

Safari Developer Tools Guide

Contents

About Safari Developer Tools 5

At a Glance 5

Enable the Developer Tools in Safari Preferences 6

Speed Up Prototyping by Interactively Testing for Errors 6

Debug Your HTML and CSS Interactively Using Web Inspector 7

Use Web Inspector to Debug JavaScript Interactively 8

Get Help with Cookies, Local Storage, the Application Cache, and HTML5 Client-Side Databases 8

Use the Instrument Navigator 9

Be a Power User 10

Enable, Build, and Debug Extensions 10

See Also 11

Developer Tools for Safari Overview 12

Developing Websites with the Developer Tools 12

Safari on the Desktop, Safari on iOS, and WebKit 12

Enabling Developer Tools in Safari on the Desktop 13

The Develop Menu Command Summary 14

Enabling Developer Tools in WebKit-Based Applications Other Than Safari 15

Enabling and Using Web Inspector in Safari on iOS Devices 15

Prototyping Your Website 16

The Prototyping Process, Improved 16

Using the Snippet Editor 17

Using the Error Console 17

Opening the Error Console 18

Viewing Issues 18

Using the Console to Prototype JavaScript 18

Changing the User Agent String 19

Switching to Another App 20

Debugging Your Website 21

Use Cases 21

Debugging HTML and CSS Using Web Inspector 22

Inspecting and Editing DOM Attributes 24

Inspecting and Editing Styles	25
Inspecting and Editing Metrics	27
Inspecting and Editing HTML Properties	28
Inspecting Listener Functions	29
Debugging JavaScript Using the Web Inspector	29
Using the Console to Debug JavaScript	31
Entering JavaScript Interactively	31
The Command-Line API	32
Safari JavaScript Console API	33
Analyzing Client-Side Storage, Databases, and Cookies	35
Inspecting the Offline Application Cache	38
Optimizing Your Website	40
Optimizing Loading, Scripting, and Rendering Times	40
Optimizing JavaScript	43
Keyboard and Mouse Shortcuts	45
General Shortcuts	45
Web Inspector Shortcuts	45
Console Shortcuts	46
DOM Tree Shortcuts	46
Style Details Shortcuts	47
Debugger Shortcuts	47
Document Revision History	48

Figures

About Safari Developer Tools 5

Figure I-1 Web Inspector 5

Developer Tools for Safari Overview 12

Figure 1-1 Enabling developer tools in Safari 13

Figure 1-2 The Develop menu 13

Prototyping Your Website 16

Figure 2-1 The Snippet Editor 17

Figure 2-2 The Error Console in the Log navigator 18

Figure 2-3 The type and version of various browsers in the User Agent submenu 19

Debugging Your Website 21

Figure 3-1 Editing DOM attributes 24

Figure 3-2 Nodal context menu 24

Figure 3-3 Viewing styles in the Details sidebar of Web Inspector 25

Figure 3-4 Editing style properties 25

Figure 3-5 Editing metric attributes 27

Figure 3-6 Editing HTML properties 28

Figure 3-7 Event Listeners 29

Figure 3-8 The Breakpoint navigator 30

Figure 3-9 Inspecting cookies 36

Figure 3-10 Inspecting databases 37

Figure 3-11 An SQL query 38

Figure 3-12 Application cache 39

Optimizing Your Website 40

Figure 4-1 Inspecting timelines 41

Figure 4-2 Network timeline view 42

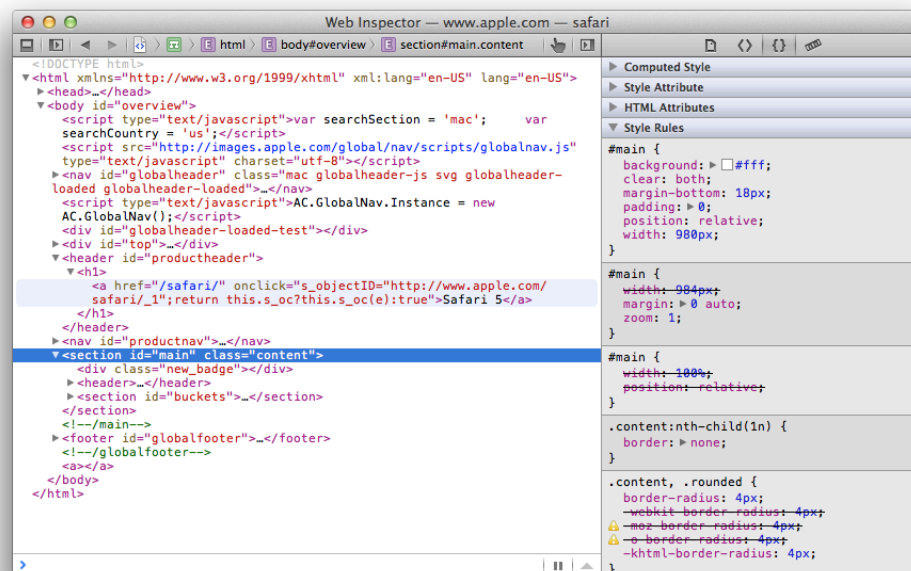
Figure 4-3 Network content and headers view 42

Figure 4-4 Profiling JavaScript 43

About Safari Developer Tools

Safari 4.0 and later includes built-in tools such as Web Inspector, shown in [Figure I-1](#) (page 5), to help you prototype, analyze, debug, and optimize websites and web apps. Safari 5.0 and later has additional tools you can use to develop and debug Safari extensions.

Figure I-1 Web Inspector



Note: This document was formerly titled *Safari User Guide for Web Developers*. This document describes the developer tools in Safari 6.0 and later. The developer tools in earlier versions of Safari are substantially different.

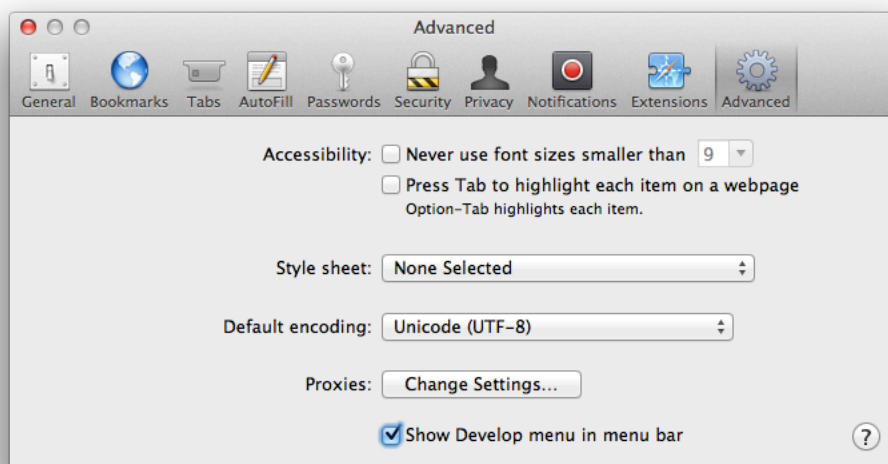
At a Glance

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

These tools are built into Safari on OS X, iOS, and Windows. You can enable them in other WebKit-based apps.

Enable the Developer Tools in Safari Preferences

Enable the developer tools in the Advanced pane of Safari preferences:



- On OS X or Windows, use Safari preferences to show the Develop menu in Safari. The full set of developer tools are then available.
- On iOS devices, use Safari Preferences to enable Web Inspector in Safari. You can then use Web Inspector on OS X to debug web content on your iOS device through a USB connection. For details, see [“Enabling and Using Web Inspector in Safari on iOS Devices”](#) (page 15).
- You can add the developer tools to other Webkit-based Mac apps as a contextual menu by modifying the application’s preferences file. For details, see [“Enabling Developer Tools in WebKit-Based Applications Other Than Safari”](#) (page 15).

Speed Up Prototyping by Interactively Testing for Errors

Show Web Inspector’s console while prototyping your website to quickly spot HTML and JavaScript structural and syntactic errors. The 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 17), [“Viewing Issues”](#) (page 18), and [“Using the Console to Prototype JavaScript”](#) (page 18).

