own code. Instead, it is just calling other functions. So, its own execution time will be approximately **~0ms**. If you want to see some values for that column, add some looping in this method.
- **Time**: This shows the duration of execution from the start point of a function to the end point of a function. For example, `foo()` has no code. So, its own execution time is approx. **~0ms**, but other functions also get invoked from that function. So, the total execution time of other functions (loop) is **0.646ms**. Therefore, it shows **0.67ms** in that column which is equal to own time taken by three `loop()` function plus own time of `foo()`.
- **Avg**: This shows the average execution time of a particular function. If you are calling a function only one time, you won't see the differences. If you are calling it more than one time, you will see the difference. The formula for calculating the average is:

  ```
  Avg = Own time / Calls
  ```

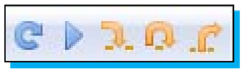
- **Min** and **Max** columns: This shows the minimum execution time of a particular function. In the example, `loop()` is invoked three times. When `1000` is passed as a parameter, it probably took only a few milliseconds (let's say `0.053ms`) and when `100000` passed to that function, it took much longer than the first time (say `0.505ms`). So, in that case, `0.053ms` will be shown in the **Min** column and `0.0505ms` will be shown in the **Max** column.
- **File**: This shows the filename of the file with the line number where the function is located.

JavaScript debugging

Firebug includes a powerful JavaScript debugger that lets you pause execution at any time and see what each variable looks like at that moment. If your code is a little sluggish, use the JavaScript profiler to measure performance and find bottlenecks faster.

Debugging JavaScript is a very straightforward process with Mozilla Firefox and Firebug. If you are a Visual Studio developer, you won't feel any differences while you are debugging the JavaScript code with Firebug, except that the debugger runs as part of the browser.

You can step debug the JavaScript by pressing one of these buttons (**Re-run**, **Continue**, **Step Into**, **Step Over**, and **Step Out**) under the **Script** tab:



Let's see the functions of these buttons:

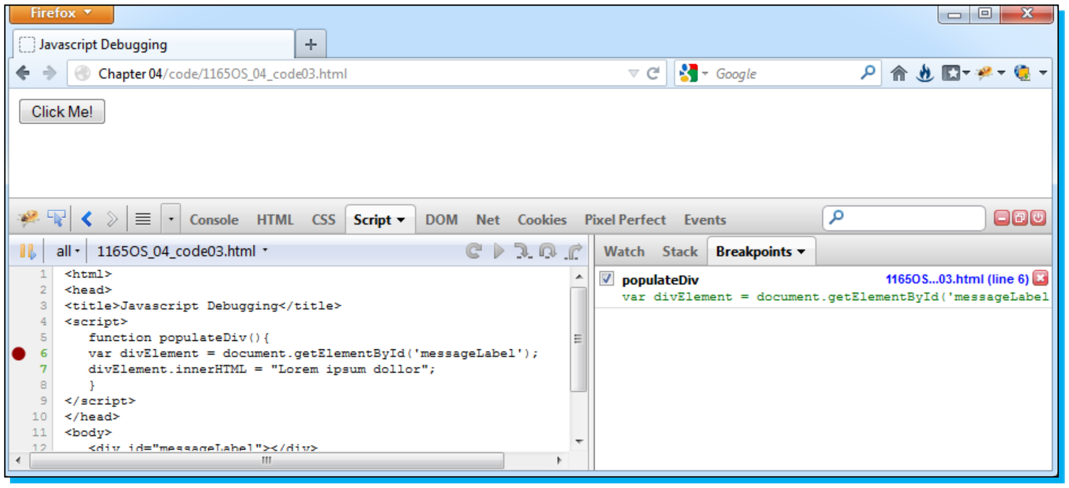
- **Re-run** (*Shift* + *F8*): Allows us to re-run the JavaScript
- **Continue** (*F8*): Allows us to resume the script execution once it has been stopped via another breakpoint
- **Step Into** (*F11*): Allows us to step into the body of the another function
- **Step Over** (*F10*): Allows us to step over the function call
- **Step Out** (*Shift* + *F11*): Allows us to resume the script execution and will stop at the next breakpoint

Applying Script breakpoints

Breakpoints in Firebug are used to debug JavaScript code. They will stop script execution at a specific point and give you control over script execution. You can set breakpoints to stop script execution as soon as it reaches them for debugging purposes. Type the following code in a text editor, save the file as `.html`, and open it in Firefox:

```
<html>
<head>
<title>Javascript Debugging</title>
<script>
   function populateDiv(){
   var divElement = document.getElementById('messageLabel');
   divElement.innerHTML = "Lorem ipsum dollor";
   }
</script>
</head>
<body>
   <div id="messageLabel"></div>
   <input type="button" value="Click Me!" onclick="populateDiv();" />
</body>
</html>
```

Now, open/activate Firebug in the browser by pressing the *F12* key. Click on the **Script** tab and insert a breakpoint on line number **6** by clicking on the gutter area next to the line number as seen in the following screenshot:

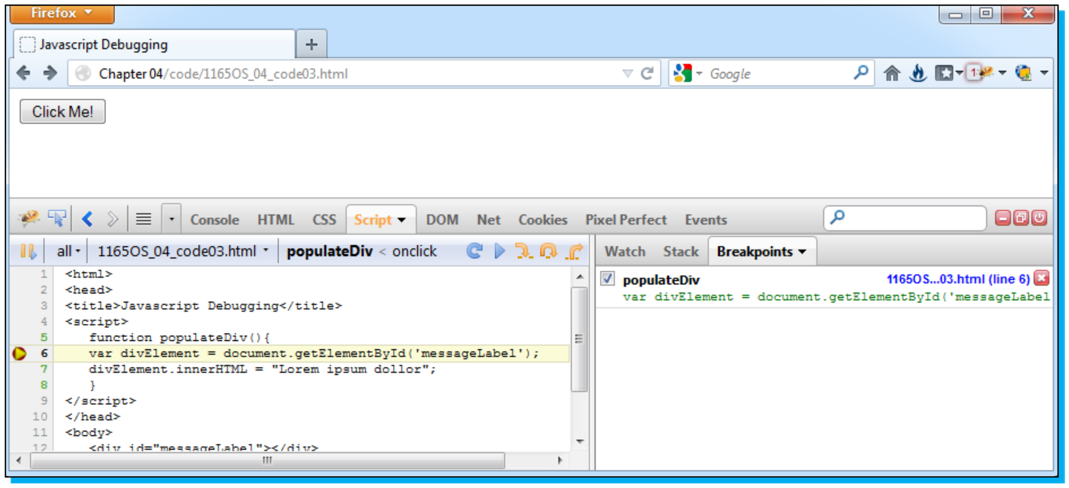


```
1  <html>
2  <head>
3  <title>Javascript Debugging</title>
4  <script>
5     function populateDiv(){
6     var divElement = document.getElementById('messageLabel');
7     divElement.innerHTML = "Lorem ipsum dollor";
8     }
9  </script>
10 </head>
11 <body>
12    <div id="messageLabel"></div>
```

```
populateDiv                                   1165OS...03.html (line 6)
   var divElement = document.getElementById('messageLabel
```

To verify that you have inserted a breakpoint, you can see the list of breakpoints in the **Breakpoints** tab on the right-hand side of the **Script** panel. Breakpoints can be created only on executable lines (so, for example, you can't break at a comment). Executable lines have a line number in green color and the rest are grayed out.

