

Authoring Tools

Tools to assist JavaScript development have lagged way behind tools for HTML page authoring, but the scripting world is the beneficiary of recent efforts by both browser makers and third parties. As you will see in this chapter, the bulk of tool activity is for development of enterprise-based scripting, either because of tool focus or pricing strategy — both of which limit the audience to serious Web application developers.

With the significant exception of Acadia's Infuse, most of the tools available so far are designed not so much to assist the serious scripter as to put the power of scripting into the hands of the nonprogrammer. This is tricky business. HTML authoring tools can hide well the ugliness of HTML tags because the metaphor of a document is widely accepted by authors who publish content. But a script is a different situation. What is the metaphor for a script? And how well does one script fit all application scenarios? If these questions could be answered easily, there would be dozens of programming tools for nonprogrammers in all languages. But that is not the case.

In the rapidly changing environment of Web authoring tools, I can't guarantee that information I provide here will be current by the time you read it. Therefore, for every tool I mention, I suggest also checking the associated URL for the latest developments.

Acadia's Infuse 2.0

Although only in a prerelease version (PR2) as I write this, Acadia's Infuse 2.0 for Windows 95 shows great potential as a helpful authoring tool for serious scripters of client- or server-side JavaScript. The product includes a source code editor that has scripters in mind, offering shortcuts for inserting both HTML and scripting construct skeletons that authors can fill in. An excellent example is the keystrokes Infuse saves when adding a `for` loop to a script. Type the word "for" and a space, and Infuse automatically enters the entire basic structure of a `for` loop, assigning the `i` variable and preselecting the placeholder where you enter the value of the condition to be tested at the beginning of each loop (see Figure 46-1).

46

CHAPTER



In This Chapter

Acadia's Infuse 2.0

Netscape Visual
JavaScript

Other server-side
authoring tools



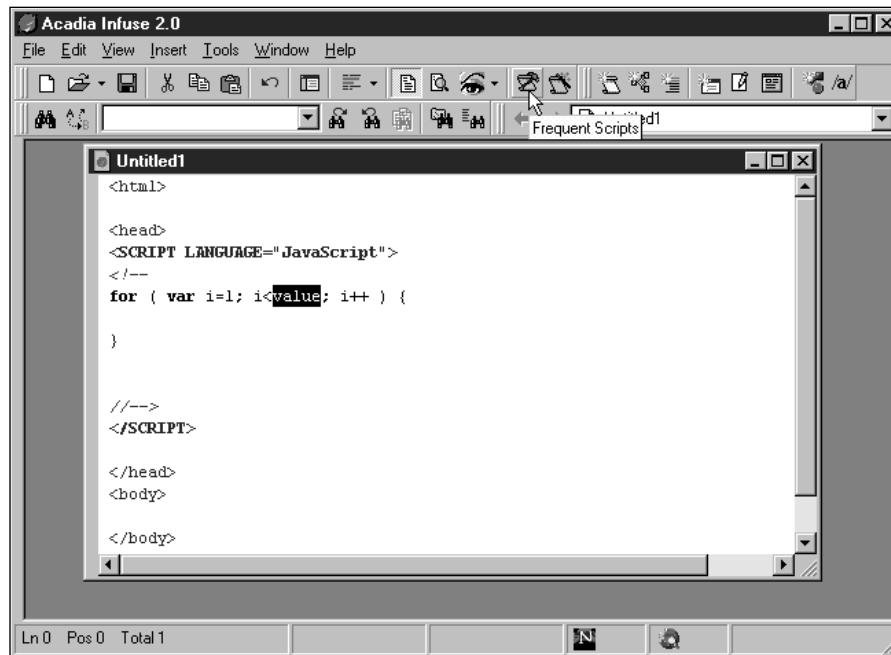


Figure 46-1: Acadia's Infuse 2.0 (prerelease)

Infuse 2.0 comes with a library of frequently used scripts. You can, of course, add your own scripts to the library and insert them into your code with a couple of clicks (and preview the code to each script before you use it).