Safari also comes with a Snippet Editor, where you can type in HTML, CSS, 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 it into your actual webpage. For details, see [“Using the Snippet Editor”](#) (page 17).

If your website has different code paths for different browsers, have Safari give different user-agent strings to test the code branches. Then, verify the behavior changes for Safari, Internet Explorer, Opera, Chrome, or Safari on iOS 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 19) and [“Switching to Another App”](#) (page 20).

Relevant Chapter: [“Prototyping Your Website”](#) (page 16)

Debug Your HTML and CSS Interactively Using Web Inspector

The Error Console finds and identifies syntactic and structural errors, but sometimes your website doesn’t look or behave as you want, even though the syntax and structure are legal. That’s when you need Web Inspector. 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.

With Web Inspector, you can:

- **Find Things Fast**

Click an element in the DOM and it is highlighted in the browser window. Use the DOM node locator button, which looks like a hand pointing a finger, and click on something in the browser window to highlight the element in the DOM.

- **Change Things on the Fly**

Click an element in the DOM and see its attributes, styles, metrics, and properties, as well as 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 arrow keys in steps of 1, or with a step of 0.1 or 10 by holding down the Option or Shift modifier keys, respectively. 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.

Relevant Section: [“Debugging HTML and CSS Using Web Inspector”](#) (page 22).

Use Web Inspector to Debug JavaScript Interactively

Web Inspector includes tools to help you set breakpoints, pause, inspect variable values, see the call stack, log messages and data to the console, set variable values and continue, and enter JavaScript on the fly to test it—all with autocompletion of function, property, and variable names to speed you along.

- **See It All**

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 source code.

- **Pause Execution When You Want**

Click in the gutter anywhere in the source code view to set a breakpoint, without having to edit the source. 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, 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. Type commands in the console with auto-completion, and see the results immediately.

Relevant Sections: [“Debugging JavaScript Using the Web Inspector”](#) (page 29), [“Using the Console to Debug JavaScript”](#) (page 31).

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

Use Web Inspector’s Storage navigator to inspect cookies, local key-value storage, the offline application cache, 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 Web Inspector, with auto-completion of SQL functions and database field names. Inspect the contents of the HTML5 application cache to better understand offline application behavior.

Relevant Sections: [“Analyzing Client-Side Storage, Databases, and Cookies”](#) (page 35), [“Inspecting the Offline Application Cache”](#) (page 38)

Use the Instrument Navigator

Web Inspector’s Instrument navigator shows you exactly where time is spent, so you know where work is needed 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 resulting problems—*before* you modify your working source.

The Instrument navigator shows all the resources your site uses—HTML files, JavaScript files, CSS style sheets, server-side output, images, fonts, 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. At a glance, see if your website is hung 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 Instrument navigator lets you profile your website in real time, showing not only the latency and load time for each resource, but when events occur, as well as the time spent executing scripts, parsing, and rendering. Complex interactions—a script that changes the DOM, causing a resource to load, and a table to be rerendered, for example—are laid out clearly. Header details and memory usage over time are also shown.

Relevant Section: [“Optimizing Loading, Scripting, and Rendering Times”](#) (page 40)

If the timeline shows that significant time is spent in your scripts, use the Profiles section of the Instrument navigator to determine where. 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 43)

Be a Power User

The Safari developer tools include dozens of keyboard and mouse shortcuts to speed up common operations, from opening Web Inspector to cycling through auto-completion suggestions. See the shortcuts table in [“Keyboard and Mouse Shortcuts”](#) (page 45) to boost your productivity.

Enable, Build, and Debug Extensions

Use Extension Builder to create extensions. You need to join the Safari Developer Program to install the extensions you create—you can’t install an extension unless it has 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.

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

The full set of developer tools for inspecting, modifying, and optimizing HTML, CSS, and JavaScript works on extensions. Once you’ve built and installed an extension, you can use Web Inspector to debug and optimize:

- Injected scripts and style sheets as if they were downloaded from the webpage’s host.

Note: You must load a webpage that the extension has access to in order to inject a script or style sheet, and you must load a webpage that causes the script or style sheet to be injected before you can inspect the injected content using Safari’s developer tools.

- Extension bars by right-clicking or Control-clicking it and using the Inspect Element contextual menu to open Web Inspector.
- Popovers can be debugged by right-clicking or Control-clicking it and using the Inspect Element contextual menu to open Web Inspector.
- A global HTML page by clicking Inspect Global Page in Extension Builder (the button is present only if the selected extension is installed and has a global page).

Important: The console does not currently recognize JavaScript elements defined in injected scripts. You can log data to the console, but you cannot interact with elements defined in an injected script from the console by typing their names.

See Also

- *Safari Extensions Development Guide* delivers step-by-step directions for creating Safari extensions using Extension Builder
- *Safari Extensions Reference* lists the JavaScript classes, methods, and properties you can access from Safari extensions
- *Safari HTML Reference* gives the supported HTML tags for Safari
- *Safari CSS Reference* gives the supported CSS tags for Safari
- *Safari Web Content Guide* offers guidance for developing web content for the iPhone
- *WebKit DOM Programming Topics* includes articles on using and modifying the Document Object Model

Developer Tools for Safari Overview

There are developer tools built into Safari on OS X, iOS, Windows, and in other WebKit-based apps. These tools can help you 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 is as follows:

- **Prototype**—Determine what combination of HTML, JavaScript, CSS, and local storage deliver the functionality you need, testing snippets of code interactively.
- **Write**—Author the website, using your favorite text editor.
- **Test and debug**—Test using several browsers and platforms (Mac, Windows, iOS), track 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, use the Snippet Editor to streamline prototyping, Web Inspector's Error Console for testing and debugging, and Web Inspector's timeline view and JavaScript profiler to help you optimize your website.

Safari on the Desktop, Safari on iOS, and WebKit

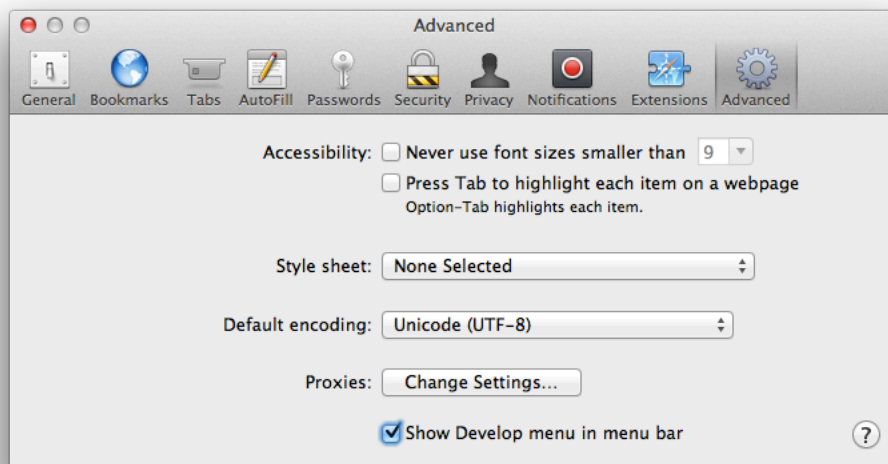
The toolset for Safari on the desktop (OS X or Windows) includes a Develop menu with several commands and a number of interactive tools. Safari on iOS (iPhone, iPad, and iPod touch) can enable a setting to allow Web Inspector to debug web content over a USB connection.

The toolset for WebKit-based apps is essentially the same as for Safari on the desktop, but it is available via the contextual menu, and you enable the tools differently.

Enabling Developer Tools in Safari on the Desktop

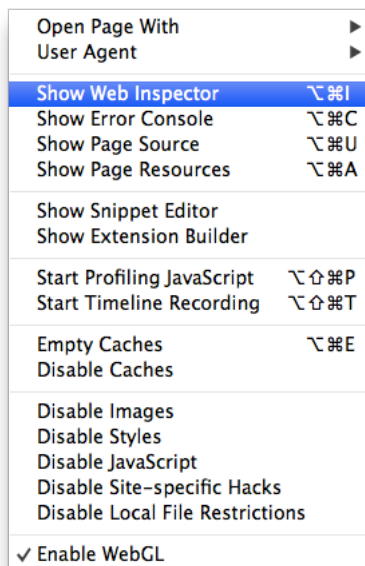
You enable the developer tools in Safari on the desktop (OS X or Windows) by turning on the Develop menu. In Safari preferences, click Advanced, then select "Show Develop menu in menu bar," as shown in Figure 1-1.

