Safari User Guide for Web Developers

Tools

2010-06-21

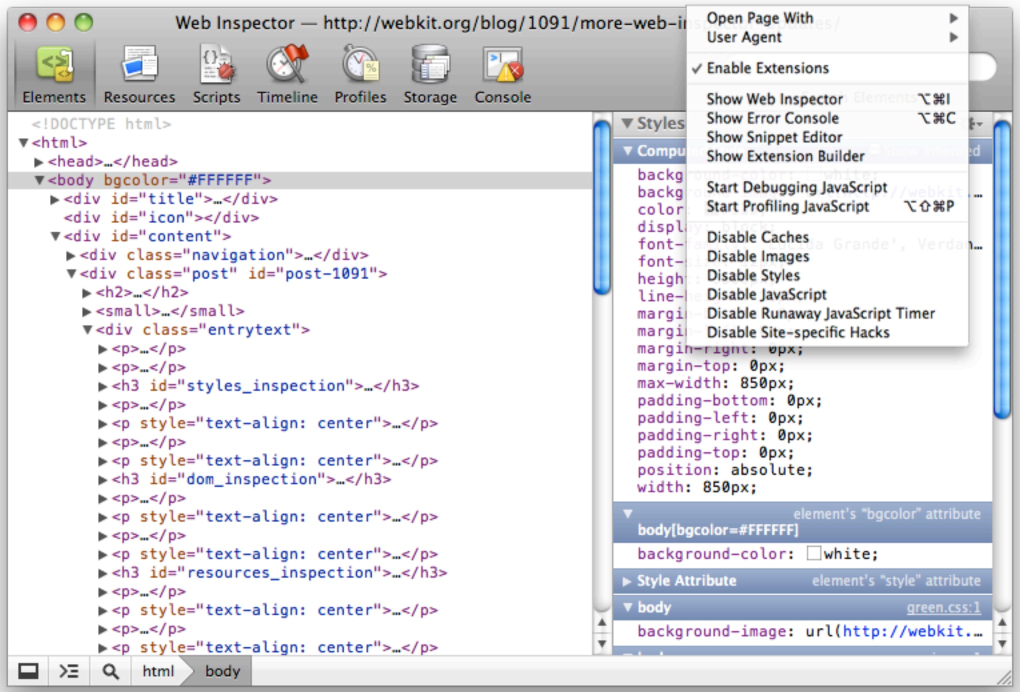
Contents

Figures

About Safari Developer Tools

Safari 4.0 and later includes built-in tools to help you prototype, analyze, debug, and optimize websites and web applications. Safari 5.0 and later has additional tools you can use to enable, develop, test and debug Safari extensions.

Figure I-1 The Safari Develop menu and Web Inspector



```
<!DOCTYPE html>
▼<html>
 ▶<head>…</head>
 ▼<body bgcolor="#FFFFFF">
   ▶<div id="title">…</div>
    <div id="icon"></div>
   ▼<div id="content">
     ▶<div class="navigation">…</div>
     ▼<div class="post" id="post-1091">
       ▶<h2>…</h2>
       ▶<small>…</small>
       ▼<div class="entrytext">
         ▶<p>…</p>
         ▶<p>…</p>
         ▶<h3 id="styles_inspection">…</h3>
         ▶<p>…</p>
         ▶<p style="text-align: center">…</p>
         ▶<p>…</p>
         ▶<p style="text-align: center">…</p>
         ▶<h3 id="dom_inspection">…</h3>
         ▶<p>…</p>
         ▶<p style="text-align: center">…</p>
         ▶<p>…</p>
         ▶<p style="text-align: center">…</p>
         ▶<h3 id="resources_inspection">…</h3>
         ▶<p>…</p>
         ▶<p style="text-align: center">…</p>
         ▶<p>…</p>
         ▶<p style="text-align: center">…</p>
```

```
margin-top: 0px;
max-width: 850px;
padding-bottom: 0px;
padding-left: 0px;
padding-right: 0px;
padding-top: 0px;
position: absolute;
width: 850px;

body[bgcolor=#FFFFFF]
background-color: white;

body
background-image: url(http://webkit.…
```

If you develop websites, web applications, or browser extensions, you should learn about Safari's built-in developer tools.

At a Glance

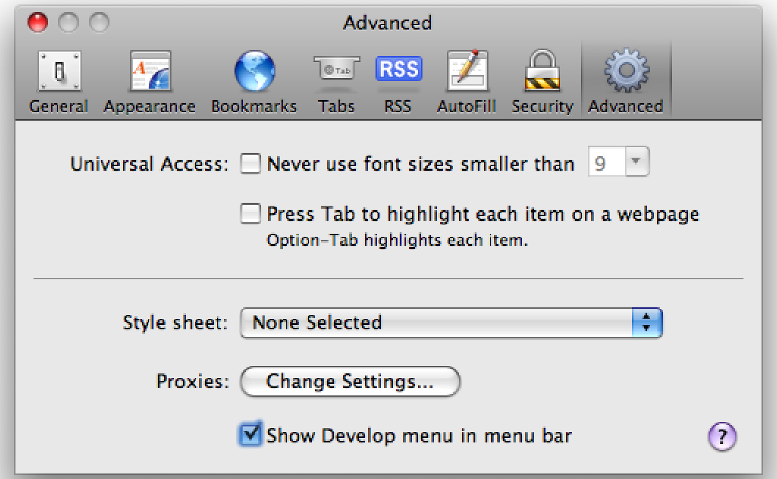
Safari has tools for prototyping HTML, CSS, and JavaScript, tools for interactively inspecting and debugging elements, attributes, properties, and styles, an integrated JavaScript debugger, and optimization tools such as a latency and download timeline and a JavaScript profiler.

These tools are built into Safari on the desktop (Mac OS X and Windows) and you can enable them in other Webkit-based applications. A subset of the tools are available in Safari on iPhone OS (iPhone, iPad, and iPod touch).

Enable the Developer Tools in Safari Preferences

You enable the developer tools Safari Preferences, as illustrated in Figure I-1.

Figure I-2 Showing the Develop menu



Universal Access: ☐ Never use font sizes smaller than 9

☐ Press Tab to highlight each item on a webpage
Option-Tab highlights each item.

Style sheet: None Selected

Proxies: Change Settings...

☑ Show Develop menu in menu bar

- On Mac OS X or Windows, use Safari Preferences to show the Develop menu in Safari, with the full set of developer tools available from the menu.
- On iPhone OS, use Safari Preferences to add an error console to Safari. For details, see "Enabling and Using Developer Tools in Safari on iPhone OS" (page 16).
- You can add the developer tools to other Webkit-based applications as a contextual menu by modifying the application's `.plist` file. For details, see "Enabling Developer Tools in WebKit-Based Applications Other Than Safari" (page 16).

Speed Up Prototyping by Interactively Testing for Errors

Show the Error Console while prototyping your website to quickly spot HTML and JavaScript structure and syntax errors. The Error Console shows you errors and warnings, highlights the line number of the problem in your source file, and tells you how Safari dealt with the error (by ignoring an extra closing tag, for example). For details, see "Using the Error Console" (page 20), "Viewing Errors" (page 21), and "Using the Error Console to Prototype JavaScript" (page 22).

Safari also comes with a Snippet Editor, where you can type in HTML and JavaScript snippets and see them evaluated interactively. No more having to create a dummy HTML page to test an element or a JavaScript function—just type in the part you're working on and see the results. When it works as you want it to, copy and paste into your actual webpage. For details, see "Using the Snippet Editor" (page 20).

If your website has different code paths for different browsers, have Safari give different user-agent strings to test the code branches—verify the behavior changes for Safari, Internet Explorer, Opera, Chrome, or Safari on iPhone OS without having to switch browsers, operating systems, or devices. When you're ready to open your website in another application, make the switch from inside Safari, without having to quit, open another application, and navigate to your site. For details, see "Changing the User Agent String" (page 22) and "Switching To Another Application" (page 23).

Relevant Chapter: "Prototyping Your Website" (page 19)

Debug Your HTML and CSS Interactively Using the Web Inspector

The Error Console finds and identifies syntax and structural errors, but sometimes your website doesn't look or behave as you want, even though all the syntax and structure is legal. That's when you need the Web Inspector. The Web Inspector shows you the DOM as it exists in memory—for a static HTML page, that's often identical to the page source, but for websites that modify the DOM using JavaScript and CSS, it can be very different.

- **Find Things Fast**

  Hover over an element in the DOM and it's highlighted in the browser window. Control-click or right-click on something in the browser window, choose Inspect Element from the contextual menu, and the element is highlighted in the DOM.

- **Change Things on the Fly**

  Click an element in the DOM and see its attributes, styles, metrics, and properties, as well any event listener functions attached to it. Double click the element to add, delete, or edit attributes interactively. Click the value of a style, property, or metric to modify or disable it. Increment or decrement numeric values using the cursor keys, in steps of 0.1, 1, or 10. Right-click or control-click to edit the DOM as if it were HTML in a text editor. Any changes you make are shown immediately in the browser window.

- **Get it Working in the Browser Before You Change the Source**

  Make changes without affecting your website, or modify any website inside the browser to better understand how it works or how you could adapt it to your needs. Find problems and fix them on a live website without making a copy to a new site or modifying your production site's source code. Copy and paste the modified code into your source after it's fully tested and works exactly the way you want it to.

Relevant Section: For details, see "Debugging HTML and CSS Using the Web Inspector" (page 26).

Use the Web Inspector and Console to Debug JavaScript Interactively

The Web Inspector and the Console work together to help you find and fix problems in your JavaScript—set breakpoints, pause, inspect variable values, see the call stack, log messages and data to the console, set variable values and continue, enter JavaScript on the fly to test it—all with auto-completion of function, property, and variable names to speed you along.

- **See It All**

The Web Inspector shows all the JavaScript sources—inline functions in the HTML, included `.js` files, output from server-side scripts, and code generated on the fly by other code. All the resources are listed. Click a resource in the list to see the working code.

- **Pause Execution When You Want**

  Click in the gutter anywhere in the code display to set a breakpoint, without having to edit the source. Set conditional breakpoints. Disable breakpoints and re-enable them with a click. Pause any time with a mouse click, without setting a breakpoint.

- **Continue When and How You Like**

  Check the call stack, examine variable values, and change values while paused, enter and execute test code, then continue, optionally stepping through the code function by function. Step over functions or step out of a function at will.

- **Use the Interactive Console**

  Safari implements the same Console API as the popular Firebug debugger. Add test code to your scripts to log branches in the code or print variable values on the fly, without pausing or setting breakpoints. You can have multiple independent consoles to monitor interaction between windows or tabs. Type commands in the console, with helpful auto-completion, and see the results immediately.

Relevant Sections: "Debugging JavaScript Using the Web Inspector" (page 31), "Using the Console to Debug JavaScript" (page 34).

Get Help with Cookies, Local Storage, and HTML5 Client-Side Databases

Use the Web Inspector's Storage panel to inspect cookies, local key/value storage, and even client-side relational databases created with HTML5. All local data is displayed in editable data grids. You can perform actions or enter data in the webpage and see the results in the grid, or enter data interactively in the grid itself. Issue SQL queries right from the Web Inspector, with auto-completion of SQL functions and database field names.

Relevant Section: "Analyzing Client-Side Storage, Databases, and Cookies" (page 38)

It Works—Now Make It Fly

Once your site is working as it should, you may be understandably reluctant to make changes to optimize performance. Fear not. The Web Inspector's Resource and Timeline panels show you exactly where time is spent, so you know where bottlenecks are before you change anything.

Once you know exactly where the bottlenecks are, you can make changes interactively without leaving Safari, optimizing your site—and fixing any problems that may cause—*before* you modify your working source.

The Resources panel shows all the resources your site uses—HTML files, JavaScript files, CSS stylesheets,`asp` output, images, and media files—in a timeline view, showing when each resource was requested, the latency for each request, when the first byte of the resource arrived, and when the resource finished loading. See at a glance if your website is being hung up by a slow server script, bandwidth-choking file sizes, network latency problems, or unexpected dependencies such as a resource that is not requested until after a script executes.

The Timeline panel lets you profile your website in real time, showing not only the latency and load time for each resource, but the time spent executing scripts and rendering the output. Complex interactions—a script that changes the DOM, causing a resource to load, and a table to be re-rendered, for example—are laid out in a clear and easy to understand fashion.

Relevant Sections: "Optimizing Download Time" (page 43), "Optimizing Loading, Scripting, and Rendering Times" (page 45).

If the timeline shows that significant time is being spent in your scripts, use the Profiles panel to see where in your script the time is being spent. Moving a single function call from the inside of a loop to the outside of the loop can sometimes have a tremendous impact. Running a profile takes the guesswork out of optimizing JavaScript. A profile shows you how much time is spent in each function, including dependent functions, and how many times each function is called.

See the output in milliseconds or percent of execution time. Sort by execution time. Then spend your effort optimizing code that you know is going to make a significant difference—modify only functions that take a proportionally long time to execute or functions that are called many times.

Relevant Section: "Optimizing JavaScript" (page 46)

Be a Power User

The Safari developer tools include dozens of keyboard and mouse shortcuts to speed up common operations, from opening the Web Inspector to cycling through auto-completion suggestions. See the shortcuts table in "Keyboard and Mouse Shortcuts" (page 49) to give your productivity a boost.

Enable, Build, and Debug Extensions

Extensions are a new feature in Safari 5.0. Because they're so new, they are disabled by default, and are enabled in the Develop menu—making them a developer feature for Safari 5.0.

Once you enable extensions, you can use Extension Builder to create them. You need to join the Safari Developer Program to create extensions—you can't install an extension without a signed certificate. Go to developer.apple.com to join the program.

When you're ready to build an extension, read *Safari Extensions Development Guide*, and refer to *Safari Extensions Reference*. If you've already developed extensions for other browsers, see *Safari Extensions Conversion Guide* for time-saving tips.

Once you've built and installed an extension, you can use the Web Inspector to debug and optimize it.

- Injected scripts and stylesheets are inspected and debugged exactly as if they were downloaded from the webpage's host.
- Extension bars can be inspected and debugged by right-clicking or control-clicking on the extension bar and using the contextual menu.

- A global HTML page can be inspected and debugged by clicking Inspect Global Page in Extension Builder (the button is present only if the selected extension is installed and has a global page).

Each webpage, extension bar, and global page has its own console. The full set of developer tools for inspecting, modifying, and optimizing HTML, CSS, and JavaScript works on extensions.

Note: When inspecting storage, remember that the domain for the global page or extension bars is the extension, but the domain of injected scripts is the domain of the webpage the script is injected into.

