Serving Up The Web

Click & Retrieve Source CODE!

BY CARL FRANKLIN

Microsoft's new Internet Information Server makes data access easy, and OLEISAPI support taps the power of VB.

Although Microsoft has been late in responding to the popularity of the Internet and the World Wide Web, slowly but surely Bill Gates and company have been putting together the pieces of an Internet tools strategy. The two key tools are Visual Basic Script, with its ability to code Internet applications that run inside a Web browser, and the new Microsoft Internet Information Server (MIIS), formerly known by its code name, Gibraltar.

Microsoft's Internet Information Server has many parts. I spent a long weekend exploring a late beta copy of MIIS and found it confusing at first. But once I understood the architecture I was blown away, especially when I grasped the importance of the Internet Database Connector and the Internet Server API (ISAPI)—technologies that are key to the power of this package. I sat in a dark room for nearly an hour to let all the possibilities sink in. In case you missed it, Microsoft formally introduced MIIS February 12. This free Web server can be downloaded from the Web at http://www.microsoft.com/infoserv. It's also bundled with Windows NT (for more on MIIS, see "Piece Together Microsoft's Internet Puzzle," by Roger Jennings, in this issue).

In short, the Internet Database Connector bypasses external Common Gateway Interface scripting for data access. ISAPI (specifically, OLEISAPI.DLL) lets users access OLE Automation servers through the Web. The power and simplicity of this approach may inspire you to rethink your development strategy. A core advantage of this new architecture is reduced coding redundancy, and we all know that redundancy is bad.

Before I go into the technical details, I'll put this new package into perspective. The Database Connector makes data access easy because it's part of the server itself. On the VB side, the Internet Server API does for CGI what IDC does for database access.

Dynamic link libraries written with ISAPI run in the server's address space, requiring less time and fewer resources to process scripts. Also, OLEISAPI lets you access an

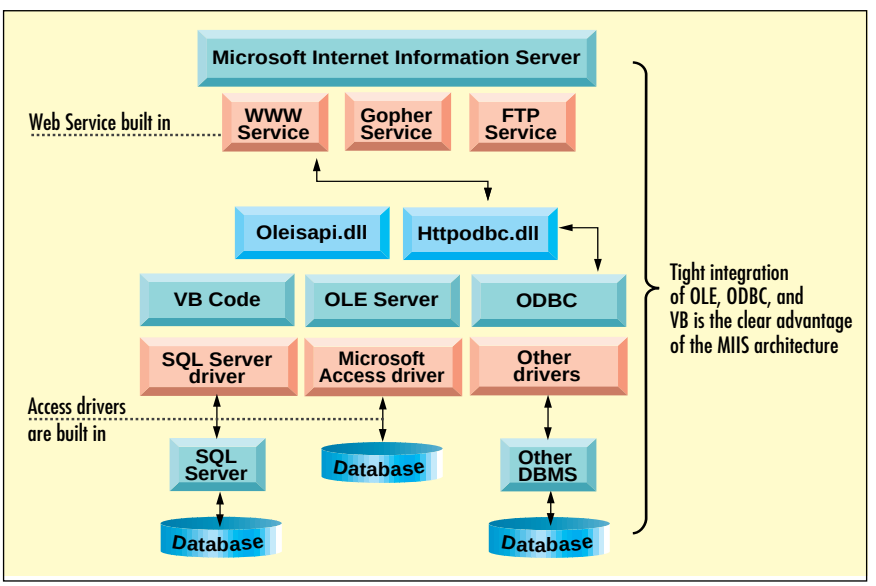
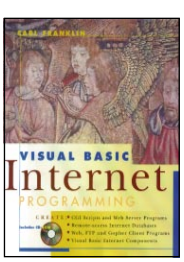


FIGURE 1 ***Architecturally Sound.*** *Microsoft is playing to its strengths by integrating OLE, ODBC, and VB capability into its Web server. The Internet Database Connector is a key component of easy Web development. Database access occurs from within the server itself using HTTPODBC.DLL to access any ODBC data source.*

Carl Franklin writes Q&A for VBPJ *and is co-owner of Carl & Gary's Visual Basic Home Page (http://www.apexsc.com/vb).* *Portions of this material may be used in Carl's book,* Visual Basic Internet Programming, *to be published by Wiley Computer Books. Material is reprinted with the publisher's permission. Reach Carl at CarlF@apexsc.com. Download sample code in OLEISAPI.ZIP on* VBPJ's *CIS, MSN, and Internet sites, and at Carl & Gary's VB File Archive on the Web (file:/ftp.apexsc.com/pub/cgvb/misc/oleisapi.zip).*



OLE Server DLL from the Web site.