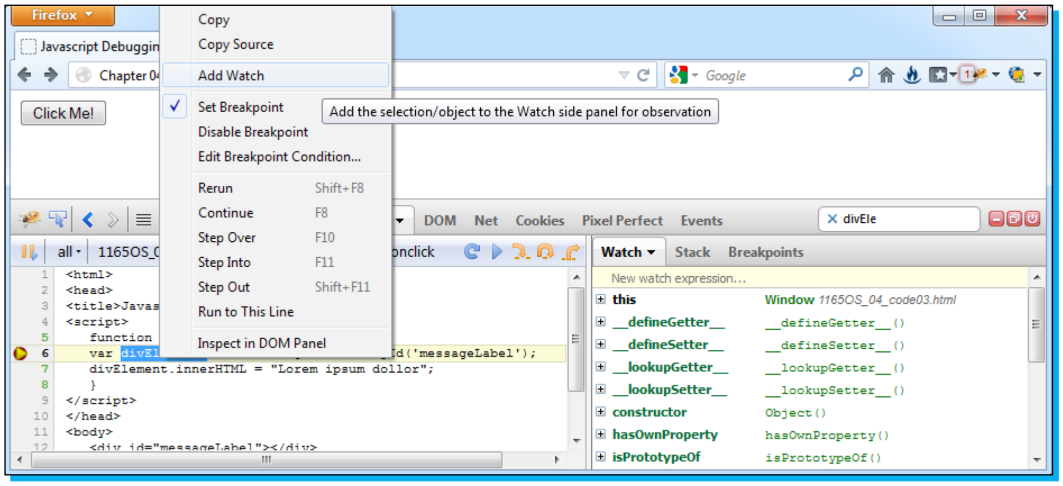
Click on the **Click Me!** button to start the execution of JavaScript. Thereafter, JavaScript execution will stop at the breakpoint that you set on line number **6** as seen in the following screenshot:



```
1  <html>
2  <head>
3  <title>Javascript Debugging</title>
4  <script>
5     function populateDiv(){
6     var divElement = document.getElementById('messageLabel');
7     divElement.innerHTML = "Lorem ipsum dollor";
8     }
9  </script>
10 </head>
11 <body>
12    <div id="messageLabel"></div>
```

```
populateDiv                                   1165OS...03.html (line 6)
   var divElement = document.getElementById('messageLabel
```

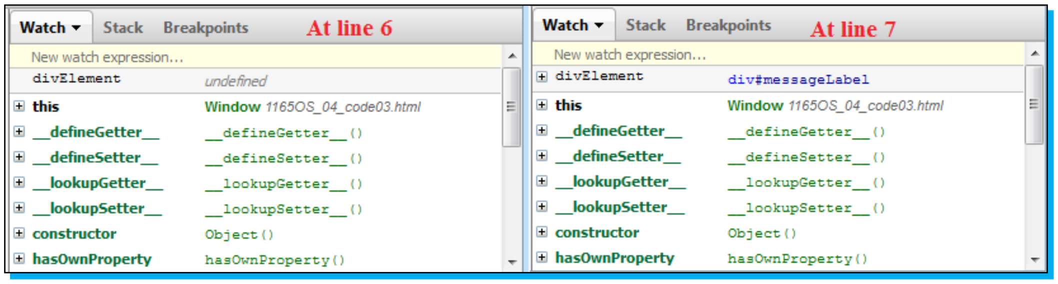
Select the variable `divElement` from JavaScript, right-click on it and choose **Add watch** (as shown in the following screenshot). This enables Firebug to track the behavior of this variable under the **Watch** tab.



```
1   <html>
2   <head>
3   <title>Javas
4   <script>
5       function
6       var divEl                        d('messageLabel');
7       divElement.innerHTML = "Lorem ipsum dollor";
8       }
9   </script>
10  </head>
11  <body>
12      <div id="messageLabel"></div>
```

this	Window 1165OS_04_code03.html
__defineGetter__	__defineGetter__()
__defineSetter__	__defineSetter__()
__lookupGetter__	__lookupGetter__()
__lookupSetter__	__lookupSetter__()
constructor	Object()
hasOwnProperty	hasOwnProperty()
isPrototypeOf	isPrototypeOf()

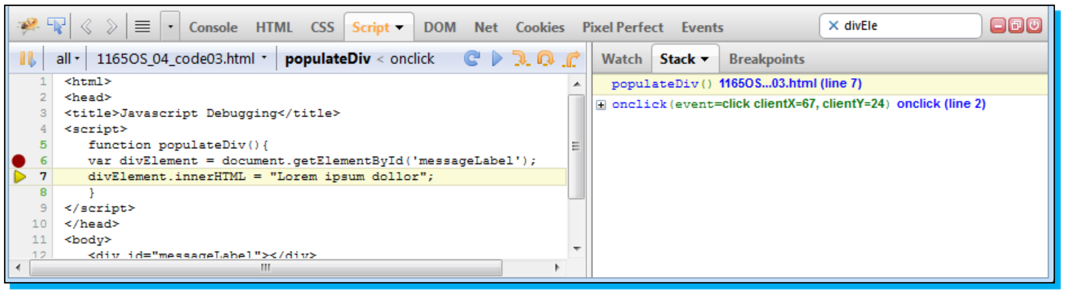
Click on **Step Over** or press the *F10* key to execute line number **6** and move on to line number **7**. Notice the value of `divElement` in the **Watch** tab on the right-hand side. Before the execution of line number **6**, the value for variable `divElement` was undefined and after the execution of line number **6** it is populated with an HTML `div` element. Notice the value of the `div` element is a link to the actual `div` element on the page. On clicking, it will take you to the position of the `div` element in the **HTML** panel.



divElement	*undefined*
this	Window 1165OS_04_code03.html
__defineGetter__	__defineGetter__()
__defineSetter__	__defineSetter__()
__lookupGetter__	__lookupGetter__()
__lookupSetter__	__lookupSetter__()
constructor	Object()
hasOwnProperty	hasOwnProperty()

divElement	div#messageLabel
this	Window 1165OS_04_code03.html
__defineGetter__	__defineGetter__()
__defineSetter__	__defineSetter__()
__lookupGetter__	__lookupGetter__()
__lookupSetter__	__lookupSetter__()
constructor	Object()
hasOwnProperty	hasOwnProperty()

You can also view the stack of call and execution flow on the **Stack** tab on the right-hand side of the **Script** panel as seen in the following screenshot:



```
<html>
<head>
<title>Javascript Debugging</title>
<script>
   function populateDiv(){
   var divElement = document.getElementById('messageLabel');
   divElement.innerHTML = "Lorem ipsum dollor";
   }
</script>
</head>
<body>
   <div id="messageLabel"></div>
```

```
populateDiv()  1165OS...03.html (line 7)
onclick(event=click clientX=67, clientY=24)  onclick (line 2)
```

If you want to debug external JavaScript files then you can select that file from the drop-down under the **Script** panel.

