Collision Course: JavaScript and VB Script

Click & Retrieve Source CODE!

BY MICHAEL TEMPLEMAN

Prepare for a head-to-head competition in a new category: HTML scripting tools.

Editor's Note: This feature article is a departure from our normal technical articles. VBPJ usually insists that its articles cover products our readers can use now, and focus more on solving problems than on product shoot-outs and future technologies.

However, given the hype, misunderstandings, and rapid changes in Internet development tools, we thought a comparison of the two exciting competitors in the emerging field of Web scripting tools justified special treatment. At this writing, JavaScript could be used only in Netscape's version 2.0 browser, which had just shipped, and VB Script was in alpha and required an alpha version of Microsoft's browser. So, what we could demonstrate about VB Script is much more limited than what we could do with its competitor.

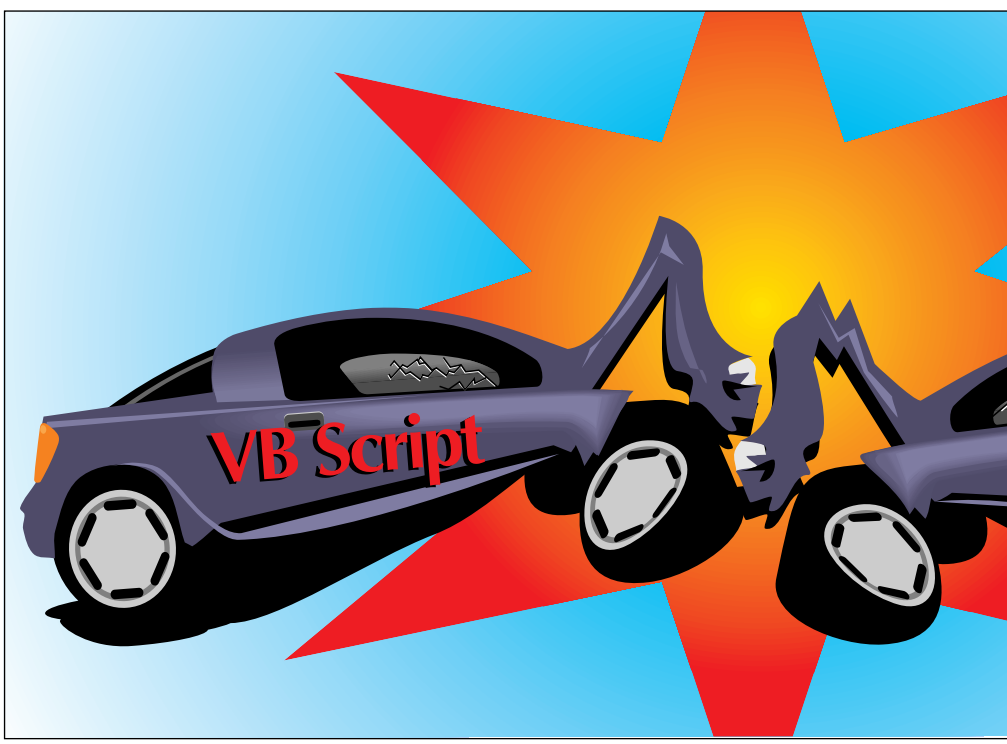
Undoubtedly, the final specifications and implementations in VB Script will change by its commercial release. Because Microsoft has promised support for JavaScript in its browsers and other Internet tools, both products are directly relevant to our readers.

Michael Templeman is president of MetaBridge Inc., a Seattle-based company providing information design and software development services for companies doing business on the Internet. Reach Michael at miket@metabridge.com on the Internet. Visit the MetaBridge Inc. Web site at www.metabridge.com.

JavaScript. Visual Basic Script (VB Script or VBS). What are they? What are the implications of this kind of technology? As I try to answer these questions in this article, I'll provide the succession of examples I used myself to demonstrate the capabilities and differences between each language.

JavaScript is a scripting language developed and promoted by Sun and Netscape and supported by the latest Netscape Navigator 2.0 Gold and several of the latest Navigator 2.0 beta browsers. JavaScript is not Java: it merely resembles Java. Although Netscape states that JavaScript is based on Java, shared functionality appears limited. Currently Netscape Navigator seems to be the only browser supporting JavaScript. For more up-to-date information from Netscape, visit http://proto.netscape.com/ and download the latest version of Netscape Navigator.

