Visit the book's companion Web site at www.wiley.com/compbooks/martiner

Visual Basic® Programmer's Guide to Web Development

William Martiner

Create powerful Web applications using:

- Visual Basic 5, ActiveX controls, ActiveX documents, and VBScript
- Visual InterDev and Active Server Scripting to create dynamic Web content
- ADO and Visual InterDev Data Access Components

Visit the book's companion Web site at www.wiley.com/compbooks/martiner

Visual Basic® Programmer's Guide to Web Development

William Martiner

Create powerful Web applications using:

- Visual Basic 5, ActiveX controls, ActiveX documents, and VBScript
- Visual InterDev and Active Server Scripting to create dynamic Web content
- ADO and Visual InterDev's Data Management Tools for fast, robust data access

Visual Basic Programmer's Guide to Web Development

William Martiner

WILEY COMPUTER PUBLISHING

John Wiley & Sons, Inc.

New York • Chichester • Weinheim • Brisbane • Singapore • Toronto

Executive Publisher: Robert Ipsen

Editor: Robert Elliott

Assistant Editor: Pam Sobotka

Managing Editor: Mark Hayden

Electronic Products, Associate Editor: Mike Green

Text Design & Composition: SunCliff Graphic Productions

This text is printed on acid-free paper.

Published by John Wiley & Sons, Inc.

Library of Congress Cataloging-in-Publication Data

ISBN: 0-471-19382-8

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents

Laying the Framework: HTTP and HTML

Together, the HTTP protocol and the HTML markup language form the basis of communication and interface design on the World Wide Web. The Hypertext Transfer Protocol (HTTP) governs the way that hypermedia (i.e., Web pages) are transferred over the Internet and the way in which the HTTP server and the HTTP client interact. Hypertext Markup Language (HTML) is the common formatting language of the World Wide Web, defining how a Web page will appear to the browser by "marking it up" with a series of tags.

In a real sense, the World Wide Web is nothing more than a set of standards. So, although HTTP and HTML are both relatively limited and simple, these standards are really what the World Wide Web *is*. Just as the HTTP protocol is the basis of the Web, the Post Office Protocol (POP) and the Simple Mail Transfer Protocol (SMTP) form the standards of e-mail; files are sent and received on the Internet using File Transfer Protocol (FTP) and people are able to send and receive messages from newsgroups because they adhere to the Network News Transfer Protocol (NNTP). Each of these standards has its positive and negative features. However, from the standpoint of the World Wide Web developer, the abilities of the standards are not as important as the fact that they are universally *understood*. It's not as if you have to decide whether to use these standards. The simple fact is that if you want to communicate to the largest audience, then you have to play by the rules.

The rules in the case of development on the World Wide Web are HTTP and HTML. This is why an understanding of both of these standards is crucial to beginning to work the Web. As you will soon see, it is this simple protocol

and markup language that will either act as the building blocks or will be used as the end result of the all of the technologies that frame the World Wide Web.

Hypertext Transfer Protocol (HTTP)

HTTP has only been around since 1989 when Tim Bernes-Lee proposed the protocol. Since then, it has become the standard for all things Web. The HTTP draft that was current at the writing of this chapter defines HTTP as "an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. It is generic, stateless, object oriented protocol. . . ."

This definition is qualified with a date because new proposals are currently being developed for the standards used on World Wide Web. These standards are constantly being revised and expanded by the members of the Internet Engineering Task Force (IETF). This group is a collection of academic and industry leaders who are assembled to guide the standards used on the Internet. If you are curious about the newest proposals for these standards, you can find them at the W3C (World Wide Web Consortium) Web site at www.w3.org.

However, there are good reasons why you should not be alarmed with this constant revision. The first is that whenever a standard changes, the governing body of the World Wide Web takes great pains to provide backward compatibility. You can also take comfort in the fact that changes to HTTP and HTML move very slowly because of the nature of their users. As long as there is diversity in the browsers being used to view World Wide Web content, these standards will be driven by consensus and use rather than the fast-paced proposal of new academic standards.

The HTTP protocol basically defines a client-server model that is designed for the most rapid and efficient means of delivering World Wide Web content. Like any transaction between a server and a client, the HTTP transaction can be most easily understood as a conversation in which the client asks the server for something and where the server gives what was asked to the client.

As stated in the earlier definition of the protocol, HTTP is a *stateless* protocol. This means that the conversations between the HTTP client and the HTTP server are succinct and non-persistent. That is, when a server returns what a client has asked, then the connection is broken and there is no memory of the exchange. It's sort of like asking a drunk college roommate to borrow some money because when the exchange is over, your roommate may (with some luck) have no memory of the transaction. Since there is normally no persistent

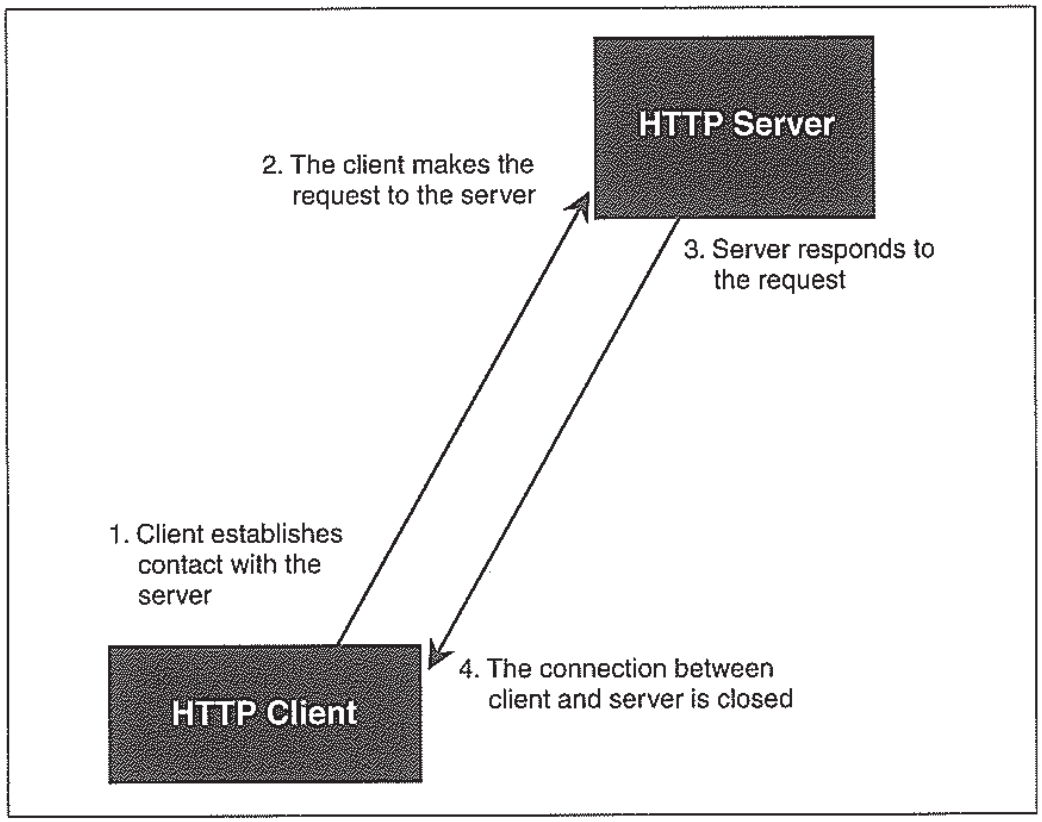

Figure 3.1 The HTTP Conversation.

data between requests to the server, HTTP's statelessness makes it difficult to program database applications that need to maintain some connection to their data in order to guard against problems like multiuser contention.

The HTTP Conversation

The conversation between the HTTP server and the HTTP client goes something like this (see Figure 3.1):

1. The client establishes a contact with the server using the Internet address of the server and port number that the server application is "listening to" (That's right; it's just connecting to a *socket*.)
2. After the connection is made, the client makes a request for a resource (file or program) on the server. This resource is typically a HTML file or some type of gateway program. In addition to the request, the client sends the server some information about itself in the form of header fields. This request is made using one of the HTTP *methods* that will be discussed later in this chapter.

3. When the server receives this request, it responds with information about the state of the transaction (success, fail, or something in between) and the data requested by the client. The data is transmitted as a stream of 8-bit characters called *octets*. This format allows for the transmission of virtually all forms of data, whether the data that is being transferred is an image, a HTML file, or an executable program.
4. After the server makes its response, it closes the connection and does not retain any information about the transaction.

These steps are taken for *each* request that the client makes of the server. This means that if a HTML document contains several different images, Java appletts, and ActiveX controls, then these steps must be taken for each object. So, although the HTTP server is fast in getting data to the HTTP client, once a connection is made, a separate connection must be made for each resource being requested. This unlucky behavior of HTTP is blamed for much of the slow response time of the Web. However, it is also this behavior that allows for the web server to be able to satisfy the requests of many concurrent clients because, in a rather socialist manner, everyone gets a little piece of his or her request and everyone is made to wait.

Getting in Touch with the Server The first step of the HTTP client-server conversation is the client opening a connection with the server. In a HTTP transaction, the server is identified just like a server in any sockets application; you can connect using a combination of an address (or name) and a socket. The subject of addressing becomes somewhat complex on the World Wide Web, as there are an unfathomable number of addresses to which a client may connect.

To solve this problem, the World Wide Web uses a form of address called a Universal Resource Identifier (URI). And, a common form of URI used on the Web is a Universal Resource Locator (URL). The difference between a URI and a URL is that the URL includes information about how the resource should be received from the server. For example, the following URIs are simply addresses of objects on the Wingspan Web server. There is no information in the URI about the protocols being used to transfer these files.

```
//207.87.44.198/Bill/Xmass.htm
//207.87.44.198/Bill/Xmasstree.gif
```

In contrast, the following URLs contain both the address and the protocol that should be used to retrieve the file from the server:

```
Http://207.87.44.198/Bill/Xmass.htm
Ftp://207.87.44.198/Bill/Xmasstree.gif
```

The URL, Itself The HTTP URL is a format that can identify any object on any server on the Internet:

```
Http://<IP address>/[<port>]/[<path>][?<query information>]
```

The first part of the URL identifies what type of protocol should be used to retrieve the object. In the case of a World Wide Web document, this would be HTTP. The second part is the machine name (IP address) of the machine upon which the desired object exists.

When a request to a named URL (like www.WingSpan-T.com) is made, the request is first handled by a Domain Name Server. This DNS takes the name requested by the client and resolves it with the IP address of the server before a formal request is made. The result of this is that the World Wide Web is a much more friendly place to visit because Web sites have memorable names like www.microsoft.com rather then being identified with an IP address like 270.87.44.198.

The protocol and the IP address are the only mandatory portions of the URL because if an URL does not specify a file name, the HTTP server will respond with a default file. In most circumstances, this file will either be named INDEX or DEFAULT. However, the person maintaining the HTTP (Web) server can set the default file name to anything that he or she wishes.