Applying conditional breakpoints

Sometimes you have an error inside a loop that can be really difficult to get to. You definitely don't want to put a breakpoint inside a loop and hit *F10* (**Step Over**) a few thousand times until you get to the error condition. Thanks to Firebug, it provides you with such a utility by which you can insert breakpoints on the basis of some condition.

Perhaps the most important tool for debugging inside loops is the conditional breakpoint. You can set a condition on a breakpoint so it will only break when a specified condition is true.

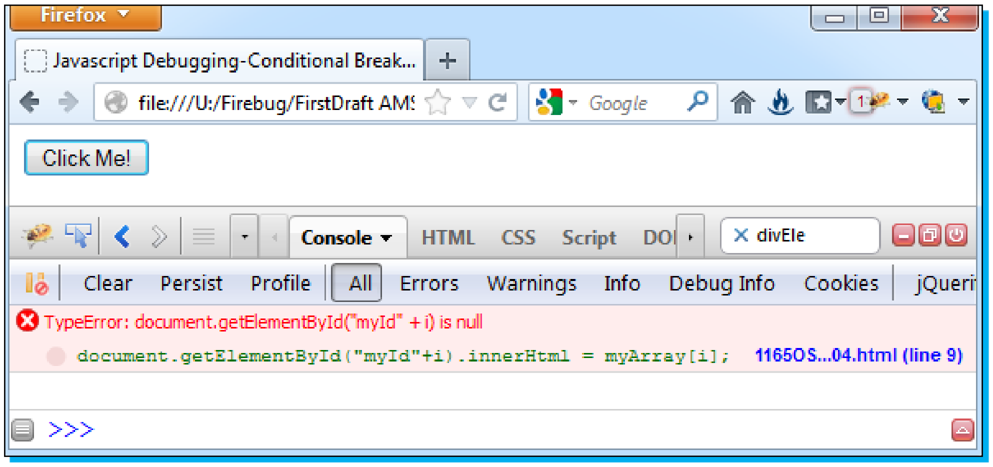
To view conditional breakpoints in action, type the following code in some text editor, save it as an HTML file, open it in Firefox, and then open Firebug:

```
<html>
<head>
<title>Javascript Debugging-Conditional Breakpoint</title>
<script>
   var myArray = new Array(9);
   function printTableOf(num){
       for(i = 1; i<=9; i++){
      myArray[i] = i*num;
      document.getElementById("myId"+i).innerHtml = myArray[i];
       }
   }
</script>
</head>
<body>
   <div id="myId1"></div>
```

```
        <div id="myId2"></div>
        <div id="myId3"></div>
        <div id="myId4"></div>
        <div id="myId5"></div>
        <input type="button" value="Click Me!" onclick="printTableOf(2);"
    />
    </body>
    </html>
```

Clicking on the **Click Me!** button will show you the error on Firebug's console informing you about the error at **line 9**.

The error contains very useful information that can be of great help while debugging the code. It shows the line number where the error occurs, the cause of error, and stack, as seen in the following screenshot:



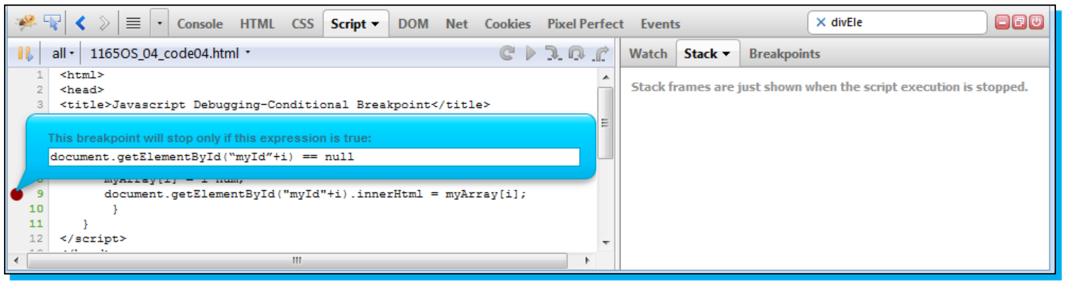
```
TypeError: document.getElementById("myId" + i) is null
document.getElementById("myId"+i).innerHtml = myArray[i];    1165OS...04.html (line 9)
```

Now follow the steps to apply conditional breakpoints to debug this one:

1. Refresh the page and go to the **Script** tab. Right-click on the line number where the error occurred. Firebug will show you a blue balloon where you can give a condition, and when to pause the execution. Generally, the condition is decided by the cause of the error. In the example, the cause is that `document.getElementById("myId"+i)` is null.

   So, the condition would be:

   ```
   document.getElementById("myId"+i) == null
   ```


Console HTML CSS Script DOM Net Cookies Pixel Perfect Events
divEle
all 1165OS_04_code04.html
Watch Stack Breakpoints

```
<html>
<head>
<title>Javascript Debugging-Conditional Breakpoint</title>
```

This breakpoint will stop only if this expression is true:

```
document.getElementById("myId"+i) == null
```

```
        myArray[i] = i*num;
        document.getElementById("myId"+i).innerHtml = myArray[i];
        }
    }
</script>
```

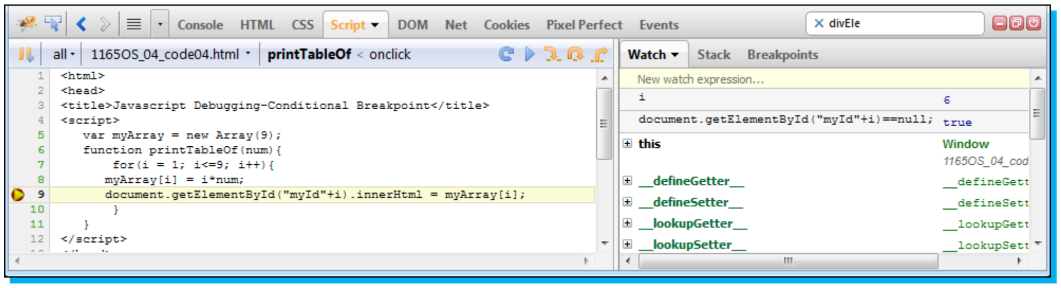
Stack frames are just shown when the script execution is stopped.

2. Press *Enter*, and you can see the breakpoint is inserted on the **Breakpoints** panel.

Removing/disabling breakpoints: To disable breakpoints, uncheck the checkbox(es) in the **Breakpoints** panel or we can simply click on the big red dot to remove them.

3. Click on the **Click Me!** button and notice that the execution of the JavaScript is paused on the line where we inserted a conditional breakpoint.

   Observe the value of variable `i = 6`. When the execution is paused it means that the conditional breakpoint is triggered in error. The error occurred because we have only five `<div>` elements in our HTML file and through JavaScript we are accessing the sixth `<div>` element that does not exist.