See Also

- *Safari Extensions Development Guide*—step-by-step directions for creating Safari extensions using Extension Builder
- *Safari Extensions Reference*—the JavaScript classes, methods, and properties you can access from Safari extensions
- *Safari HTML Reference*—the supported HTML tags for Safari
- *Safari CSS Reference*—the supported CSS tags for Safari
- *Safari Web Content Guide*—guidance for developing web content for the iPhone
- *Web Page Development: Best Practices*—Apple recommendations for webpage development
- *WebKit DOM Programming Topics*—articles on using and modifying the Document Object Model
- *WebKit DOM Reference*—syntax rules for working with the DOM for Safari and other WebKit-based applications

Overview of Developer Tools for Safari

There are developer tools built into Safari on the desktop (Mac OS X and Windows), Safari on iPhone OS, and in other WebKit-based applications. These tools can help you to prototype, debug, and optimize your website. This chapter gives a quick overview of how to enable and use these developer tools.

Developing Websites with the Developer Tools

The development process for websites can be accelerated by the Safari toolset at several points. The usual process of development is as follows:

- Prototype—Determine what combination of HTML, JavaScript, CSS, and database deliver the functionality you need, testing snippets of code interactively.
- Write—Author the website, typically using tools such as Dashcode and HTML editors.
- Test and debug—Test using several browsers and platforms (Mac, Windows, iPhone), track down errors and correct them.
- Optimize—Make your website more responsive, shorten load times, and improve JavaScript execution.

Once you've enabled and familiarized yourself with the developer tools, you can use the Snippet Editor to streamline prototyping, the Error Console and Web Inspector for testing and debugging, and the Web Inspector's timeline view and JavaScript profiler to help you optimize your website.

Differences Between Safari on the Desktop, Safari on iPhone OS, and WebKit

Safari on iPhone OS (iPhone, iPad, and iPod touch) contains a simple Debug Console that you can enable to help you debug websites and web applications directly from the mobile device.

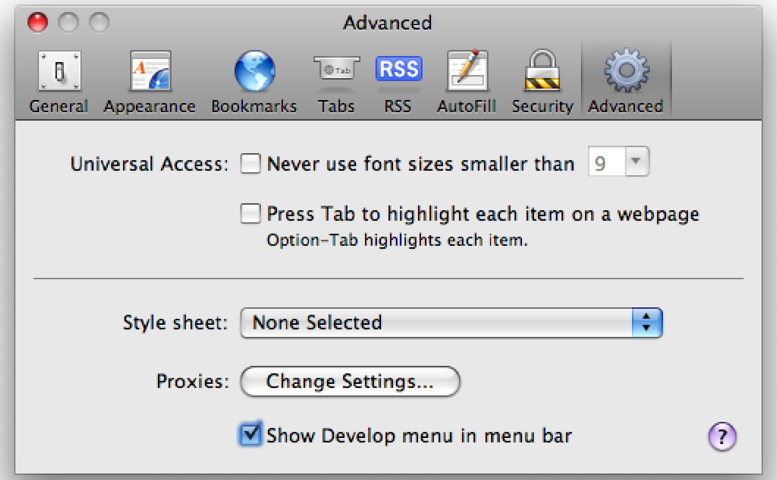
The toolset for Safari on the desktop (Mac OS X or Windows) is far more extensive. It includes a Develop menu with several commands and a number of interactive tools.

The toolset for WebKit-based applications is essentially the same as for Safari on the desktop, but you enable the tools differently, and the Develop menu is a contextual menu.

Enabling Developer Tools in Safari on the Desktop

You enable the developer tools in Safari on the desktop (Mac OS X or Windows) by turning on the Develop menu. In Safari preferences, click Advanced, then select "Show Develop menu in menu bar," as shown in Figure 1-1.

Figure 1-1 Safari preferences



Universal Access: ☐ Never use font sizes smaller than 9
☐ Press Tab to highlight each item on a webpage
Option-Tab highlights each item.

Style sheet: None Selected

Proxies: Change Settings...

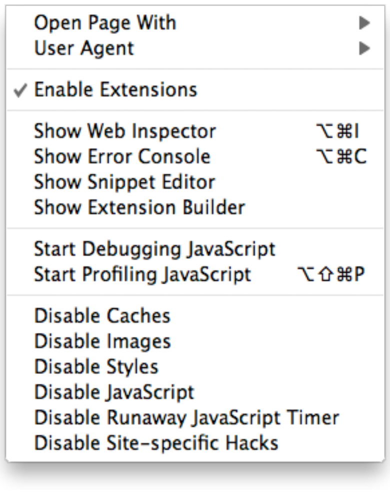
☑ Show Develop menu in menu bar

Selecting this option adds a Develop menu to your menu bar (Figure 1-2 (page 15)). The Safari developer tools are now enabled.

The Develop Menu Command Summary

The Develop menu contains a set of tools to assist you in prototyping, debugging, and optimizing your website.

Figure 1-2 The Develop menu



- Open Page With—Open the current webpage in another application.
- User Agent—Browsers send a user agent string that identifies the browser type and version to the server. The same string is sent in response to a JavaScript request for the user agent string. Use this menu item to modify the user agent string Safari sends.
- Enable Extensions—Extensions are a new feature of Safari. Because they are new in Safari 5.0, they are disabled by default, and thy can be turned on only in the Develop menu. For the time being, they are a developer feature. For details, see *Safari Extensions Development Guide*.
- Show Web Inspector—Open the Web Inspector window to inspect or modify the DOM, HTML attributes, and CSS properties.
- Show Error Console—Open the Error Console window to see any HTML or JavaScript errors and any corrective actions taken by Safari.
- Show Snippet Editor—Open the Snippet Editor window to interactively prototype HTML, CSS, or JavaScript snippets.
- Show Extension Builder—Open Extension Builder to install, modify, create, or uninstall a Safari extension. For more information, see *Safari Extensions Development Guide*.
- Start Debugging JavaScript—Turn on the interactive JavaScript debugger to set breakpoints, inspect variables, and so on.
- Start Profiling JavaScript—Turn on the JavaScript profiler to see how many times each function is called, how long it takes, and so on.
- Disable Caches—Turn off caching to see how a website loads the first time.
- Disable Images—Turn off image display and view websites as text only.
- Disable Styles—Turn off CSS style properties to view the page purely as HTML and JavaScript.
- Disable JavaScript—View websites with the JavaScript interpreter disabled.
- Disable Runaway JavaScript Timer—Do not prematurely terminate JavaScript functions, no matter how long they take.
- Disable Site Specific Hacks—If Apple engineers have modified Safari specifically to work around a problem with your website, use this to disable the modifications to Safari and test your site for correct operation.

Enabling Developer Tools in WebKit-Based Applications Other Than Safari

Applications make use of a `.plist` file, a set of key-value pairs in the preferences folder, that are used to configure the application. To enable the developer tools in a WebKit-based application other than Safari, set the `WebKitDeveloperExtras` key to the Boolean value `True` in the `.plist` file.

From the command console, type:

```
defaults write com.myApp WebKitDeveloperExtras bool true
```

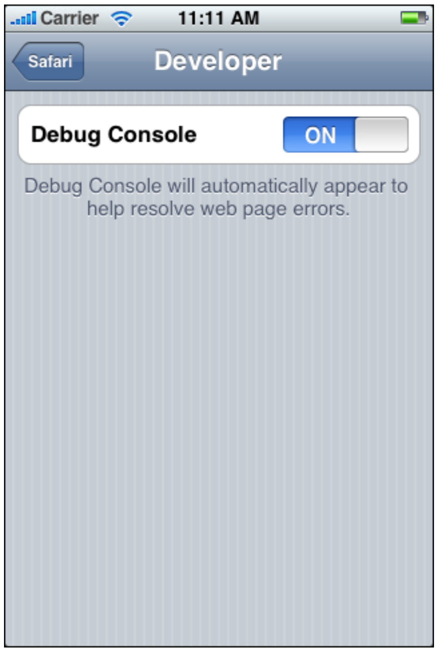
replacing `myApp` with the bundle identifier of your application.

You must also enable contextual menus in your application. Once this is done, launch the application. The Develop menu can now be accessed by a control-click or right-click from within the application.

Enabling and Using Developer Tools in Safari on iPhone OS

You can enable a Debug Console in Safari on iPhone or iPod touch, which allows you to see HTML, CSS, and JavaScript errors directly on the iPhone or iPod touch. To enable the console, tap the Settings icon, then tap Safari and scroll down to the bottom of the screen, then tap Developer. From here you can turn the console on or off, as shown in Figure 1-3

Figure 1-3 Safari iPhone settings

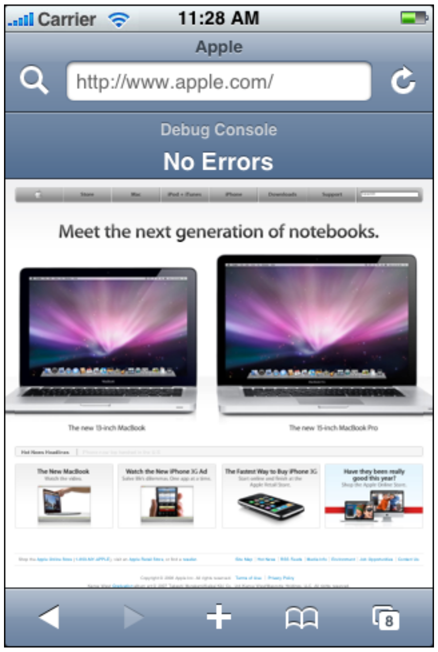


Debug Console ON

Debug Console will automatically appear to help resolve web page errors.

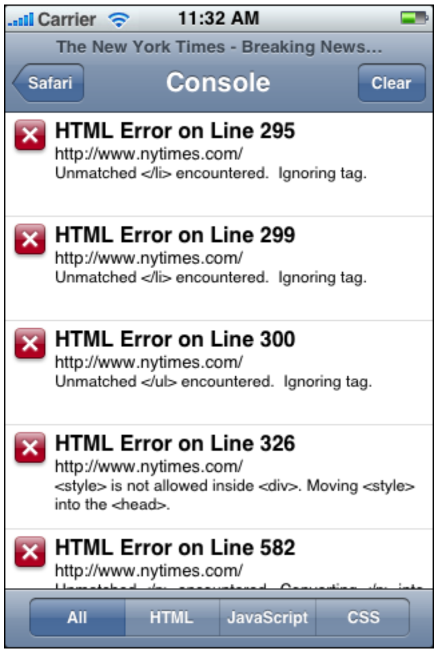
Once the Debug Console is enabled, Safari records any errors it encounters when accessing a website. A Debug Console report appears at the top of all displayed webpages, as illustrated in Figure 1-4.

Figure 1-4 The Debug Console report



If there are no errors, the words “No Errors” are displayed. If any errors are encountered in the HTML, JavaScript, or CSS, the number of errors is displayed, followed by a right-pointing arrow. Tapping the carat brings up the Debug Console.

Figure 1-5 The Debug Console



HTML Error on Line 295
http://www.nytimes.com/
Unmatched </li> encountered. Ignoring tag.

HTML Error on Line 299
http://www.nytimes.com/
Unmatched </li> encountered. Ignoring tag.

HTML Error on Line 300
http://www.nytimes.com/
Unmatched </ul> encountered. Ignoring tag.

HTML Error on Line 326
http://www.nytimes.com/
<style> is not allowed inside <div>. Moving <style> into the <head>.

HTML Error on Line 582
http://www.nytimes.com/

A scrollable list of errors is displayed. You can choose to see all errors or limit the display to only the HTML, JavaScript, or CSS errors. The line number on which the error occurred is displayed, along with a brief description of the error.

If you choose to see JavaScript errors, you also see JavaScript log events. All output from the JavaScript functions `console.log()`, `console.info()`, `console.warn()`, and `console.error()` are logged to the Debug Console. By judiciously placing log entries into your JavaScript, you can trace the code path that executes on an iPhone or iPod touch.

Note: Currently, long log entries are truncated to fit the space available on the Debug Console. For best results when using the Debug Console to track JavaScript log events in Safari on iPhone or iPod touch, keep your log entries terse.

Prototyping Your Website

The first step in designing a website is primarily esthetic—how do you want it to look and feel? The next several steps are technical—can you achieve what you want using a given combination of HTML, CSS, and JavaScript? Does a particular JavaScript function do what you need? How does applying a specific CSS style affect the page? Does the combined set work on the browsers and platforms you want to support?

The Safari developer tools can help you answer the technical questions in a streamlined and efficient manner.

The Prototyping Process, Improved

It's common to prototype a website by creating a combination of HTML, style sheets, and scripts, load the combination into a browser, see problems, modify the source—or multiple sources—then reload the page to see the results. It's a cumbersome and sometimes painful process.

It's inefficient to create a complete webpage before you can test any part of it, then test it using several browsers, rewriting and reloading the page each time you find an error, going back and forth between your HTML, CSS, and JavaScript sources. If your webpage contains a hidden error, and different browsers deal with the error differently, it can be frustrating to find the problem.

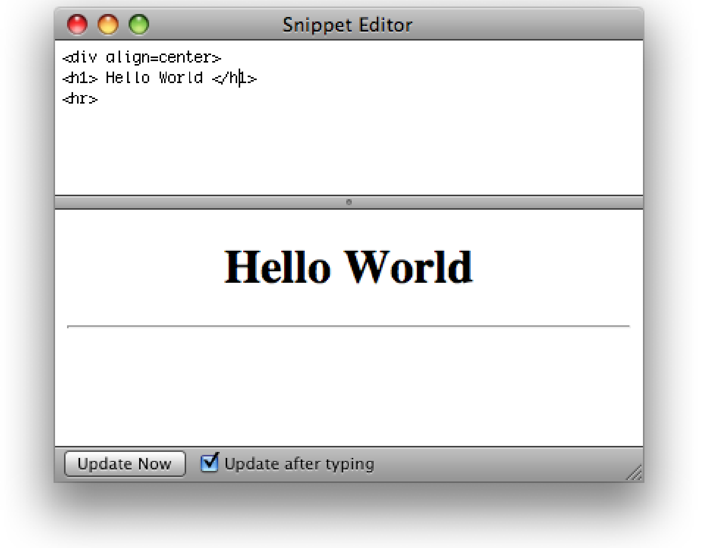
Safari can help.

- Use the Snippet Editor to interactively test elements of your page in a sandbox, debugging your syntax and testing different atributes and elements without going through the process of creating a whole website. Test the snippets first—set an HTML attribute, apply a CSS style, call a JavaScript function—and see the results immediately, then build your prototype website using a combination of parts that you already know work.
- Once you have a prototype, load it in Safari and use the Error Console to spot hidden errors that Safari is dealing with for you—other browsers may deal with them differently, resulting in a different experience. Correct the errors to maximize your chances of compatibility.
- If you use the user agent string to execute different code branches that are intended to execute on different browsers, choose User Agent String from the Develop menu to modify the user agent string and invoke the different branches, making sure that each browser is being shown what you intend. Resolve the question of proper code branch versus browser differences in advance.
- Finally, you can test your prototype using other browsers or web applications by invoking them directly from the Develop menu in Safari.