VB Script is a scripting language under development by Microsoft and will be supported in the next version of Microsoft's browser, Internet Explorer 3.0. VB Script is a reduced version of Visual Basic for Applications. No browsers other than Internet Explorer support VB Script. However, the price seems right—it's free as long as you don't change the language, and you agree to Microsoft's compatibility tests. Visit http://www.microsoft.com/intdev/sweeper/sweeper.htm to download the tools for VB Script development and other Microsoft Internet development tools (code-named Sweeper).



It's important to grasp what these languages are *not*. Neither are compiled into intermediate code. The source is embedded in an HTML page for all to see and borrow. The browsers parse and interpret the scripting. Neither language is object oriented and there is no inheritance, polymorphism, or operator overloading. Both languages are object based, which usually means at least the capability to package code and data together.

I developed the samples for this article using a 24 MB Pentium 90 running Windows 95. I tested the JavaScript code using Netscape Navigator 2.0. I tested the VB Script samples using Internet Explorer 3.0 (alpha) running under Windows 95, and they use the libraries and controls provided by the Sweeper SDK. I did not use Java applets in the JavaScript samples.

HELLO, JAVA HEADS!

JavaScript is a simple, C-like language that is passed to the browser embedded within an HTML document. The language supports creation of objects, events, functions, methods, and a serviceable set of functions. An object model integrates the browser, document, forms, elements, applets, and plug-ins. JavaScript, like Java, cannot read or write to your hard disk for security reasons. It is loosely typed and interpreted by the browser.

I always start with the canonical "Hello World" application (see Figure 1) to ensure that the environment is working well enough to write a string of text. To create the standard Hello World application, I needed a way to put a string on the output stream. I had to find a method and make certain that the script called the method:

```
<!– My first JavaScript Page. –>

<HTML>
<HEAD>

<!–   to hide script contents from _
      old browsers that don't _
      understand the SCRIPT tag, _
      use HTML comment

<SCRIPT LANGUAGE="JavaScript">
document.write("Hello world")
</SCRIPT>

// end hiding contents from _
   old browsers  –>

</HEAD>
<BODY>

</BODY>
</HTML>
```

The method is a standard HTML document with a script tag and some code. The script is executed by the browser when the page is loaded. Functions are stored on load and called by events. The script puts the string onto the document by invoking the write document method.

I wrote a simple script that validates a registration form before submitting it. This has real-world validity: why load the server with sanity checks on input forms?

I needed to create a registration form, access the fields in the form during validation, validate the registration fields when the user tries to submit the form, and alert the user to a problem in the registration form without disturbing the form. I also needed to keep the code as tight as possible (see Listing 1).

When I first wrote the checkForm function, I assumed I would pass it the form object I wanted validated. This function is closely coupled with the form elements. If you change the form elements, you change the function. Thus I prefer to put the brute-force validation into other functions I could easily reuse if the form changed.

I wrote the CheckAlpha and CheckNum functions to validate alphameric and numeric fields. I also added field range checking. The checkForm function calls the different validation functions, and if the field does not pass validation, the alert method is invoked. A tighter solution would be to put the alert in the validation functions and concatenate the functions in one expression. JavaScript provides C-type short-circuit logical expression evaluation, which would be a desired enhancement. The CheckNum and CheckAlpha functions demonstrate string operations. Given a string, you can extract a substring, make comparisons, change the case, and check the length.