Console HTML CSS Script DOM Net Cookies Pixel Perfect Events
divEle
all 1165OS_04_code04.html printTableOf < onclick
Watch Stack Breakpoints
New watch expression...

```
<html>
<head>
<title>Javascript Debugging-Conditional Breakpoint</title>
<script>
   var myArray = new Array(9);
   function printTableOf(num){
       for(i = 1; i<=9; i++){
       myArray[i] = i*num;
       document.getElementById("myId"+i).innerHtml = myArray[i];
       }
   }
</script>
```

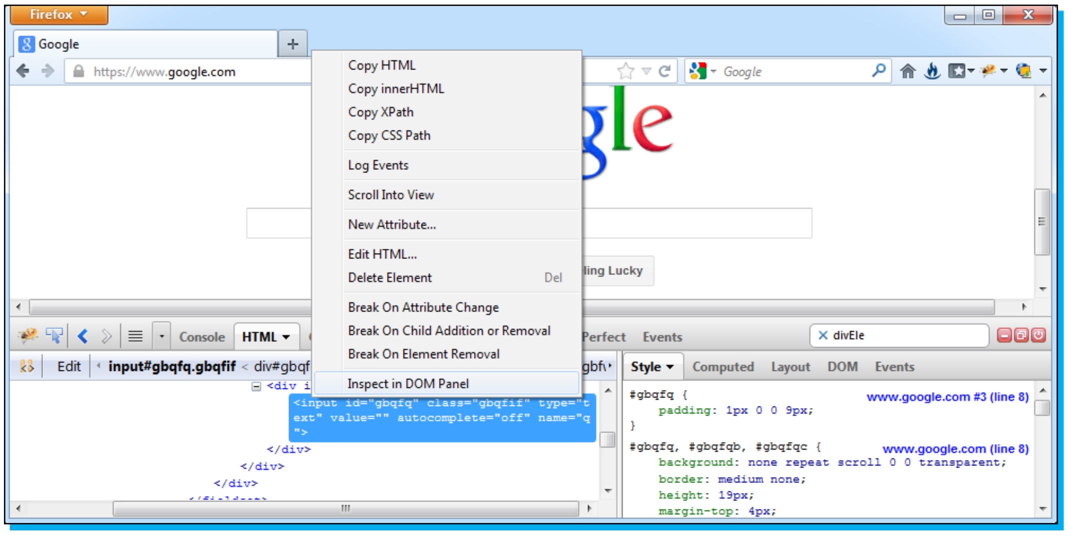
```
i                                        6
document.getElementById("myId"+i)==null;  true
this                                     Window
                                         1165OS_04_cod
__defineGetter__                         __defineGett
__defineSetter__                         __defineSett
__lookupGetter__                         __lookupGett
__lookupSetter__                         __lookupSett
```

Inspecting DOM

The DOM inspector allows for full in-place editing of your document structure, not just text nodes. In the DOM inspector, Firebug autocompletes the property value when you hit the *Tab* key. The following are the steps to inspect an element under the **DOM** tab:

1. Inspect an HTML element on the page by pressing *Ctrl* + *Shift* + *C*. This is a shortcut key to open Firebug in inspect mode.

2. Move the mouse pointer over the HTML element that you want to inspect and click on that element. The HTML of that element will be shown on Firebug's **HTML** tab as seen in the following screenshot:

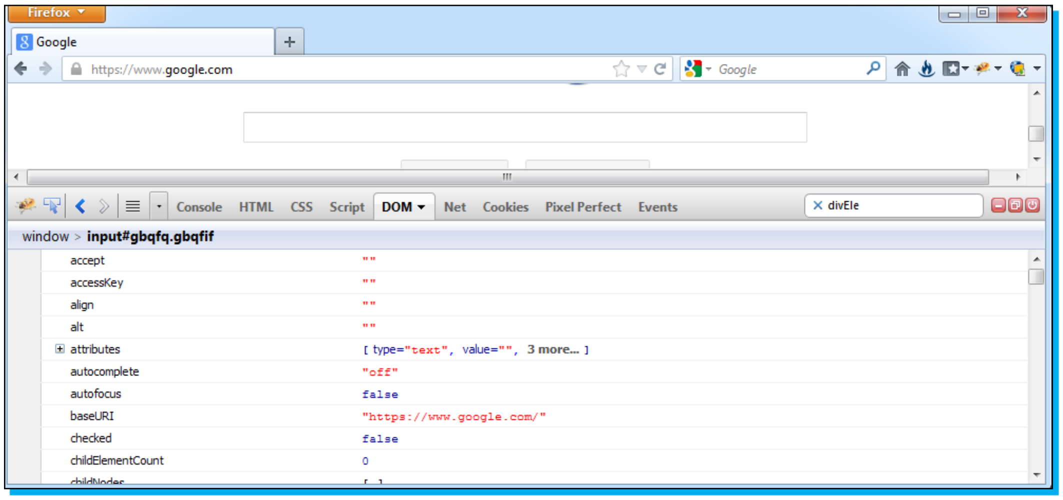


```
<div i
  <input id="gbqfq" class="gbqfif" type="text" value="" autocomplete="off" name="q">
</div>
</div>
</div>
```

```
#gbqfq {
    padding: 1px 0 0 9px;
}
#gbqfq, #gbqfqb, #gbqfqc {
    background: none repeat scroll 0 0 transparent;
    border: medium none;
    height: 19px;
    margin-top: 4px;
```

3. Right-click on the selected DOM element, and a context menu will appear. Select the **Inspect in DOM Panel** option from the context menu.

4. As soon as you select **Inspect in DOM Panel**, Firebug will take you to its **DOM** panel as seen in the following screenshot:



accept	""
accessKey	""
align	""
alt	""
⊞ attributes	[type="text", value="", 3 more...]
autocomplete	"off"
autofocus	false
baseURI	"https://www.google.com/"
checked	false
childElementCount	0