Also, because you are connecting to a socket when you request something from a HTTP server, you must specify not only a machine name and a resource, but also a port. However, it's not necessary to include the port of the URL, either. This is because the default port of a HTTP server is port 80. So, since both resource and port are specified by default, the user can enter www.MSN.com into the top of the browser's screen and he or she will be connected to the Microsoft Network's Web server at port 80 and sent its default file.

The last portion of the URL can contain query information that is being sent to a resource. We will be exploring the implementation of this later in this chapter.

Making Your Order When you go to the deli, very few people simply step to the counter and say "tunafish" without first seeing how the tuna looks and without specifying the right type of bread, cheese, condiments, and tomatoes (if they look good, and never after November). It's the same way when you request something from a Web server; not only do you want the content of the Web page, but you might want to limit the types of files that can be sent (let's say you have a text-only browser). You might want to tell the server how you got to a specific page. Or, you might not even want what you are asking for if it hasn't been updated since the last time you made a request.

For all of these reasons, when you send a request to a server for a resource, you are not only sending the name of the file that you would like to have returned, but you are also giving the HTTP server information about the client asking for the resource. Therefore, a request goes to the server not only with the name of the resource, but also with information about the client doing the asking.

This information is sent to a server in a HTTP request header containing several fields that make the request to the server and tell the server details about the order. The most important thing that a HTTP request header will send is a HTTP method. For example, when you type an URL into the top of a browser and hit enter, a GET method is sent to the server. There are several different HTTP methods that can be used to make a request:

- **GET:** This method instructs the server to return the resource specified by the client. If the resource specified by the request is an application, then the server will return the output of the application rather that the application file, itself.
- **HEAD:** The HEAD method is like GET, but will return only the header information of the requested resource. Often, the HEAD method is used by spiders and other search agents to return general information about a Web site.
- **POST:** This method will send data to a specified URL and is often used with HTML forms to send information to the server. Typically, this method is used with some type of gateway application to handle the data POSTED to it appropriately.
- **PUT:** PUT writes data sent by the client into the specified URL. Few HTTP servers implement the PUT method.
- **DELETE:** Deletes the resource at the specified URL. Thankfully, this is not currently used by many HTTP servers.
- **LINK:** Links one existing object to another object. This method is not used by many HTTP servers.
- **UNLINK:** Removes link information inserted by the LINK method. Like its counterpart, this method is not currently implemented in many HTTP servers.

Aside from the method that the request header sends to the server, there are also a number of header fields sent in a HTTP request. These header fields provide the server with information about the specifics of what the client wants and what type of a client is making the request. One of the types of header fields that can exist in a request header is the *General Message* header fields that can be inserted into either request headers that are sent to the server by the client or response headers that are sent to the client by the server. These general message header fields are:

- **DATE:** Specifies the date and time that the request was made.
- **PRAGMA:** Designed for passing implementation-specific information to servers. This field can also be used to tell a proxy server to always fetch the source from the actual server and not from its cache.
- **FORWARDED:** Can be used to trace the message through machines other than the client and the server. This could be used to trace the route of a message through proxy servers.
- **MESSAGE-ID:** Used to uniquely identify the message.

Aside from these general HTTP header fields, messages will carry other types of header fields. Each of these fields will tell the server more about the capabilities of the client that made the request. (Table 3.1 highlights some of the common header fields that a client can use to add detail to the request that it is making of the server.)

- **ACCEPT:** A number of fields sent to the server to notify it of what data types and the size types that can be accepted by the client. Most modern browsers will send */* in this field to designate that they will accept all types.
- **AUTHORIZATION:** Used to present some type of information to the server to bypass security and encryption. If this is not required by the server, then this field will be absent.
- **FROM:** Used if the client application wishes to provide the server with information about its email address. Typically, HTTP clients will not send this header field. However, web crawlers and other robots should provide this field to designate the email address of the person who started it.
- **IF-MODIFIED-SINCE:** Used to provide a conditional GET. If the document being requested has not been changed since the specified date, then the server will not send the object. If the date sent is in some invalid format or if the date sent is later than the server's date, then this field is ignored by the server.
- **REFERRER:** The object from which the request from the server resource was made.
- **MIME-VERSION:** The MIME protocol version that is used to work with files of different types. The current version is 1.0.
- **USER-AGENT:** Information about the client (i.e., browser) that is making the request.

Getting Your Stuff As soon as the server receives a request, it interprets the method being used by the request and begins to process its response. Just as with the request message from a client, the response message from the server includes information about the

Table 3.1 HTTP Status Codes

Status Code	Reason Phrase and Meaning
200	*Success*: Response has been made without error
202	*Accepted*: Request has been made successfully. However, the results of the process are unknown (not finished processing).
204	*No content*: Request was processed. However, there is no new information to send and the client application should continue to display the current document.
301	*Moved Permanently*: The object has been moved. In this case, the server should include the new location in the header of the response. Some browsers will simply connect to the new address without notifying the user.
304	*Not Modified*: A request was made using an If-Modified-Since header field designating a date later than the date that the document was last modified. When this is the case, the document is not sent.
400	*Bad Request*: A syntax error was made by the client application requesting an object on the server.
401	*Unauthorized*: Security on the server required that the request included an Authorization field. If this status is returned, then it must also return a special *WWW-Authenticate* header. This header is basically a challenge to the client. If the client is to pass this security, then it must provide authentication information before it may pass.
403	*Forbidden*: Just like it sounds, the client has asked for a resource that the server will not provide.
404	*Not Found*: The request was made for an URL that is not known to the server.
500	*Internal Server Error*: The server crashed and cannot process the request.
501	*Not Implemented*: A legal request was made, but the method that was used for the request was not supported by the server.

Status Code	Reason Phrase and Meaning
502	*Bad Gateway*: In an attempt to satisfy the client's request, the server attempted to gain access to a resource from another server, or through a gateway. This secondary server or gateway did not return a response.
503	*Service Unavailable*: The server is too busy to respond to the request. In this case, the server may respond with this code, a reason phrase and a *retry-after* header that can be used by the browser application to govern the time that it should wait before trying to request the resource from the server again.

message in the form of header fields. These *response header* fields are included to inform the client of the server's status, the type of information that it is returning, the time at which its response was returned, and other data about the message.

The first line of the response header is the status line and usually follows the following format:

```
<HTTP Version> <Status-Code> <Reason-Phrase>
```

The status code returned from the server is a three-digit code that can be used by the client to determine how the server was able to handle its request. The first digit of this code indicates the broad category of the status of the response message. The numbers following the first provide further detail about the status of the server's response. Status codes are split into four basic types:

1. If the number starts with a 2, then the request was processed successfully.
2. If the status code begins with a 3, then the request was redirected.
3. A status code starting with 4 indicates that the request from the client was in some way erroneous.
4. Finally, if the status code begins with a 5, then the server has failed to fulfill the request.

Along with the status code, a reason phrase will be returned with the message to provide some human readable explanation of the status code. The browser application will usually display this reason phrase to the user when a problem is encountered with the response (i.e., a status code is returned designating some type of failure). Table 3.1 details some common server responses, their associated reason phrase, and the reason for the code.

Aside from the status code that the HTTP server will return, there are several other header fields that will be returned to the client. Many of these header fields are only used under the circumstance that the server has passed a status code other than OK (i.e., 200). However, others fields exist to provide the browser application with information about data about to be sent. The following list details some of the Response Header Fields that can be sent to the client:

- **STATUS CODE /REASON PHRASE:** Indicates the status of the response (success/fail or somewhere inbetween) and a human readable explanation of the status.
- **DATE:** The date and time upon which the data was sent to the client.
- **LAST_MODIFIED:** Just as it sounds, this field will give the date and time when the resource was last modified. Note that this is always given in Greenwich Mean time. (After all, what is the local time of the World Wide Web?)
- **SERVER:** Information about the server from which the data originated.
- **CONTENT_TYPE:** Designates the type of data that will be sent in the response.
- **RETRY_AFTER:** This field is returned if the server was too busy to handle the response (status code 503). The information returned in this field is used by the client to determine how long it should wait before retrying the server.
- **WWW_AUTHENTICATE:** This header field is sent when the server's security requires that the client pass some authentication (status code 401). It is basically a challenge to the browser to return authentication data.

Of course, this is not all of what the server returns to the client. The bulk of the traffic on the World Wide Web is made in the form of response messages. This is because, aside from the header information that is returned to the client, the server also includes *Entity* information. Essentially, entity information in a response message is the contents of a resource on the HTTP server. In other words, this is the stuff that you asked for in the first place.

Entity information is also sometimes used in a request message. This happens when a HTML form uses a POST method to pass data to some server resource like a server-side script or a gateway program. In this instance, the information that the user input into the form's controls is submitted to the server as the request message's entity.

However, more typically, this information is sent from the server to the client as a text file formatted with HTML tags or some type of information that the client application has requested (e.g., .avi files, .gif files, Java appletts, . . .). Later, there will be an examination of exactly how HTML is used to format these documents being delivered as the entity section of the HTTP response message.

What follows is an example of what a server's response message may look like. Notice that the header information comes first and allows the browser to know about the message before the bulk of it is processed. The remainder of the response message is the contents of a HTML document that will serve as the message's entity.

```
HTTP/1.0 200 Document Follows
Date: Thu, 26 Dec 1996 01:35:15 GMT
Server: NCSA/1.4.1
Content-type:  text/html
Last-Modified: Wed, 25 Dec 1996 12:59:15 GMT
Content length: 344
<html>
<head>
<title> Now What?? </title>
</head>
<body background = xmasstree.gif>
<h1>Boxing Day???</h1>
<p>What exactly does one do on boxing day?
<p>It happens to be my wife's birthday, but what do the rest of you poor slobs
do? <br>
<FORM METHOD="Post" Action="Boxing.ASP">
<textarea name="Boxing" Rows=25 Cols=50></textarea>
<input type=submit value="Tell me, please."
</form>
</body>
</html>
```

Goodbye The last step in the HTTP transaction is simple. Once the server sends the appropriate header fields and entity contents to the client, the connection is dropped. Remember that HTTP has been defined as a stateless protocol. So, once the transaction is finished, the conventional HTTP server will forget that it ever happened and will move quickly to the next request.

HTTP Summary

I know that this protocol material makes for a fairly dry and difficult way to make entry into this new technology. I feel your pain and I have tried to keep it short.

However, in a real sense, Web-based programming is simply the different creative implementations of this messaging technology. So, HTTP (or future derivatives of this protocol) is

the enabling technology of development on the Web. As you will see when you begin to actually build Web-based applications, a basic understanding of this type of protocol is crucial.