The final task, validating the form before sending, remained. I tried putting an onClick method into the submit button field, which achieved the input validation, but sent the form regardless of whether it was valid. I found in the documentation an event for the form called OnSubmit. I tried to use this single line:

```
onSubmit = "checkForm(frmRegister)"
```

but, it still submitted the form. Apparently the function return did not affect the onSubmit event. Instead I tried inserting the code directly in the tag:

```
onSubmit="if
(!checkForm(frmRegister))_
   return false"
```

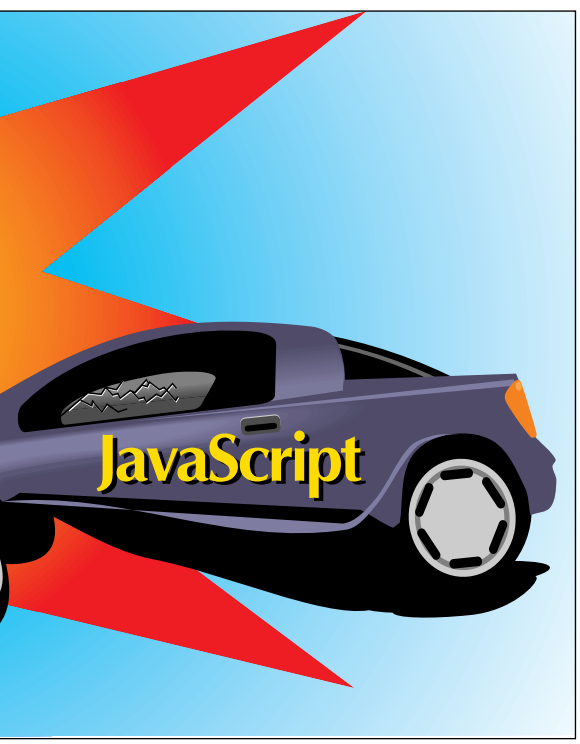
which worked. Apparently, the function didn't return a compatible result for the submit event, but putting code in line did. This code will check each field. The name must be filled in with text. At least one address field must be filled in, and the city name must be filled in with text. The state code must be exactly two text characters, and a numeric zip code must be there. When run, the code performs these simple checks. If the form fields pass validation, the form is sent.

USING CLOCK FUNCTIONS

My final JavaScript sample is aimed at recurring events. Specifically, I wanted to see if a timer was available in JavaScript. I had encountered a clock example on the World Wide Web that seemed to do the trick, so I looked at what already existed and added some comments and minor changes to demonstrate a couple more aspects of JavaScript.

My goal was to build an analog clock applet and use JavaScript to drive the timing. This proved impossible in the time available, so I compromised with an input field. However, if you have a Java applet that displays an analog or digital clock, it's a simple matter to swap the input field with the Java applet. The interface to Java applet properties is the same as that of form elements.

To build a clock, you need only a timer, a way to field timer events, a way to get the current system time, and an output field (see Listing 2). JavaScript functions provide a one-shot timer and a simple way to provide a timer callback function. Globals are declared in the script block and initialized when the page is loaded. You need a global to hold the timer identifier so that you don't stop a non-existent timer. You also need a flag for the timer to indicate



that the timer is active, and functions to initialize the clock, field the event, and update the clock fields. These operations are in the Showtime() function. Showtime() pulls the current system time out of the JavaScript-provided GetDate() function, which returns the current date and time in milliseconds as a date object. The date object provides methods to extract the current time, which the Showtime() function uses to build a string containing the current time. You could use the ToLocalString() method of a date object to have a date/time string created for you, but the result is limited, so I used the other approach.

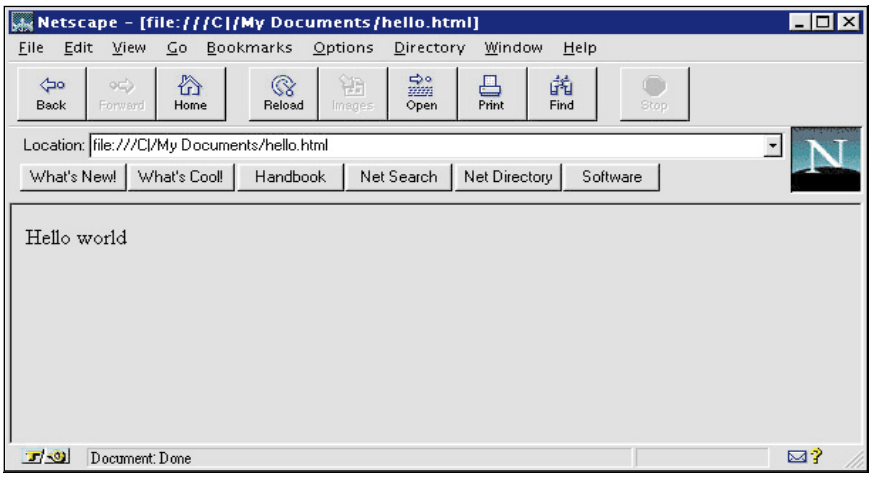


FIGURE 1 ***Hello Java World.*** *This "Hello world" was written with JavaScript (ready for embedding in a Web site), downloaded from the Web, and run in a browser. It's a standard HTML document with only a script tag and some code.*