Using the Snippet Editor

Choose Show Snippet Editor from the Develop menu to open the Snippet Editor. The Snippet Editor contains an upper pane, in which you can type any combination of HTML, CSS, or JavaScript, and a lower pane that shows how it displays in Safari (or any WebKit-based application), as shown in "The Snippet Editor."

Figure 2-1 The Snippet Editor



```
<div align=center>
<h1> Hello World </h1>
<hr>
```

The Snippet Editor provides an interactive interface for quickly prototyping or debugging your HTML, CSS, and JavaScript without having to create complete HTML pages and open them in a browser, or to switch between your editor and browser repeatedly to edit the code and refresh the browser display. Simply type in fragments of code and the display refreshes immediately. When your code produces the effect you want, you can copy and paste it into a working document.

If the "Update after typing" option is selected, the display pane is updated each time you press a key. This is ideal for working with short snippets of HTML or debugging the syntax in a line of JavaScript. For longer snippets, where the display of unfinished code would be distracting, deselect the option and click the Update Now button to refresh the display when you are ready.

Using the Error Console

The Error Console is the most basic tool for debugging a website, so it's an appropriate tool for prototyping. The Error Console notifies you of any syntax or structural errors that Safari detects in your HTML, CSS, or JavaScript, gives you the location of the error, including the source file and line number, and includes a brief description of how Safari dealt with the error (such as by ignoring an extra closing tag).

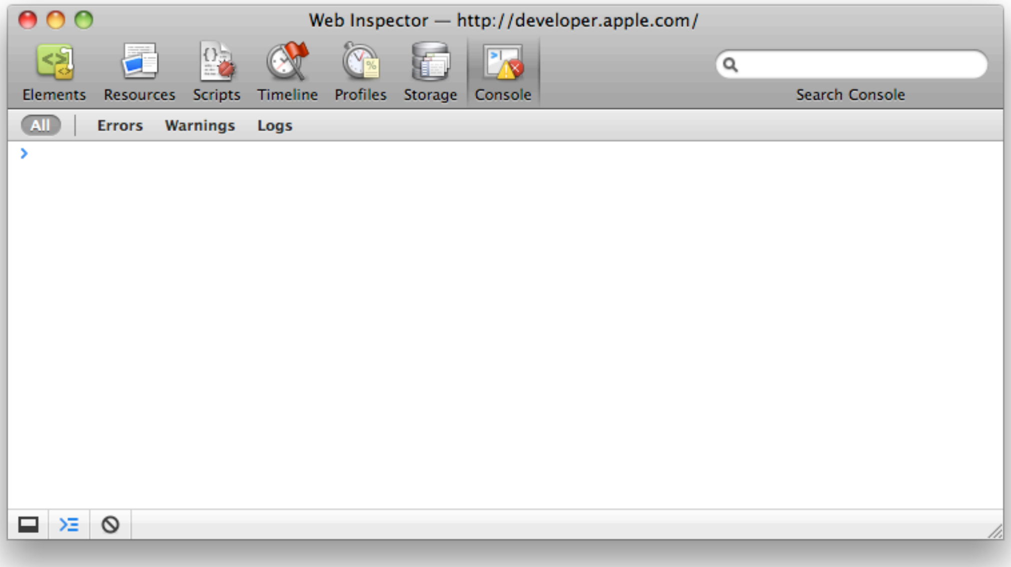
Note: Safari on iPhone OS also has an Error Console, although it is more limited. If you are testing content on the iPhone, enable the Error Console as described in "Enabling Developer Tools in Safari on iPhone and iPod Touch" (page 16).

Opening the Error Console

There are several ways to open the Error Console. You can choose Show Error Console from the Develop menu, or, if the Web Inspector is open, you can click the Console button (greater-than sign and horizontal lines), press the Esc key, or click the error or warning button in the bottom bar.

The Error Console opens at the bottom of the Web Inspector, as shown in Figure 2-2.

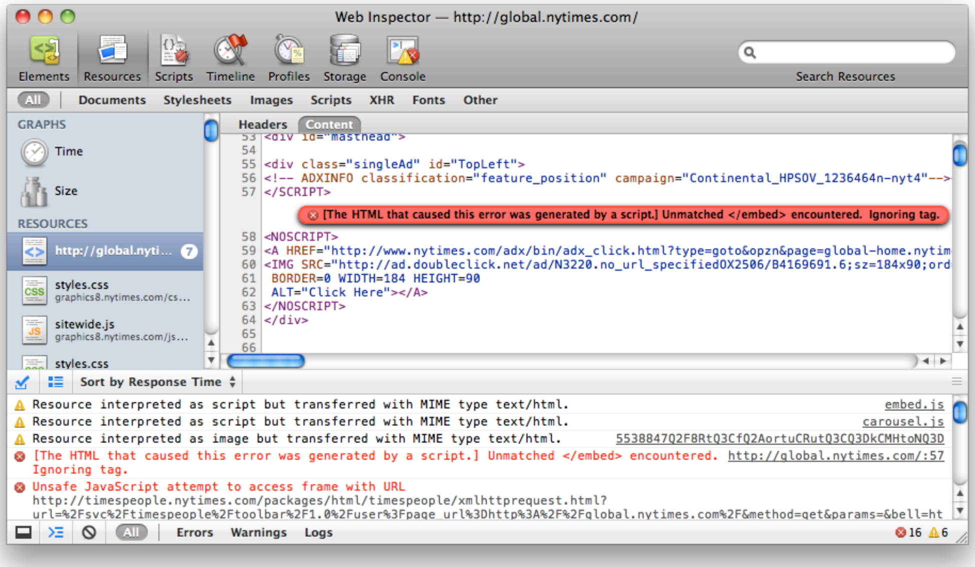
Figure 2-2 The Error Console



To close the Error Console again, click the Console button in the bottom bar.

Viewing Errors

If any errors or warnings are encountered when loading a website, the number of each is displayed in the bottom bar. Clicking the error or warning button opens the Error Console and displays all of the errors and warnings.



```
<div id="masthead">

<div class="singleAd" id="TopLeft">
<!-- ADXINFO classification="feature_position" campaign="Continental_HPSOV_1236464n-nyt4"-->
</SCRIPT>
    [The HTML that caused this error was generated by a script.] Unmatched </embed> encountered.  Ignoring tag.
<NOSCRIPT>
<A HREF="http://www.nytimes.com/adx/bin/adx_click.html?type=goto&opzn&page=global-home.nytim
<IMG SRC="http://ad.doubleclick.net/ad/N3220.no_url_specifiedOX2506/B4169691.6;sz=184x90;ord
 BORDER=0 WIDTH=184 HEIGHT=90
 ALT="Click Here"></A>
</NOSCRIPT>
</div>
```

```
Resource interpreted as script but transferred with MIME type text/html.                embed.js
Resource interpreted as script but transferred with MIME type text/html.             carousel.js
Resource interpreted as image but transferred with MIME type text/html.   5538847Q2F8RtQ3CfQ2AortuCRutQ3CQ3DkCMHtoNQ3D
[The HTML that caused this error was generated by a script.] Unmatched </embed> encountered. http://global.nytimes.com/:57
Ignoring tag.
Unsafe JavaScript attempt to access frame with URL
http://timespeople.nytimes.com/packages/html/timespeople/xmlhttprequest.html?
url=%2Fsvc%2Ftimespeople%2Ftoolbar%2F1.0%2Fuser%3Fpage_url%3Dhttp%3A%2F%2Fglobal.nytimes.com%2F&method=get&params=&bell=ht
```

Errors and warnings are shown with a URL link to the resource that generated the problem, the line number (where applicable), and a brief description of how Safari dealt with the problem.

Click the link in the error listing to open the source in the upper pane. The error or warning is also displayed in the source.

Using the Error Console to Prototype JavaScript

Like the Snippet Editor, the Error Console allows you to enter JavaScript interactively and see the results immediately. In addition, a number of `console` functions can be used to log data to the Error Console (see "Safari JavaScript Console API" (page 36)).

The console has auto-completion support for JavaScript. As you type, JavaScript variables, properties, and function names are suggested in gray. Pressing the Right Arrow key accepts the current suggestion. If multiple selections begin with the same prefix, you can cycle through the suggestions using the Tab key. If there is only one suggestion, the Tab key accepts it.

Typing a variable name and pressing Enter displays the variable's current value.

Any changes you make to the DOM using JavaScript from the console are immediately displayed in the Web Inspector's Elements pane, as well as in the browser window.

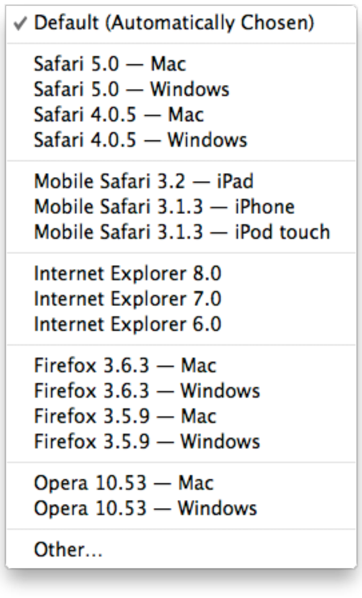
Changing the User Agent String

Every browser has a user agent string that identifies its type and version number. The browser sends this string to the server. Your website can also read the user agent string using JavaScript. This is one way to determine what version of which browser a user is running. You can choose what Safari reports as its user agent from the User Agent submenu.

This can be useful to quickly test your code to see if it is reacting to various user agents as you expect, without having to actually load the page in multiple versions of multiple browsers. An example of the User Agent submenu is shown in Figure 2-3.

Note: The browser versions listed in the submenu are updated frequently to reflect current availability.

Figure 2-3 An example of the User Agent submenu



You can choose the common versions of most popular browsers from the submenu. Note that the list includes the versions of Safari found on iPhone, iPad, and iPod touch.

The Other... menu item opens a sheet showing the default user agent string, which you can inspect and edit to any string you like.

If your website has different code branches for different browsers, and loading the site in a given browser reveals problems, one of the first questions is whether the code has actually branched as expected. By changing the user agent string in Safari, you can isolate the code branch from the browser differences. You can also log the code branch to the console and check for it using Safari's Error Console.

Switching To Another Application

When first testing a website, you typically open it in several browsers, such as Safari, Internet Explorer, and Firefox, to make sure that it works correctly in all cases. The Open Page With command is a convenient way to open the current webpage in another browser, without having to leave Safari, open the other browser, and navigate to the page.

Choose Open Page With from the Develop menu. A submenu is displayed listing all the applications known to the operating system that can open the page.

Debugging Your Website

Safari has a number of tools for finding and correcting problems with your website. The best tool for the job depends on what kind of problems you're experiencing. Basic testing with the Error Console will reveal most syntax and structural problems. More complex problems can usually be resolved using the Web Inspector.

Use Cases

Here are some common cases and the best ways to deal with them.

1. **You have a new website that you're ready to start testing, or a half-finished website that you're developing.**

   See the section on "Prototyping Your Website" (page 19). It includes basic testing.

2. **You have a website designed for Internet Explorer on Windows, and you are having trouble making it work in Safari on the desktop or iPhone OS.**

   Use the Error Console to see if Safari detects any syntax or structural errors in your HTML, CSS, or JavaScript, and if so, what corrective action it is taking. This will also reveal use of extensions that may be proprietary to Explorer.

   Note: If Safari reports a tag that works in Explorer as an error, it is not a standard tag, and you need to use an equivalent tag instead, or include a branch in your code that uses one tag for Explorer and another tag for other browsers.

   See "Using the Error Console" (page 20) for a description of basic testing. Start by correcting the reported errors. In most cases, that will solve the problem. If no errors are reported, or you correct the reported errors and problems persist, see the following use cases.

3. **Your website doesn't work on iPhone, but the Error Console shows no errors.**

   Enable the Error Console for iPhone or iPod touch (see "Enabling and Using Developer Tools in Safari on iPhone OS" (page 16)) and check for errors on the device itself. See *Safari Web Content Guide* for guidance on specific design considerations for iPhone web content.

4. **Your website doesn't look or behave as you expect, but the Error Console shows no errors.**

   See the sections in this chapter, "Debugging HTML and CSS Using the Web Inspector" (page 26) and "Debugging JavaScript Using the Web Inspector" (page 31), to learn how to use the developer tools to analyze and debug website behavior.

5. **You are having problems with an HTML5 client-side database.**

   See the last section in this chapter, "Analyzing Client-Side Database Storage" (page 38).

6. **Your website works, but is sluggish or unresponsive.**

   See the next chapter, "Optimizing Your Website" (page 43).

Debugging HTML and CSS Using the Web Inspector

If your website doesn't look or act as you expect, and the Error Console doesn't report any errors, analyze your site using the Web Inspector.

Choose Show Web Inspector from the Develop Menu. This opens the Web Inspector window. Click the Elements button on the Web Inspector toolbar.

In the Elements pane, the left half of the Web Inspector contains the DOM of the current webpage, as a collapsable and expandable structure of nested elements. Click the disclosure triangle to expand or collapse the view of a given element and its contents. A breadcrumb path is added to the bottom bar, allowing you to see where you are in the DOM hierarchy. You can click a breadcrumb to move back up the hierarchy.

The DOM displayed is the symbolic structure of the webpage that Safari has constructed in memory. In a simple static webpage with no errors, the DOM is identical to the HTML source. In websites where the DOM changes interactively, the Elements pane gives you the current state of the DOM. If there are errors in the webpage, the Elements pane shows you the DOM that Safari has constructed, which may differ significantly from the source.

When you hover over an element in the DOM, the corresponding element is highlighted in the browser window. If you control-click in the browser window, a contextual menu is displayed with an "Inspect Element" choice. Choosing Inspect Element highlights the corresponding element in the DOM. This makes it easy to zoom in on a given element and find its location in the source, even in a complex website.

Note: Another way to get from an element in the browser display to its definition in the DOM is to click the magnifying glass button in the bottom bar, then move the cursor over the browser window. Elements are highlighted as the mouse passes over them. Clicking an element zooms the DOM tree to the element's definition and highlights it.

The right side of the pane displays the styles, metrics, and properties of the currently selected element.

Using the DOM view along with the styles, metrics, and properties, you can inspect and interactively modify any element on a webpage. More significantly, you can quickly grasp the inheritance structure that gives each element its appearance, placement, and behavior.