Hypertext Markup Language (HTML)

What you see in Figure 3.2 is HTML in action. The first part of this chapter dealt with the way that the World Wide Web data is requested and transferred. The final section of this chapter will deal with the creation of the documents that usually make up the entity section of the HTTP response message. A full explanation and exposition of style in the creation of Web pages is certainly beyond the scope of this book. Therefore, it is not my intention to deal with the height and breadth of HTML, but simply to give you a foundation from which to begin your work. If you want more information on HTML and how to use it, please visit

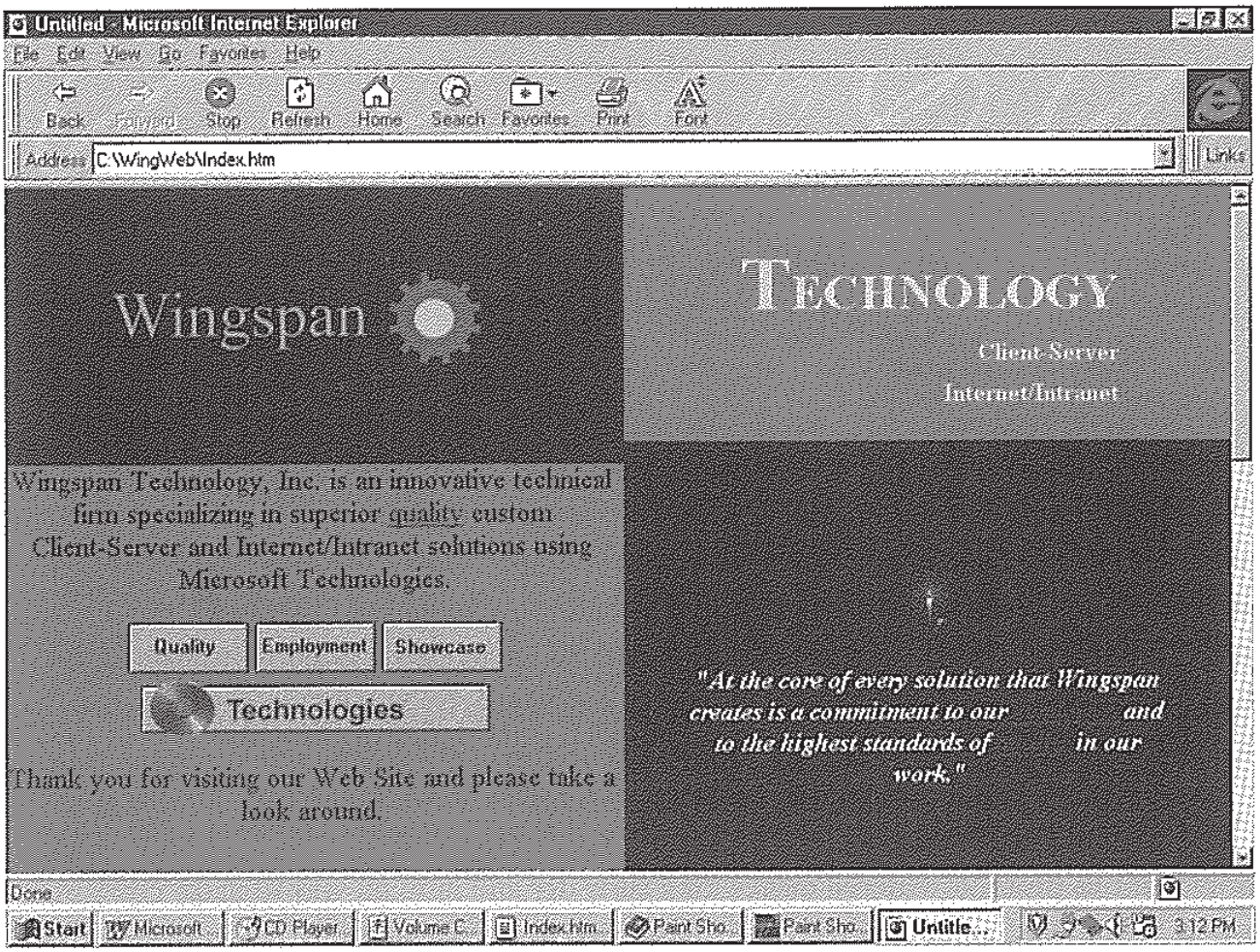

Figure 3.2 HTML in action.

the official HTML resource at the W3C Web site (HTTP://WWW.W3.org/pub/www/markup/). This site can provide you with guides, tutorials, and style manuals.

Hypertext Markup Language or HTML is a structured markup language that defines the various components of a Web page. As a markup language, HTML must contain embedded instructions about the appearance of the document to be presented.

These formatting instructions are implemented as tags that can be used by inserting them at either end of the text to be formatted. For example, the tags that I would use to underline text are **<U>** and **</U>**. Tags come in pairs; the first tag begins the section to be formatted and the last tag ends the formatting. So, if I wanted to underline a sentence, I would enclose the sentence between the **<U>** and **</U>** tags:

```
<u>Here is an example of an underlined sentence.</u>
```

This is a rather simple example of using HTML. Believe me, it is capable of much more than simple underlining. HTML is able to format rich user interfaces using tables, frames, and even forms.

When an HTML is sent to a browser application as the entity section of a HTTP response message, it is the job of the browser to parse and interpret the HTML code. Different browsers will view HTML code in different ways. This is why much of the skill involved in creating HTML documents is understanding the ways that popular browsers will interpret the code that you write. So, the HTML code that you write is more like giving formatting hints to the browser application than it is really like creating a user interface with Visual Basic forms.

That is true, if you are in the unlucky situation of creating either an Internet application or an intranet application that will be used in conjunction with a variety of browsers. However, if you are creating an intranet application that will be viewed by only one type of browser (e.g., Internet Explorer), then you are in luck. If you only write HTML for one type of browser, then you can know with relative certainty that your design will be viewed as it was intended. Further, you will be able to take advantage of the specifics of the HTML that is implemented in the browser. For example, if I were writing an application that I knew was only going to be viewed by Microsoft Internet Explorer, I would be at liberty to use frames, style sheets, and other niceties of the browser at present.

At the most basic level, HTML documents consist of a HTML element, a document head, a document body, heading elements, and formatting tags. These tags work together to specify

that the text is in HTML format, to specify titles and document body sections, and to format the text in the browser. HTML formatting tags are used within a hierarchy to describe the format of the document and the style of the text. These tags are case-insensitive and are almost always paired in a **<Format> </Format>** syntax.

The HTML Document's Basic Structure

Any HTML document must begin and end with the **<HTML> </HTML>** tags. These tags indicate to the browser that what is enclosed in them is HTML code. This may be necessary for a number of reasons, including when a browser requests a file that does not have a name that specifies it as a HTML document (e.g., webpage.txt) and in circumstances where the server is able to send documents that exist in other markup languages (e.g., SGML).

After the **<HTML>** tag, there is usually a document head section contained in **<Head > </Head>** tags. This section serves as a header and is able to provide the browser with information about the document, itself. This section can be used to provide the browser with a title for the document that will be displayed separately from the rest of the HTML text, typically in the browser application's title bar:

```
<HTML>
<HEAD><TITLE>This is my document's title!</TITLE><HEAD>
<HTML>
```

The next element included in a HTML document is the body element, defined by **<Body></Body>** tags. This section is where the user-viewed text of the HTML document is placed.

Inside the body of the document, formatting logical headings can be defined. That is, these special tags do not define text in a static format (like 12 Point Courier), but rather, they leave the formatting up to the browser (see Figure 3.3). Headings are called logical formats because every browser will format the higher level headings such as Heading 1 (**<H1>**) and Heading 2 (**<H2>**) more prominently than the lowest level headings like Heading 5 (**<H5>**) and Heading 6 (**<H6>**). So, the following HTML code will be displayed in a similar format in all browsers:

```
<html>
<head> <title>This is an example of headings<title></head>
<body>
<h1>Heading 1</h1> <h2>Heading 2</h2> <h3>Heading 3</h3>
<h4>Heading 4</h4> You get the idea…
</body>
</html>
```

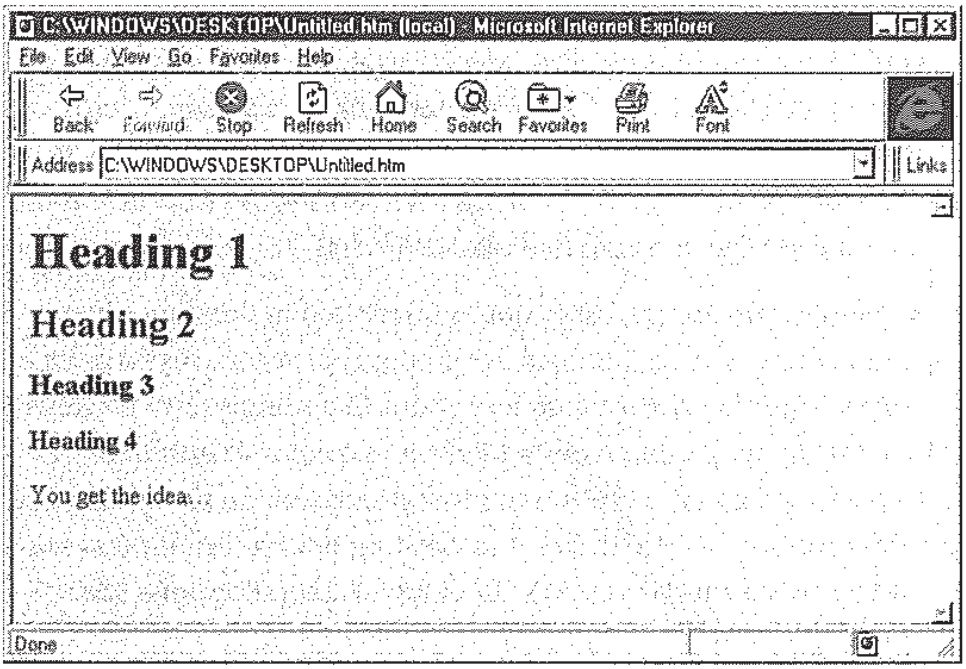

Figure 3.3 The HTML example code as interpreted by Internet Explorer.

Not only are you able to make use of the heading tags to format the text of your document, but also you can use a wide assortment of other tags that allow you to explicitly format font and size of font, insert horizontal lines, designate background colors, and include pictures. HTML is quickly becoming a rich formatting language that, through the use of its large and growing library of formatting tags, is capable of constructing elegant matter for the Web.

If you need more particulars about using HTML to create attractive documents, the Web is brimming with HTML guides. Once again, the resources that are available to you at HTTP://WWW.W3.org/pub/www/markup/ are exhaustive. Also, for your convenience, this book includes a quick reference to HTML in Appendix A.

However, if you really want to learn HTML quickly, spend more time surfing. When you come upon a page that is particularly attractive for some reason, find out why. If you are using Netscape Navigator, go to your **View** menu and select your **View Source** option to see the underlying HTML. If you are using Microsoft Internet Explorer, simply right-click on the page and select **View Source** from your popup menu. You will be amazed by how quickly you learn just by being nosy.

So, with HTML you can develop richly formatted documents that use tags to format the way that the text appears. Whoopee.

The fact is that if you can use Microsoft Word, Excel, PowerPoint, or Access, you already can create dynamite looking HTML by simply upgrading to Microsoft Office 97. The Office 97 package allows you to create HTML as easily and as seamlessly as you can now print from these applications. Further, if you have an eye for design and the time to invest in learning Microsoft FrontPage, you will be able to develop Web content in a WYSIWYG interface that will rival anything that you see on even the most professional Web sites.