You could try to initialize the clock in the same way as the registration form example: call the Showtime function from the JavaScript block when the page is loaded. This will cause a JavaScript error telling you that the object frmClock has no properties: frmClock does not yet exist and cannot be assigned values. The JavaScript code was loaded and executed before the page was created. You must initialize the clock after the window is created.

Navigator fires an event at just the right time. It is called the onLoad event and you can see how to link the event to the function in the tag:

```
<BODY
bgcolor="#ffffff"_
onLoad="startclock()">
```

Once the window has been loaded (onLoad can be used for frames, too), the onLoad event is fired. This calls the startclock() function, which formats the clock object, starts the timer, and tells the timer to call startclock() when the timer fires.

The clock example also provides the option to write out the time for Navigator's status bar (the text field at the bottom of the window). Access this field by setting the window status field to the time string. If the global variable fStatusDisplay is true, then the Showtime() function will also set the window status field to the current time.

FStatusDisplay is initialized to false when the page is loaded. When clicked, the fldStatFlag check box element generates an event to which I tie a snippet of JavaScript code to the FStatusDisplay global to the value of the check box:

```
onClick="fStatusDisplay = _
   this.form.fldStatFlag.checked"
```

I first assigned fStatusDisplay the result of this form, fldStatFlag.value, but doing so returned a string. I determined that the Checked() method returns a Boolean that could be assigned to the fStatusDisplay global, so after adding this, everything worked as advertised.

JavaScript is available now for Navigator users. it is object based; and it provides a usable object model for Navigator, HTML, Java applets, and in-line plug-ins. JavaScript also allows a programmer to define and create new objects, provides dynamic binding, and is very much like C. JavaScript appears to be secure: programmers cannot read or write to the hard disk or system properties.

JavaScript appears to be well integrated with the Navigator and HTML object model. You can trap events at reasonable states, read and write Navigator and HTML object properties, and invoke methods for those objects. I did not see any way for plug-ins and Java applets to fire events that could be fielded by JavaScript code, a real limitation to integrating Java and JavaScript.

Developing with JavaScript is challenging. I developed the code in Developer Studio and ran code in a Navigator window. To test the page, I switched to the test Navigator window and reloaded the page. When Navigator encounters a parsing error or a runtime error, it puts up a dialog and reports the error. I debugged the scripts using alerts as variable watches and breakpoints.

Like HTML, once you implement in JavaScript, anyone on the Internet can incorporate your code in their own pages. This is also true of VB Script, so put your

VB3 VB4

User Tip

CHANGING THE ATTACHED PICTURE BACK TO NONE

Once a graphic (bitmap, icon, or metafile) is attached to a form or picture control, Visual Basic will seem to thwart any attempt to change the property to the original default of <none>. To work around this behavior, click the Picture property in the Properties list before highlighting the graphic value. Another way is to change the focus to any other property on the Properties window and reselect the Picture property. The DEL key then correctly deletes the graphic value and changes the value to <None>. The graphic correctly disappears from the form or picture control.