Figure 1-1 Enabling developer tools in Safari



Selecting this option adds a Develop menu to your menu bar (Figure 1-2 (page 13)). The Safari developer tools are now enabled.

Figure 1-2 The Develop menu



The Develop Menu Command Summary

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

- 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, to simulate visiting your site using a different browser or device type.
- Show Web Inspector—Open the Web Inspector window to inspect or modify the DOM, HTML attributes, and CSS properties.
- Show Error Console—Open the console in Web Inspector to see any HTML or JavaScript errors and any corrective actions taken by Safari.
- Show Page Source—Open the Source Code in Web Inspector to see the HTML of the current page.
- Show Page Resources—Open the Resource navigator in Web Inspector to view all images, scripts, and style sheets attached to the current page.
- 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 Profiling JavaScript—Turn on the JavaScript profiler to see how many times each function is called, how long it takes, and so on.
- Start Timeline Recording—Record detailed information about the status of incoming HTTP requests, JavaScript events, and layout rendering.
- Empty Caches—Delete all caches stored by the browser.
- 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 Site-specific Hacks—Use this to disable the modifications to Safari and test your site for correct operation (if Apple engineers have modified Safari specifically to work around a problem with your website).
- Disable Local File Restrictions—Disable security checks that may prohibit local development.
- Enable WebGL—Turn on the ability to view OpenGL content in Safari.

Enabling Developer Tools in WebKit-Based Applications Other Than Safari

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 a web view in the application.

Enabling and Using Web Inspector in Safari on iOS Devices

You can enable an advanced setting in Safari on iOS to debug web content on your device directly from your desktop. To do this, turn on Web Inspector in the Advanced pane under Safari in the Settings app. Once it is enabled, connect your device to your desktop machine with a USB cable. Alternately, you can use Simulator to take advantage of Web Inspector's debugging capabilities, which comes free with Xcode from the Mac App Store. If you are a registered iOS developer, you can even inspect the web content of a `UIWebView`. For more information, read the chapter "Debugging" in *Safari Web Content Guide*.

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 browsers deal with the error differently, it can be frustrating to find the problem.

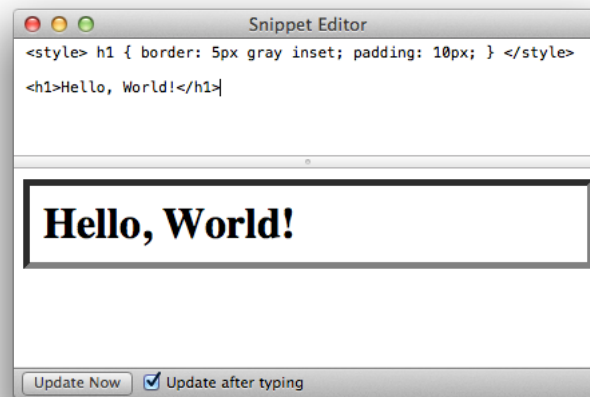
Here's how Safari can help:

- Use the Snippet Editor to interactively test elements of your page in a sandbox, debugging your syntax and testing different attributes 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 console to spot hidden errors that Safari is dealing with for you—other browsers may deal with them differently. Correct the errors to maximize your chances of compatibility.
- If you use the user agent string to execute various code branches that are intended to execute on different browsers or devices, choose User Agent String from the Develop menu to modify the user agent string and invoke the code branches in your website, 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 apps 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 Figure 2-1.

Figure 2-1 The Snippet Editor



The Snippet Editor provides an interactive interface for quickly prototyping or debugging your HTML, CSS, and JavaScript. You don't have to create complete HTML pages and open them in a browser, or 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. You can freely edit text in the upper pane of the Snippet Editor—cut, copy, paste, insert text, and so on. 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.

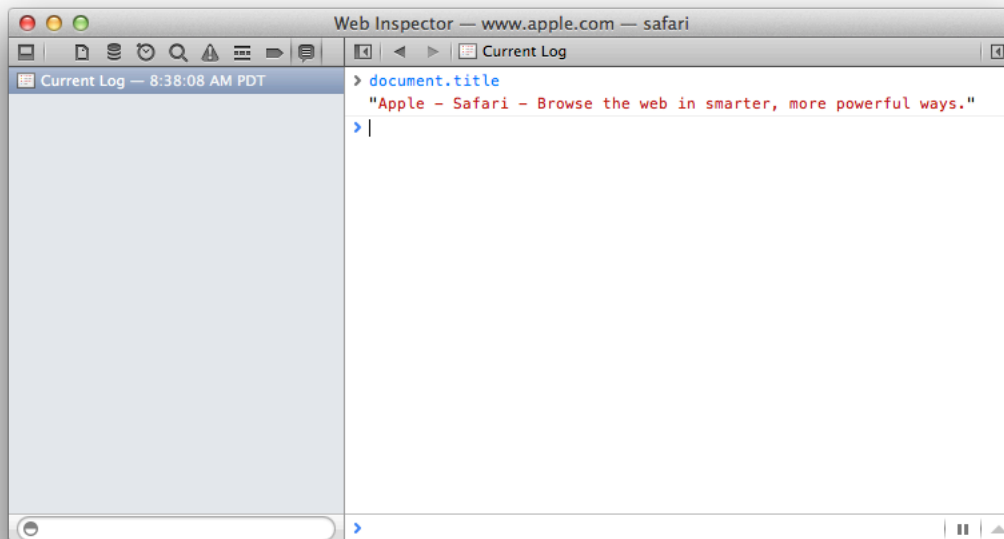
Using the Error Console

The Error Console of Web Inspector is the most basic tool for debugging a website, so it's an appropriate tool for prototyping. The Error Console notifies you of any syntactic 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 handled the error (such as by ignoring an extra closing tag).

Opening the Error Console

There are several ways to open the Error Console. You can choose Show Error Console from the Develop menu, or you can click Current Log in the Log navigator of Web Inspector, as shown in Figure 2-2.

Figure 2-2 The Error Console in the Log navigator



You also have access to the Quick Console—a console available at the bottom of the content browser of Web Inspector—by clicking the Console area (the blue greater-than sign), or by pressing the Esc key.

Viewing Issues

Any errors encountered when loading a website appear under the Issue navigator. Clicking an error opens the troublesome line in the source code.

To view warnings along with errors, click Current Log under the Log navigator. 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.

Using the Console to Prototype JavaScript

Like the Snippet Editor, the console of Web Inspector 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 console (see [“Safari JavaScript Console API”](#) (page 33)).

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, a menu appears, and you can cycle through the suggestions using the Up and Down Arrow keys. If there is only one suggestion, the Tab key accepts it.

To display a variable's current value, type the variable name and press Enter.

Any changes you make to the DOM using JavaScript from the console are immediately displayed in Web Inspector's content browser, 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 use JavaScript to read the user agent string to determine which version of a 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. 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 The type and version of various browsers in 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, first question 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 Web Inspector's console.

Note: If you have a mobile version of your website, optimized for iPhone and iPod touch, you should not display the mobile version to visitors using Safari on iPad. You can test your website by setting the user agent string to Safari on iPad to verify that iPad users see the normal website, not the mobile version.

Switching to Another App

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 apps that the operating system knows can open the page.

Debugging Your Website

Safari's powerful Web Inspector tool can find and correct problems with your website. Syntactic and structural problems can be revealed by using the interactive console. Data transmitting over the network can be analyzed and dissected. Script execution can be paused and examined in the Debug navigator. Even the most complex problems can usually be resolved using Web Inspector.