That's OK because, as a developer, you are probably not as interested in creating nice documents as you are with creating a workable interface for your applications. If you are a database developer (like myself) you probably care a lot less about including pretty pictures in your HTML than you are worried about feeding your addiction to grids and intuitive, useful forms. As you know, to provide a robust solution to most programming situations, you need to work with code. These tools may be very helpful in creating interfaces, but you will still have to know your stuff when you want to make use of more advanced options like creating these interfaces-on-the-fly.

This is not to say that these advanced Web site designing tools are not extremely helpful. Believe me, it is much easier to create an attractive HTML document using the intuitive, WYSIWYG interface of Visual InterDev or the advanced features of other tools like HomeSite than it is to continue using Notepad (my personal favorite up to just last year). These tools allow for fast formatting, error checking, link checking, and a lot of ready-done programming that will allow you to be much more productive in much less time than would be otherwise possible. It's just that it has become so easy to create attractive HTML, that it doesn't really warrant discussion in the context of a book like this one.

That is why I am going to skip a lot of the particulars of how you format text, create horizontal lines, and include pictures on your Web pages, and include them in the appendix at the back of the book. At a really basic level, most of the formatting work that you do is just using the design tool of your choice to put the right tags in the right place.

However, this view of HTML starts to break down when you begin to work with advanced features. In other words, to be able to provide real solutions in this technology, things are going to get a little more complicated.

Creating Links between Documents

Probably one of the first things that you are wondering about if you have never worked with HTML before is how links between documents are created. This ability to link regions of text or images to other related documents on the World Wide Web is one or the chief powers of HTML. Once again, this is a type of special formatting done through the use of tags.

In this case, the tags are *Anchor* tags that are added into a HTML document through the use of **<A>** and **</A>** tags. Inside the initial tag you specify the document to which you want to link entering the **HREF=** parameter and the URL of the document. Then, you can enclose in those tags either the text that will serve as the link or a reference to a graphic element. In the following example, the first line of the body will use the text "Link to Wingspan" as its link and the second line will use an embedded image as a link:

```
<HTML>
<HEAD><Title>Link Example</Title></Head>
<Body>
<A HREF="http://www.wingspan-t.com">Link to Wingspan</A><br>
<A HREF="http://www.microsoft.com/ie"><IMG SRC="ie_static.gif"></A>
</Body>
</HTML>
```

When these links are displayed in the browser, they will appear as in Figure 3.4. When either of these links is clicked upon, the browser application will send a HTTP request message to the appropriate server using a GET method. When the contacted server responds with its HTTP response message, the browser will interpret the new document's HTML and display it again in the browser.

Creating Tables

The use of HTML tables is becoming almost universal. As you would expect, tables are used to display data in a grid. However, because of the nature of formatting text and graphics in HTML, they also have wide use in the creation of user interface. The reason for the latter is

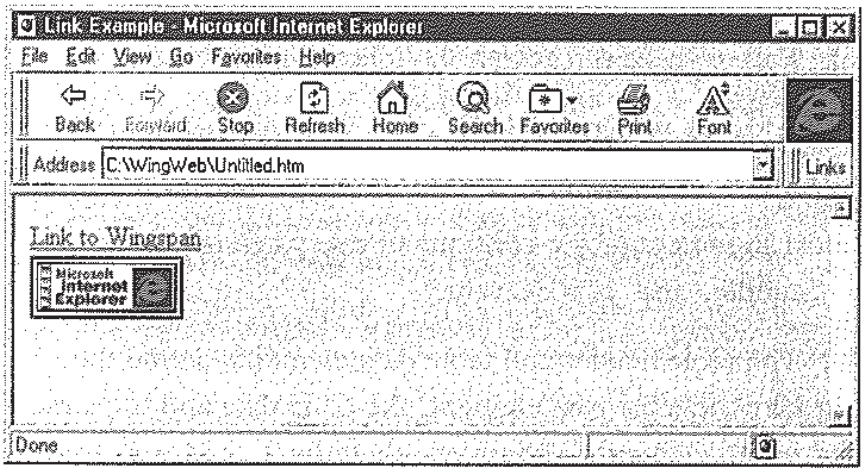

Figure 3.4 Text and a graphic element used as links.

that the exact placement of text and graphical elements in an HTML document is problematic. The physical placement of text in a HTML file doesn't correspond exactly to what the user will see because it's the browser that decides where the elements are shown, not really the HTML.

That is, it's not the HTML as long as the placement of elements is outside of a table. Because the cells of a table can be used to designate exactly where a certain element should be placed, tables can be used for both the organization and display of data, and of graphical information on the screen.

The table is a relative newcomer to HTML, being introduced only in HTML 3.0. This new feature provides the developer with a number of table elements that allow for the creation of simple, yet effective tables. Tables are defined through the use of four separate tags:

1. **<Table></Table>:** This is the outermost element of the table that surrounds the other elements used in the creation of a table. The table tag can have the following attributes:
    - **Border:** This attribute designates the thickness of the border that is drawn around each cell of the table. If this attribute is set to 0, then no cell borders will be drawn.
    - **Cellpadding:** Defines the thickness of the space left in-between the cell border and the content of the cell.
    - **Cellspacing:** Like the Border attribute, this attribute describes the space left between the individual cells in a table.
    - **Width:** Describes the desired width of the table. This measurement can either be made in pixels or as a percentage of total space.
2. **<TR></TR>:** This element defines the table row, which can contain table headers (**<TH>**) and table data (**<TD>**) elements. This element can accept the following attributes:
    - **Align:** This attribute defines the alignment of the content of every cell in the row. The Align attribute can accept "right," "left," and "center" as values.
    - **Valign:** Defines the vertical alignment of the contents of every cell in the row. Valign accepts "bottom," "middle," and "top" as values.
3. **<TH></TH> and <TD></TD>:** The table header and table data elements define the individual cells in a table's row. Like the table row element, the table data and the table header elements have align and valign elements. However, when these attributes are included with the table header or the table data elements, the alignment is only defined

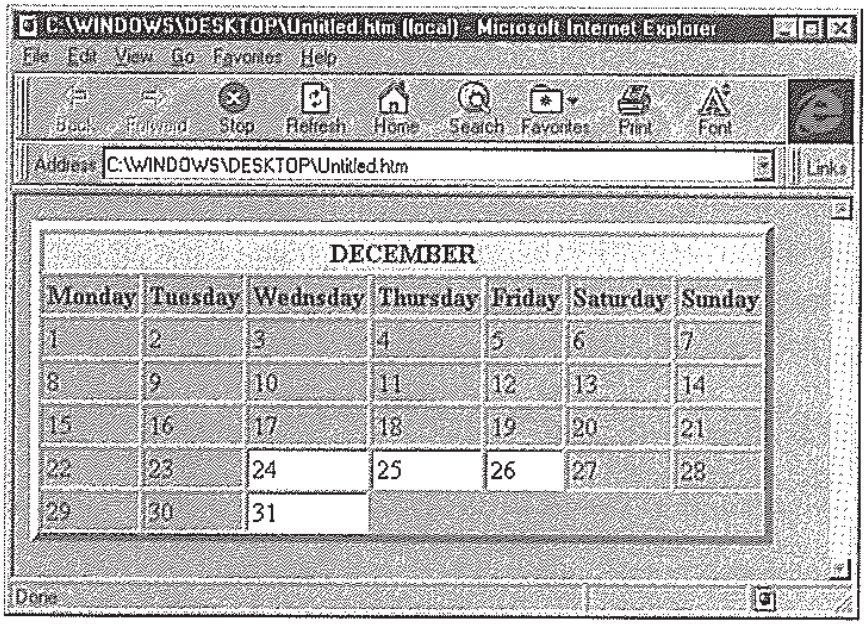

Figure 3.5 The Internet Explorer's rendition of the HTML.

for the individual cell. Aside from align and valign, the table header and the table data elements have the following attributes:

- **Colspan:** This attribute defines the number of column widths spanned by the individual cell. The default value for this attribute is 1.
- **Rowspan:** Like the Colspan attribute, the Rowspan attribute defines the number of rows that the cell will cover. The default value for this attribute is also 1.

4. **BGColor:** Defines the background color of the cell. The colors can either be specified by RGB (red-green-blue) triplet values or (more simply) by the name of the color you would like in quotes. The supported colors for this attribute are : "red," "green," "blue," "black," "white," "gray," "silver," "yellow," "aqua," "navy," "lime," "teal," "fuchsia," and "olive."

The following HTML defines a simple table that forms a calendar for the month of December. Notice that the holidays will appear in the table as white cells. Figure 3.5 shows the Internet explorer's presentation of this HTML.

```
<TABLE CELLPADDING=2 WIDTH=50 BORDER=5 ALIGN="LEFT">
<TR>
<TH ALIGN="Center" VALIGN="Middle" BGCOLOR="Yellow" COLSPAN=7>DECEMBER</TH>
</TR>
<TR Align="center">
```

```
      <TH>Monday</TH><TH>Tuesday</TH><TH>Wednsday</TH>
      <TH>Thursday</TH><TH>Friday</TH>
      <TH>Saturday</TH><TH>Sunday</TH>
</TR>
<TR VALIGN="top">
      <TD>1</TD><TD>2</TD><TD>3</TD><TD>4</TD>
<TD>5</TD><TD>6</TD><TD>7</TD>
</TR>
<TR VALIGN="top">
      <TD>8</TD><TD>9</TD><TD>10</TD><TD>11</TD>
      <TD>12</TD><TD>13</TD><TD>14</TD>
</TR>
<TR VALIGN="top">
      <TD>15</TD><TD>16</TD><TD>17</TD><TD>18</TD>
      <TD>19</TD><TD>20</TD><TD>21</TD>
</TR>
<TR VALIGN="top">
      <TD>22</TD>      <TD>23</TD><TD BGCOLOR="white">24</TD>
<TD BGCOLOR="white">25</TD><TD BGCOLOR="white">26</TD>
<TD>27</TD><TD>28</TD>
</TR>
<TR VALIGN="top">
      <TD>29</TD><TD>30</TD><TD BGCOLOR="white">31</TD>
      <TD></TD><TD></TD><TD></TD><TD></TD>
</TR>
</TABLE>
```

Creating Forms

By far, the most important HTML elements to the serious Web developer are those involved in the creation of forms. If you remember from earlier in this chapter, World Wide Web documents can be used to send information to HTTP (Web) servers. The HTML elements that are responsible for calling HTTP methods are those involved in the creation of forms.

The topmost level of a form is the **<Form></Form>** element, itself. You can place any of the control tags inside these two tags to create control on your form. The form element has three attributes that govern what the form does when it is submitted and the type of data that it will send to the server.

Action The action attribute specifies the URL of the server resource to which the data filled into the form should be submitted. Typically, this is some type of *gateway* program. A gateway program is one that can receive information from the HTTP method and do something with the information. Typically, gateway programs are either Perl scripts, CGI (common gateway interface) programs or, in the case of a site being installed on a Microsoft Internet Information Server, it could be some type of ISAPI (Internet server application programming Interface) program typically written in C. However, this book will concentrate on the newest generation of gateway interfaces, in particular, the creation of Active Server Scripts.

