



## Get with the script

***How can you hope a user will buy your product when due to browser limitations he can't even read your Web site?***

One of the problems with Internet development is uncertainty over the abilities of clients. By this, I don't mean whether Mr X can juggle, but whether his browser can.

In the "perfect" world of the intranet, you should be able to control everything, resulting (probably) in a perfect harmony between browser and GIS application. You will know whether the standard browser will support a particular plug-in; indeed, you may install it yourself. Yet intranets are primarily for sharing information within a company. On the other hand, the Internet is mass-market, high-volume and, unfortunately, uncontrollable. How can you hope a user will buy your product when, due to browser limitations, he or she can't even read your Web site? To answer this, we need to look at a more fundamental question. How can one bring order to the chaotic world of the Internet?

Scripting languages are simple programming languages that can create executable commands based upon a set of supplied parameters or files. The common gateway interface (CGI) is one such language, for transferring data between Web servers and other applications. This was sufficient when there were very few things a browser could do other than read text and display images. As browsers developed and more sophisticated features and complicated demands were asked of the technology, many servers have become over-burdened by the complexity and volume of requests. This gave the spark for the initial drive for scripting languages contained as part of the client browser. This meant that some code can appear in the Web page, moving the burden from the server to the client, because scripts are not pre-compiled but interpreted by the browser.

Unfortunately, increased competition in the market forced divisions in the abilities of browsers. Standards were extended proprietarily and new features added by the bucket load. While Microsoft and Netscape both support versions of JavaScript, these are not completely compatible. To complicate the matter, Microsoft developed VBScript from Visual Basic.

This is okay for the mass of Web pages that contain information about families, games or even weddings, but for serious commercial ventures, the alienation of a significant proportion of your client base can be devastating.

To overcome the client-bound limitations of scripting languages, both Microsoft and Netscape went back to basics and integrated their scripting engines into their servers. Microsoft's Active Server Pages (based on JavaScript and VBScript) and Netscape's LiveWire (based on JavaScript) are implementations of server-side scripting. This contains code executed on the server before being sent to the client. Not only does this mean that the page can be pre-configured for a particular browser before it even sees a piece of HTML, but as the page is pre-processed, the code is not even seen by the browser. This is especially important if you wish to protect intellectual property or just don't want users being able to see what is being done in the background.

Server-side scripts can also communicate with other applications, such as mapping applications or databases. Output from these can be included in the Web page, allowing you to tightly control the output from these systems. You can use server-side scripting with client-side scripts, resulting in the most comprehensively developed sites.

So it's not which language you use, but how you combine scripting on both the server and the client. With the Web moving into full swing as an applications platform, the planning stages are vital in any Web development today.

*MATTHEW TOON is a member of the MA editorial board.*