While using the API instead of CGI scripts to directly access the database is a powerful capability, MIIS offers nothing like the base of shareware utilities and pragmatic maintenance features of more mature Unix servers. For example, the plethora of free programs available in the Unix world doesn't yet exist for Gibraltar. Nonetheless, these should evolve in time.

The first technology to consider is the Internet Database Connector. This approach makes it easy to post data for Web pages. The simplest way to understand the power of this architecture is by comparing how VB programmers had to use Common Gateway Interface coding only last winter. My article titled, "Spin Your Own Web Site," in the December issue of *VBPJ*, showed methods for posting data from a Web page using Windows CGI with Visual Basic and O'Reilly's WebSite HTTP server.

In brief, here's how it works. A user fills out your HTML form and clicks on the submit button. The server writes the entered data to an INI file and runs your VB application, passing the name of the INI file. The VB app then opens a database, executes a query using the specified data, and returns an HTML result string to the server, which passes it to the user.

However, using MIIS, the server does the hard work for you. A user fills out your HTML form and clicks on the submit button. The server reads an IDC script, a small text file that contains an ODBC data source, user name, user password, a SQL query, and an HTX file (the file extension is short for Extended HTML). The server then opens the database and executes the query using the specified data.

```
<html>
<body>
<title>Sample IDC Form</title>
<p>
This is a simple example of using
the Database Connector
that queries an ODBC data source
called Biblio, an Access
database (BIBLIO.MDB).
<p>
Enter the partial name of a
publisher below (ex: "Ran*"
might return "Random House")
<p>
<form action="http:/scripts/
test.idc" method="POST">
<input name=Search>
<input type=submit value="Query">
</Form>
<p>
</body>
</html>
```

LISTING 1 ***Data Access Made Easy.*** *The Internet Database Connector makes it easy to establish data access without using VB. This HTML code queries an Access ODBC data source called Biblio (BIBLIO.MDB).*

Lastly, the server sends back an HTML response that includes the resulting data that is displayed using the HTML Extension file as a template.

Excuse me, but where does Visual Basic fit in? So far, it doesn't. The process never leaves the server. But don't worry: you're VB skills aren't becoming obsolete. That's the second half of the story. You can easily integrate VB applications to your server because IDC works smoothly with the ISAPI.

Instead of firing up a VB executable to do the database work, the server calls HTTPODBC.DLL, which is already resident in memory, to query the database and return the results.

There is no down time in opening the database after the first call because HTTPODBC has its own connection management scheme (see Figure 1). The power of IDC lies in the use of three types of files:

- HTM Files. These are standard HTML format files, with no special consideration required except for the fact that they reference an IDC file rather than any other type of CGI script.
- IDC Files. Internet Database Connector scripts are small text files that contain, at minimum, an ODBC data source name, a