—Douglas Haynes

SEND YOUR TIP

If it's cool and we publish it, we'll pay you $25. If it includes code, limit code length to 10 lines if possible. Be sure to include a clear explanation of what it does and why it is useful. Send to 74774.305@compuserve.com or Fawcette Technical Publications, 209 Hamilton Ave., Palo Alto, CA, USA, 94301-2500.

```
<HTML>
<HEAD>
<TITLE>Registration Validation Sample</TITLE>
<SCRIPT LANGUAGE="JavaScript">

//Validate that this string contains no digits
function CheckAlpha(str, min, max)
{
  if (str.length < min || str.length > max)
    return false

  for (var i = 0; i < str.length; i++) {
    var ch = str.substring(i, i + 1)
    if (ch >= "0" && ch <= "9") {
      return false
    }
  }

  return true
}


//Validate that this string contains only
// digits
function CheckNum(str, min, max)
{
  if (str.length < min || str.length > max)
     return false

  for (var i = 0; i < str.length; i++) {
    var ch = str.substring(i, i + 1)
    if (ch < "0" || ch > "9") {
          alert(alertStr)
       return false
    }
  }

  return true
}


// Check all the fields of the form
// for validity.
function checkForm(form)
{
  // first, check that the name has no numbers in it
  // that isn't a valid name
  if (!CheckAlpha(form.fldName.value, 1, 65)) {
    alert("You must enter a valid name")
    return false
  }


  // Next, check that there was at least one address
  // field and a city field entered.
  // Note that numbered cities are also invalid
  if (form.fldAddr1.value == "" _
    && form.fldAddr2.value == "") {
    alert("You must enter at least one address line")
    return false
  }
  if (!CheckAlpha(form.fldCity.value, 1, 65)) {
    alert("Invalid city name")
    return false
```

CONTINUED ON PAGE 53.

LISTING 1 ***JavaScript Form Validation.*** *This code performs a series of simple checks. For example, the name must be filled in and digits are not allowed. At least one address field must be filled in with some text, and the state code must appear and must be exactly two text characters. If the form fields pass validation, the form is sent.*

proprietary code into OCXs, Java applets, and plug-ins, and glue them together with these scripting languages.

HELLO WORLD OCX?

Because VB Script is earlier in its development phase than JavaScript and using it requires the alpha release of Internet Explorer 3.0, developing with the VB Script alpha was shaky. Both the feature set and procedures in VB Script may change by the time it's released. Take the code shown here with a grain of salt.

Internet Explorer, with the VB Script interpreter OCX, does not report any errors. The rewrite for VB Script describes Navigator, not Internet Explorer. The VB Script OCX provided in the SDK had no error reporting, and the MsgBox function didn't work. A parser error seemed to invalidate the entire script, but without any kind of error report it was hard to tell. It was all code and pray. Not all OCX properties appeared to be accessible, and not all events fired properly. Finally, inserting objects by their CLASSID is definitely not the most programmer-friendly way to go about referencing OCXs.

I intended to build a form validation field, but without any kind of working message box and HTML Programming directions, I had to forgo that exercise. According to the online documentation for Sweeper, Microsoft intends to provide that information and the version of Internet Explorer that will support it. We might get access to it at the Professional Developers Conference in March 1996.

CONTINUED FROM PAGE 51.

```
  }

  // Better check the state, too
  if (!CheckAlpha(form.fldState.value, 2, 2)) {
    alert("Invalid state code")
    return false
  }

  // Now validate the zip code
  if (!CheckNum(form.fldZip.value, 5, 9)) {
    alert("Invalid zipcode")
    return false
  }

  // Passed all the tests.
  // The form may be submitted
    return true

}

</script

// end hiding contents from old browsers  -->

</HEAD>

<!-- Now, the body of the form -->

<H1 align=center>Mike's JavaScript Form validator</
H1>

<FORM NAME="frmRegister" METHOD="POST" onSubmit="if _
  (!checkForm(frmRegister)) return false">

<P>
  Name: <INPUT NAME="fldName" VALUE=""
MAXLENGTH="65" _
    SIZE=65><BR>
Address1: <INPUT NAME="fldAddr1" VALUE="" _
  MAXLENGTH="65" SIZE=65>
<BR>
Address2: <INPUT NAME="fldAddr2" VALUE="" _
  MAXLENGTH="65" SIZE=65>
<BR>
  City: <INPUT NAME="fldCity" VALUE=""
MAXLENGTH="35" _
    SIZE=35> State:
<INPUT NAME="fldState" VALUE="" MAXLENGTH="2" SIZE=2>
Zipcode<input name="fldZip" VALUE="" MAXLENGTH=9 _
  size=9><BR><BR>
<INPUT TYPE=SUBMIT VALUE="Submit" _
  NAME="btnSubmit">
</FORM>


</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>JavaScript Clock</TITLE>
<SCRIPT LANGUAGE="JavaScript">

<!--  to hide script contents from old browsers

// First set up the global variables needed for the
// clock
var idTimer = null;
var fTimerOn = false;
var fStatusDisplay = false;

// We need an initialize/clear function for the timer
// stop the clock if it is on, then clear the timer
// flag
function stopclock (){
  if(fTimerOn)
    clearTimeout(idTimer);
  fTimerOn = false;
}

// Start up the timer clock.
// lets try to only start it once!
function startclock () {
  // Make sure the clock is stopped
  stopclock();
  showtime();
}

// Compute the current time
// and stuff it into the timer
// field
function showtime () {

  // get the date and time
  var now = new Date();
  var hours = now.getHours();
  var minutes = now.getMinutes();
  var seconds = now.getSeconds()

  // convert to a 12 hour clock
  // until the US understands a
  // normal 24 hour clock
  var timeValue = "" + ((hours >12) ? hours -12 _
    :hours)
  timeValue += ((minutes < 10) ? ":0" : ":") + _
    minutes
  timeValue += ((seconds < 10) ? ":0" : ":") + _
    seconds
  timeValue += (hours >= 12) ? " P.M." : " A.M."
  document.frmClock.fldClock.value = timeValue;
```