Note: If you are a registered OS X or iOS developer, you can test the behavior and appearance of your website on iOS devices using the Simulator application. Simulator is part of the Xcode installation. To open Simulator, first open Xcode, then Control-click on the Xcode icon in the Dock, and select iOS Simulator from the Open Developer Tool submenu. You can switch between iPad and iPhone simulation using Simulator's Hardware menu.

Use Cases

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

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

See the section ["Prototyping Your Website"](#) (page 16). It includes basic testing.

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

Use the Error Console to see if Safari detects any syntactic or structural errors in your HTML, CSS, or JavaScript, and if so, what corrective action it is taking. This will also reveal the 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, 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 17) 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.

- **Your website doesn't work on iOS devices, but Web Inspector shows no errors.**

Enable Web Inspector for iPhone or iPod touch (see [“Enabling and Using Web Inspector in Safari on iOS Devices”](#) (page 15)) and check for errors on the device itself. See *Safari Web Content Guide* for guidance on specific design considerations for web content on iOS devices.

- **Your website doesn't look or behave as you expect, but Web Inspector shows no errors.**

See the sections in this chapter, [“Debugging HTML and CSS Using Web Inspector”](#) (page 22) and [“Debugging JavaScript Using the Web Inspector”](#) (page 29), to learn how to use the developer tools to analyze and debug website behavior.

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

See the last section in this chapter, [“Analyzing Client-Side Storage, Databases, and Cookies”](#) (page 35).

- **Your website works, but it is sluggish or unresponsive.**

See the next chapter, [“Optimizing Your Website”](#) (page 40).

Debugging HTML and CSS Using Web Inspector

If your website doesn't look or act as you expect, analyze your site using Web Inspector.

Choose Show Web Inspector from the Develop menu. This opens the Web Inspector window.

iOS Note: You can use Web Inspector to inspect web content on iOS 6 and later by enabling Web Inspector on iOS in the Settings app. Once it's enabled, you can follow along with the rest of this document, as features of Web Inspector work with iOS as well as OS X. To learn more about enabling Web Inspector for iOS, read the [“Debugging Web Content on iOS”](#) chapter of *Safari Web Content Guide*.

The left sidebar of Web Inspector is known as the Navigation sidebar. Here you'll find the main sections of Web Inspector under each navigator icon. At the bottom of the Navigation sidebar is the Filter input field. Anything typed into this field will filter the results displayed within the currently selected navigator. For example, in the Resource navigator (Control-1), typing `*.min.js` will refine the results shown to only display files containing `.min.js` in their filepaths.

The center of Web Inspector, known as the content browser, contains the content view of the selected item in the Navigation sidebar. You can access the DOM of the current webpage as a collapsable and expandable structure of nested elements by clicking a source file in the Resource navigator. Click the disclosure triangle to expand or collapse the view of a given element and its contents. A breadcrumb path is added to the top bar, allowing you to see where you are in the DOM hierarchy. You can click a breadcrumb to move back up the hierarchy, as well as toggle between viewing the DOM and the source.

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 content browser gives you the current state of the DOM. If there are errors in the webpage, the content browser shows you the DOM that Safari has constructed, which may differ significantly from the source.

When you click an element in the DOM, the corresponding element is highlighted in the browser window. If you Control-click or right-click in the browser window, a contextual menu is displayed with an Inspect Element option. Choosing Inspect Element highlights the corresponding element in the DOM, making it easy to zoom in on a given element and find its location in the source, even in a complex web app.

Tip: Another way to get from an element in the browser display to its definition in the DOM is to click the DOM node locator button to the right of the breadcrumb bar, then move the cursor over the browser window. The DOM node locator button looks like a hand pointing its finger. Elements are highlighted as the mouse passes over them, and a summary of the element's style and metrics are shown in a tooltip under the element. Clicking an element opens the DOM tree to the element's definition and highlights it.

The Details sidebar on the right side of Web Inspector displays the properties, styles, and metrics of the currently selected element, as well as information about the page itself, such as query parameters and HTTP headers.

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 or right-click on the part of the page that looks wrong in Safari's browser window, select Inspect Element, then look at the highlighted element in the DOM panel of Web Inspector to see how the element is defined. You can interactively modify the HTML parameters in 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 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 29).

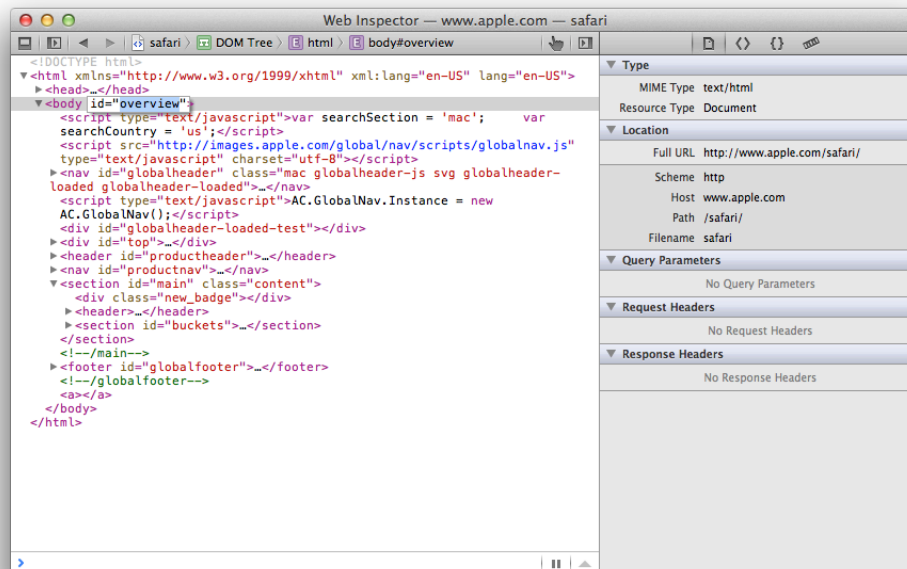
If you know the name of an element in the DOM that you want to inspect (for example, to find all instances of a particular class), use the Search navigator (Control-4) in the upper-left corner of the Inspector. The Search navigator conducts a plain-text search across all resources linked from the current page.

The following subsections show how to inspect and modify HTML and CSS using 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 Figure 3-1.

Figure 3-1 Editing DOM attributes

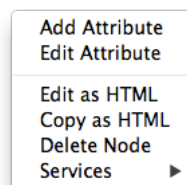


Use the Tab key and Shift-Tab key combination to traverse the attributes.

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.

Double-click an element or attribute to edit it, or Control-click or right-click the element to bring up a contextual menu, as shown in Figure 3-2 (page 24).

Figure 3-2 Nodal context menu

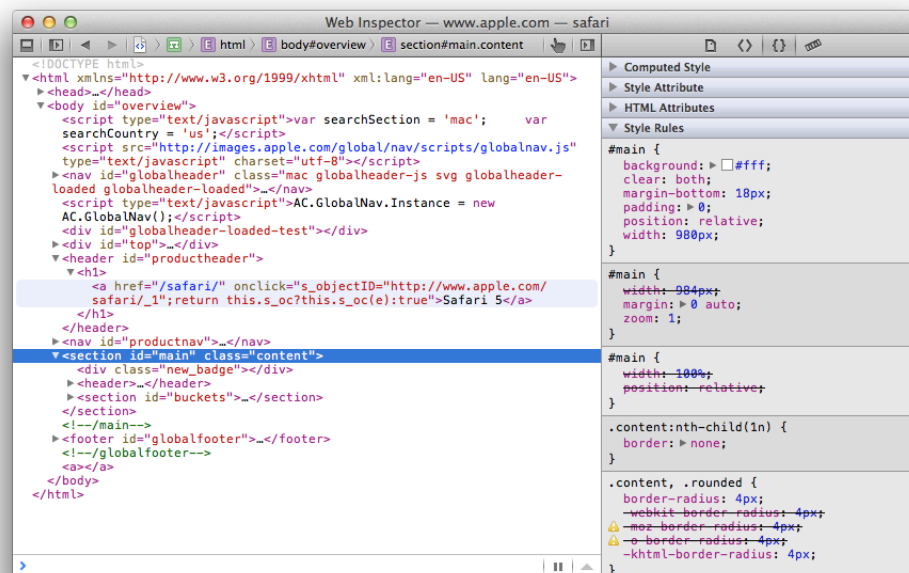


You can add a new attribute to the element, edit the DOM as if it were an HTML file in a text editor, copy the HTML for the element and its children to the clipboard, or delete the element and all of its children.