Method This attribute defines the HTTP method used to submit data to the server. If you use a GET method, then the information typed into the form's controls by the user is submitted to the server as part of the URL in the HTTP request header:

```
Get vb-bin/input/pslobs.pl?text1=Bill&text2=Martiner HTTP/1.0
```

In contrast, if the POST method is used, then the information filled into the form's controls is sent to the server as the HTTP request method's entity (not in its header information, but in its body).

There is an advantage to using the POST method to send large amounts of information because transmitting information in the body of the HTTP message is more efficient for most servers. However, if you wish for your users to be able to put the results of some server process (e.g., the results of a search) into their collection of "favorites" or "bookmarks," then you should use the GET method because all of the information sent to the server is transmitted with the URL and can be stored as a string to be returned to at some later time. (This is why you can store the results of a search in many Web searching utility sites.)

Enctype This rarely used attribute can specify the type of data sent via the form's method. If this attribute is not included, then it will pass the default value of **`application/x-www-form-urlencoded`** which is what you will want (unless you are doing something very specialized) anyway.

So, the first element in the creation of a form typically looks something like this:

```
<FORM METHOD=POST ACTION="www.wingspan-t.com/vb-bin/scam.asp">
</FORM>
```

The Input Element

Now that you have specified the resource that the form will be acting upon and the HTTP method that it will be using, you can make use of the **<Input></Input>** element to design

your form's user interface. Although this is just one element, when it is combined with its **Type** attribute, it can be used to create textboxes, submit and reset command buttons, invisible textboxes, checkboxes, image controls, password-type textboxes, and radio buttons. The **<Input>** element's common attributes are as follows:

- **Name:** Assigns the variable name specified to be equal to the contents of the control. For example, if I created a textbox named txtFirstname, then when the user typed information into the textbox and submitted the form, the server would receive the string `txtFirstname=WhatWasEntered` in the client's HTTP request message.
- **Type:** Designates the control's type. The various types are as follows:
  - **checkbox:** Will create a checkbox on the form. A HTML checkbox is by default *off*. When the form is submitted, only checkboxes with a value of *on* are submitted. This type will take a **Checked** attribute that will turn the checkbox to *on*.
  - **radio:** Will create a radio button (i.e., option button) type of control. Radio elements can be made mutually exclusive by assigning them the same name. If this is done, then, when one radio button in the group is *on*, all the others will be set to *false*. Like the checkbox type, this type also supports a **Checked** attribute. A **Value** attribute must be added to a radio button. This value will reflect the data to be passed to the server when the form is submitted.
  - **text:** A normal, single-line Visual Basic type textbox is created. This type takes an additional **Size** attribute which defines the maximum size that is to be displayed, an **Align** attribute that can be used to define the alignment of the text input to the control, and a **Maxlength** attribute that specifies the maximum number of characters that can be input into the control.
  - **hidden:** Like a textbox, but it is not displayed to the user. This type of control is helpful for passing hidden information to the server.
  - **password:** This is another control like a textbox. However, the text entered into this element will be obscured by asterisks.
  - **image:** The use of this attribute has the exact same effect as using the **Ismap** attribute of the **<IMG>** tag (used for inserting images). When this element is clicked upon, the form is submitted immediately, sending the X, Y coordinates to the form's **Action** attribute. The gateway program on the server can use these coordinates to judge where the user has clicked on the image and respond appropriately.

- **reset:** When this attribute is used, a command button that will clear the user's input into the form will be created. This type of element will take a **Value** attribute that will define the command button's text.
- **submit:** This attribute will create a command button that will submit the data typed into the form's controls to the server using the form's HTTP method.

So, through the use of the **<Form></Form>** and the **<Input></Input>** elements, a robust form interface can be built for your Web-based applications. The following is an example of the use of these elements to create a simple form interface and Figure 3.6 shows the document.

```
<FORM METHOD=POST ACTION="www.wingspan-t.com/vb-bin/creditscam.asp">
<H1>Be Careful!</H1>
Please input your first name:<INPUT TYPE="TEXT" NAME="txtFirstName" VALUE=""
SIZE=10><BR>
Please input your last name:<INPUT TYPE="TEXT" NAME="txtLastName" VALUE=""
SIZE=20><BR>
Please input your Social Security number:<INPUT TYPE="TEXT" NAME="txtSS#"
VALUE="" SIZE=11 MAXLENGTH=11><BR>
I trust you absolutely with this information. <INPUT TYPE="CHECKBOX"
NAME="chkTrust" ><BR><BR>
Are you really sure that you want to do this?<BR>
Yes<INPUT TYPE="RADIO" NAME="optSure" VALUE="yes"> No<INPUT TYPE="RADIO"
NAME="optSure" VALUE="no"><BR><BR>
<INPUT TYPE="SUBMIT" NAME="cmdOK" VALUE="OK">
</FORM>
```

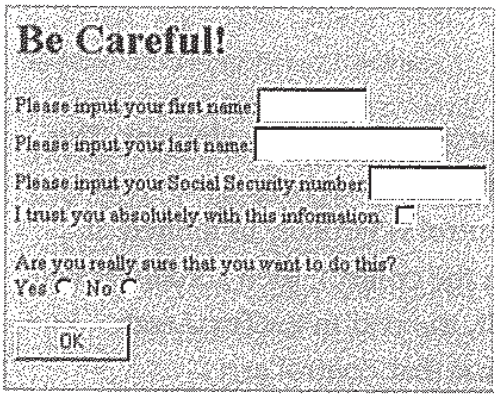

Figure 3.6 Display of the document shown in the HTML.

The Select Element

Aside from the types of the **<Input>** element, there are two other types of HTML form controls. The first of these is the **<Select>** element that can be used to create a Visual Basic type listbox. The **<Select>** element is used to allow the user to select one or many predefined values. The value that is currently selected when the form is submitted will be passed to the server as the value of the control. The attributes of this form element are:

- **Name:** As with the **<Input>** element, this attribute defines the variable name that will be associated with this control.
- **Size:** This attribute is used to define the number of values that the listbox will display. If this attribute is given the value of 1, then the control will act as a drop-down listbox.
- **Multiple:** If this attribute is present, then multiple values can be selected from the listbox. The default value for this element is single value selection.

The available values of the listbox are defined as **<Option>** elements that exist within the **<Select></Select>** tags. **<Option>** elements can only be text; no markup can be included. The attributes of the **<Option>** element are:

- **Value:** This attribute defines a paired value of the **<Option>** element's text. For example, if you had a listbox that showed your users a list of products, but stored the users' selections in your database as SKU numbers, you would place the name of the product between the **<Option></Option>** tags and you would designate the SKU number as the **Value** of the **<Option>** element.
- **Selected:** A *true* value for this attribute will define the **<Option>** element as the currently selected option when the form is displayed.
- **Disabled:** A *true* value for this attribute will define the **<Option>** element as disabled. A disabled element will be displayed in the list as grayed or faded.

The TextArea Element

The final type of HTML defined form control is the **<TextArea></TextArea>** element. This element is used to create a form control that allows the user to input a large amount of text. The amount of text that can be input into a **<TextArea>** control is virtually unlimited. If more text is input into the control than can be displayed in the control, then scrollbars will appear to allow easy navigation through the text. By default, the text in a **<TextArea>** control will not wrap. The attributes that allow you to format that control are:

- **Name:** As with every other form element, the **<TextArea>** control has a mandatory **Name** attribute that can be used to define the variable name that will carry the content of the control to the server.
- **Rows:** This attribute is used to specify the height of the **<TextArea>** control in rows. This attribute is mandatory for the creation of the control.
- **Columns:** Like the **Rows** attribute, the **Columns** attribute defines the dimensions of the control. The **Columns** attribute will define the width of the control in columns.
- **Wrap:** Defines the way that text will "wrap" in the control. If you define this attribute to be *off*, then word wrapping will be completely disabled. When its value is set to *virtual* then new line characters are inserted to achieve the effect of word wrapping on the client machine. However, these line characters will not be returned to the server when the form is submitted. Finally, when the value of the **Wrap** attribute is set to *physical* then line characters will cause word wrapping both in the client interface and in the data sent to the server.

In this final document, a combobox and a textarea control are added to the form.

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="www.wingspan-t.com/vb-bin/creditscam.asp">
<H1> Be Careful!H1>
Please input your first name:
<INPUT TYPE="TEXT" NAME="txtFirstName" VALUE="" SIZE=10><BR>
Please input your last name:
<INPUT TYPE="TEXT" NAME="txtLastName" VALUE="" SIZE=20><BR>
Please input your Social Security number:
<INPUT TYPE="TEXT" NAME="txtSS#" VALUE="" SIZE=11 MAXLENGTH=11><BR>
I trust you absolutely with this information. <INPUT TYPE="CHECKBOX"
NAME="chkTrust" ><BR><BR>
Are you really sure you want to do this? Yes<INPUT TYPE="RADIO" NAME="optSure"
Value="yes"> No<INPUT TYPE="RADIO" NAME="optSure" Value="no"><BR><BR>
Now that I have this information, how would you like to be abused?<BR>
<SELECT NAME="cboAbuse">
     <OPTION>Apply for cards and trash your credit rating
     <OPTION>Bury you in marketing
     <OPTION>Find out private information
```