VB4

```
Sub ParseParameters(szRequest As String)

  Dim nPos      As Integer
  Dim szTemp    As String
  Dim szNextParam        As String
  'Replace the plusses with spaces
  szTemp = szReplaceString(szRequest, "+", " ")
  NumPairs = 0
  Do
    nPos = InStr(szTemp, "&")
    If nPos Then
      NumPairs = NumPairs + 1
      ReDim Preserve Pairs(1 To NumPairs)
      szNextParam = Left$(szTemp, nPos - 1)
      szTemp = Mid$(szTemp, nPos + 1)
      nPos = InStr(szNextParam, "=")
      If nPos Then
        Pairs(NumPairs).Parameter = Left$(szNextParam, nPos - 1)
Pairs(NumPairs).Data = _
Mid$(szNextParam, nPos + 1)
      Else
        Exit Do
      End If
    Else
      If Len(szTemp) Then
        NumPairs = NumPairs + 1
        ReDim Preserve Pairs(1 To NumPairs)
        nPos = InStr(szTemp, "=")
        If nPos Then
          Pairs(NumPairs).Parameter = Left$(szTemp, nPos - 1)
          Pairs(NumPairs).Data = Mid$(szTemp, nPos + 1)
        End If
      End If
      Exit Do
    End If
  Loop
End Sub
Function szReplaceString(szOriginal As String, szFind As String, _
  szReplace As String)
  Dim szTemp  As String
  Dim nPos As Integer
  szTemp = szOriginal
  Do
    nPos = InStr(szTemp, szFind)
    If nPos Then
      szTemp = Left$(szTemp, nPos - 1) & szReplace & Mid$(szTemp, _
        nPos + Len(szFind))
    Else
      Exit Do
    End If
  Loop
  szReplaceString = szTemp
End Function
```

LISTING 2 ***Parse Field/Value Pairs.*** *The ParseParameters routine copies all of the field/value pairs specified by the client into a Pairs array, for easy access. ParseParameters calls the szReplaceString function.*

SQL query, and a reference to an HTX file. Using variables in the query that represent the data passed by the form, you can allow users to supply data for the query.

• HTX Files. HTML extension files look like normal HTML files, but like the IDC file, they contain variables representing data fields. HTML Extension also defines flow control tags so you can return specified results if a certain condition is true, false, or whatever. The HTX file is merely a template—it does not represent the HTML sent back to the user. The server uses it to create the final output with the user data.

UNDERSTANDING FILE TYPES

Let's look at these file types, beginning with a simple HTML file that runs an IDC script (see Listing 1). The form for the HTML code consists of one field (Search) and a button that runs TEST.IDC in the \scripts directory off of the Web server root.

TEST.IDC contains:

```
Datasource: Biblio
Template: test.htx
SQLStatement:
+SELECT Name, Address FROM Publishers
+WHERE Name Like "<%Search%>"
```

The Datasource field specifies an ODBC data source (not the database name), while the Template field specifies an HTML Extension file to be used as the output template, and the SQLStatement field specifies a SQL Query. Although a few fields are optional, three fields are required.

First, you must have a plus sign in front of each line of the query. When the server reads this IDC file, it opens the ODBC data source called "Biblio" and executes the SQL statement:

```
"SELECT Name, Address FROM Publishers
WHERE Name Like <search term>"
```

The variable <%Search%> is replaced with the contents of the Search field on the form when the query is run. You can reference any of the HTML form's fields within your query in this way.

When the data comes back from HTTPODBC.DLL, it is merged with the TEST.HTX file:

```
<html>
<body>
<title>Query Results</title>
Here are the results of your query:
<p>
<%begindetail%>
<pre>
   Company Name: <%Name%>
      Address: <%Address%>
</pre>
<%enddetail%>
</body>
</html>
```

Remember, the HTX file is a template that the server uses to construct the final HTML output, and it does not represent the code that is actually sent. The <%begindetail%> and <%enddetail%> sections delimit where rows returned from the database will appear in the document. Columns returned from the query are surrounded by <%%>, such as <%Name%> and <%Address%>.

HTML RESERVED WORDS

When writing HTX files you have considerably more control than HTML alone gives you. For example, you can test conditions within the HTX file by using this syntax:

```
<%if condition %>
   HTML text
[<%else%>
   HTML text]
<%endif%>
```

where condition is of the form:

```
value1 operator value2
```

and operator can be one of these:

```
EQ       if value1 equals value2
LT       if value1 is less than value2
GT       if value1 greater than value2
CONTAINS if any part of value1
contains the string value2
```

For example, let's say that you want to return a special message if one of the names returned from the earlier query was "Acme":

```
<%begindetail%>
<%If Name EQ "Acme"%>
   <H2>
   Acme is having a sale on all
   size boxes of magnetic
   bird seed!
   </H2><p>
<%EndIf%>
<%enddetail%>
```

The operands value1 and value2 can be column names, one of the built-in variables (CurrentRecord or MaxRecords), an HTTP variable name, or a constant. When used in an <%if%> statement, values are not delimited with <% and %>. The CurrentRecord built-in variable contains the number of times the <%begindetail%> section has been processed.