CONTINUED ON NEXT PAGE.

LISTING 2 ***Build a JavaScript Digital Clock.*** *You need a timer, a way to field timer events, a way to get the current system time, and an output field. JavaScript functions provide a one-shot timer and a simple way to provide a timer callback function.*

CONTINUED FROM PREVIOUS PAGE.

```
   // If we want to have the status bar update the time
   // set this flag to true
   if (fStatusDisplay) {
     window.status = timeValue;
   }

   // set the timer to fire once a second
   // and put the timer identifier into our global
   // call showtime every time the timer fires
   idTimer = setTimeout("showtime()",1000);

   // and set our timer on flag to true
   fTimerOn = true;
}

// end hiding Javascript from old browsers -->
</script

</HEAD>
```

```
<BODY bgcolor="#ffffff" onLoad="startclock()">

<H1 align=center>One Second timer Example</H1>

<FORM NAME="frmClock" onSubmit="0">

<P>
The current time is: <INPUT NAME="fldClock" VALUE="" _
  MAXLENGTH="15" SIZE=15><BR>
<P>
See a clock on the status bar? <INPUT TYPE="checkbox" _
  value = false NAME="fldStatFlag" _
  onClick="fStatusDisplay = _
  this.form.fldStatFlag.checked">
</FORM>


</BODY>
</HTML>
```

As before, the canonical Hello World is the simplest possible way to make certain that the environment works. The goal is to reduce programmer error to verify that everything else works.

Without an object model or OLE Automation interface for Internet Explorer, I could not write "Hello World" to the HTML document in the same simple way as with JavaScript. Nor could I know how to reference an input field element. So I wrote the string to a rich-text OCX, using the sample supplied by Microsoft in the Sweeper SDK as a starting point. Only some aspects of the sample worked, but the rich text control accepted strings assigned from VB Script.

The next problem I encountered was a lack of a way to write text to the rich-text OCX at page-load time. I found no event to provide a point in time when all objects had been instantiated and were ready for business. I inserted a ThreeD Button OCX and used the click event as a way to write "Hello World" to the Rich Text control. The click event function for the ThreeD button, Cmd1_Click(), assigns the Hello World string to the Rich Text control. Click the button and Hello World appears in the rich-text field:

```
<HTML>
<HEAD>
<TITLE>Hello world home page</TITLE>
</HEAD>
<BODY>
<H1 align=center>Hello World Test</H1>

<!- SSCommand - threed32.ocx>
<pre>
Make it say hello world:   <INSERT _
   CLSID="{0ba686b4-f7d3-101a-993e-_
   0000c0ef6f5e}" HEIGHT=30 _
   WIDTH=100 ID=Cmd1>
</pre>
```

```
<!- RichText - richtx32.ocx>
<pre>
Output field:  <INSERT _
   CLSID="{0ba686b4-f7d3-101a-993e-_
   0000c0ef6f5e}" HEIGHT=30 _
   WIDTH=100 =ID=Cmd1>
</pre>

<!- RichText - richtx32.ocx>
<pre>
Output field:  <INSERT =_
   CLSID="{3b7c8860-d78f-101b-b9b5- _
   04021c009402}" HEIGHT=40 _
   WIDTH=300 =ID=RichText1>
</pre>

<!- abasic.ocx - Hidden ActiveBasic
Control>
<INSERT CLSID="{9A03A790-_
   2C33-11CF-B2E2-00AA00C0178F}"
   HEIGHT=10 WIDTH=10 ID=Page
   CODE="Sub Cmd1_Click()
         RichText1.text = _
            'Hello world'
      End Sub"
>
</BODY>
</HTML>
```