```
        <OPTION>Sell it to those more devious than myself
</SELECT><BR><BR>
Do you have any comments on our service?<BR>
<TEXTAREA NAME="taComments" ROWS=2 COLS=50>
</TEXTAREA><BR><BR>
<INPUT TYPE="SUBMIT" NAME="cmdOK" VALUE="OK">
</FORM>
</BODY>
</HTML>
```

Figure 3.7 displays the HTML document seen earlier, complete with the recent additions. Notice how the **<Select>** element is displayed as a drop-down list because only one row is being displayed.

Summary

HTTP and HTML define the foundations of development on the World Wide Web. Now that you have been introduced, you will see them (in one form or another) throughout the remainder of this book. This is not to say that these two technologies are perfect, easy to work with, easily made secure, efficient, or wonderfully powerful. However, since the Internet is built on commonality, it is not always the best technologies that become dominant; it is simply the ones that are agreed upon. Really, it is not all that different from the rest of the

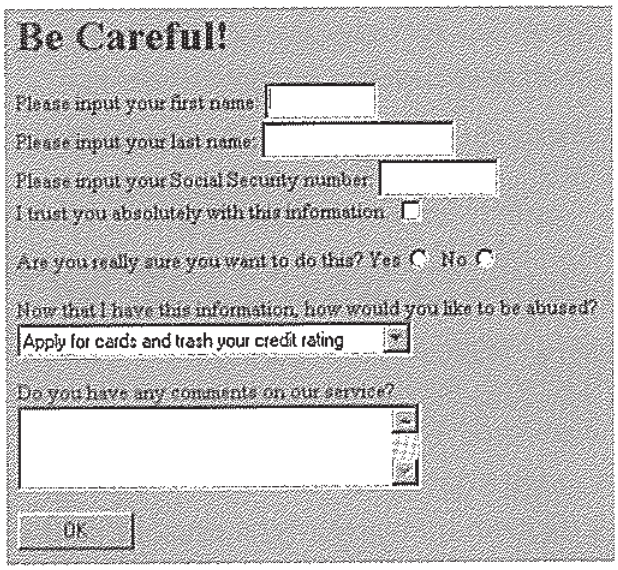

Figure 3.7 The finished HTML form.

world of development, in which technology that becomes dominant is the one that becomes most popular.

Some of these problems with the HTTP/HTML technologies have been solved through the adoption of new standards. Other problems will be solved through the creation of smart servers and gateway applications that will work within the boundaries of the current HTTP and HTML specifications, but will provide documents most appropriate for the browser making the request. Another way of dealing with the problems of HTTP and HTML is to use the ability of these technologies to serve ActiveX objects, developed in traditional development environments, like Visual Basic 5.0.

The Microsoft technologies that have been created to assist in these solutions deal extensively with Visual Basic syntax and other Visual Basic paradigms. You will be able to move from development using HTML and HTTP to an event-driven paradigm using event procedures for controls written in Visual Basic Script. The HTTP server can be programmed using a combination of HTML and Visual Basic Script. Finally, ActiveX components can be inserted into HTML documents and served to the browser. Through the use of these technologies and others being developed by Microsoft, we are beginning to see a model of the World Wide Web that is not so centered around the formats and protocols, but rather one that is focused on the creation of solutions. But, I guess that's why you're reading this book.

Expanding Limits with ActiveX Components

Coming from a community that has a long history with the use of custom controls, the inclusion of ActiveX controls on the Web may seem like an obvious solution to the limits of HTML.

However, it represents sweeping change for the Internet because, when you use an ActiveX component in a Web page, you change the way that the World Wide Web works. HTTP is still the basis of the Web, but since this protocol can be used to transmit anything, you no longer need to be tied to the sole use of HTML. In fact, ActiveX represents such a fundamental change in the way that the World Wide Web works that some see it as the death of HTML and the birth of a more active, useful Internet.

It also represents a paradigm shift away from the Internet's traditional mass of development tools, towards the tools and thinking that the Visual Basic development community espouses. In Figure 4.1, you can see the inclusion of two familiar entities from Visual Basic: the RichTextBox control and the Grid Control. The inclusion of these types of components in Web pages and the ability to use Visual Basic Script to set properties, call methods, and react to events should make you feel right at home.

Since Visual Basic can now be used to create these components and script their interactions, the model of the new World Wide Web is not built only on the clumsy implementations of HTML, but also on the robust and familiar environment of Visual Basic. More important than this is the fact that through the use of this technology, the Visual Basic community is given access to one of the largest promises of development on the World Wide Web—the powerful

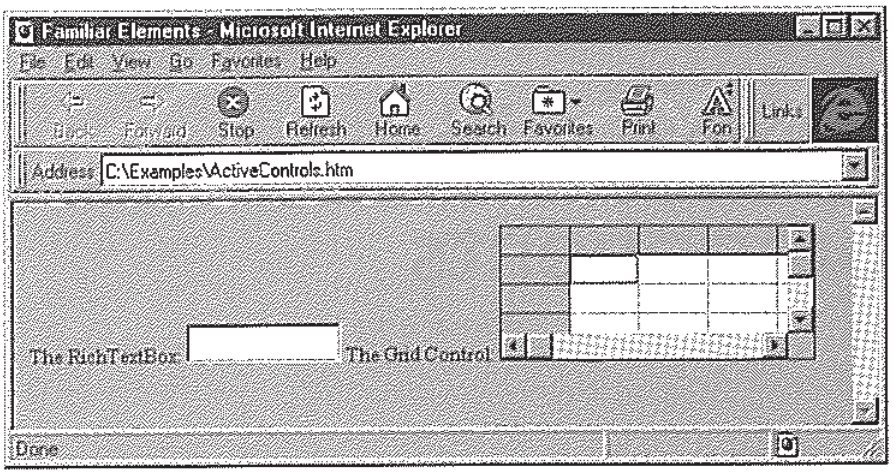

Figure 4.1 The RichTextBox and Grid Control .ocx controls at play on the Web.

ability to maintain an application from one location. If you can imagine making a bug fix on one machine and that fix seamlessly exporting itself to 10,000 of your users, then you have a basic understanding of what this technology is going to be able to do for your application development.

The <Object> Tag

You are able to integrate ActiveX components into your HTML through the use of the **<Object></Object>** tag. This tag is used not only to deliver ActiveX objects but also Java applets and even images. Like any of the complex HTML tags, the **<Object>** tag takes many different attributes. It is through these attributes that the component is downloaded and given the properties that you desire on your Web page. The following code is an example of the use of the **<Object>** tag and its attributes to embed a Timer ActiveX control into a Web page, enable it, and set its **Interval** property to 200:

```
<OBJECT ID="timer1" CLASSID="clsid:59CCB4A0-727D-11CF-AC36-00AA00A47DD2"
CODEBASE=
"http://activex.microsoft.com/controls/iexplorer/ietimer.ocx#Version=4,70,0,1161"
ALIGN=middle>
     <PARAM NAME="Interval" VALUE="200">
     <PARAM NAME="Enabled" VALUE="True">
</OBJECT>
```

The attributes that can be used with the **<Object>** tag are as follows:

Table 4.1 Align Values

Align Value	What it does
Baseline	Aligns the bottom of the object with the baseline of the surrounding text.
Center	Centers the object between the left and right margins.
Left	Aligns the object with the left margin and causes the surrounding text to wrap along the right side of the object.
Middle	The middle of the object aligns to the baseline of surrounding text.
Right	Aligns the object to the right margin of the page.
TextBottom	The bottom of the object aligns with the bottom of the text.
TextMiddle	The middle of the object aligns with the middle of the surrounding text.
TextTop	The top of the object aligns with the top of the surrounding text.

- **ALIGN:** Assigns the placement of the object on the Web page. The values that can be set to this attribute are detailed in Table 4.1.
- **BORDER:** Defines the width of the border around the object.
- **CLASSID:** When any ActiveX component is added to the system, it must first be added to the Windows Registry and given a global unique identifier (GUID). As is defined under the Common Object Model (COM), GUID can be used to instance its corresponding ActiveX class. In this way, the process of instancing an ActiveX class in the context of the browser is just like using an OLE component on a Visual Basic form. When the browser requires the creation of an ActiveX class, the Windows Registry is consulted and the ActiveX component's class will be instanced.

  As the browser receives information about an ActiveX component embedded in a Web page, it first uses the CLASSID attribute in conjunction with a call to the **CoGetClassObject** API function to query the Windows Registry. If it has already been installed and registered, the browser will use the CLASSID attribute to instance the ActiveX component. Only if this attribute does not correspond to a GUID in the Windows Registry will the Internet Download Service be employed to download, validate, and install the control.
- **CODEBASE:** As was previously stated, if the client system has not yet installed the component, then the browser must first download and install the component before it is

used. For this reason, when you include an ActiveX component in a Web page, you must also specify the **CodeBase** (i.e., URL) from which the browser application can download the control.

If you look at the HTML example that preceded Table 4.1, you will see that the **Codebase** is defined as follows:

```
CODEBASE=
"http://activex.microsoft.com/controls/iexplorer/ietimer.ocx#Version=4,70,0,1161"
```

So, when the browser application receives an object tag that includes an ActiveX component that it does not yet have, it calls the **CoGetClassObjectFromURL** API function and passes it the information about the location of the control.

- **CODETYPE:** Used to ensure that the viewing application is compatible with the object to be downloaded.
- **DATA:** This property can be used to specify a file containing information that is used by the object. For example, if a Multimedia control was being instanced, then the control's **Data** property would specify a .avi (video) file.
- **DECLARE:** Specifies that the object is not to be instanced; only its class is to be declared in the context of the page. This can be used when creating cross-references to the object later in the document or when using the object as a parameter to another object in VBScript.
- **HEIGHT:** This is used to define the height of the object as it will appear on the Web page.
- **HSPACE:** Defines the space to keep between the right and left borders of the visible area of the object.
- **ID:** This attribute defines the name of the object that will be used in scripting the object. This **ID** value is very much like the **Name** property of .ocx controls in Visual Basic as it is the name of the control that is used when properties are set or methods are called in code.
- **NAME:** If the object exists within a **<Form></Form>** block, it will be submitted to the server in the HTTP method if it is supplied with this attribute. By using this property, you are able to use ActiveX controls rather than HTML controls in your on-line forms.
- **SHAPES:** This attribute can be used to assign anchor tags over certain areas of the embedded object.

- **STANDBY:** Defines the text that will appear on the Web page as the object is downloading or instancing.
- **TYPE:** Specifies the Internet Media Type.
- **VSPACE:** Defines the space to keep between the top and bottom borders of the visible area of the object.
- **WIDTH:** Defines the width of the object.

PARAM Tags

PARAM tags allow you to set the properties of the object that you have instanced in your HTML. The PARAM tag is used in conjunction with its NAME attribute to set the value of a certain property of the embedded control. The following code uses the PARAM attribute of the Microsoft Label control to format a label (Figure 4.2).