Inspecting and Editing Styles

Click Style (Control-Shift-3) in the Details sidebar to see the CSS styles that are applied to the currently selected element, as shown in Figure 3-3.

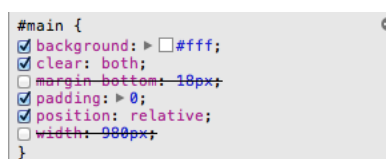
Figure 3-3 Viewing styles in the Details sidebar of Web Inspector



The first section of text in the Style details sidebar 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.

Clicking an editable style brings up a series of checkboxes. Deselecting a checkbox 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 Figure 3-4. Reselecting the checkbox enables the style property again.

Figure 3-4 Editing style properties



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:

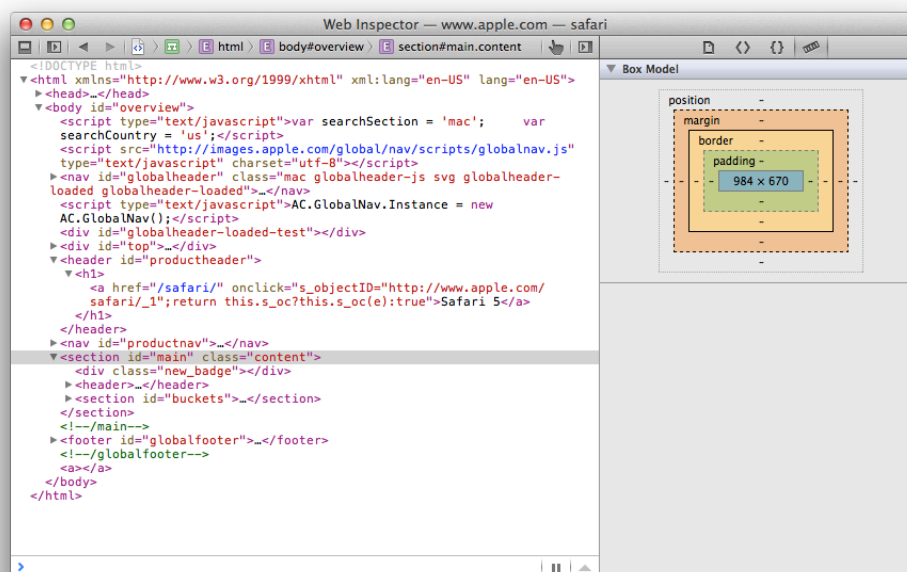
- Press Delete with the property selected to delete the property, if allowed.
- Use the keyboard to change the property values.
- Use the arrow keys to increment or decrement the value by 1 (for numerical values only). Holding down the Option or Alt key increments or decrements the value by 0.1, while holding down the Shift key increments or decrements by 10.
- 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.
- Edit a selector by double-clicking it.
- Create a new rule by choosing New Style Rule from the gear menu.
- Cycle through different color representations—`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 code.

Inspecting and Editing Metrics

Click Appearance 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 a metric value displayed on the right to edit it directly, as shown in Figure 3-5.

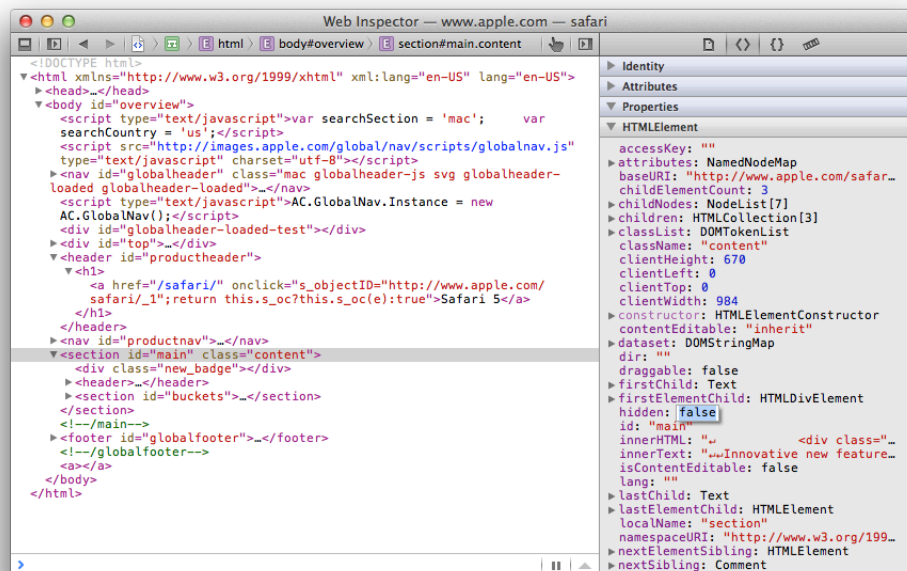
Figure 3-5 Editing metric attributes



Inspecting and Editing HTML Properties

Click the disclosure triangle for `HTMLElement` under the Node navigator in the Details sidebar to view the highlighted element's HTML properties. Double-click a property to edit its value, as illustrated in [Figure 3-6](#) (page 28).

Figure 3-6 Editing HTML properties

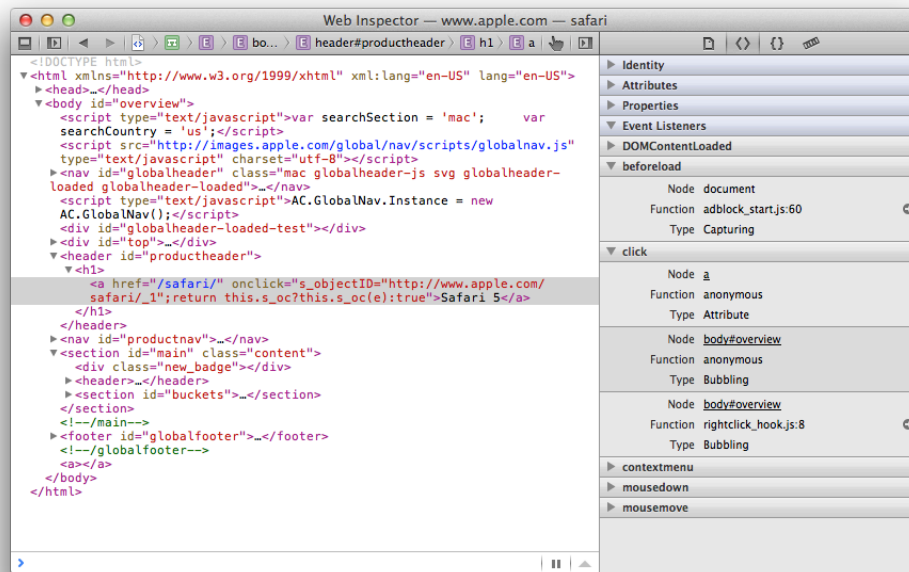


In this image, you can see the properties of the element with an `id` of `main`. The `hidden` property is set to `false`. Double-clicking the property allows you to edit its value.

Inspecting Listener Functions

If any JavaScript functions have been added as event listeners, you can inspect them by clicking the HTML node of interest in the content browser, and then clicking the Event Listeners disclosure triangle in the Details sidebar, as shown in Figure 3-7.

Figure 3-7 Event Listeners



Clicking the go-to arrow to the right of the function name will take you to where the event handler is declared in the code.