The first time through the <%begindetail%> section, the value is zero. Subsequently, the value of CurrentRecord changes every time another record is fetched from the database. The MaxRecords built-in variable contains the value of the MaxRecords field in the Internet Database Connector file.

You can access parameters from the IDC file in the HTX file by prefixing the name of the parameter with "idc" and a period. For example, if you want to get the actual value the user entered in the Search field on the form, you can do something like this:

```
<%begindetail%>
<pre>
   Company Name: <%Name%>
      Address: <%Address%>
</pre>
<%enddetail%>
<%if CurrentRecord EQ 0 %>
   There are no matches with the term,
<%idc.Search%>
<%endif%>
```

Several variables used in HTML extension files give a lot of information about the environment and the Web client connected to the server. While I can't document all of them here, I've described some of the important ones (see Table 1).

ENTER VB AND ISAPI

Let's move on to integrating VB 4.0 applications in your server using the Internet Server Application Programming Interface (ISAPI). ISAPI was developed by Microsoft and Process Software Corp., of Framingham, Massachusetts, in 1995 to improve the performance of Windows EXE files used as CGI scripts. Both companies are promoting ISAPI as a free public standard. You can download ISAPI technical specifications from the Process Software Web site at http://www.process.com/news/spec.htp. ISAPI lets Windows programmers write their CGI scripts as in-process DLLs. The DLLs are loaded into the same address space as the server, and the entry points are called by their actual address, not by going through an import library and creating a new process. This method is faster than loading an EXE file, because a DLL stays loaded in memory, and because ISAPI DLLs reside in the same address space as MIIS.

VB4

User Tip

LOADING A STATIC COLOR CURSOR (CUR)

As with VB3, you can specify what mousepointer should be visible when the pointer is over a form or control. Additionally, VB4 permits you to provide your own cursors.

To use a custom cursor, set the MousePointer property of the form or control to vbCustom and load the cursor into MouseIcon. You can also use LoadPicture at runtime.

You don't even have to use a cursor file! VB will make a cursor for you out of an icon, but you'll have no control over the hot spot (always in the middle, for better or for worse) and few icons really look like cursors. Also, Win16 doesn't support color cursors, so if you load color icons (most of those supplied with VB), you'll get garbage because the colors will be more or less randomly converted to monochrome.

For this reason, always use cursors rather than icons for the MouseIcon property. Unfortunately, VB doesn't come with cursors, but you can create them or modify them from icon or bitmap images with Imagedit (in the tools directory on the VB4 CD).

—A. Nicklas Mali

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.

OLE Automation makes ISAPI technology available to any OLE Automation object, which could be created with tools other than VB4 (such as VC++). The key to calling methods in OLE Automation servers as your CGI interface is the OLEISAPI.DLL, written by David Stutz, one of the founding fathers of OLE technology at Microsoft. OLEISAPI.DLL lets you call OLE methods as CGI programs that execute and return HTML text to the client. OLEISAPI.DLL comes standard with MIIS.

But why use OLE Automation with VB when you can directly access data using the Internet Database Connector? It's the same reason that both Access programmers and Visual Basic programmers exist. With the IDC you can read and write to a database.

Tapping Visual Basic through OLE Automation gives you greater functionality. Evaluate both OLEISAPI and the Internet Database Connector and develop a thorough understanding of what both can do. This way, you will be prepared to decide what to use in a real-world situation.