When debugging a webpage, it's typically best to control-click on the part of the page that looks wrong in Safari's browser window. Then look at the highlighted element in the DOM panel of the Web Inspector to see how the element is defined. You can interactively modify the HTML parameters in the Web Inspector to see how that changes the behavior in the browser window. If the HTML attributes seem correct, check the applied CSS styles by selectively disabling them or modifying them in the Web Inspector. The effects are immediately visible in the browser window. If this solves the problem, copy the modified HTML or CSS and paste it into your source. If not, the problem may be caused by an errant script. See "Debugging JavaScript Using the Web Inspector" (page 31).

If you know the name of an element in the DOM that you want to inspect (for example, to find all instances of a particular class), use the search bar in the upper right corner of the Inspector.

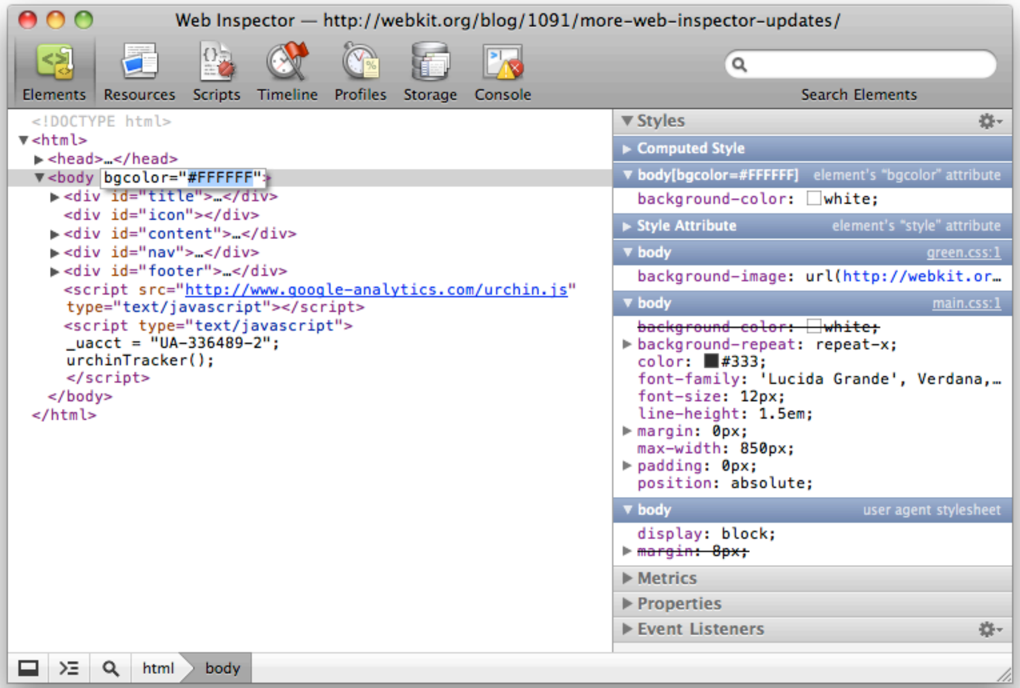
Note: In the Elements pane, the search field accepts Xpath and CSS selectors as well as plain text. For example, searches can be conducted as plain text, with an Xpath query using `document.evaluate()`, and with a CSS selector using `document.querySelectorAll()`. All search results are highlighted in the DOM tree, with the first match revealed and selected.

The following subsections show how to inspect and modify HTML and CSS using the Web Inspector.

Inspecting and Editing DOM Attributes

The left pane shows the DOM attributes associated with the currently selected element. Double-click an element name, attribute name, or attribute value to edit it interactively, as shown in "Editing DOM attributes."

Figure 3-1 Editing DOM attributes


Web Inspector — http://webkit.org/blog/1091/more-web-inspector-updates/
Elements
Resources
Scripts
Timeline
Profiles
Storage
Console
Search Elements
Styles
Computed Style
body[bgcolor=#FFFFFF] element's "bgcolor" attribute
Style Attribute element's "style" attribute
body green.css:1
body main.css:1
body user agent stylesheet
Metrics
Properties
Event Listeners
html
body

```
  <!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body bgcolor="#FFFFFF">
    ▶<div id="title">…</div>
     <div id="icon"></div>
    ▶<div id="content">…</div>
    ▶<div id="nav">…</div>
    ▶<div id="footer">…</div>
     <script src="http://www.google-analytics.com/urchin.js"
     type="text/javascript"></script>
     <script type="text/javascript">
     _uacct = "UA-336489-2";
     urchinTracker();
     </script>
   </body>
 </html>
```

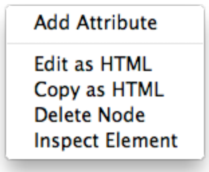
```
background-color: white;

background-image: url(http://webkit.or…

~~background-color: white;~~
▶ background-repeat: repeat-x;
  color: #333;
  font-family: 'Lucida Grande', Verdana,…
  font-size: 12px;
  line-height: 1.5em;
▶ margin: 0px;
  max-width: 850px;
▶ padding: 0px;
  position: absolute;

  display: block;
▶ ~~margin: 8px;~~
```

Use the tab key and shift-tab key combination to traverse the attributes.

You can edit values using the letter and number keys as you would expect. For numerical values, you can also use the arrow keys to increment or decrement the value by 1. Holding down the option or alt key increments or decrements the value by 0.1, while holding down the shift key increments or decrements by 10.

Right-click or control-click the element to bring up a contextual menu, as shown in Figure 3-2 (page 28).

Figure 3-2 Nodal context menu

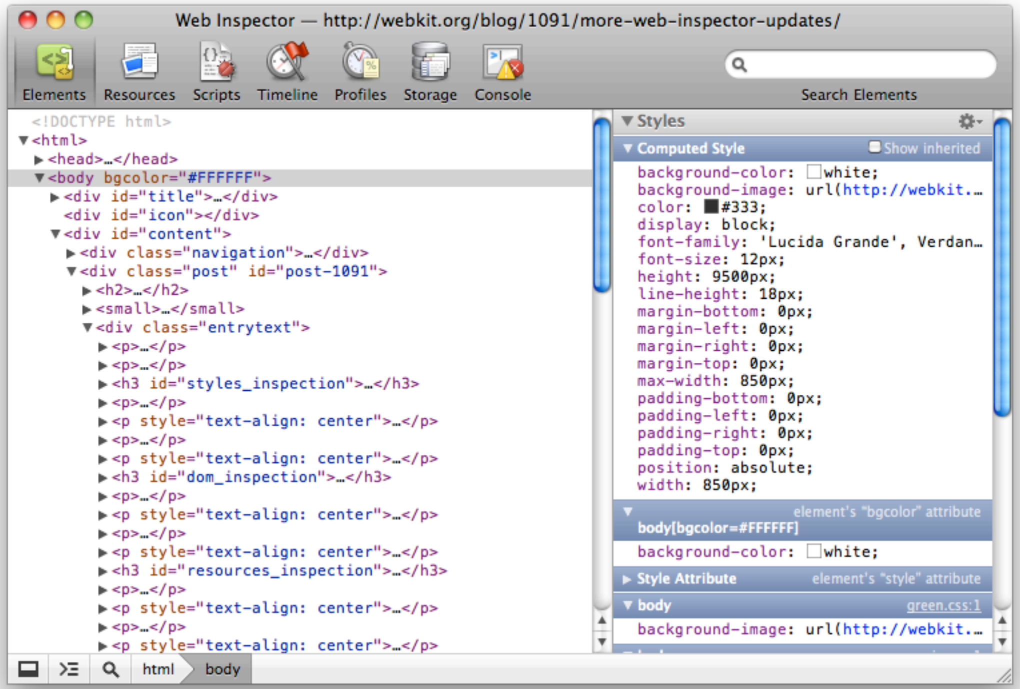


You can add a new attribute to the element, edit the DOM as if it were an HTML file in a text editor, copy the HTML for the element and all its children to the clipboard, of delete the element and all of its children. Choosing Inspect Element in this context allows you to inspect the Web Inspector itself.

Inspecting and Editing Styles

Click Styles in the list on the right, to see the CSS styles that are applied to the currently selected element, as shown in "Viewing styles."

Figure 3-3 Viewing styles



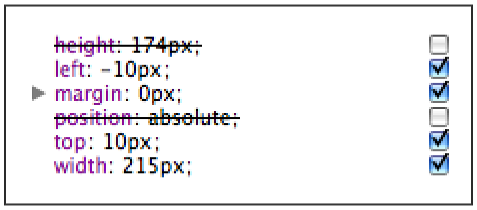
```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body bgcolor="#FFFFFF">
    ▶<div id="title">…</div>
     <div id="icon"></div>
    ▼<div id="content">
      ▶<div class="navigation">…</div>
      ▼<div class="post" id="post-1091">
        ▶<h2>…</h2>
        ▶<small>…</small>
        ▼<div class="entrytext">
          ▶<p>…</p>
          ▶<p>…</p>
          ▶<h3 id="styles_inspection">…</h3>
          ▶<p>…</p>
          ▶<p style="text-align: center">…</p>
          ▶<p>…</p>
          ▶<p style="text-align: center">…</p>
          ▶<h3 id="dom_inspection">…</h3>
          ▶<p>…</p>
          ▶<p style="text-align: center">…</p>
          ▶<p>…</p>
          ▶<p style="text-align: center">…</p>
          ▶<h3 id="resources_inspection">…</h3>
          ▶<p>…</p>
          ▶<p style="text-align: center">…</p>
          ▶<p>…</p>
          ▶<p style="text-align: center">…</p>
```

```
▼ Styles
▼ Computed Style                          Show inherited
  background-color: white;
  background-image: url(http://webkit.…
  color: #333;
  display: block;
  font-family: 'Lucida Grande', Verdan…
  font-size: 12px;
  height: 9500px;
  line-height: 18px;
  margin-bottom: 0px;
  margin-left: 0px;
  margin-right: 0px;
  margin-top: 0px;
  max-width: 850px;
  padding-bottom: 0px;
  padding-left: 0px;
  padding-right: 0px;
  padding-top: 0px;
  position: absolute;
  width: 850px;
▼                            element's "bgcolor" attribute
  body[bgcolor=#FFFFFF]
  background-color: white;
▶ Style Attribute              element's "style" attribute
▼ body                                        green.css:1
  background-image: url(http://webkit.…
```

The first section of text in the right pane shows the computed style for the selected element, which is the sum of all inherited and overridden styles specified for that element and its containers. This section is followed by the sections containing the CSS specifications that apply to the element, in hierarchical order. Select the "Show inherited" option to see the inherited default styles being applied as well.

Hovering over an editable style brings up a series of check boxes. Unchecking a box disables the application of that style property. The results are immediately displayed in the browser window. The style is then displayed in strikethrough text, as shown in "Editing style properties." Re-checking the box enables the style property again.

Figure 3-4 Editing style properties



```
~~height: 174px;~~         ☐
left: -10px;               ☑
▶ margin: 0px;             ☑
~~position: absolute;~~    ☐
top: 10px;                 ☑
width: 215px;              ☑
```

Double-clicking a style allows you to edit it on the fly and immediately see the difference in your browser window. You can edit properties in a few different ways:

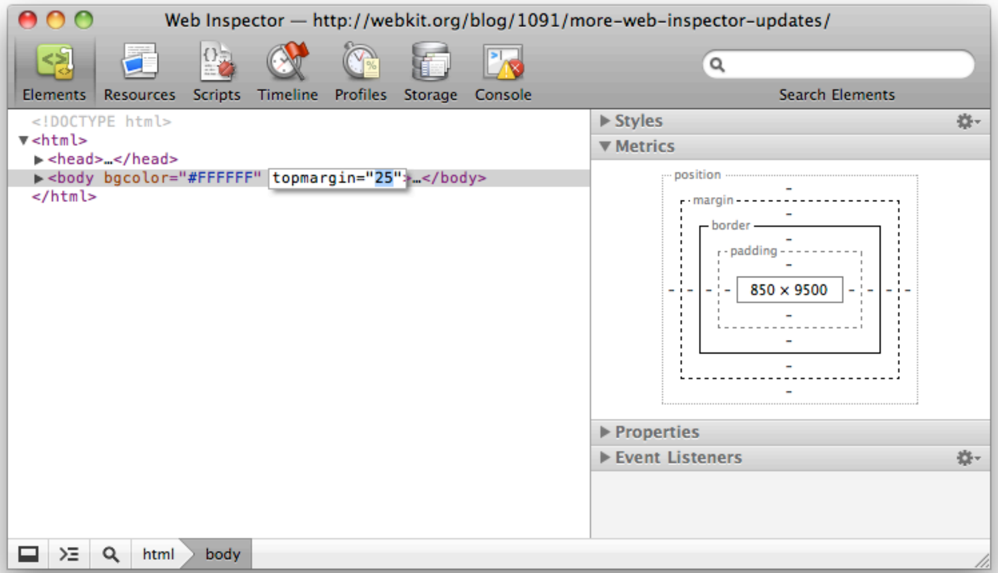
- Pressing Delete with the property selected deletes the property, if allowed.
- You can edit the property values using the keyboard.
- For numerical values, you can use the arrow keys to increment or decrement the value by 1. Holding down the option or alt key increments or decrements the value by 0.1, while holding down the shift key increments or decrements by 10.
- You can add style attributes by clicking in the white space or tabbing past the last attribute, or by appending a semicolon to the end of a line and typing in new style attributes.
- You can edit a selector by double-clicking it.
- You can create a new rule by choosing "Add New Style Rule" from the gear menu.
- You can cycle through different color representations—such as `white`, `#ffffff`, or `rgb(255,255,255)`, for example—by clicking on the color swatch beside a color value.

Because style properties are interactively editable, you can modify them until you have exactly the effect you want, before you change a line of source.

Inspecting and Editing Metrics

Click Metrics to see the spatial metrics for a given element—its height and width, along with the height and width of any borders, margins, or padding. Double-click the value of a metric attribute to edit it interactively, as shown in Figure 3-5.