Even experienced scripters should appreciate one of the tools built into the product that scans a page for language words and objects that may not be supported by all browsers. You can ask Infuse to examine the scripts for Navigator and Internet Explorer compatibility back to the earliest versions of the scriptable browsers. Right up to the current technologies, Infuse 2.0 supports development of Internet Explorer scriptlets. A separate tool by Acadia, BeanBuilder, assists in the creation of Navigator 4 JavaScript Bean (JSB) components. Acadia Software is at <http://www.acadians.com>.

Netscape Visual JavaScript

The name of this authoring tool may mislead the scripting community into thinking that this is a separate version of JavaScript, much like Visual Basic is its own incarnation of Basic. But Visual JavaScript is an authoring tool for creating the same JavaScript code that you would create by hand or with a tool such as Acadia's Infuse 2.0. Visual JavaScript can be used for both client- and server-side JavaScript development. The tool itself is a Java application and will be available for all operating system platforms.

In many ways Visual JavaScript (VJS) resembles an integrated development environment (IDE). Each application you work on is assembled as a project. A project may contain any number of HTML files, Java applets, JavaScript files, images, and server data sources — all managed inside a VJS project.

Each HTML document (called a page) can be viewed and edited in three different views. The Layout view is how you usually assemble the basic content and layout of a page by positioning elements on a dummy page. The plan for the final release indicates that you will have the same level of layout composition tools as is provided in Netscape Composer (part of Communicator). Even so, this view does not purport to be a WYSIWYG (what you see is what you get) display. Many elements that let you connect things like form elements and database connections to forms aren't really seen when displayed in the browser, but must be shown here to let authors make those connections. The two other views are a hierarchical Structure view of the entire document and a Source view that shows the HTML source code generated by the tool.

One of the most powerful parts of Visual JavaScript is the Component Palette (see Figure 46-2). Components consist primarily of Java Beans or JavaScript Beans (JSBs). As a result, they all reflect the introspection feature of beans. When you drop a component onto a page under construction, you double-click on the item to reveal an Inspector window for that component. In that window you can set default properties and assign scripts to event handlers (if the component supports events). For items that are normally HTML objects (for example, form elements), setting these properties instructs VJS to write the HTML tag and attributes for that object. Some of the property settings are quite useful. One of my favorites is the one for setting a color (see Figure 46-3). A pop-up list of the color names with the actual color sample makes it easy to find a suitable color for the object's attribute.



Figure 46-2: Visual JavaScript

HTML generated by VJS includes a lot of commented statements and tag attributes that you may not recognize. All of this extraneous material is used by VJS to display and manage the elements and component connections inside that environment. The browser does not need this information. By the same token, you can import an existing application into VJS, but not all of the connections you've made manually in your page will be converted to the VJS format for further editing visually (although you can continue to edit the source code as you wish).

Despite the ease with which Visual JavaScript lets authors work with preassembled script components, authors are not prevented from adding or enhancing scripts manually. For example, if you wish to create a function executed by a button's `onClick=` event handler, you can add the script statements of that function in the inspector for the button object. You don't have to assign names to the functions, because VJS automatically assigns names to such functions based on the name of the object.



Figure 46-3: Visual JavaScript property inspector and color selector

Visual JavaScript is also endowed with powers to simplify the creation of server-side applications for Enterprise Server 3 and the LiveWire connection between applications and databases. As you work with an application under development in VJS, the tool connects with the database to let you generate connections between table data and HTML form elements with click-and-drag ease.

You can get more information about Visual JavaScript at <http://developer.netscape.com>.

Other Server-Side Tools

Assisting in development of server-side applications — especially making the connection between HTML documents and databases via LiveWire in Enterprise Server 3 — has attracted at least two other tools developers. Elemental Software (<http://www.elementalsoftware.com>) has been shipping Drumbeat since early 1997. A more recent entrant is NetObjects (<http://www.netobjects.com>) with its JavaScript Components add-on to the popular Fusion Web site authoring toolset. Both tools are designed to simplify the writing of server-side JavaScript code to allow nonprogrammers to hook up databases to Web pages.

I expect the JavaScript authoring tools race to heat up as the adoption of level 4 browsers increases. So much more of Web authoring — from Dynamic HTML to server-side CGIs — is relying on JavaScript (or at least the common denominator ECMAScript) that tools vendors will be attracted to solving the problem of increasing complexity of HTML and scripted pages.