Microsoft included a sample program to show how OLEISAPI works, but it wasn't particularly interesting. In lieu of that, I created an OLE server called CFOISAPI—which consists of CFOISAPI.VBP,

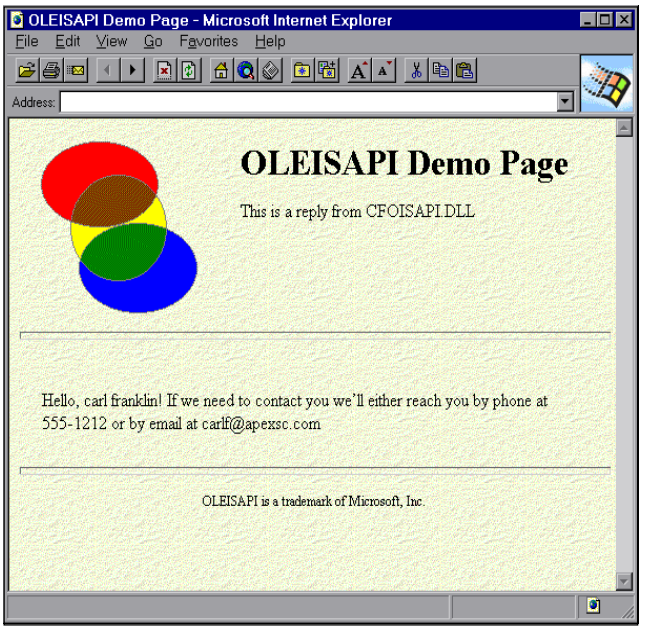


FIGURE 2 ***Craft a Standard Page.*** *You can automate creation of standard Web pages using code in Listing 4 and Listing 5 to create a sample form. To build this page, an OLE server calls the StandardPage string function.*

VB4

```
Public Sub Method_Get (Request As String, Response As String)
  Dim szName As String
  Dim szEmailAs String
  Dim szPhoneAs String
  Dim szReplyAs String
  Dim nIndex As Integer
  'Copy the returned data into the Pairs array
  ParseParameters Request
  'Set local variables
  For nIndex = 1 to NumPairs
    Select Case UCase$(Pairs(nIndex).Parameter)
      Case "NAME"
        szName = Pairs(nIndex).Data
      Case "EMAILADDRESS"
        szEmail = Pairs(nIndex).Data
      Case "PHONENUMBER"
        szPhone = Pairs(nIndex).Data
    End Select
  Next
  'Create a reply message
  szReply = "Hello, " & szName & "! If we need to " & _
    "contact you we'll either reach you by phone " & _
    "at " & szPhone & " or by email at " & szEmail
  'Send the reply
  Response = "Content-Type: text/html" & _
    vbCrLf & vbCrLf & "<html><head><title>" & _
    szReply & "</body></html>"
End Sub
```

LISTING 3 ***Calling ParseParameters.*** *One of the advantages of MIIS is the ability to call ParseParameters from within your OLE server. This code retrieves all the parameters sent by the client as discrete strings using the Pairs array.*

CFOISAPI.BAS, and CFOISAPI.CLS, which I will demonstrate.

OLEISAPI-callable methods must take two string arguments. The first argument is the request data passed from the server. The second argument is a response string—an HTML reply from your application created by your method. Consider this object definition, which could be called using OLEISAPI.DLL:

```
DLL Name:      OLETEST.DLL
Class Name:    TestClass
Public Sub Method_Get (Request As _
   String, Response As String)
   Response = "Content-Type: _
      text/html" & vbCrLf & vbCrLf & _
      "<html><head><title>" & _
      "<H1>Here is the request " & _
      "string you sent: " & _
      "</H1>" & _
      Request & vbCrlf & _
      "</body></html>"
End Sub
```

Within your HTML form, you might call this method like this:

```
<form action="oleisapi.dll/_
   OLETEST.TestClass.Method_Get"_
    method="POST">
First Name: <input _
   name="FirstName"><p>
Last Name: <input _
   name="LastName"><p>
<input type="submit" value="Click _
   here">
</form>
```

This is a simple form with a submit button and two text boxes, one for the first name and one for the last name.