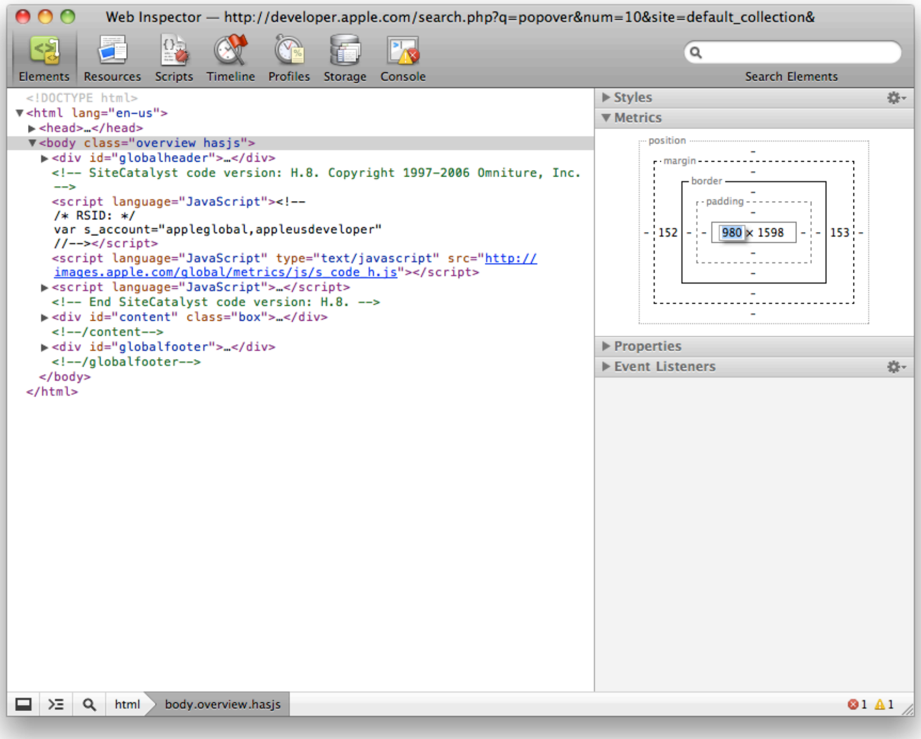
Figure 3-5 Editing metric attributes



```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▶<body bgcolor="#FFFFFF" topmargin="25">…</body>
  </html>
```

Alternatively, double-click a metric value displayed on the right to edit it directly, as shown in Figure 3-6

Figure 3-6 Editing metrics directly

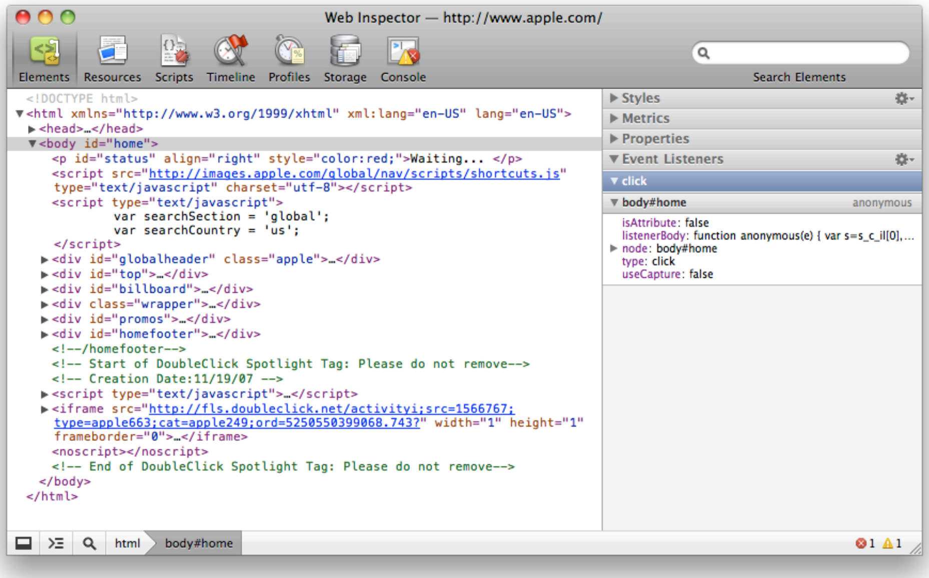


```
<!DOCTYPE html>
▼<html lang="en-us">
  ▶<head>…</head>
  ▼<body class="overview hasjs">
    ▶<div id="globalheader">…</div>
     <!-- SiteCatalyst code version: H.8. Copyright 1997-2006 Omniture, Inc.
     -->
     <script language="JavaScript"><!--
     /* RSID: */
     var s_account="appleglobal,appleusdeveloper"
     //--></script>
     <script language="JavaScript" type="text/javascript" src="http://
     images.apple.com/global/metrics/js/s_code_h.js"></script>
    ▶<script language="JavaScript">…</script>
     <!-- End SiteCatalyst code version: H.8. -->
    ▶<div id="content" class="box">…</div>
     <!--/content-->
    ▶<div id="globalfooter">…</div>
     <!--/globalfooter-->
   </body>
 </html>
```

Inspecting Listener Functions

If any JavaScript functions have been added as event listeners, you can inspect them by clicking Event Listeners in the right hand pane, as shown in Figure 3-7.

Figure 3-7 Event Listeners



```
<!DOCTYPE html>
▼<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-US" lang="en-US">
  ▶<head>…</head>
  ▼<body id="home">
     <p id="status" align="right" style="color:red;">Waiting... </p>
     <script src="http://images.apple.com/global/nav/scripts/shortcuts.js"
     type="text/javascript" charset="utf-8"></script>
     <script type="text/javascript">
             var searchSection = 'global';
             var searchCountry = 'us';
     </script>
    ▶<div id="globalheader" class="apple">…</div>
    ▶<div id="top">…</div>
    ▶<div id="billboard">…</div>
    ▶<div class="wrapper">…</div>
    ▶<div id="promos">…</div>
    ▶<div id="homefooter">…</div>
     <!--/homefooter-->
     <!-- Start of DoubleClick Spotlight Tag: Please do not remove-->
     <!-- Creation Date:11/19/07 -->
    ▶<script type="text/javascript">…</script>
    ▶<iframe src="http://fls.doubleclick.net/activityi;src=1566767;
     type=apple663;cat=apple249;ord=5250550399068.743?" width="1" height="1"
     frameborder="0">…</iframe>
     <noscript></noscript>
     <!-- End of DoubleClick Spotlight Tag: Please do not remove-->
   </body>
</html>
```

```
isAttribute: false
listenerBody: function anonymous(e) { var s=s_c_il[0],…
▶ node: body#home
type: click
useCapture: false
```

The gear menu gives you the choice of seeing event listeners that have been added to any node or only those added to the currently selected node.

Debugging JavaScript Using the Web Inspector

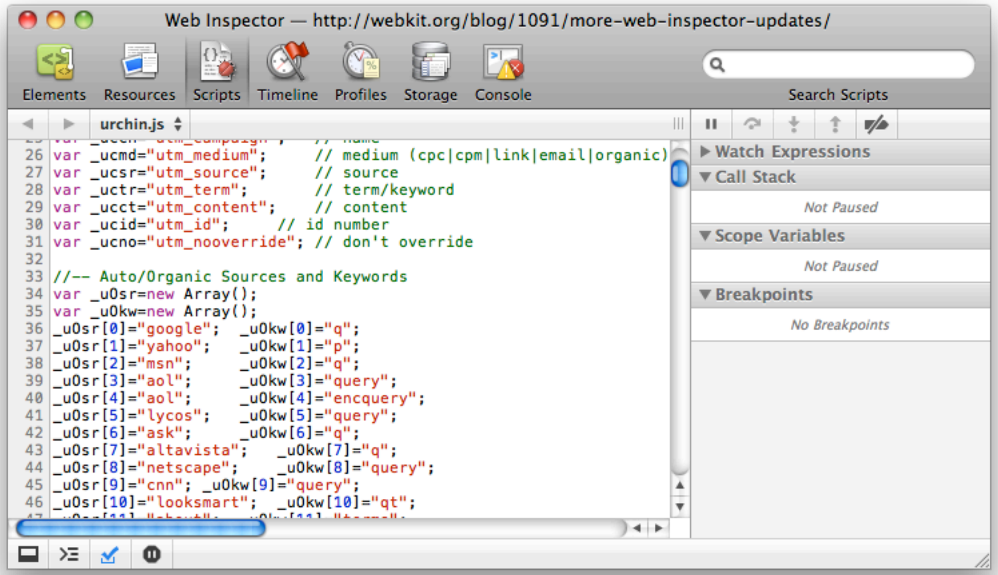
Choose Start Debugging JavaScript from the Develop menu. Open the Web Inspector and click Scripts in the toolbar to view the Scripts pane

If you do not have JavaScript debugging enabled, you see a prompt to enable debugging when you click the Scripts button. You can toggle debugging on and off by clicking the checkmark button in the bottom bar.

The pause button in the bottom bar causes the debugger to pause on exceptions. The icon turns blue when pause-on-exceptions is active.

Once debugging is active, a pop-up menu of JavaScript sources is displayed, above a listing of the currently selected source, as shown in .

Figure 3-8 The Scripts pane



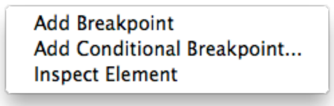
```
var _ucmd="utm_medium";      // medium (cpc|cpm|link|email|organic)
var _ucsr="utm_source";      // source
var _uctr="utm_term";        // term/keyword
var _ucct="utm_content";     // content
var _ucid="utm_id";      // id number
var _ucno="utm_nooverride"; // don't override

//-- Auto/Organic Sources and Keywords
var _uOsr=new Array();
var _uOkw=new Array();
_uOsr[0]="google";  _uOkw[0]="q";
_uOsr[1]="yahoo";   _uOkw[1]="p";
_uOsr[2]="msn";     _uOkw[2]="q";
_uOsr[3]="aol";     _uOkw[3]="query";
_uOsr[4]="aol";     _uOkw[4]="encquery";
_uOsr[5]="lycos";   _uOkw[5]="query";
_uOsr[6]="ask";     _uOkw[6]="q";
_uOsr[7]="altavista";   _uOkw[7]="q";
_uOsr[8]="netscape";    _uOkw[8]="query";
_uOsr[9]="cnn"; _uOkw[9]="query";
_uOsr[10]="looksmart";  _uOkw[10]="qt";
```

When the source of the script is a JavaScript file, the filename is listed. If the source is in-line JavaScript in an HTML file, the URL of the HTML is listed. If the JavaScript is the result of a string passed through `eval( )` or another anonymous source, the resource is listed as "(program)".

Choosing a source from the pop-up menu displays the listing for that source in the left pane. If the script is not paused, the right pane shows headings for the call stack, breakpoints, watch expressions, and scope variables, but no content is displayed.

You can set a breakpoint in any script by clicking in the gutter by the line number. The script will pause at the breakpoint. The script name, line number, and text of the breakpoint appear in the Breakpoints section on the right of the Web Inspector—clicking a breakpoint on the right jumps the text on the left to the line with the breakpoint. A checkbox allows you to enable and disable the breakpoint without removing it.

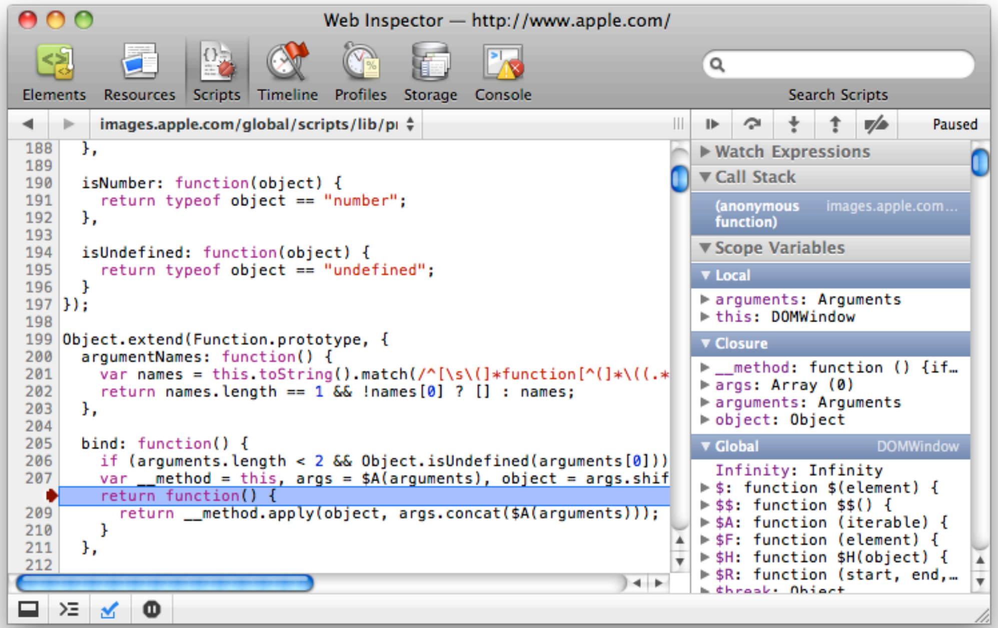
Right-click or control-click in the gutter to bring up a contextual menu:



If choose Add Conditional Breakpoint, you are prompted to enter an expression. When execution reaches a conditional breakpoint, the script pauses only if the expression evaluates as true, non-zero, or not null.

Clicking the pause icon in the toolbar above the right pane also pauses the script. When the script is paused, the line of JavaScript last executed is highlighted and the call stack, breakpoints, watch expressions, and scope variables are displayed, as illustrated in .