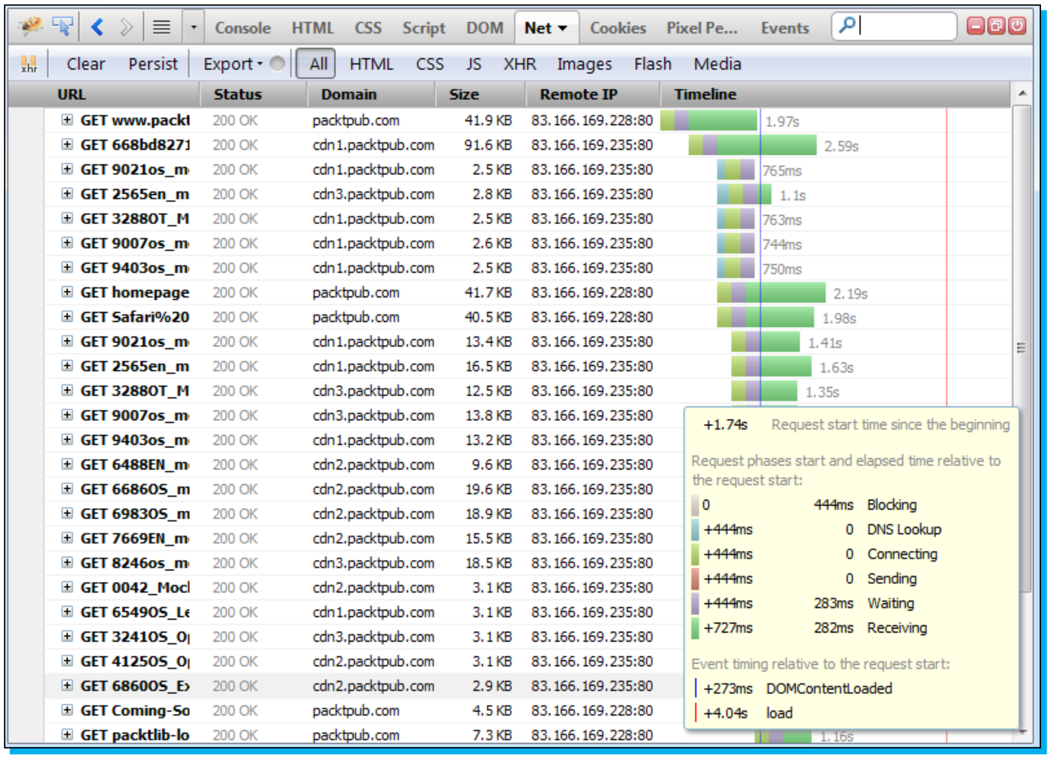
Network monitoring

Even if the server does not take much time to process a request, a web-application might appear to be slow to an end-user because of various reasons:

- Network latency
- The order in which the files are loaded
- The number of requests that are made to the server
- Browser caching (or the absence of it!)

Firebug's Net panel helps you detect such problems very easily. The main purpose of the **Net** panel is to monitor HTTP traffic initiated by a web page and simply present all the collected and computed information to the user in a graphical and intuitive interface.

The following screenshot shows various requests that are made by the Firefox to load the homepage of the `www.packtpub.com` website:



URL	Status	Domain	Size	Remote IP	Timeline
GET www.packt	200 OK	packtpub.com	41.9 KB	83.166.169.228:80	1.97s
GET 668bd8271	200 OK	cdn1.packtpub.com	91.6 KB	83.166.169.235:80	2.59s
GET 9021os_m	200 OK	cdn1.packtpub.com	2.5 KB	83.166.169.235:80	765ms
GET 2565en_m	200 OK	cdn3.packtpub.com	2.8 KB	83.166.169.235:80	1.1s
GET 32880T_M	200 OK	cdn1.packtpub.com	2.5 KB	83.166.169.235:80	763ms
GET 9007os_m	200 OK	cdn1.packtpub.com	2.6 KB	83.166.169.235:80	744ms
GET 9403os_m	200 OK	cdn1.packtpub.com	2.5 KB	83.166.169.235:80	750ms
GET homepage	200 OK	packtpub.com	41.7 KB	83.166.169.228:80	2.19s
GET Safari%20	200 OK	packtpub.com	40.5 KB	83.166.169.228:80	1.98s
GET 9021os_m	200 OK	cdn1.packtpub.com	13.4 KB	83.166.169.235:80	1.41s
GET 2565en_m	200 OK	cdn1.packtpub.com	16.5 KB	83.166.169.235:80	1.63s
GET 32880T_M	200 OK	cdn3.packtpub.com	12.5 KB	83.166.169.235:80	1.35s
GET 9007os_m	200 OK	cdn3.packtpub.com	13.8 KB	83.166.169.235:80	
GET 9403os_m	200 OK	cdn1.packtpub.com	13.2 KB	83.166.169.235:80	
GET 6488EN_m	200 OK	cdn2.packtpub.com	9.6 KB	83.166.169.235:80	
GET 66860S_m	200 OK	cdn2.packtpub.com	19.6 KB	83.166.169.235:80	
GET 69830S_m	200 OK	cdn2.packtpub.com	18.9 KB	83.166.169.235:80	
GET 7669EN_m	200 OK	cdn2.packtpub.com	15.5 KB	83.166.169.235:80	
GET 8246os_m	200 OK	cdn3.packtpub.com	18.5 KB	83.166.169.235:80	
GET 0042_Mocl	200 OK	cdn2.packtpub.com	3.1 KB	83.166.169.235:80	
GET 65490S_Le	200 OK	cdn1.packtpub.com	3.1 KB	83.166.169.235:80	
GET 32410S_O	200 OK	cdn3.packtpub.com	3.1 KB	83.166.169.235:80	
GET 41250S_O	200 OK	cdn2.packtpub.com	3.1 KB	83.166.169.235:80	
GET 68600S_E	200 OK	cdn2.packtpub.com	2.9 KB	83.166.169.235:80	
GET Coming-So	200 OK	packtpub.com	4.5 KB	83.166.169.228:80	
GET packtlib-lo	200 OK	packtpub.com	7.3 KB	83.166.169.228:80	1.16s

+1.74s Request start time since the beginning

Request phases start and elapsed time relative to the request start:

0	444ms	Blocking
+444ms	0	DNS Lookup
+444ms	0	Connecting
+444ms	0	Sending
+444ms	283ms	Waiting
+727ms	282ms	Receiving

Event timing relative to the request start:

+273ms	DOMContentLoaded
+4.04s	load

Description of information in the Net panel

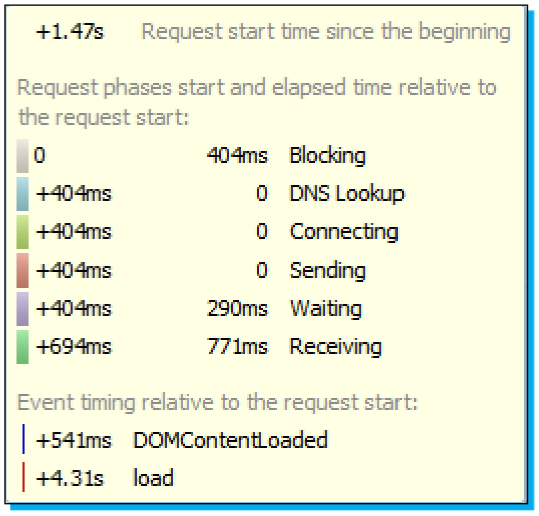
The requests shown in the **Net** panel are sorted based on the order of how Firefox loaded those requests and is presented to you in a tabular form. You can also sort them later on the basis of **Data size**, **URL**, **Status**, **Domain**, **Protocol**, and so on by clicking on the heading of a particular column.

Column No.	Column Name	Description
1	**URL**	The URL of the file/request that was loaded. The `GET` prefix for most of the requests depicts the method of the request (`GET`, `POST`, and so on).
2	**Status**	Status of the HTTP request and the code. For example, `code 200` denotes a successful HTTP request and `code 304` denotes that the file was not modified since the last request (based on some caching time-limit).
3	**Domain**	The base URL for the request. If you are loading files from other sites (For example: linking images from other sites, putting up ads from an ad server) then a different URL will be shown for that particular file/request.
4	**Size**	The size of the response data.
5	**Remote IP**	The IP address with the port number of the server from which the request is served.
6	**Timeline**	The time taken to load the particular file/request. It also shows whether or not the file is loaded from the cache. The bar shows you when the file started and stopped loading relative to other files.