When the user clicks the button, the server accesses OLEISAPI.DLL and passes OLETEST.TestClass.Method_Get. The <form …> tag defines the action that occurs when you click the submit button like this:

```
<form action = "[full path to _
   oleisapi.dll]/ [DLL Name].[Class _
   Name].[Method]"
   method = "[Access method: either _
      POST or GET]">
```

The request is passed as a string:

```
[Field1]=[Value1]&[Field2]=[Value2]…
```

Each field/value pair is separated with an ampersand character. For example:

```
FirstName=Joe&LastName=Schmoe
```

Within the fields themselves, spaces are encoded into plus signs. So the string:

```
Joe Schmoe
```

Variable	Description
CONTENT_TYPE	The content type of the information supplied in the body of a POST request.
HTTP_ACCEPT	Special case HTTP header. Values of the Accept: fields are concatenated, separated by ", "; for example, if these lines are part of the HTTP header: accept: */*; q=0.1 accept: text/html accept: image/jpeg the HTTP_ACCEPT variable will have a value of: */*; q=0.1, text/html, image/jpeg
PATH_INFO	Additional path information, as given by the client. This is the trailing part of the URL after the script name but before the query string (if any).
QUERY_STRING	The information following the ? in the URL that referenced this script.
REMOTE_ADDR	The IP address of the client.
REMOTE_HOST	The host name of the client.
REMOTE_USER	This contains the user name supplied by the client and authenticated by the server.
REQUEST_METHOD	The HTTP request method.
STRING	
SERVER_NAME	The server's host name (or IP address) as it should appear in self-referencing URLs.

TABLE 1 ***HTTP Variables For HTML Extension Files.*** *These are the most important HTTP variables that can be accessed from within an HTML extension file. MIIS gives you full CGI header access and more. Key variables include content type, special header information, path information, query specifications, remote object names, and server name.*

VB4

```
Public Sub Method_Get(Request As String, _
  Response As String)
  Dim szName As String
  Dim szEmailAs String
  Dim szPhoneAs String
  Dim szTitleAs String
  Dim szBody As String
  Dim szDescrip          As String
  Dim szFooter  As String
  Dim nIndex As Integer
  'Copy the returned data into the Pairs array
  ParseParameters Request
  'Set local variables
  For nIndex = 1 To NumPairs
    Select Case UCase$(Pairs(nIndex).Parameter)
      Case "NAME"
        szName = Pairs(nIndex).Data
      Case "EMAILADDRESS"
        szEmail = Pairs(nIndex).Data
      Case "PHONENUMBER"
        szPhone = Pairs(nIndex).Data
    End Select
  Next
  'Create the title
  szTitle = "OLEISAPI Demo Page"
  'Create a reply message
  szBody = "Hello, " & szName & "! If we need to _
    contact " & "you we'll either reach you by phone _
    at " & szPhone & " or by email at " & szEmail
  'Now the footer.
  szFooter = "<center><font size=1>OLEISAPI is a " & _
    "trademark of Microsoft, Inc.</font " & _
    "size=1></center>"
  'Create the description
  szDescrip = "This is a reply from CFOISAPI.DLL"
  'Send the reply
  Response = StandardPage(szTitle, szBody, _
    szDescrip,szFooter, "lyellow.gif", "logo.gif", _
    "http://my.url.com")
End Sub
```

LISTING 4 ***Automate Web Page Creation.*** *Calling StandardPage is as easy as executing this code. The StandardPage function returns the HTML text for a standard Web page that contains all the specified elements. You can modify this function to fit your Web page design criteria.*

VB4

```
Function StandardPage( szTitle As String, _
  szBodyText As String, Optional Description _
  As Variant, Optional Footer As Variant, _
Optional BackgroundImage As Variant, _
Optional LogoImage As Variant, _
Optional LogoURL As Variant) As String
  'This is just a work string used internally
  Dim szWorking As String
  'Flags for the optional arguments
  Dim nBackGroundImage   As Integer
  Dim nLogoImage         As Integer
  Dim nLogoURL           As Integer
  Dim nDescription       As Integer
  'Set the argument flags
  nBackGroundImage = Not IsMissing(BackgroundImage)
  nLogoImage = Not IsMissing(LogoImage)
  nLogoURL = Not IsMissing(LogoURL)
  nDescription = Not IsMissing(Description)
  'Start with the HTML header and title
  szWorking = "Content-Type: text/html" & _
    vbCrLf & vbCrLf & "<html><head><title>" & _
    szTitle & "</title><body"
'Use background bitmap?
If nBackGroundImage Then
  szWorking = szWorking & " background =" & _
    BackgroundImage & ">"
Else
  szWorking = szWorking & ">"
End If
'Logo image?
If nLogoImage Then
  's there a link in the image?
  If nLogoURL Then
    szWorking = szWorking & _
      "<a href=" & LogoURL & ">"
  End If
  'Here's the image itself
  szWorking = szWorking & "<img src=" & _
    LogoImage & " align=left " & "hspace=20 border=0>"
  'If there's a link then close the anchor.
  If nLogoURL Then
    szWorking = szWorking & "</a>"
  End If
```