Figure 3-9 A paused script



```
  },

  isNumber: function(object) {
    return typeof object == "number";
  },

  isUndefined: function(object) {
    return typeof object == "undefined";
  }
});

Object.extend(Function.prototype, {
  argumentNames: function() {
    var names = this.toString().match(/^[\s\(]*function[^(]*\((.*
    return names.length == 1 && !names[0] ? [] : names;
  },

  bind: function() {
    if (arguments.length < 2 && Object.isUndefined(arguments[0]))
    var __method = this, args = $A(arguments), object = args.shif
    return function() {
      return __method.apply(object, args.concat($A(arguments)));
    }
  },
```

```
(anonymous function)    images.apple.com...

Local
▶ arguments: Arguments
▶ this: DOMWindow
Closure
▶ __method: function () {if…
▶ args: Array (0)
▶ arguments: Arguments
▶ object: Object
Global                  DOMWindow
  Infinity: Infinity
▶ $: function $(element) {
▶ $$: function $$() {
▶ $A: function (iterable) {
▶ $F: function (element) {
▶ $H: function $H(object) {
▶ $R: function (start, end,…
▶ $break: Object
```

Note: The pause/play button on the upper right pauses or resumes the script. The pause button in the bottom bar cycles the debugger through three states:

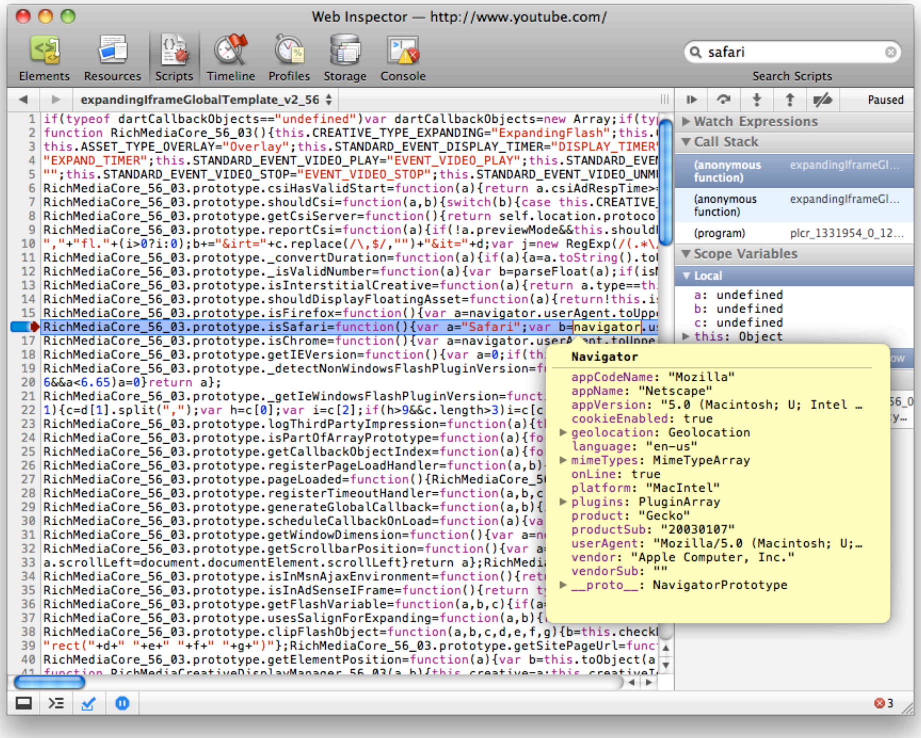
- Black—Do not pause on exceptions.
- Blue—Pause on exceptions.
- Purple—Pause on uncaught exceptions.

The Web Inspector has a unique feature regarding in-scope variables: It shows closures, "with" statements, and event-related scope objects separately. This gives you a clearer picture of where your variables are coming from and why things might be breaking (or even working correctly by accident).

The pause icon changes to a continue icon when the script is paused. The toolbar above the call stack display has additional controls, allowing you to step past the next function, step into the next function, or step out of the current function. These controls allow you to step through any script, function by function, skipping functions as needed, and examine the call stack and variables at each point.

While the script is paused, if you hover over an expression in a script, a popover appears showing the evaluation of the expression, typically the properties of the object, as illustrated in Figure 3-10

Figure 3-10 Popover



```
if(typeof dartCallbackObjects=="undefined")var dartCallbackObjects=new Array;if(ty
function RichMediaCore_56_03(){this.CREATIVE_TYPE_EXPANDING="ExpandingFlash";this.
this.ASSET_TYPE_OVERLAY="Overlay";this.STANDARD_EVENT_DISPLAY_TIMER="DISPLAY_TIMER
"EXPAND_TIMER";this.STANDARD_EVENT_VIDEO_PLAY="EVENT_VIDEO_PLAY";this.STANDARD_EVE
"";this.STANDARD_EVENT_VIDEO_STOP="EVENT_VIDEO_STOP";this.STANDARD_EVENT_VIDEO_UNM
RichMediaCore_56_03.prototype.csiHasValidStart=function(a){return a.csiAdRespTime>
RichMediaCore_56_03.prototype.shouldCsi=function(a,b){switch(b){case this.CREATIVE_
RichMediaCore_56_03.prototype.getCsiServer=function(){return self.location.protoco
RichMediaCore_56_03.prototype.reportCsi=function(a){if(!a.previewMode&&this.should
","+"fl."+(i>0?i:0);b+="&irt="+c.replace(/\,$/,"")+"&it="+d;var j=new RegExp(/(.*\
RichMediaCore_56_03.prototype._convertDuration=function(a){if(a){a=a.toString().to
RichMediaCore_56_03.prototype._isValidNumber=function(a){var b=parseFloat(a);if(is
RichMediaCore_56_03.prototype.isInterstitialCreative=function(a){return a.type==th
RichMediaCore_56_03.prototype.shouldDisplayFloatingAsset=function(a){return!this.i
RichMediaCore_56_03.prototype.isFirefox=function(){var a=navigator.userAgent.toUpp
RichMediaCore_56_03.prototype.isSafari=function(){var a="Safari";var b=navigator.u
RichMediaCore_56_03.prototype.isChrome=function(){var a=navigator.userAgent.toUppe
RichMediaCore_56_03.prototype.getIEVersion=function(){var a=0;if(th
RichMediaCore_56_03.prototype._detectNonWindowsFlashPluginVersion=f
6&&a<6.65)a=0}return a};
RichMediaCore_56_03.prototype._getIeWindowsFlashPluginVersion=funct
1){c=d[1].split(",");var h=c[0];var i=c[2];if(h>9&&c.length>3)i=c[c
RichMediaCore_56_03.prototype.logThirdPartyImpression=function(a){t
RichMediaCore_56_03.prototype.isPartOfArrayPrototype=function(a){fo
RichMediaCore_56_03.prototype.getCallbackObjectIndex=function(a){fo
RichMediaCore_56_03.prototype.registerPageLoadHandler=function(a,b)
RichMediaCore_56_03.prototype.pageLoaded=function(){RichMediaCore_5
RichMediaCore_56_03.prototype.registerTimeoutHandler=function(a,b,c
RichMediaCore_56_03.prototype.generateGlobalCallback=function(a,b){
RichMediaCore_56_03.prototype.scheduleCallbackOnLoad=function(a){va
RichMediaCore_56_03.prototype.getWindowDimension=function(){var a=n
RichMediaCore_56_03.prototype.getScrollbarPosition=function(){var a
a.scrollLeft=document.documentElement.scrollLeft}return a};RichMedi
RichMediaCore_56_03.prototype.isInMsnAjaxEnvironment=function(){ret
RichMediaCore_56_03.prototype.isInAdSenseIFrame=function(){return t
RichMediaCore_56_03.prototype.getFlashVariable=function(a,b,c){if(a
RichMediaCore_56_03.prototype.usesSalignForExpanding=function(a,b){
RichMediaCore_56_03.prototype.clipFlashObject=function(a,b,c,d,e,f,g){b=this.check
"rect("+d+" "+e+" "+f+" "+g+")"};RichMediaCore_56_03.prototype.getSitePageUrl=func
RichMediaCore_56_03.prototype.getElementPosition=function(a){var b=this.toObject(a
```

```
(anonymous function)    expandingIframeGl…
(anonymous function)    expandingIframeGl…
(program)               plcr_1331954_0_12…
```

```
a: undefined
b: undefined
c: undefined
▶ this: Object
```

```
Navigator
  appCodeName: "Mozilla"
  appName: "Netscape"
  appVersion: "5.0 (Macintosh; U; Intel …
  cookieEnabled: true
▶ geolocation: Geolocation
  language: "en-us"
▶ mimeTypes: MimeTypeArray
  onLine: true
  platform: "MacIntel"
▶ plugins: PluginArray
  product: "Gecko"
  productSub: "20030107"
  userAgent: "Mozilla/5.0 (Macintosh; U;…
  vendor: "Apple Computer, Inc."
  vendorSub: ""
▶ __proto__: NavigatorPrototype
```

You can also use the console to assist in debugging JavaScript.

Using the Console to Debug JavaScript

Click the Console icon to open the Console panel. You have the choice of viewing all console messages, just errors, just warnings, or just log entries.

Note: You can have multiple independent consoles open—one for each window or tab being inspected, as well as consoles for each component of a Safari extension being inspected, such as extension bars, injected scripts, and the global page.

You can use the console to debug JavaScript in two distinct ways:

- You can enter JavaScript interactively in the console and see the results immediately.
- You can include various `console` functions in your JavaScript to log data to the Error Console while the script is running. These functions use the same syntax as the popular Firebug debugger.

Entering JavaScript Interactively

You can enter JavaScript in the terminal interactively to help debug your script. For example, you can call functions defined in the script and see the results; you can evaluate expressions that include variables or functions declared in your script; and you can query the values of variables directly.

This is particularly helpful when used in combination with breakpoints in your code, allowing you to pause and inspect the script interactively at any point.

The console has auto-completion support when entering JavaScript. As you type, JavaScript variables, properties, and function names are suggested in gray. Pressing the Right Arrow key accepts the current suggestion. If multiple selections begin with the same prefix, you can cycle through the suggestions using the Tab key. If there is only one suggestion, the Tab key accepts it.

Typing a variable name and pressing Enter displays the variable's current value.

Any changes you make to the DOM using JavaScript from the console are immediately displayed in the Elements pane, as well as in the browser window.

The Command Line API

In addition to the usual JavaScript methods, and the functions and variables defined in your script, you can enter some Firebug command line API's interactively at the console. The following commands are supported interactively:

- `$0-$4`

  Variables that contain the current and previous three selected nodes in the Web Inspector.

- `$(id)`

  Returns the element with the specified ID. Similar to `getElementById()`.

- `$$(selector)`

  Returns the array of elements that match the given CSS selector. Similar to `querySelectorAll`.

- `$x(xpath)`

  Returns the array of elements that match the given XPath expression.

- `clear()`

  Clears the console.

- `debug(functionName)`

  Adds a breakpoint to the first line of a function.

- `dir(object)`

  Prints an interactive listing of all properties of the object. Similar to the popover from hovering over an object when a script is paused.

- `dirxml(node)`

  Prints the XML source tree of an HTML or XML element. This looks identical to the view that you would see in the Elements panel of the Web Inspector. You can click on any node to inspect it in the Web Inspector.

- `inspect()`,

  Takes an element, database, or storage area as an argument and automatically jumps to the appropriate panel to display the relevant information.
- `keys(object)`

  Returns an array containing the names of all properties of the object. (Properties are key/value pairs; the name of a property is the key.)
- `monitor(functionName)`

  Turns on logging for all calls to a function.
- `monitorEvents(object[, types])`

  Turns on logging for all events dispatched to an object. The optional argument types may specify a specific family of events to log. The most commonly used values for types are "mouse" and "key".

  The full list of available types includes "composition", "contextmenu", "drag", "focus", "form", "key", "load", "mouse", "mutation", "paint", "scroll", "text", "ui", and "xul".
- `profile([title])`

  Turns on the JavaScript profiler. The optional title is a label for the profile.
- `profileEnd()`

  Stops a running profile.
- `unmonitor(functionName)`

  Stops logging calls to a function.
- `unmonitorEvents(object[, types])`

  Stops logging events, optionally events of a particular type, that are dispatched to an object.
- `values(object)`

  Returns an array containing the values of all properties of the object. The values are returned in the same order as the keys in `keys(object)`.

To make working with these APIs easier, they are included in the Console's auto-completion capability.

Safari JavaScript Console API

Safari supports several JavaScript `console` functions for debugging. As an alternative to setting breakpoints, you can log branches in your code path, print variable values, and so on, using `console` functions. Safari supports many of the same `console` functions used in the Firebug API.

Note: You type the Firebug *command line* APIs interactively in the console. You insert the Firebug `console` functions into your scripts.

Many `console` functions take a **message-object** as a parameter. This message-object is logged to the error console. When Safari logs a message-object, it appends a hyperlink to the line in the source code where the logging `console` function appears. A message-object can contain a string, one or more variables, or a combination. You can use `printf`-style string substitution using numeric or string variable values. If variables are included, but not used for string substitution, the variable values are logged, space delimited.

Examples of valid message-objects:

```
"It got this far..."
```

```
"Item and count:", item, count
```

```
"Item: %s Count: %d", item, count
```

```
"Item: %s Count:", item, count
```

```
count
```

The following `console` functions are supported in Safari:

- `console.assert(expression, message-object)`

  If `expression` evaluates false, logs the message.

- `console.count([title])`

  Logs the number of times this line of code has executed, and an optional title.

- `console.debug([message-object])`

  Logs the message object.

- `console.dir(object)`

  Logs the current properties of the object.

- `console.dirxml(node)`

  Logs the DOM tree of an HTML or XML element.

- `console.error(message-object)`

  Logs an “error” icon followed by a color-coded message object.

- `console.group(message-object)`

  Logs the message object and begins an indented block for further log entries.

- `console.groupEnd()`

  Ends an indented block of log entries.

- `console.info(message-object)`

  Logs the message object.

- `console.log(message-object)`

  Logs the message-object.

- `console.log(message-object)`

  Logs the message-object.

- `console.profile([title])`

  Begins profiling JavaScript—tracking the number of times each function is called, the time spent in that function, and the time spent in nested groups of functions. If a title is provided, the profile is named. See “Optimizing JavaScript” (page 46).

- `console.profileEnd([title])`

Ends one or more JavaScript profiles. If a title is provided and a running profile has a matching title, only the current run of that profile is ended. Otherwise, the current run of all profiles is ended.

- `console.time(name)`

  Starts a timer and gives it a name.

- `console.markTimeline("string")`

  Adds a label to the timeline view marking when the point when the method was called.

- `console.trace()`

  Logs a JavaScript stack trace at the moment the function is called. The stack trace lists the functions on the call stack (functions that have been called and have not yet finished executing and returned) and the values of any arguments passed to those functions.

- `console.warn(message-object)`

  Logs a "warning" icon followed by a color-coded message-object.

Note: The functions `console.log`, `console.info`, `console.warn`, `console.debug`, and `console.error` all log a message. The only difference between the functions is the color-coding of the log entry and the inclusion of marker icons for warnings and errors.

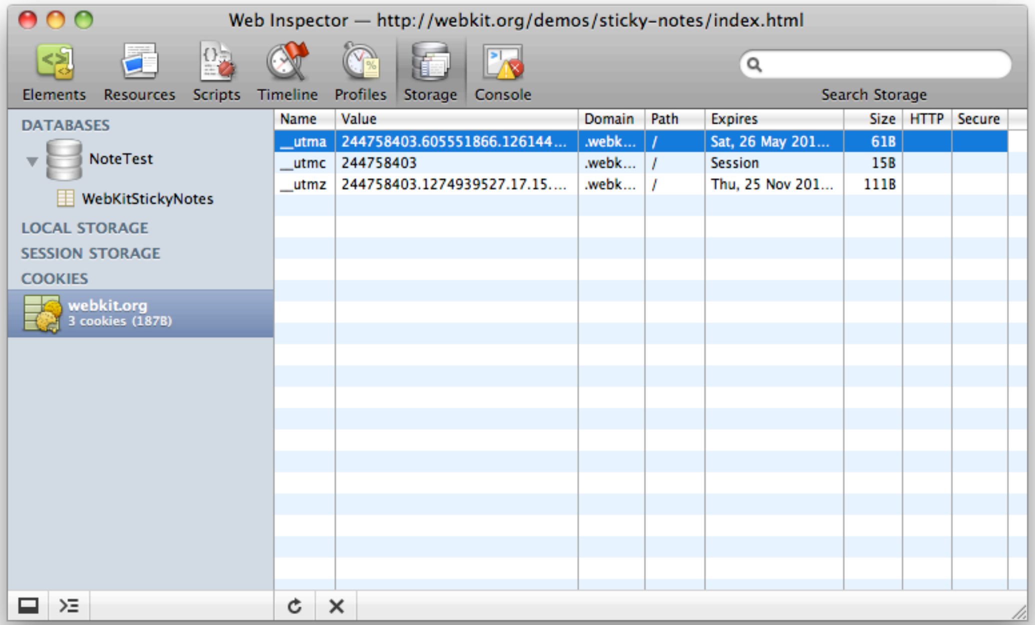
Analyzing Client-Side Storage, Databases, and Cookies

You can use the Web Inspector's Storage pane to inspect HTML5 client-side databased, local storage, session storage, and cookies.