It is also possible to hide/show certain columns that the **Net** panel shows you by default. In order to customize the columns that you want to see in the **Net** panel, simply click on the table header and choose the columns that you want to view.

Load-time bar color significance

As we know, every request-response round trip is shown as a horizontal bar in the **Timeline** panel and is composed of several phases, represented by different colors. Hovering over a **Request Timeline** offers more detailed information about the timings of the different phases, as seen in the following screenshot:



```
+1.47s     Request start time since the beginning

Request phases start and elapsed time relative to
the request start:
0              404ms   Blocking
+404ms             0   DNS Lookup
+404ms             0   Connecting
+404ms             0   Sending
+404ms         290ms   Waiting
+694ms         771ms   Receiving

Event timing relative to the request start:
+541ms   DOMContentLoaded
+4.31s   load
```

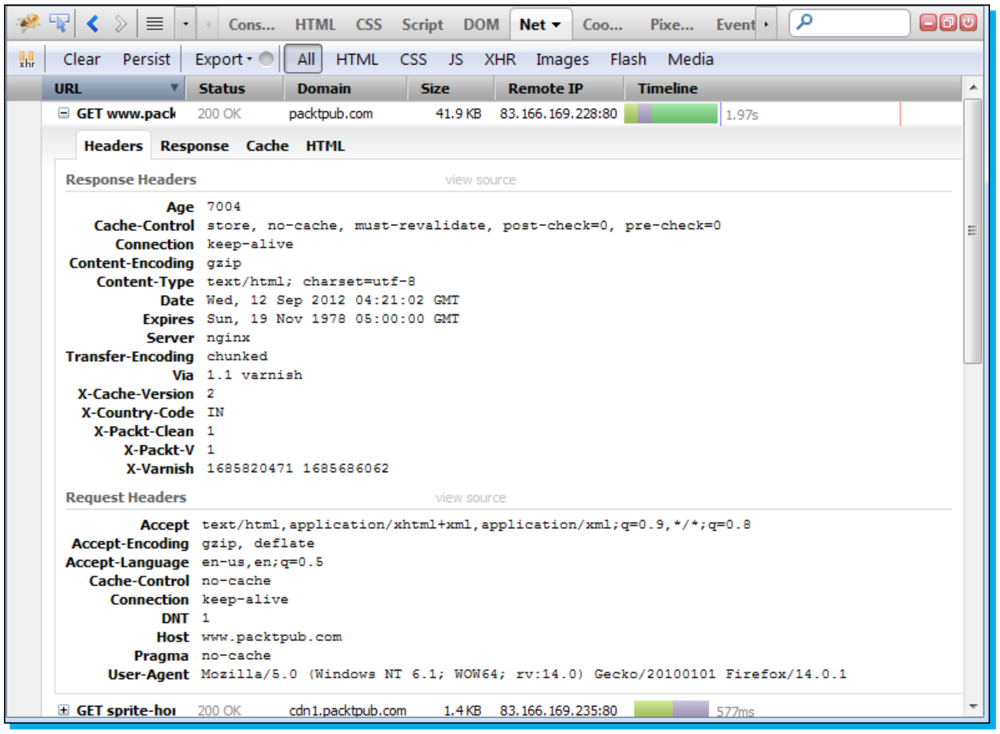
The different colors used in the timeline bar have significance. The following table describes what each color means in the timeline bar:

Display Color	Value	Description
Olive grey	**Blocking**	Time spent in a browser queue waiting for a network connection (formerly called Queuing)
Sky blue	**DNS Lookup**	Time for DNS Lookup or DNS resolution time.
Light green	**Connecting**	Elapsed time required to create a TCP connection
Light brown	**Sending**	Time for which the request had to wait in the queue.
Purple	**Waiting**	Time waiting for a response from the server
Dark grey	**Receiving**	Request was sent to server, request served by the server and not from browser cache
Light grey	**Receiving** (From cache)	Request was sent to the server, `304 Not Modified` received from server, response loaded from the browser cache
Blue line	**DOMContentLoaded** (event)	Point in time when `DOMContentLoaded` event was fired (since the beginning of the request, can be negative if the request has been started after the event)
Red line	**Load** (event)	Point in time when the page load event was fired (since the beginning of the request, can be negative if the request has been started after the event)

Display Color	Value	Description
Green line	**MozAfterPaint** (event)	Point in time when a `MozAfterPaint` event was fired (since the beginning of the request, can be negative if the request has been started after the event)
Olive line	**Time stamp**	Time stamp created via `console.timeStamp()`

Examining HTTP headers

HTTP headers contain a wealth of interesting information such as the mime type of the file, the type of web server, caching directives, the cookie, and lots more. To view the HTTP headers, just click on the **+** button to the left-hand side of each request to expand it as seen in the following screenshot:



```
URL              Status   Domain          Size     Remote IP            Timeline
GET www.pack     200 OK   packtpub.com    41.9 KB  83.166.169.228:80    1.97s

Response Headers
              Age  7004
    Cache-Control  store, no-cache, must-revalidate, post-check=0, pre-check=0
       Connection  keep-alive
 Content-Encoding  gzip
     Content-Type  text/html; charset=utf-8
             Date  Wed, 12 Sep 2012 04:21:02 GMT
          Expires  Sun, 19 Nov 1978 05:00:00 GMT
           Server  nginx
Transfer-Encoding  chunked
              Via  1.1 varnish
  X-Cache-Version  2
   X-Country-Code  IN
    X-Packt-Clean  1
        X-Packt-V  1
        X-Varnish  1685820471 1685686062

Request Headers
           Accept  text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Accept-Encoding  gzip, deflate
  Accept-Language  en-us,en;q=0.5
    Cache-Control  no-cache
       Connection  keep-alive
              DNT  1
             Host  www.packtpub.com
           Pragma  no-cache
       User-Agent  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1

GET sprite-hoi   200 OK   cdn1.packtpub.com  1.4 KB  83.166.169.235:80   577ms
```

For each HTTP request, Firebug displays the following tabs when you click on the **+** button:

- **Headers**
- **Response**

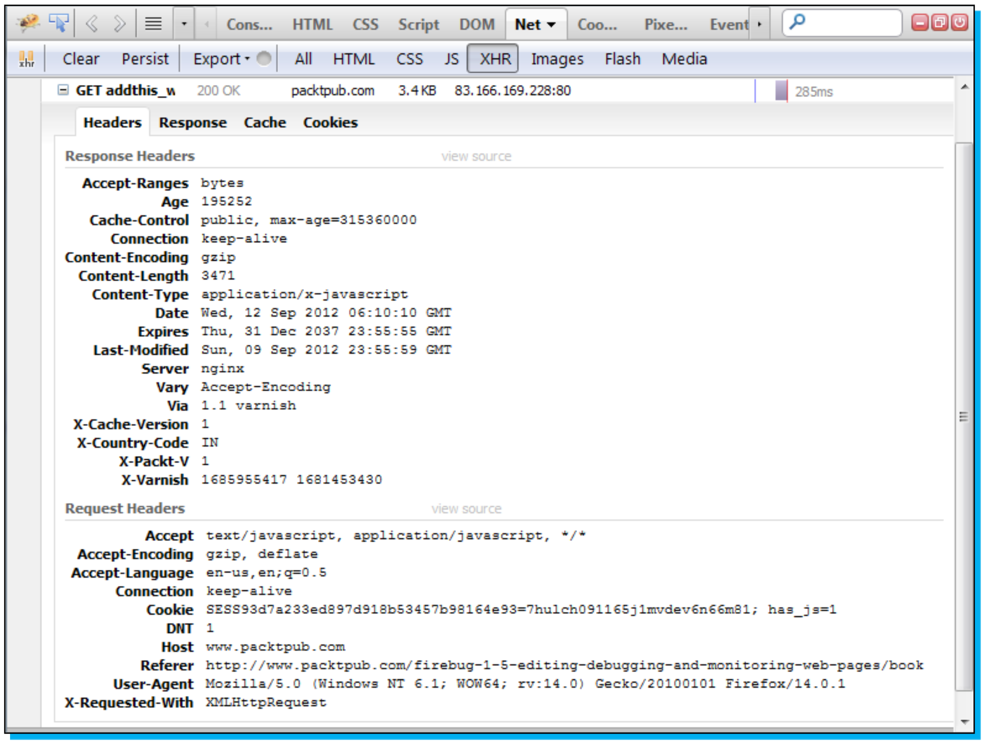
In addition to these tabs, Firebug displays the following tabs if applicable:

- **HTML**
- **Params**
- **Cache**
- **Post**
- **Cookies**

In order to get a complete understanding of what each HTTP request/response header means, please refer to `http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`.

Monitoring AJAX requests

You know about how to analyze the requests and responses that are made and received when a page is loaded by the browser. However, today's web applications make a lot of asynchronous XML requests (yes, we are talking about AJAX requests). In order to view the Ajax requests that are made by a web page, take a look at Firebug's **XHR** (**XML HTTP Request**) tab under the **Net** panel. This tab filters out the only AJAX requests as seen in the following screenshot:



```
GET addthis_w   200 OK   packtpub.com   3.4 KB   83.166.169.228:80   285ms
Headers  Response  Cache  Cookies

Response Headers                                         view source
    Accept-Ranges  bytes
              Age  195252
    Cache-Control  public, max-age=315360000
       Connection  keep-alive
 Content-Encoding  gzip
   Content-Length  3471
     Content-Type  application/x-javascript
             Date  Wed, 12 Sep 2012 06:10:10 GMT
          Expires  Thu, 31 Dec 2037 23:55:55 GMT
    Last-Modified  Sun, 09 Sep 2012 23:55:59 GMT
           Server  nginx
             Vary  Accept-Encoding
              Via  1.1 varnish
  X-Cache-Version  1
   X-Country-Code  IN
       X-Packt-V  1
       X-Varnish  1685955417 1681453430

Request Headers                                          view source
           Accept  text/javascript, application/javascript, */*
  Accept-Encoding  gzip, deflate
  Accept-Language  en-us,en;q=0.5
       Connection  keep-alive
           Cookie  SESS93d7a233ed897d918b53457b98164e93=7hulch091165j1mvdev6n66m81; has_js=1
              DNT  1
             Host  www.packtpub.com
          Referer  http://www.packtpub.com/firebug-1-5-editing-debugging-and-monitoring-web-pages/book
       User-Agent  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
X-Requested-With  XMLHttpRequest
```

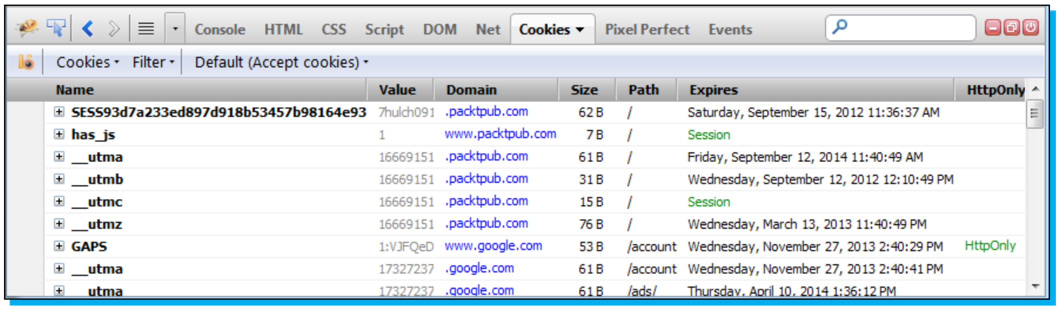
The **XHR** tab displays the AJAX requests made and responses received by the web page that you are currently viewing. You can see the start and completion of these AJAX requests and if they were successful and what was returned from the server as response.

As you will notice, an **XHR** (`XMLHttpRequest`) is not much different from a normal request. The preceding screenshot shows the AJAX requests made by `www.packtpub.com` when the user tries to add a book to the cart.

The **XHR** tab displays the AJAX events as they happen on a page. If your view becomes too crowded while viewing and analyzing AJAX events, simply click on the **Clear** button to clear the events from the panel that are currently being displayed.

Managing cookies

Firebug allows you to manage cookies via its **Cookies** panel. As you already know, this panel displays a list of all cookies associated with all the domains of the currently displayed page. And each entry in the list displays basic information about a cookie (**Name**, **Value**, **Domain**, **Path**, **Expiry date**, and so on) as seen in the following screenshot:



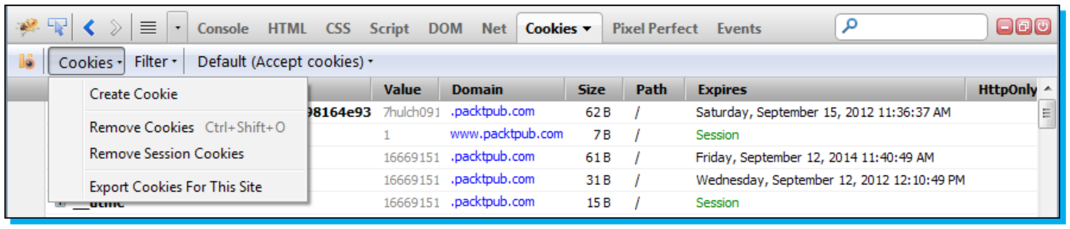
Name	Value	Domain	Size	Path	Expires	HttpOnly
SESS93d7a233ed897d918b53457b98164e93	7hulch091	.packtpub.com	62 B	/	Saturday, September 15, 2012 11:36:37 AM	
has_js	1	www.packtpub.com	7 B	/	Session	
__utma	16669151	.packtpub.com	61 B	/	Friday, September 12, 2014 11:40:49 AM	
__utmb	16669151	.packtpub.com	31 B	/	Wednesday, September 12, 2012 12:10:49 PM	
__utmc	16669151	.packtpub.com	15 B	/	Session	
__utmz	16669151	.packtpub.com	76 B	/	Wednesday, March 13, 2013 11:40:49 PM	
GAPS	1:VJFQeD	www.google.com	53 B	/account	Wednesday, November 27, 2013 2:40:29 PM	HttpOnly
__utma	17327237	.google.com	61 B	/account	Wednesday, November 27, 2013 2:40:41 PM	
__utma	17327237	.google.com	61 B	/ads/	Thursday, April 10, 2014 1:36:12 PM	

This panel has been available since Firebug 1.10 and was formerly a separate extension called *Firecookie*.