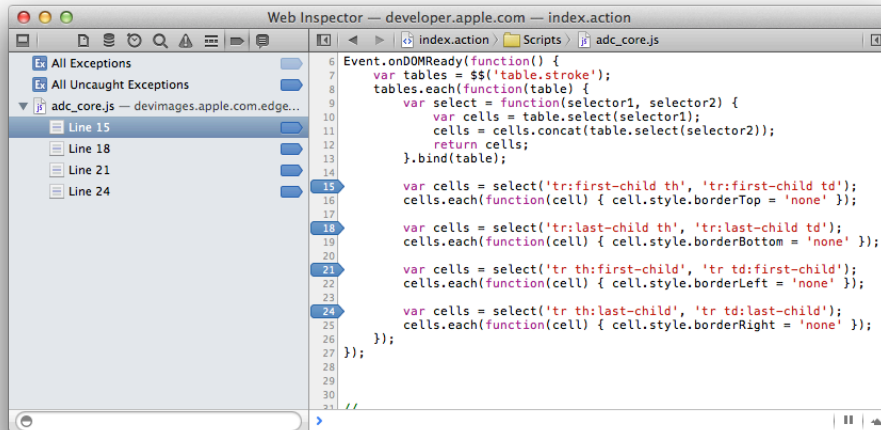
Debugging JavaScript Using the Web Inspector

To start debugging JavaScript, open Web Inspector and click Breakpoint in the Navigation sidebar.

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 left of the Web Inspector window. Clicking a breakpoint on the left jumps the text in the content browser to the line with the breakpoint. The breakpoint icon to the right of the line number allows you to enable and disable the breakpoint without removing it.

You can add breakpoints to your code by selecting a JavaScript file in the Resource navigator. Clicking in the gutter will add blue breakpoints. You also have the option to break on all exceptions or all uncaught exceptions in the Breakpoint navigator, as shown in Figure 3-8.

Figure 3-8 The Breakpoint navigator



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 Anonymous Script.

The pause button in the Quick Console at the bottom causes the debugger to pause JavaScript execution. While paused, all JavaScript calls are blocked, and you are presented with additional controls to help you understand your code:

- On the left, the call stack is displayed in the Debug navigator. Viewing the call stack helps you trace the path in which functions execute.
- On the right, properties of the scoped object, as well as local and global variables, are displayed in the Scope Chain details sidebar. These details help you visualize the current values of objects at that exact position in the code.
- At the bottom, the pause button changes into more debugging buttons, 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.

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).

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

Using the Console to Debug JavaScript

Click the Log navigator to open the Current Log. You can also press the Esc key anywhere in Web Inspector to move focus to the Quick Console in the bottom bar.

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 Current Log 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 console 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 also has auto-completion support. As you type, JavaScript variables, properties, and function names are suggested in a pop-up selection menu. Pressing Tab or the Right Arrow key accepts the current suggestion. If multiple selections begin with the same prefix, you can cycle through suggestions by pressing the Up and Down Arrow keys.

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 content browser, 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 APIs interactively at the console. The following commands are supported:

- `$0-$4`

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

- `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 content browser of Web Inspector. You can click on any node to inspect it.

- `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)`
Logs the message, if `expression` evaluates false.
- `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.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 43).
- `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 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.

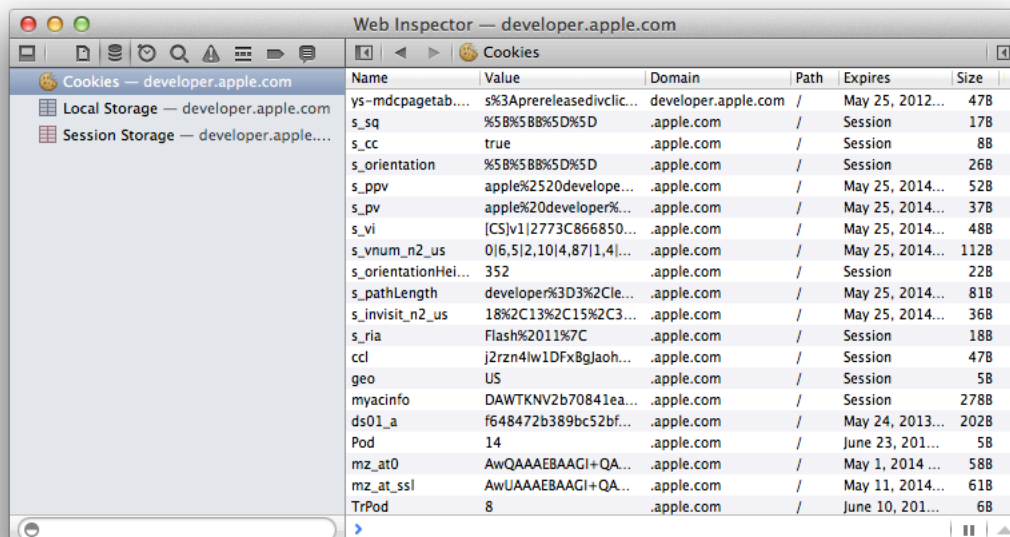
Note: The functions `console.log`, `console.info`, `console.warn`, `console.debug`, and `console.error` all log a message. The only difference between the functions is the color-coding of the log entry and the inclusion of marker icons for warnings and errors.

Analyzing Client-Side Storage, Databases, and Cookies

You can use Web Inspector’s Storage navigator to inspect HTML5 client-side databases, local storage, session storage, cookies, and the application cache.

Local storage and session storage are displayed as an editable data grid of key-value pairs. As shown in Figure 3-9, cookies are displayed in a table that lists each cookie's name, value, domain, path, expiration date, and size. Pressing the Delete key while a cookie is selected deletes the cookie.

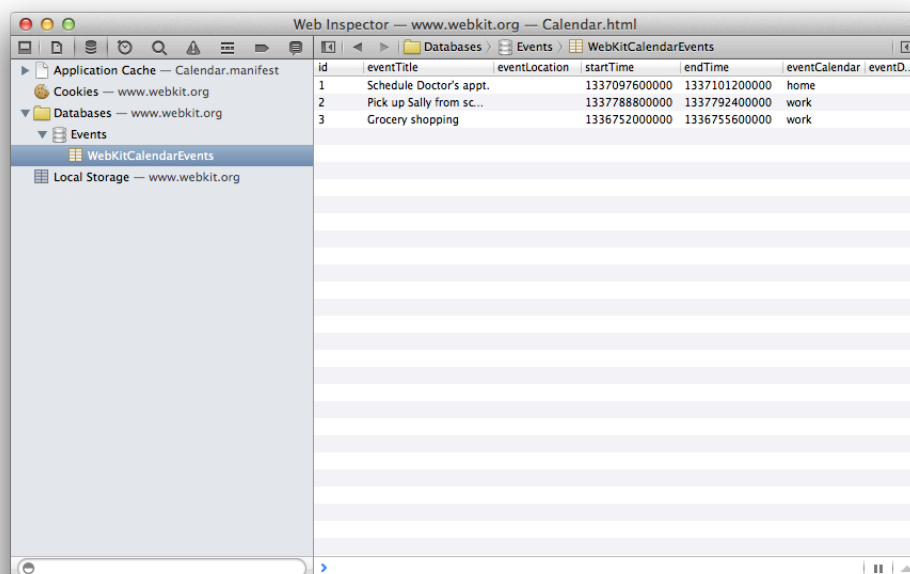
Figure 3-9 Inspecting cookies



Name	Value	Domain	Path	Expires	Size
ys-mdcpageta...	s%3Aprereleasedivcl...	developer.apple.com	/	May 25, 2012...	47B
s_sq	%5B%5B%5D%5D	.apple.com	/	Session	17B
s_cc	true	.apple.com	/	Session	8B
s_orientation	%5B%5B%5D%5D	.apple.com	/	Session	26B
s_ppv	apple%2520develope...	.apple.com	/	May 25, 2014...	52B
s_pv	apple%20develope...	.apple.com	/	May 25, 2014...	37B
s_vi	[CS]v1 2773C866850...	.apple.com	/	May 25, 2014...	48B
s_vnum_n2_us	0 6,5 2,10 4,87 1,4apple.com	/	May 25, 2014...	112B
s_orientationHei...	352	.apple.com	/	Session	22B
s_pathLength	developer%3D3%2Cle...	.apple.com	/	May 25, 2014...	81B
s_invisit_n2_us	18%2C13%2C15%2C3...	.apple.com	/	May 25, 2014...	36B
s_ria	Flash%2011%7C	.apple.com	/	Session	18B
ccl	j2rzn4lw1DFx8gjaoh...	.apple.com	/	Session	47B
geo	US	.apple.com	/	Session	5B
myacinfo	DAWTKNV2b70841ea...	.apple.com	/	Session	278B
ds01_a	f648472b389bc52bf...	.apple.com	/	May 24, 2013...	202B
Pod	14	.apple.com	/	June 23, 201...	5B
mz_at0	AwQAAAEBAAGI+QA...	.apple.com	/	May 1, 2014 ...	58B
mz_at_ssl	AwUAAAEBAAGI+QA...	.apple.com	/	May 11, 2014...	61B
TrPod	8	.apple.com	/	June 10, 201...	6B