Local storage and session storage are displayed as an editable data grid of key/value pairs.

Cookies show the name, value, domain, path, expiration date, and size of each cookie, as shown in Figure 3-11.

Figure 3-11 Inspecting cookies

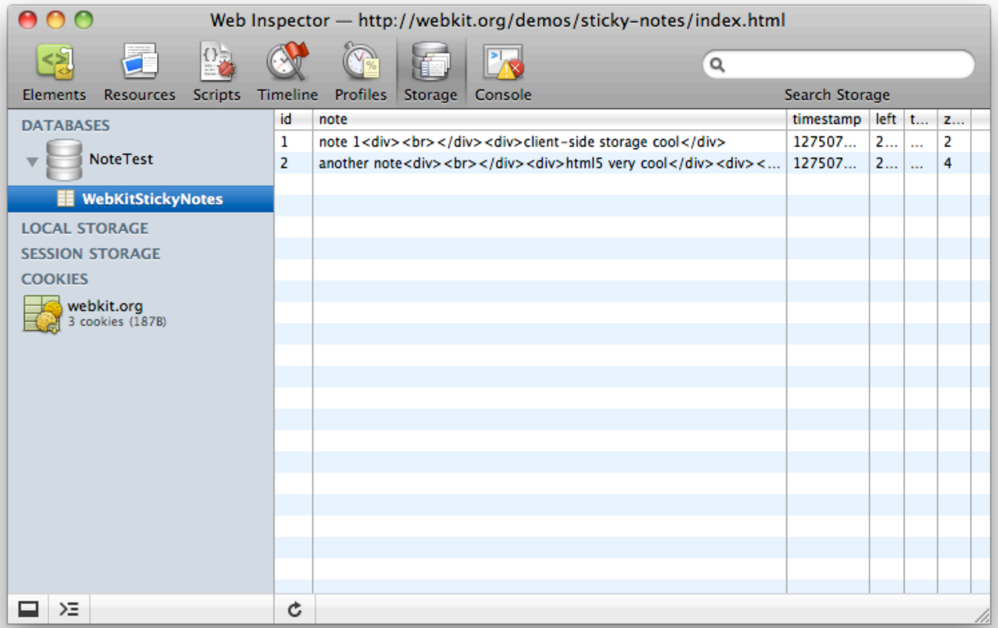


Name	Value	Domain	Path	Expires	Size	HTTP	Secure
__utma	244758403.605551866.126144...	.webk...	/	Sat, 26 May 201...	61B		
__utmc	244758403	.webk...	/	Session	15B		
__utmz	244758403.1274939527.17.15....	.webk...	/	Thu, 25 Nov 201...	111B		

When inspecting cookies, you can see `httpOnly` cookies and cookies that are sent only over HTTPS in the rightmost columns. Clicking the X in the bottom bar deletes the cookie.

Open databases are shown in the sidebar. Clicking a database's disclosure triangle shows the database's tables. Selecting a database table displays a data grid containing all the columns and rows for that table, as shown in Figure 3-12.

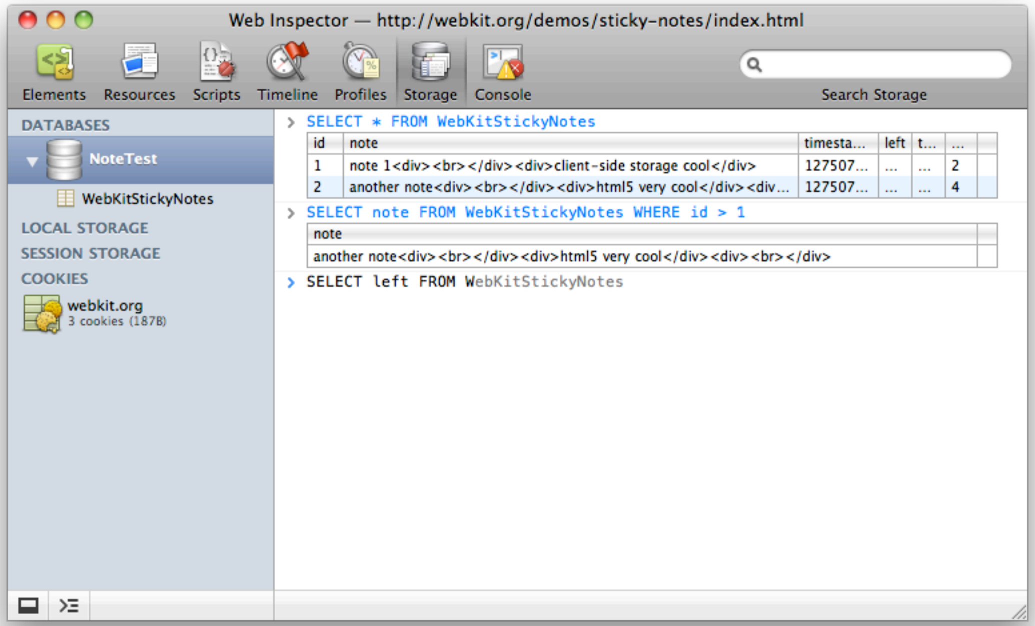
Figure 3-12 Inspecting databases



id	note	timestamp	left	t...	z...
1	note 1<div> </div><div>client-side storage cool</div>	127507...	2...	...	2
2	another note<div> </div><div>html5 very cool</div><div><...	127507...	2...	...	4

In addition to inspecting HTML 5 databases, you can interact with them by issuing SQL queries against any of the displayed databases. Select a database in the sidebar to see an interactive console for evaluating SQL queries. The input to this console has auto-completion and tab-completion for table names in the database, as well as for common SQL words and phrases, as illustrated in "An SQL query."

Figure 3-13 An SQL query



```
> SELECT * FROM WebKitStickyNotes
```

id	note	timesta…	left	t…	…
1	note 1<div> </div><div>client-side storage cool</div>	127507…	…	…	2
2	another note<div> </div><div>html5 very cool</div><div…	127507…	…	…	4

```
> SELECT note FROM WebKitStickyNotes WHERE id > 1
```

note
another note<div> </div><div>html5 very cool</div><div> </div>

```
> SELECT left FROM WebKitStickyNotes
```

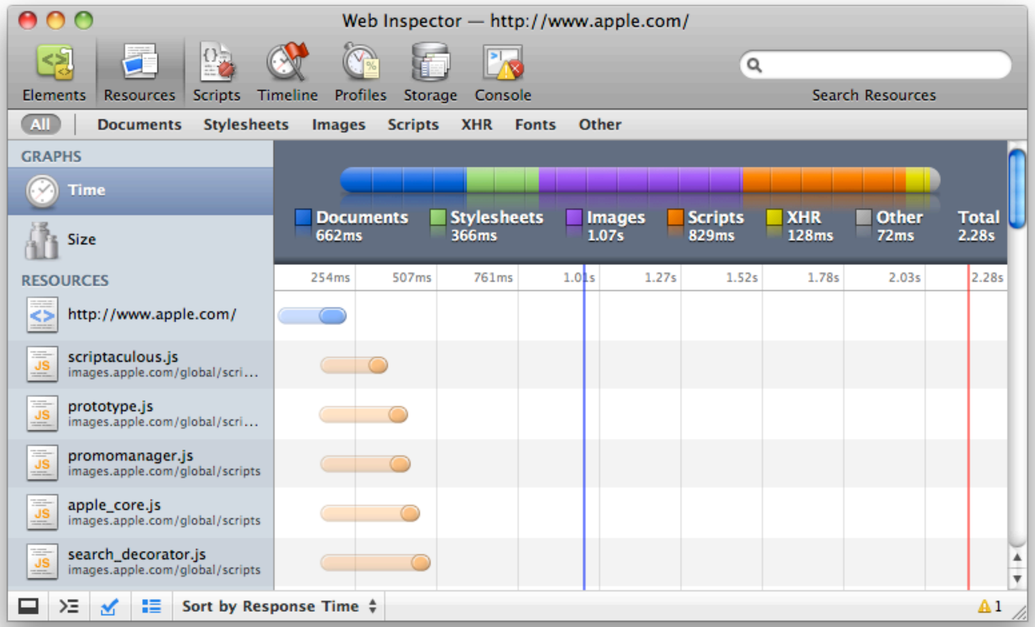
Optimizing Your Website

The final step in website development is optimization—making your website load and your scripts run as quickly and responsively as possible. Safari has two tools designed to help you: a visual download analyzer and a JavaScript profiler.

Optimizing Download Time

Open the Web Inspector. Click Resources in the toolbar to see a list of resources that make up the website: HTML, CSS, and JavaScript files, images, XMLHttpRequests, and so on, as shown in .

Figure 4-1 The Resources pane



Note: If resource tracking is not enabled, you are prompted to enable it, either for this session only or always. You can toggle resource tracking on and off by clicking the checkmark button in the bottom bar.

All the resources used in the website are shown in a list on the left. On the right, they are displayed graphically and color-coded by type—documents are blue, style sheets are green, images are purple, XHR resources are yellow, and so on.

The graph shows you when each resource was requested, the combined network and server delay, and the load time for each resource.

The blue line is when the `DOMContentReady` event is dispatched. The red line is when the body `onload` event is dispatched.

You can display the resources graphically by size or time. If displayed by size, they are sorted from largest to smallest. You can choose to sort by the transfer size (Transfer) or uncompressed size (Size), as some resources by be compressed for transmission.

If displayed by time, you can sort in several ways:

- Start time—the time at which the resource is requested.
- Response time—the time at which the first byte of the resource is received.
- End time—the time at which the last byte of the resource is received.
- Duration—the length of time the resource takes to load.
- Latency—the delay between requesting the resource and getting the first byte.

No matter which way you sort, each resource appears as a lozenge whose left edge shows the request time. The lozenge is translucent during the latency, then opaque for the duration (from the response time to the end time).

If your website is slow to appear, use this pane to see why. You can see immediately if the delay is caused by a server-side script that is not responding promptly, a large resource that takes a long time to load, or a script that takes a long time to execute prior to requesting another resource.

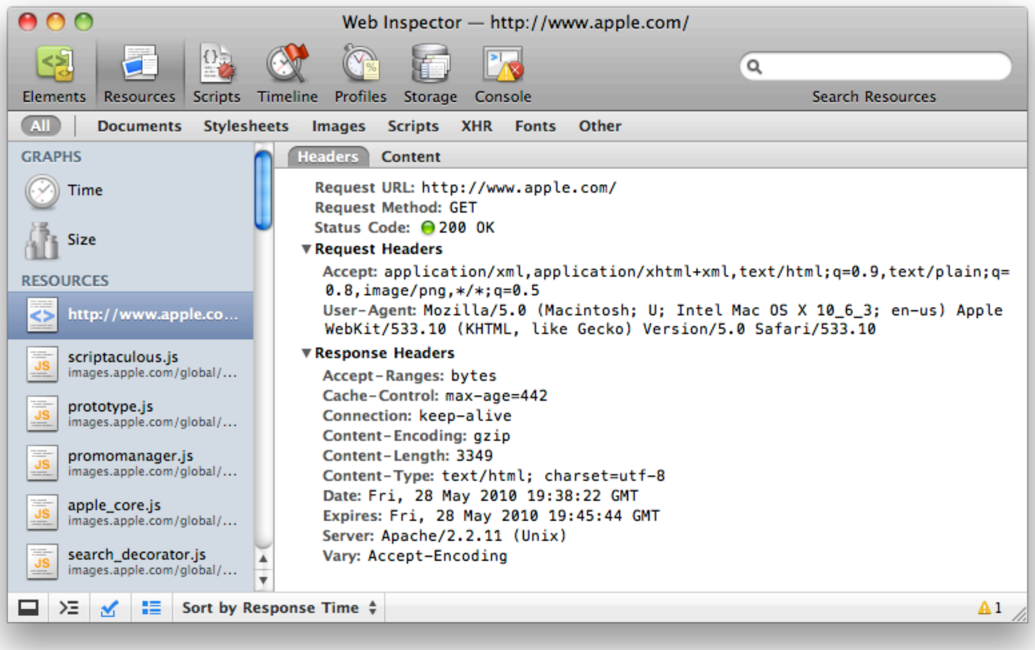
You can also use this pane to scan for resources that are not loading at all, such as missing style sheets or requested resources that are misspelled.

You can click any resource in the list to inspect that resource more closely. Exactly what you see depends on the type of resource.

Clicking a document resource, such as a CSS, HTML, or JavaScript document, displays the source code for the document with line numbers. Resources that are not documents, such as XMLHttpRequests, display the data that was delivered to the browser with line numbers.

You can choose to view the contents of the resource or the header information. If you choose Headers, the number of request and response headers associated with the resource is listed. You can choose to display the request and response headers by clicking the disclosure triangle, as shown in "Inspecting resources."

Figure 4-2 Inspecting resources



```
Request URL: http://www.apple.com/
Request Method: GET
Status Code: 200 OK
▼Request Headers
 Accept: application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
 0.8,image/png,*/*;q=0.5
 User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_3; en-us) Apple
 WebKit/533.10 (KHTML, like Gecko) Version/5.0 Safari/533.10
▼Response Headers
 Accept-Ranges: bytes
 Cache-Control: max-age=442
 Connection: keep-alive
 Content-Encoding: gzip
 Content-Length: 3349
 Content-Type: text/html; charset=utf-8
 Date: Fri, 28 May 2010 19:38:22 GMT
 Expires: Fri, 28 May 2010 19:45:44 GMT
 Server: Apache/2.2.11 (Unix)
 Vary: Accept-Encoding
```

Clicking an image resource displays the image, its dimensions, file size, and MIME type. The request and response headers are also enumerated and inspectable.

Note: Values sent in XHR requests are also shown here; you may find this useful when debugging AJAX sites.