The following is a list of information that is displayed for each cookie:

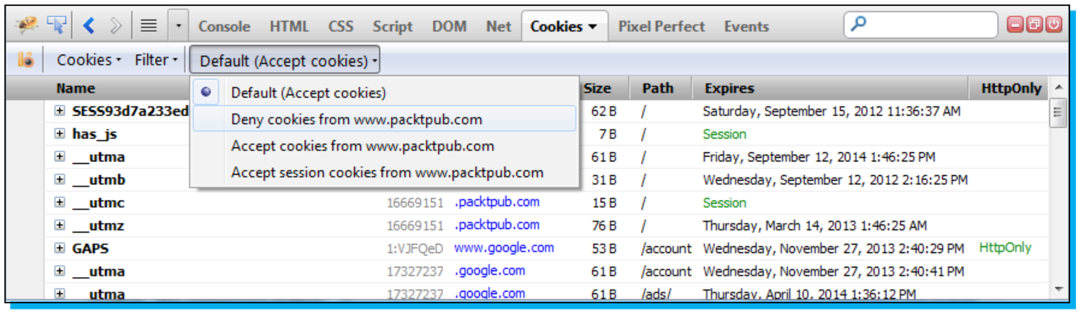
- Name of the cookie
- Value of the cookie
- Domain the cookie is set for
- Size of the cookie value in bytes
- Path the cookie is set for
- Expiration date or **Session** for session cookies
- Flag specifying whether **HttpOnly** is set for a cookie
- Flag specifying whether it's a secure cookie

Under the **Cookies** panel there is a drop-down named **Cookies**. On clicking this drop-down you will find options for creating, removing, and exporting cookies as seen in the following screenshot:



	Value	Domain	Size	Path	Expires	HttpOnly
98164e93	7hulch091	.packtpub.com	62 B	/	Saturday, September 15, 2012 11:36:37 AM	
	1	www.packtpub.com	7 B	/	Session	
	16669151	.packtpub.com	61 B	/	Friday, September 12, 2014 11:40:49 AM	
	16669151	.packtpub.com	31 B	/	Wednesday, September 12, 2012 12:10:49 PM	
	16669151	.packtpub.com	15 B	/	Session	

You can also manage cookie permissions for the current site directly from a drop-down available under the **Cookies** panel. The permission drop-down displays the current status as a label and it's automatically updated if the permission is changed as seen in the following screenshot:


Console
HTML
CSS
Script
DOM
Net
Cookies
Pixel Perfect
Events
Cookies
Filter
Default (Accept cookies)
Default (Accept cookies)
Deny cookies from www.packtpub.com
Accept cookies from www.packtpub.com
Accept session cookies from www.packtpub.com

Name			Size	Path	Expires	HttpOnly
SESS93d7a233ed			62 B	/	Saturday, September 15, 2012 11:36:37 AM	
has_js			7 B	/	Session	
__utma			61 B	/	Friday, September 12, 2014 1:46:25 PM	
__utmb			31 B	/	Wednesday, September 12, 2012 2:16:25 PM	
__utmc	16669151	.packtpub.com	15 B	/	Session	
__utmz	16669151	.packtpub.com	76 B	/	Thursday, March 14, 2013 1:46:25 AM	
GAPS	1:VJFQeD	www.google.com	53 B	/account	Wednesday, November 27, 2013 2:40:29 PM	HttpOnly
__utma	17327237	.google.com	61 B	/account	Wednesday, November 27, 2013 2:40:41 PM	
__utma	17327237	.google.com	61 B	/ads/	Thursday, April 10, 2014 1:36:12 PM	

People and places you should get to know

Official Sites

- Official site of Firebug: `https://getfirebug.com/`
- Firebug plugins page on Firebug site: `https://getfirebug.com/wiki/index.php/Firebug_Extensions`
- Manual and documentation: `http://getfirebug.com/docs.html`
- Wiki: `https://getfirebug.com/wiki/`
- Blog: `https://blog.getfirebug.com/`
- Source code: `https://github.com/firebug/`
- Official issue list: `http://code.google.com/p/fbug/issues/list`

Articles and Tutorials

- A seven minute video covering an introduction of Firebug by Rob Campbell: `https://getfirebug.com/video/Intro2FB.htm`
- An eight minute video covering what's new in Firebug 1.6: `http://vimeo.com/18411877`
- Also an extended edition of *What's new in Firebug 1.6* is available on: `http://vimeo.com/18080380`
- Wiki link for Firebug Lite: `https://getfirebug.com/wiki/index.php/Firebug_Lite`
- Blog series on Firebug *Tips and Tricks* by Jan Odvarko: `http://www.softwareishard.com/blog/firebug-tips/`
- *Getting started with Firebug 1.5*, Chandan Luthra and Deepak Mittal, Dzone Refcardz: `http://refcardz.dzone.com/refcardz/getting-started-firebug-15`

Community

- Official mailing list: `http://groups.google.com/group/firebug`
- Official forums; this forum is an archive for the official mailing list `firebug@googlegroups.com`: `http://old.nabble.com/FireBug-f17524.html`
- Unofficial forums, `Stackoverflow.com`; it's a language-independent collaboratively edited question and answer site for programmers:
  - `http://stackoverflow.com/questions/tagged/firebug`
  - `http://stackoverflow.com/questions/tagged/firebug-lite`
- Official IRC channel: `irc://irc.mozilla.org/firebug`
- User FAQ: `http://getfirebug.com/faq`

Blogs

- The official Firebug blog link: `https://blog.getfirebug.com/`
- The blog of Jan Odvarko, Firebug Technical Lead: `http://www.softwareishard.com/blog/`
- Mozhack blog: `http://hacks.mozilla.org/category/firebug/`
- The blog of Joe Hewitt, initial Firebug developer : `http://joehewitt.com/blog`
- The blog of Michael Sync, Firebug user and reviewer of *Firebug 1.5* book `http://michaelsync.net/category/firebug`

Twitter

- Follow Firebug news on Twitter: `http://twitter.com/#!/firebugnews`
- Joe Hewitt at `http://twitter.com/joehewitt`
- Jan Odvarko at `https://twitter.com/janodvarko`
- Follow Planet Mozilla at `https://twitter.com/planetmozilla`
- For more Open Source information, follow Packt Publishing at `http://twitter.com/#!/packtopensource`