LISTING 5 ***Manage Web Page Elements.*** *The StandardPage function controls construction of your pages, including the display of headers, footers, titles, background graphics, and text. Once elements are specified, the HTML text builds your standard Web page. You can modify this function to fit your Web page design criteria.*

becomes:

```
Joe+Schmoe
```

Before you can access this data in your class, you must parse the request string into field/value pairs. I wrote a routine called ParseParameters (included in CFOSAPI.CLS) that does exactly that (see Listing 2). ParseParameters requires these declarations in your class:

```
Private Type ParameterType
   Parameter   As String
   Data  As String
End Type
Dim Pairs() As ParameterType
Dim NumPairs   As Integer
```

Call ParseParameters passing the request string, and all the Field/Value pairs are copied into the Pairs array. The NumPairs member contains the number of fields.

If you had a form with three text fields on it, such as Name, EmailAddress, and PhoneNumber, you would use ParseParameters to extract these three pieces of data when the Method_Get method is invoked (see Listing 3).

Once you have the data from the form, you can run a query against a database and return the results in the Response string. I'll skip over the database part now and show you the next piece of what I consider to be required code for any OLEISAPI-aware object.

StandardPage is a function that returns a string containing the HTML for a standard document. In other words, you can set up a template for your pages so they all have the same look and feel.

Instead of having to custom program the HTML for each and every page, you can just create the critical elements with code, and let the StandardPage routine format it to your standard using this syntax:

```
HTML$ = StandardPage(Title$, _
   TextBody$[, Description$] _
   [, Footer$][, BackgroundImage] _
   [, LogoImage][, LogoURL])
```

The first two arguments (Title$ and TextBody$) are required. All the other arguments are optional. Keep in mind that you can alter StandardPage to suit your needs.

Title$ is the title of the page. This is usually displayed on the TitleBar of the Web browser and also at the top of the page. TextBody$ designates the main content of the page—it should be already encoded as HTML text. Description$ is a short description of the contents of the page that will appear at the top under the title. Footer$ is an HTML string that appears at the very bottom of the page (copyright information, text links, things like that).

BackgroundImage is the name of an image file that will be used as the background. This should be a small image (composed of no more than 50 by 50 pixels) because it gets tiled automatically by the Web browser.

LogoImage is another small bitmap you can use as a logo. It appears in the upper left-hand corner of the page. If you specify a URL following the LogoImage argument (LogoURL), clicking on that image will jump to the site at the specified URL. Execute the code in Listing 4 to build your standard HTML Web page (see Figure 2). The StandardPage function itself is the key to crafting this page (see Listing 5).

OLEISAPI AND REMOTE SERVERS

The folks at Microsoft told me that you can't access remote objects with OLEISAPI now, but this feature may be added in the future. You can, however, create a remote object in your local OLEISAPI object to access the remote object indirectly. For example, consider this object definition:

```
DLL Name:      OLETEST.DLL
Class Name:    TestClass
Private RemoteObject As New
RemoteClass
Public Sub Method_Get (Request As _
   String, Response As String)
   'Access the remote object here and
   'return the results
   Response = RemoteObject.GetResults_
      (...)
End Sub
```

In the absence of direct support, you can use your VB class to access the remote object.

I'm impressed with the technology in MS Internet Information Server—I reserve final judgement on performance, however. As of this writing, I've seen only the second release, and thus cannot fairly comment on MIIS performance under heavy load.

However, I think the potential for this server is impressive, and that your worries about doing powerful CGI scripting could be long gone. ×