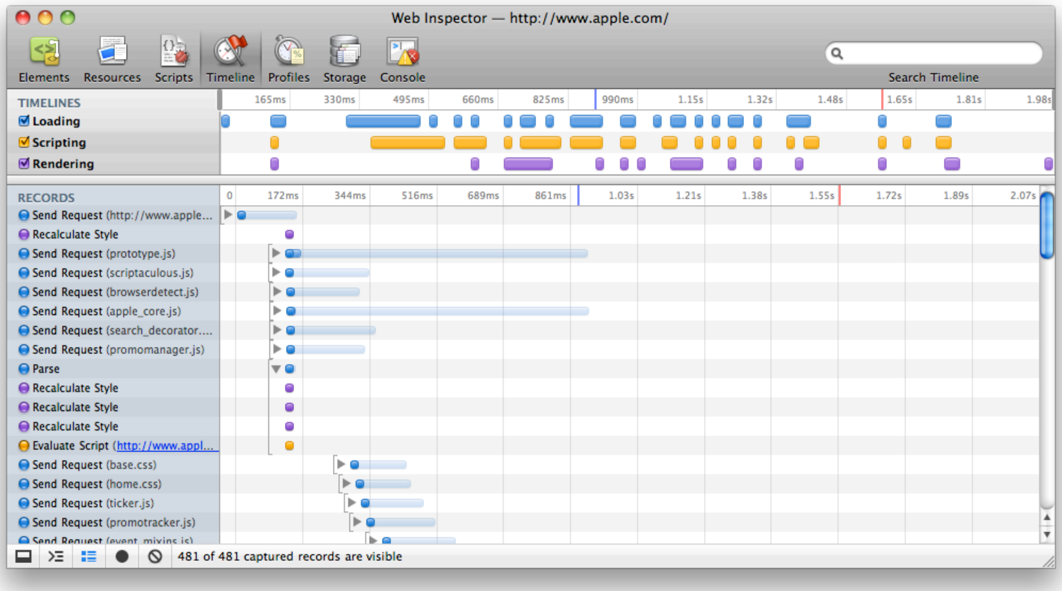
Optimizing Loading, Scripting, and Rendering Times

Not all of the time taken to display a webpage is used to load resources. Considerable time can be spent rendering the webpage and running scripts. There can be complex interactions where a resource needs to load before a script executes, modifying the DOM before it can be rendered.

To get a detailed look at exactly where time is being spent on your website, including dependencies, select Timeline in the Web Inspector.

Capturing a timeline is similar to capturing a JavaScript profile. Click the black record button in the botom bar to start recording events, then click it again to stop recording. The recorded events are shown in a timeline, with time spent loading, scripting, and rendering broken out separately. For an example, see Figure 4-3.

Figure 4-3 Timeline


Web Inspector — http://www.apple.com/
Elements
Resources
Scripts
Timeline
Profiles
Storage
Console
Search Timeline
TIMELINES
Loading
Scripting
Rendering
165ms 330ms 495ms 660ms 825ms 990ms 1.15s 1.32s 1.48s 1.65s 1.81s 1.98s
RECORDS
0 172ms 344ms 516ms 689ms 861ms 1.03s 1.21s 1.38s 1.55s 1.72s 1.89s 2.07s

```
Send Request (http://www.apple...
Recalculate Style
Send Request (prototype.js)
Send Request (scriptaculous.js)
Send Request (browserdetect.js)
Send Request (apple_core.js)
Send Request (search_decorator....
Send Request (promomanager.js)
Parse
Recalculate Style
Recalculate Style
Recalculate Style
Evaluate Script (http://www.appl...
Send Request (base.css)
Send Request (home.css)
Send Request (ticker.js)
Send Request (promotracker.js)
Send Request (event_mixins.js)
481 of 481 captured records are visible
```

The window has a TIMELINES section and a RECORDS section. Click and drag in the TIMELINES section to zoom in on the records for that period of time.

Clicking a record shows a popup with its name and details. Each record includes a disclose triangle if it contains sub-records. For example, a request event might wait for a reply, and a parse event might have to recalculate styles—waiting for the reply and recalculating styles would be dependent sub-records. The solid portion of a record is the time spent at the top level—the translucent portion shows time spent in the cascade of dependent events.

The blue tool in the bottom bar toggles visibility of records that are under 15 msec.

Optimizing JavaScript

Clicking the Profiles button in the Web Inspector opens the Profiles pane. This pane allows you to see where execution time is being spent in your JavaScript. Use the Profiles pane to find bottlenecks in your scripts and optimize their performance.

Note: If profiling is not enabled, you are prompted to enable it, either for this session only or always. You can toggle profile enabling on and off by clicking the checkmark button in the bottom bar.

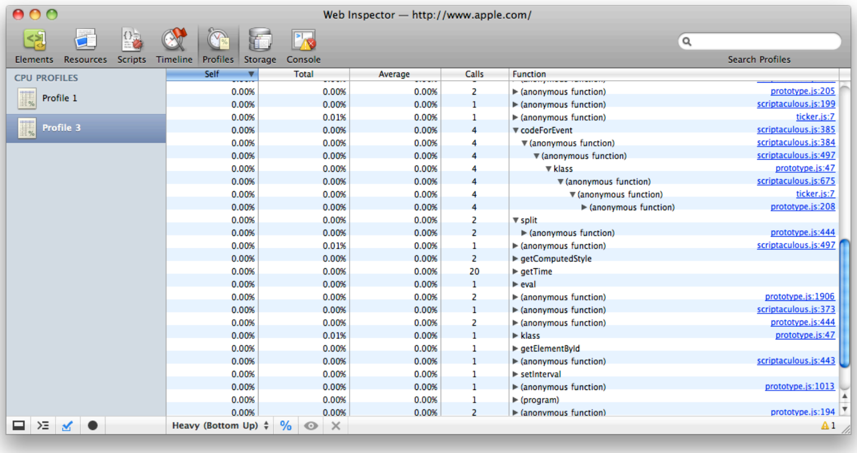
To use the Profiles pane, you must start profiling, either manually or by including a `console.profile()` call in your script. To start profiling manually, click the record button (black circle) in the bottom bar. The record button turns red. To stop the profile, click the record button again. No profile is displayed until you stop profiling, either manually or through a call to `console.profileEnd()`. Each time you begin and end profiling, another profile is captured.

The Web Inspector's JavaScript profiler is fully compatible with the Firebug functions `console.profile()` and `console.profileEnd()`, but in Safari you can optionally specify a title in `console.profileEnd()` to stop a specific profile if multiple profiles are being recorded.

Once you have captured one or more profiles, they are listed on the left side of the Web Inspector. Manual profiles are named sequentially (Profile 1, Profile 2, and so on). Profiles created by calls to `console.profile()` are named with the title of the profile provided in the script. If multiple profiles are captured under the same name, a disclosure triangle reveals multiple runs within the profile.

The time spent in each function executed during the profile is displayed, as well as the number of times each function is called, as shown in "The profiles pane." The time can be displayed either as a percentage of total time or in milliseconds, toggled by the % button in the bottom bar.

Figure 4-4 The Profiles pane



Self	Total	Average	Calls	Function	
0.00%	0.00%	0.00%	2	▶ (anonymous function)	prototype.js:205
0.00%	0.00%	0.00%	1	▶ (anonymous function)	scriptaculous.js:199
0.00%	0.01%	0.00%	1	▶ (anonymous function)	ticker.js:7
0.00%	0.00%	0.00%	4	▼ codeForEvent	scriptaculous.js:385
0.00%	0.00%	0.00%	4	▼ (anonymous function)	scriptaculous.js:384
0.00%	0.00%	0.00%	4	▼ (anonymous function)	scriptaculous.js:497
0.00%	0.00%	0.00%	4	▼ klass	prototype.js:47
0.00%	0.00%	0.00%	4	▼ (anonymous function)	scriptaculous.js:675
0.00%	0.00%	0.00%	4	▼ (anonymous function)	ticker.js:7
0.00%	0.00%	0.00%	4	▶ (anonymous function)	prototype.js:208
0.00%	0.00%	0.00%	2	▼ split	
0.00%	0.00%	0.00%	2	▶ (anonymous function)	prototype.js:444
0.00%	0.01%	0.00%	1	▶ (anonymous function)	scriptaculous.js:497
0.00%	0.00%	0.00%	2	▶ getComputedStyle	
0.00%	0.00%	0.00%	20	▶ getTime	
0.00%	0.00%	0.00%	1	▶ eval	
0.00%	0.00%	0.00%	2	▶ (anonymous function)	prototype.js:1906
0.00%	0.00%	0.00%	1	▶ (anonymous function)	scriptaculous.js:373
0.00%	0.00%	0.00%	2	▶ (anonymous function)	prototype.js:444
0.00%	0.01%	0.00%	1	▶ klass	prototype.js:47
0.00%	0.00%	0.00%	1	▶ getElementById	
0.00%	0.00%	0.00%	1	▶ (anonymous function)	scriptaculous.js:443
0.00%	0.00%	0.00%	1	▶ setInterval	
0.00%	0.00%	0.00%	1	▶ (anonymous function)	prototype.js:1013
0.00%	0.00%	0.00%	1	▶ (program)	
0.00%	0.00%	0.00%	2	▶ (anonymous function)	prototype.js:194

Clicking a profile name displays the functions that were executed during the profile, the time spent in each function, and the number of times each function was called. The time spent is broken down into three categories: Self, the total time spent in the function itself; Total, the total time spent in the function and any subordinate functions it calls in turn; and Average, the average time spent in the function itself during each call (the Self time divided by the number of calls).

You can toggle between absolute time or percentage of total time in the profile by clicking the percent button in the bottom bar.

If a function is declared with a name, the function name is displayed. If a function is created programmatically by an `eval()` statement or inline `<script> </script>` tagset, it is labeled (program) in the profile. Other unnamed functions, for example a function defined within a variable declaration, are labeled (anonymous function).

Note: To assist yourself in debugging, assign a `displayName` to this kind of otherwise anonymous function. The `displayName` is used as the function name in profiles.

Where applicable, the source URL and line number of the function declaration is shown in grey to the right of the function name. The source URL is a link. Clicking it opens the source in the Resources pane, scrolled to the line number where the function is declared.

There are two ways to view a profile: bottom up (heavy) or top down (tree). Each view has its own advantages. The heavy view shows you which functions have the most performance impact and the calling paths to those functions. The tree view gives you an overall picture of the script's calling structure, starting at the top of the call stack.

Below the profile is a pair of data-mining controls to facilitate the dissection of profile information. The focus button (eye icon) filters the profile to show only the selected function and its callers. The exclude button (X icon) removes the selected function from the profile and charges its callers with the excluded function's execution time. While any of these data-mining features are active, a reload button is available that restores the profile to its original state.

The Profiles pane supports plain text searches of function names and resource URLs. Numeric searches are also supported that match rows in the profile's Self, Total, and Calls columns. To facilitate powerful numeric searching, there are a few operators and units that work to extend or limit your results. For example you can search for "> 2.5ms" to find all the functions that took longer than 2.5 milliseconds to execute. In addition to **ms**, the other supported units are **s**, for time in seconds, and **%**, for percentage of time. The other supported operators are < , <=, >= and =. When no units are specified, the Calls column is searched.

Keyboard and Mouse Shortcuts

There are a number of mouse and keyboard shortcuts in the Web Inspector. These can be significant time savers if you use the Web Inspector frequently.

General Shortcuts

	Mac	Windows
Show Web Inspector	Option-Command-I	Ctrl-Alt-I
Show Console	Option-Command-C	Ctrl-Alt-C
Start Profiling JavaScript	Option-Shift-Command-P	Ctrl-Alt-P

Web Inspector Shortcuts

	Mac	Windows
Next Panel	Command-]	Ctrl-]
Previous Panel	Command-[	Ctrl-[
Toggle Console	esc	esc
Focus Search Box	Command-F	Ctrl-F
Find Next	Command-G	Ctrl-G
Find Previous	Shift-Command-G	Ctrl-Shift-G

Console Shortcuts

	Mac	Windows
Next Suggestion (auto-completion)	Tab	Tab
Previous Suggestion (auto-completion)	Shift-Tab	Shift-Tab

Accept Suggestion (auto-completion)	Right Arrow	Right Arrow
Next Line / Command	Down Arrow	Down Arrow
Previous Line / Command	Up Arrow	Up Arrow
Next Command	Ctrl-N	
Previous Command	Ctrl-P	
Clear History	Command-K or Ctrl-L	Ctrl-L
Execute	Return	Enter

Elements Panel Shortcuts

	Mac	Windows
Navigate	Up and Down Arrows	Up and Down Arrows
Expand / Collapse	Left and Right Arrows	Left and Right Arrows
Edit Node	Returns	Enter
Expand	Double-click tag	Double-click tag
Edit Attribute	Double-click attribute or tab to next attribute	Double-click attribute or tab to next attribute
Add Attribute	Tab past last attribute	Tab past last attribute

Styles Pane Shortcuts

	Mac	Windows
Edit Rule	Double-click	Double-click
Next / Prev Property	Tab / Shift-Tab	Tab / Shift-Tab
Insert New Property	Double-click whitespace	Double-click whitespace
Increment / Decrement Value	Up and Down Arrows	Up and Down Arrows
Increment / Decrement by 10	Shift-Up and Shift-Down Arrows. PgUp and PgDn	Shift-Up and Shift-Down Arrows, PgUp and PgDn
Increment / Decrement by 100	Shift-PgUp and Shift-PgDn	Shift-PgUp and Shift-PgDn

Increment / Decrement by 0.1	Option-Up / Option-Down	Alt-Up and Alt-Down Arrows

Debugger Shortcuts

	Mac	Windows
Next Call Frame	Ctrl-.	Ctrl-.
Prev Call Frame	Ctrl-,	Ctrl-,
Continue	F8	F8
Step Over	F10	F10
Step Into	F11	F11
Step Out	Shift-F11	Shift-F11
Evaluate Selection	Shift-Command-E	Ctrl-Shift-E
Toggle Breakpoint Condition	Click line number	Click line number
Edit Breakpoint Condition	Right-click line number	Right-click line number

Document Revision History

This table describes the changes to *Safari User Guide for Web Developers*.

Date	Notes
2010-06-21	Updated for Safari 5.0
2010-01-20	Added description of JavaScript 'console' API.
2009-11-17	Revised to be task-oriented, with sections on prototyping, debugging, and optimizing websites.
2009-06-01	Revised and expanded for Safari 4.0.
2009-01-06	Added description of Safari Mobile debug console.
	Corrected typos.
2008-10-15	Corrected minor typos and adjusted for style and consistency.
2008-09-09	Describes hidden developer tools introduced in Safari 3.1