The "Hello World" example has turned into an event-driven page that loads the VB Script interpreter, the Rich Text control, and the ThreeD button (see Figure 2). The documentation and sample code show that VB Script is indeed a subset of VBA as Microsoft states. While the language itself is important, the browser/HTML/applet interfaces are more so. Microsoft has developed a new OLE interface for scripting, IScriptEngine, which *any* script engine can support. The current VB Script uses the interface to drive Internet Explorer. Implementing VB Script through such an interface means that VB Script updates can be made independently of Internet Explorer updates. Also, com-

FIGURE 2 ***OLE Overkill.*** *Writing "Hello World" in VB Script was a challenge. I had to send a string to a rich text OCX. But in fairness, this is an early version of VB Script: running it requires the alpha release of Microsoft Internet Explorer 3.0.*

petitors may deliver an alternative solution to VB Script in Internet Explorer.

COMPARING THE TWO

Java is much further than VB Script in the development cycle. Netscape's JavaScript has a way to go before release, but it's on its way. VB Script is clearly in an alpha state at this time. I don't know if this state reflects the current development of VB Script, the IScriptEngine interface, Internet Explorer 3.0, or all three.

For both JavaScript and the Navigator object model, substantial documentation is available online. The Netscape JavaScript page, as well as many third-party JavaScript pages, provide a suite of samples in which the functions provided by the interpreter were operational. Because it's pre-alpha, the VB Script documentation is limited and no third parties have put up VB Script sample pages.

For C and Java programmers, JavaScript's reserved-word list and operator list are easy to understand. VB Script is a natural for VB developers.

Technically, the weakest architectural aspect of JavaScript is its apparent inability to extend itself to handle events fired by applets and plug-ins. Thus if a designer were to add a Java applet to a document that fired an event, there would be no way to write a JavaScript function to field that event. If an event passed parameters to the event method, there should be a way for the object generating the event to pass the parameters to the event method.

One method might be a simple timed trigger from an audio player. Presume that an audio player fires events to Navigator at predetermined times during play. This event could be picked up by a JavaScript function and used to change a label, change color, or force a load of a new picture, for example. This is not feasible with the current implementation of JavaScript, but it is feasible with VB Script.

Market acceptance of JavaScript depends not only on technical issues, but also on JavaScript's incompatibility with 30 percent of the browser market. Current estimates indicate that Netscape Navigator is used by about 70 percent of the market, and many of those users don't have JavaScript-capable browsers. For JavaScript to become the standard, Netscape must dominate the market.

Microsoft is giving away the VB Script binary, freeing itself from the need to dominate the browser market with Internet Explorer. If a significant percentage of browser providers use the VB Script engine, a good chunk of the browser market will deliver a competing but incompatible standard, resulting in script-enhanced HTML pages that are incompatible with many browsers on the market. Embedding scripts into tags that are ignored by incompatible browsers is no solution. These scripts are the "glue" holding together pages: imagine how the pages will behave when the glue is removed.

Microsoft's response is that the OLE Scripting interface allows the browser to download the OCX that implements the language. This is like thinking that C++ will solve the cross-platform problem, when it just turns API incompatibility to a framework incompatibility.

Both scripting languages promise to provide glue for applets and HTML. JavaScript has partly delivered on that promise and Microsoft will do so soon. Both languages are sound, but the transition to widespread use will engender many compatibility problems. ×