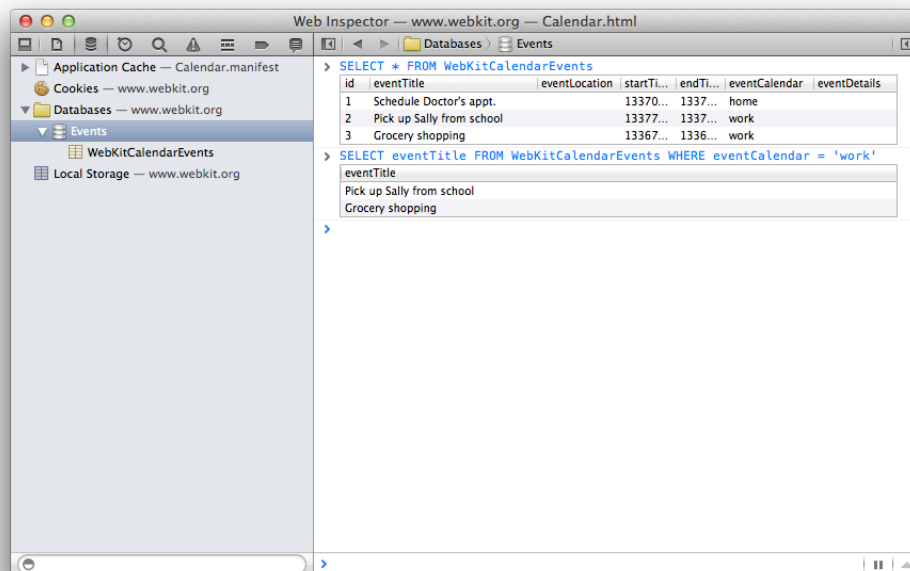
If a webpage uses HTML5 databases, they will display in the Storage navigator. Click the Databases disclose triangle to see a list of open databases and their tables. Selecting a database table displays a data grid containing all the columns and rows for that table, as shown in Figure 3-10.

Figure 3-10 Inspecting databases



In addition to inspecting HTML5 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, as illustrated in Figure 3-11.

Figure 3-11 An SQL query



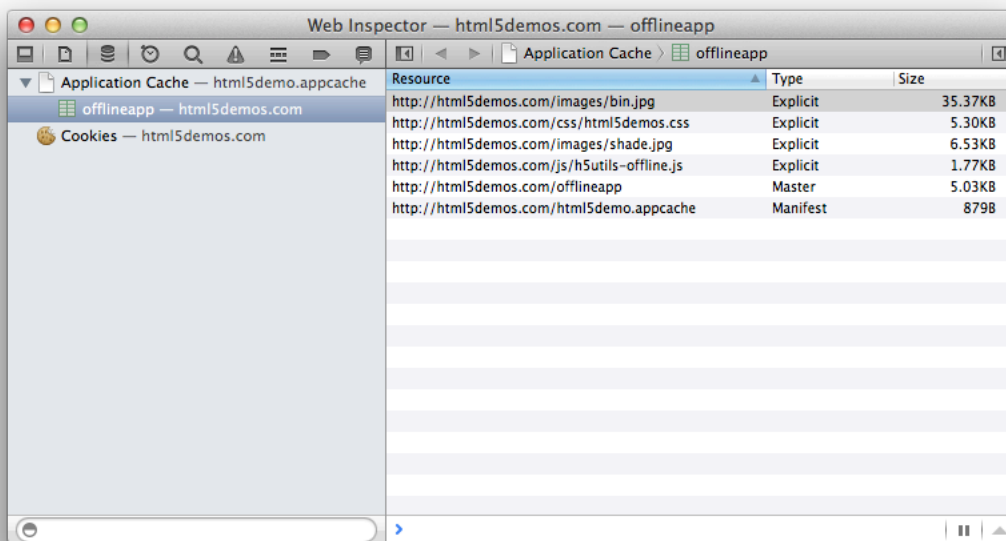
The input to this console has auto-completion and tab-completion for table names in the database, as well as common SQL words and phrases.

Inspecting the Offline Application Cache

Websites that provide a manifest file can have items stored in Safari's offline application cache. On subsequent visits to the website, Safari loads these items from the cache instead of loading them from the website again. This provides a mechanism for websites to offer features such as canvas-based games that users can play, even when their device's browser has no Internet connection.

You can inspect all the current contents of the application cache by opening the Storage navigator of the Web Inspector and clicking the Application Cache disclosure triangle. Safari shows a list of domains that have cached files. Select a domain to see the files in that domain's cache, showing the URL from which they were loaded, the type of entry the resource was specified as (explicit, manifest, master, or fallback), and the file size, as shown in Figure 3-12.

Figure 3-12 Application cache



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 a visual download analyzer and a JavaScript profiler to help you.

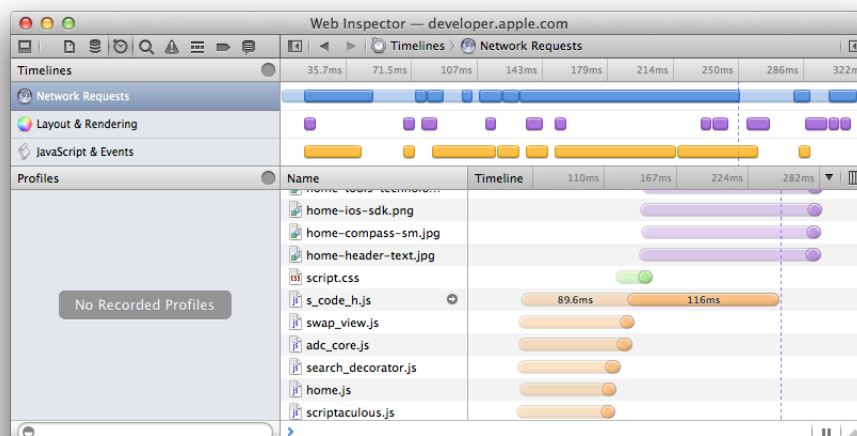
Optimizing Loading, Scripting, and Rendering Times

If your website loads slowly, use the timeline view in the Instrument navigator 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 the timeline view to scan for resources that are not loading at all, such as missing style sheets or requested resources that have been misspelled.

To see how long it takes to load a website, including the time spent evaluating scripts and rendering, click the gray record circle button toward the top-right corner of the Instrument navigator. When the circle turns red to indicate that recording has started, press Command-R to reload the website. Wait until the site has loaded, then click the red circle button to stop recording.

At the top of the content browser, Safari lists the network requests, CSS layout calculations and rendering processes, and JavaScript events involved in loading and displaying the website when they occur chronologically. They appear as a series of blue-, purple-, and yellow-colored bars, as shown in Figure 4-1.

Figure 4-1 Inspecting timelines



When the Network Requests timeline is selected, the bottom portion of the content browser displays each individual file requested from the webpage. The horizontal bar graph shows you when each resource was requested, the latency of the server, and the download time for each resource. Hover over any bar to see additional details in a tooltip.

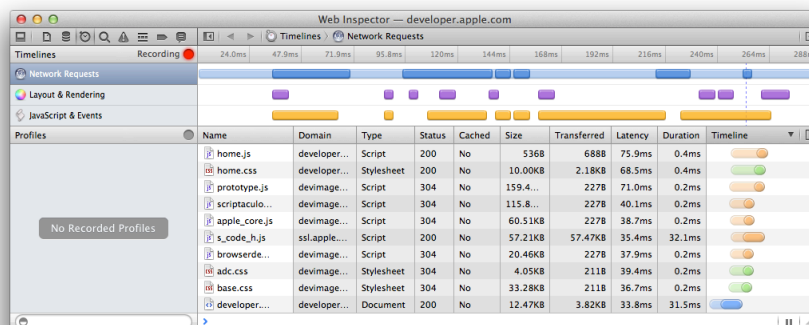
The vertical dashed blue line indicates when the DOM is available to Safari, and is equivalent to the `DOMContentLoaded` JavaScript event. The red line indicates when all resources have loaded, and is equivalent to the `load` JavaScript event.

Click the record button or press Command-R again to record a new timeline.

To see an alternate timeline view showing network latency, resource size, load time, HTTP request type, and status, click the Columns button on the right edge of Web Inspector. This brings up the network timeline view, as shown in Figure 4-2. Drag the divider between any two columns to make a column wider or narrower, and

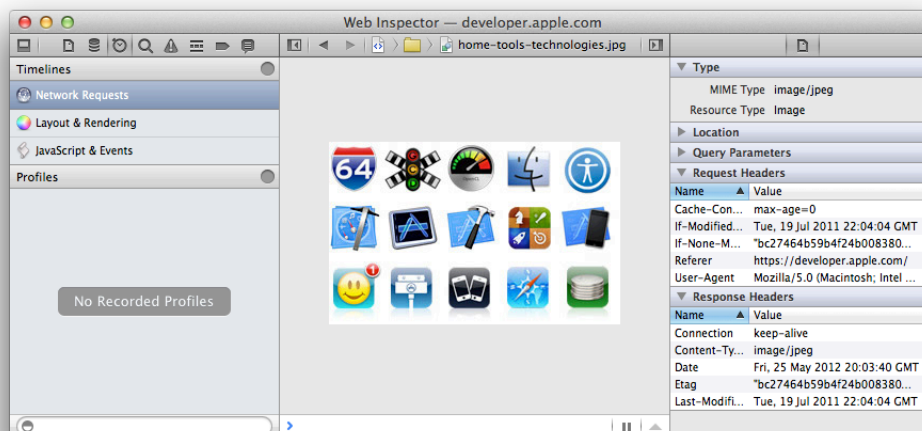
click any column header to sort that column. Sorting by Transferred, for example, will show you the largest resources coming over the wire. Sorting by Latency will show you the files that took the longest for the remote host to serve.

Figure 4-2 Network timeline view



Click the go-to arrow next to an item in the Name column to preview the file's contents, and to see the request and response headers, as illustrated in Figure 4-3. Click the Back arrow to return to the timeline view.

Figure 4-3 Network content and headers view



Just as you can see great detail about every resource that has loaded, you can see great detail about every CSS rendering process and every JavaScript event. Click on Layout & Rendering or JavaScript & Events in the Timelines pane to display a table of each paint and event that has been captured.

Optimizing JavaScript

Underneath the Timelines pane in the Instrument navigator is 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.

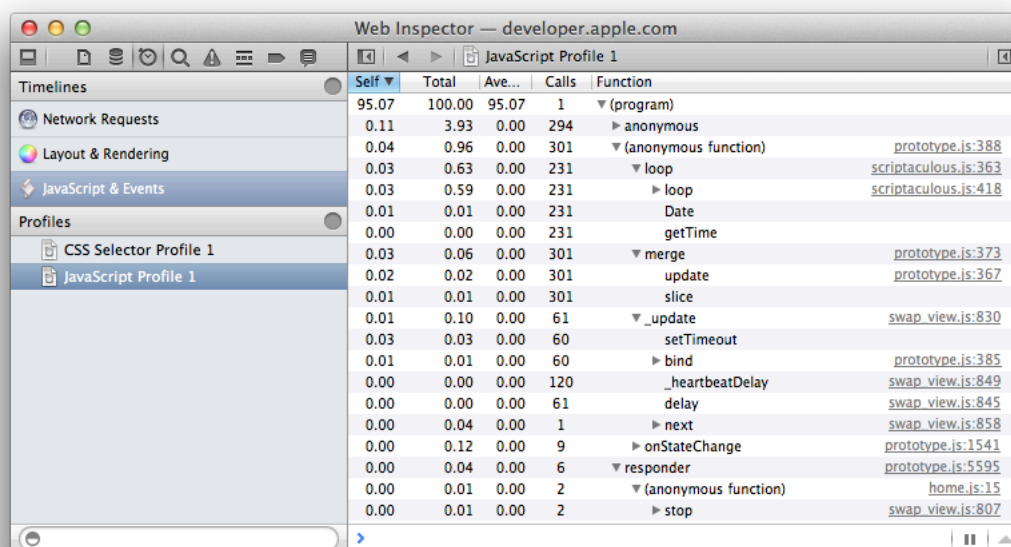
To use the Profiles pane, you must start profiling, either manually or by including a `console.profile()` call in your script. To start profiling manually, click the record button (gray circle) in the top right of the Profiles pane. The record button turns red. To stop the profile, click the record button 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 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 Web Inspector. Manual profiles are named sequentially (JavaScript Profile 1, JavaScript 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 Figure 4-4. The time is displayed as a percentage of total time.

Figure 4-4 Profiling JavaScript



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).

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 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 gray to the right of the function name. The source URL is a link. Clicking it opens the source in the content browser, scrolled to the line number where the function is declared.

Keyboard and Mouse Shortcuts

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

General Shortcuts

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

Web Inspector Shortcuts

	Mac	Windows
Resource navigator	Control-1	Ctrl-1
Storage navigator	Control-2	Ctrl-2
Instrument navigator	Control-3	Ctrl-3
Search navigator	Control-4	Ctrl-4
Issue navigator	Control-5	Ctrl-5
Debug navigator	Control-6	Ctrl-6
Breakpoint navigator	Control-7	Ctrl-7
Log navigator	Control-8	Ctrl-8
Resource details	Control-Shift-1	Ctrl-Shift-1

Node details	Control-Shift-2	Ctrl-Shift-2
Style details	Control-Shift-3	Ctrl-Shift-3
Appearance details	Control-Shift-4	Ctrl-Shift-4
Quick Console	Esc	Esc

Console Shortcuts

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

DOM Tree Shortcuts

	Mac	Windows
Navigate	Up and Down Arrows	Up and Down Arrows
Expand / Collapse	Left and Right Arrows	Left and Right Arrows
Edit Node	Return	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

Style Details 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 0.1	Option-Up and Option-Down Arrows	Alt-Up and Alt-Down Arrows
Increment / Decrement by 10	Shift-Up and Shift-Down Arrows, or PgUp and PgDn	Shift-Up and Shift-Down Arrows, or PgUp and PgDn
Increment / Decrement by 100	Shift-PgUp and Shift-PgDn	Shift-PgUp and Shift-PgDn

Debugger Shortcuts

	Mac	Windows
Step Out	F8 or Command-Shift-;	F8 or Ctrl-Shift-;
Step Into	F7 or Command-;	F7 or Ctrl-;
Step Over	F6 or Command-'	F6 or Ctrl-'
Continue	Command-/ or Command-Shift-Y	Ctrl-/ or Ctrl-Shift-Y
Toggle Breakpoint	Click line number	Click line number

Document Revision History

This table describes the changes to *Safari Developer Tools Guide*.

Date	Notes
2012-07-23	Updated screenshots to reflect new Web Inspector UI.
2011-07-21	Removed references to runaway JavaScript timer. Updated for Safari 5.1. Changed title from Safari User Guide for Web Developers.
2010-08-26	Updated for Safari 5.0.1. Corrected typos and minor errors.
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



Apple Inc.

© 2012 Apple Inc.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Apple Inc.

1 Infinite Loop

Cupertino, CA 95014

408-996-1010

Apple, the Apple logo, iPad, iPhone, iPod, iPod touch, Mac, OS X, Safari, and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries.

App Store and Mac App Store are service marks of Apple Inc.

Java is a registered trademark of Oracle and/or its affiliates.

OpenGL is a registered trademark of Silicon Graphics, Inc.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

iOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.