```
<OBJECT ID="MyShinyNewLabel" WIDTH=116 HEIGHT=24
 CLASSID="CLSID:978C9E23-D4B0-11CE-BF2D-00AA003F40D0">
    <PARAM NAME="Caption" VALUE="Impressive!">
    <PARAM NAME="Size" VALUE="3069;635">
    <PARAM NAME="FontName" VALUE="Braggadocio">
    <PARAM NAME="FontHeight" VALUE="225">
    <PARAM NAME="FontCharSet" VALUE="0">
    <PARAM NAME="FontPitchAndFamily" VALUE="2">
</OBJECT>
```

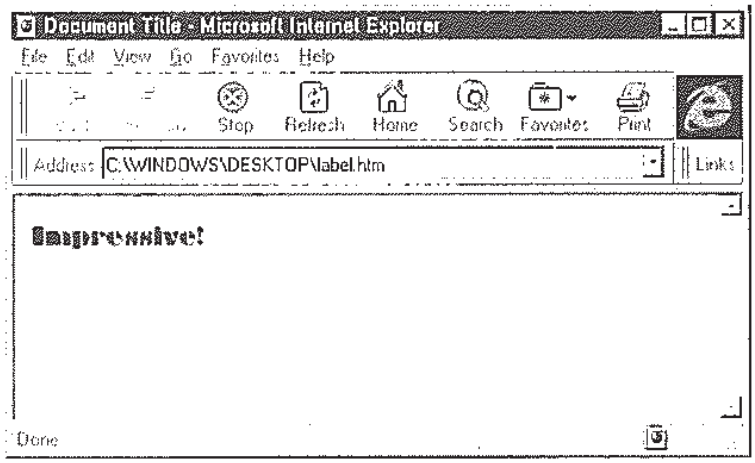

Figure 4.2 The described instance of the Microsoft Label control.

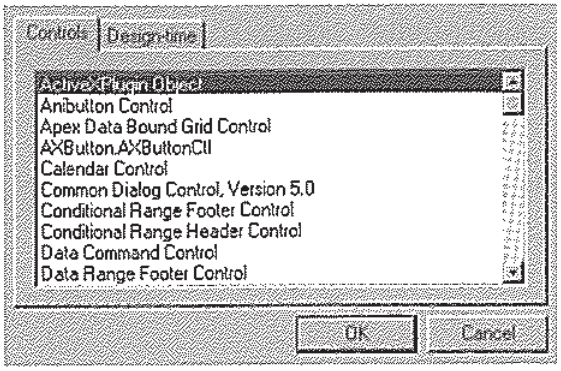

Figure 4.3 The Insert ActiveX dialog.

Just like .ocx controls that can be used in a Visual Basic form, every ActiveX component has its own set of properties that can be set. So, in order to effectively use ActiveX controls, you must also know the properties that can be written to or read from for each component. As you can imagine, this would be a daunting task if it was not for the help of design tools.

Using Visual InterDev to Insert an ActiveX Component

The process of inserting new ActiveX content into a HTML document is much like inserting a custom control into a Visual Basic form. You can start by either selecting **Insert ActiveX Control** from the File Window's Popup Menu or selecting **Into HTML, ActiveX Control** from the **Insert** menu. Once you have done this, you will be presented with the **Insert ActiveX Control** dialog shown in Figure 4.3.

If you select the control that you wish to insert and click on the OK button, you will find that the File Window changes and becomes an interface like the Visual Basic IDE form building interface. In this window, you can size the control and set its properties using property sheets that are very similar to the ones that you are used to in Visual Basic (Figure 4.4).

Once you are finished using this interface, you can finally add the HTML **<OBJECT>** tag to your HTML document by clicking on the window's close button on the upper right corner. Once this interface is closed, Visual InterDev will insert the HTML code describing the control into the HTML document (Figure 4.5).

Once again, Visual InterDev is easy to use and it provides a valuable resource. If it were not for this tool, not only would you have to remember the different parameters that each ActiveX control was able to take, but you would also have to find the appropriate GUID for each control. If you think for a moment about the number of ActiveX components that are available for your use, the value of this tool becomes immediately apparent.

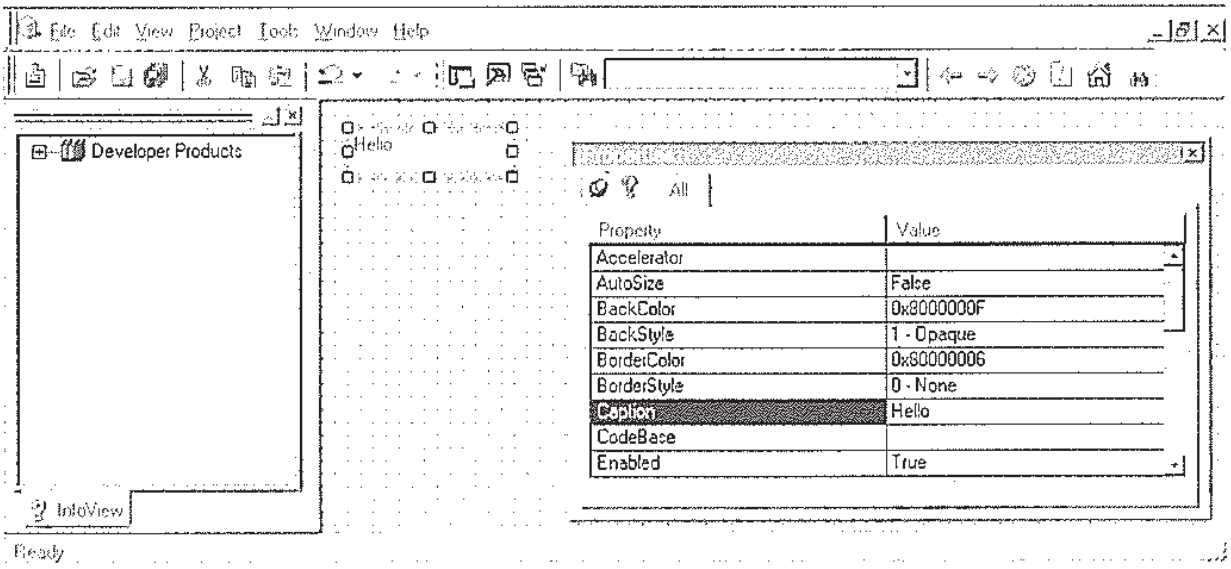

Figure 4.4 The ActiveX Insertion Interface.

Specialized ActiveX Controls

Although there are many noteworthy ActiveX controls currently available that handle everything from data manipulation to the display of multimedia, there are two particular controls that I would like to highlight. The first is important because it can have the effect of dramati-

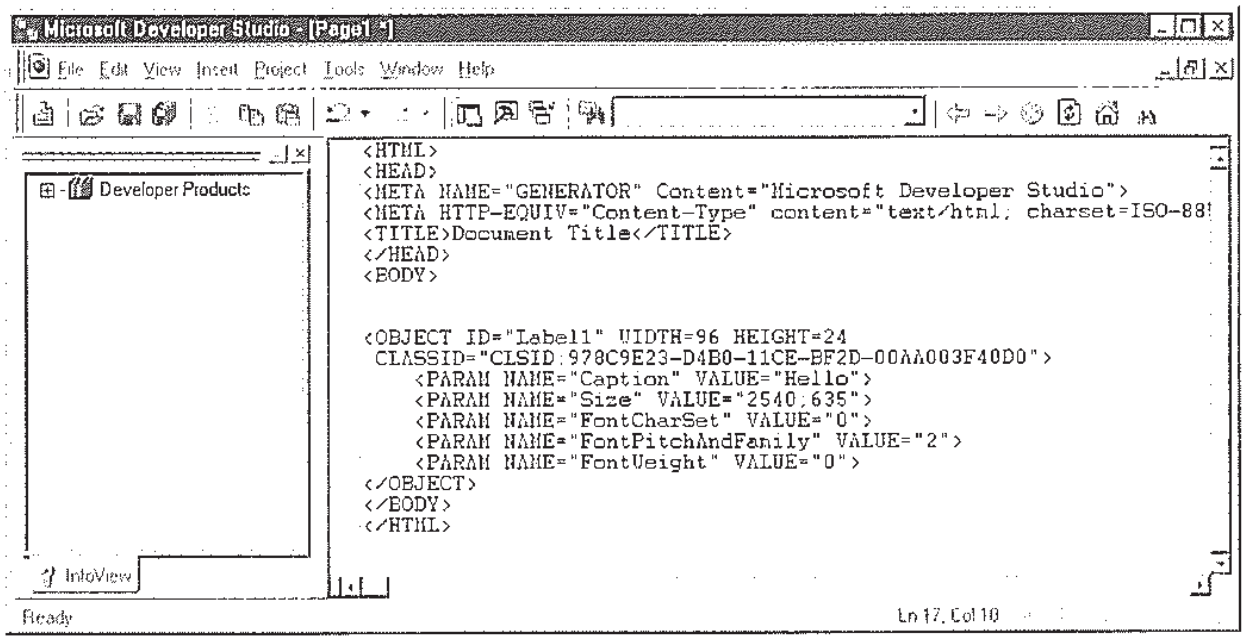

Figure 4.5 The resulting <OBJECT> code.

cally optimizing the performance of your Web-based application. The second is important because it is used extensively by the visual design environment of Visual InterDev and it marks a significant advancement in the way that World Wide Web documents can be created.

The Preloader Control

If you have ever surfed the Web using a slow connection, you have probably spent hours cursing at your screen as images or other large files crawled their way to your browser. This is exactly the reason why Microsoft has created the Preloader Control.

This control is able to download a specified file into the browser's cache while another page is viewed. Although this is not actually increasing the speed at which information is downloaded from the Web, it does make your users feel as if you have found them some extra bandwidth. The Preloader control is invisible to your users when viewed in a Web page and it will fire a COMPLETE event when it is finished downloading its file. If something happens (such as the Internet connection being broken) to stop the preloading, then the Preloader control will fire an ERROR event.

The Preloader control's only four properties (<PARAM> attributes) are shown in Table 4.2.

The HTML that follows is an example of using the Preloader control to download a large amount of information into the browser's cache for later viewing:

```
<OBJECT
ID=Preload1
ClassID="clsid:16E349E0-702C-11CF-A3A9-00A0C9034920"
CODEBASE="http://www.microsoft.co/jp/ie/download.activex/ieprld.ocx"
width=1
height=1
<PARAM NAME ="URL" value="huge.htm">
<PARAM NAME="ENABLE" value="1">
</OBJECT>
```

The HTML Layout Control

One of the most difficult things about working with HTML is the small amount of control that you really have in constructing your interface. Although, this type of fixed two-dimensional layout is simply assumed in every other modern development environment, it was only recently that such innovations as tables and frames became available for the reliable two-dimensional formatting of Web content. Still, these innovations do not come close to providing HTML with the type of formatting strength that is expected in a modern development tool.

Table 4.2 PRELOADER Control Param Attritbutes

Property	What It Defines
URL	The URL that is to be downloaded into the browser's cache.
ENABLE	Can be set either to enable (ENABLE=1) or disable (ENABLE=0) the Preloader control.
CACHEFILE	The read-only name of the locally cached file
DATA	The read-only contents of the cached file

The World Wide Web Consortium (W3C) is very aware of this problem and has already proposed a solution by extending the HTML language to include full two-dimensional layout capabilities. However, the universal adoption of this proposal has been slow in coming.

So, Microsoft has created an ActiveX Control that provides a preliminary implementation of this new technology. The HTML Layout control allows developers to work in the proposed syntax and thereby create more compelling and controllable World Wide Web matter as they wait for the newest implementations of HTML to be accepted by the general public. Microsoft is committed to supporting the finally adopted W3C format and will provide a free utility for the conversion of the HTML Layout Control to the W3C format at the time that the new standard is widely accepted.

However, when the time comes that you no longer need the HTML layout control, you may miss it because it has advantages other than simply providing the correct space between controls on a Web page. First, because it is an ActiveX control, it is able to supply efficiencies in the speed at which graphical data can be displayed. Secondly, the idea of implementing an interface in a control allows for significant reuse. For example, if you created an HTML Layout that defined a simple name and address form, then you could simply drop it onto any page that had a need for that type of interface again and again.

Microsoft Internet Explorer 4.0

The 4.0 version of Microsoft Internet Explorer supports all of the features available to many browsers through the HTML Layout Control. Once again, if you are able to develop a Web application that needs only to cater to one type of browser, then Microsoft Internet Explorer 4.0 allows you to work with the VBScript language and all of the features of *dynamic* HTML.

How the HTML Layout Control Works

Aside from being able to format and display the new two-dimensional layout of HTML, the HTML Layout control also acts as a container for all of the other controls that you place in it. The way that this control works is through the use of two files. The first file is the ActiveX component and the other is a text file with a **.ALX** extension. This text file formats the layout of HTML components using the preliminary HTML syntax for two-dimensional layout. Although this currently requires the use of these two files, when the 2-D layout specification is widely accepted, most browsers will support it directly.

The first thing that must be done to be able to use this control is to insert the ActiveX component in the native HTML. As with any ActiveX component, the HTML Layout control is defined by the **<OBJECT>** tag and **<PARAM>** tags that define the properties of the control, itself. Aside from the attributes that any object can have, the following list describes the parameters and attributes that the HTML Layout Control supports:

- **ID:** An optional ID or name to identify the control for the use with scripting (Visual Basic Script or JavaScript).
- **Style:** Inline style for the HTML Layout control. This attribute has LEFT, TOP, WIDTH, and HEIGHT attributes that define the control's placement in the HTML document.
- **ALXPATH:** The URL of the .ALX file that will be used in the instance of the HTML Layout Control.

The following code is an example of the object tag that would be used to define a basic HTML Layout Control embedded into a HTML document:

```
<OBJECT CLASSID="CLSID:812AE312-8B8E-11CF-93C8-00AA00C08FDF"
codebase="http://activex.microsoft.com/controls/mspert10.cab#version=1,0,5,1"
ID="download_alx" STYLE="LEFT:0;TOP:0">
     <PARAM NAME="ALXPATH" REF VALUE="layout.alx">
 </OBJECT>
```

The .ALX File This supporting file includes objects and event procedure code that will be displayed and run when HTML Layout Control is viewed. According to the preliminary specifications of the W3C, this file is defined within a pair of **<DIV>** tags. The objects that are included within this tag take additional style attributes that define their placement in the HTML Layout Control. What follows is a .ALX **<DIV>** tag that defines the placement of an ActiveX label, an ActiveX textbox, and an ActiveX command button:

```
<DIV ID="Layout1" STYLE="LAYOUT:FIXED;WIDTH:173pt;HEIGHT:53pt;">

    <OBJECT ID="Label1"
     CLASSID="CLSID:978C9E23-D4B0-11CE-BF2D-00AA003F40D0"
STYLE="TOP:8pt;LEFT:8pt;WIDTH:41pt;HEIGHT:8pt;ZINDEX:0;">
        <PARAM NAME="Caption" VALUE="First Name:">
        <PARAM NAME="Size" VALUE="1455;291">
        <PARAM NAME="FontCharSet" VALUE="0">
        <PARAM NAME="FontPitchAndFamily" VALUE="2">
        <PARAM NAME="FontWeight" VALUE="0">
    </OBJECT>

    <OBJECT ID="txtFirstName"
     CLASSID="CLSID:8BD21D10-EC42-11CE-9E0D-00AA006002F3"
STYLE="TOP:17pt;LEFT:8pt;WIDTH:74pt;HEIGHT:17pt;TABINDEX:1;ZINDEX:1;">
        <PARAM NAME="VariousPropertyBits" VALUE="746604571">
        <PARAM NAME="Size" VALUE="2620;582">
        <PARAM NAME="FontCharSet" VALUE="0">
        <PARAM NAME="FontPitchAndFamily" VALUE="2">
        <PARAM NAME="FontWeight" VALUE="0">
    </OBJECT>

    <OBJECT ID="CommandButton1"
     CLASSID="CLSID:D7053240-CE69-11CD-A777-00DD01143C57"
STYLE="TOP:17pt;LEFT:91pt;WIDTH:41pt;HEIGHT:17pt;TABINDEX:2;ZINDEX:2;">
        <PARAM NAME="Caption" VALUE="OK">
        <PARAM NAME="Size" VALUE="1455;582">
        <PARAM NAME="FontCharSet" VALUE="0">
        <PARAM NAME="FontPitchAndFamily" VALUE="2">
        <PARAM NAME="ParagraphAlign" VALUE="3">
        <PARAM NAME="FontWeight" VALUE="0">
    </OBJECT>
</DIV>
```

As you can see in the preceding code, there are many attributes that must be used in order to define the placement of ActiveX components inside **<DIV>** tags. Luckily, Visual

InterDev provides a visual interface for the creation of these files. Remember, there are two parts to a HTML Layout control. So, before the control can be placed, the .ALX file must be created.

Using Visual InterDev to Create .ALX Files and Place HTML Layout Controls

You can use Visual InterDev to create a .ALX file by first selecting **New** from the **File** menu, then selecting **HTML Layout** from the **New** dialog. After you click on the **OK** button on the bottom of this dialog, an interface for building the .ALX file is presented that appears very much like the Visual Basic form design environment (Figure 4.6). You can use this interface just as you would create a form in Visual Basic; through the addition and placement of controls in the window and the setting of the control's properties.

Once you have finished building the .ALX file in this interface, you can save the file that you created and close the window. If you want to view the .ALX code that was generated with this interface, select the newly created file in the Project Window and select **Open With** from that file's popup menu. When the **Open With** dialog is shown, select the **Source Editor** and click on the **Open** button. By doing this, the .ALX code will not be shown in its visual design representation, but rather as a text file displayed and color-coded in the normal Visual InterDev file editor.

Once you have a .ALX file built, you can insert it into any number of HTML documents. When you wish to insert a HTML Layout control into an HTML document, simply display the

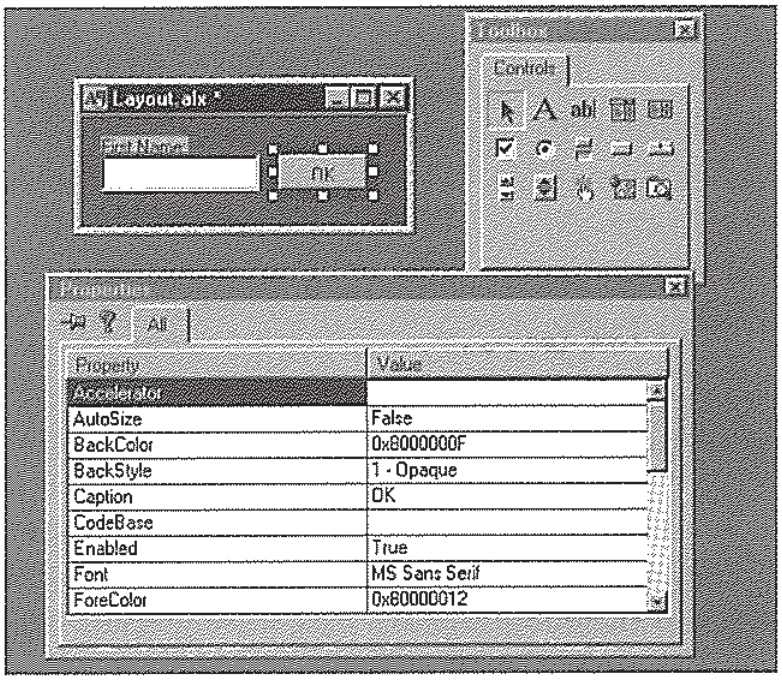

Figure 4.6 The .ALX Visual design environment.

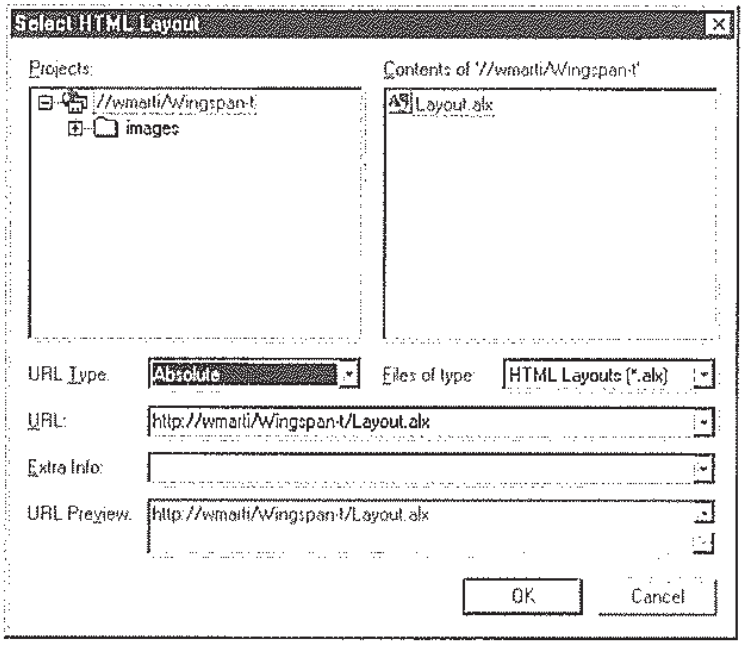

Figure 4.7 The Insert HTML Layout dialog.

document in the Visual InterDev File Window and place your cursor where you would like to insert the control. Then select **Into HTML, HTML Layout** from the **Insert** menu. When the **Select HTML Layout** dialog is displayed, select the appropriate .ALX file and click the **OK** button and Visual InterDev will insert the HTML Layout Control into your HTML document (Figure 4.7).

The Way That ActiveX Works

Through the creation of the ActiveX standard, Microsoft has moved its Windows operating system from the desktop to the Internet. This technology is basically OLE (i.e., ActiveX) at work on the Internet. Because the Visual Basic community is so used to working with this Windows service, the movement of this technology to the Internet represents a very smooth transition from desktop Visual Basic development to Visual Basic development on the World Wide Web.

Visual Basic forms can be used as OLE containers. For example, when you place a .ocx control on a Visual Basic 4.0 form, it is instanced as an embedded, in-process OLE. Once an .ocx control is *sited* on a form, the client (the Visual Basic .exe) can use the server by changing properties and calling methods of the embedded object. So, if you have ever used an OLE custom control in Visual Basic development, the idea of using OLE components in World Wide Web development should seem natural. Microsoft has been able to integrate

ActiveX controls into its browser because its browser, like a Visual Basic form, is an OLE container. In fact, the component that is responsible for the display of Web content in the Microsoft Internet Explorer is an ActiveX component itself.

When an HTTP message is sent to the client for viewing, it may carry with it a call to instance an ActiveX object class. If you already have that type of object on your system, then it is created in the context of the browser. However, if you do not have that component, or (in some situations) the most recent version of the control on your system, then the component will be downloaded, validated, and instanced in your browser. Once you have the component installed on your system, the next time that your browser needs to instance that type of component, no download is needed; it is simply instanced when it is called. Since this file transfer only needs to happen once over the Internet, the client perceives ActiveX controls as being loaded very quickly.

Through this behavior, the browser provides a flexible and intriguing interface with which to develop. Although a browser works in a way much like the Visual Basic form, it is also much more like a television screen than a conventional form. That is, in conventional programming, a different form is used for each screen in your application. In contrast, when you are developing content for the Web, the browser interface is as constant as a television screen. It is only the content shown through the interface that is changing as it receives information from a remote server.

The Internet Download Service The Internet Explorer and other browsers that can make use of ActiveX technology are able to provide this through a system service known as the *Internet Download Service*. It is this service that is responsible for the retrieval, verification, and installation of ActiveX components. When the Internet Explorer encounters an ActiveX component that it does not yet have, it makes one call to the API function **GoGetClassObjectFromURL** and passes it the **CodeBase** information about the control that is contained in the HTML. This function will then asynchronously download the ActiveX component's code onto the user's system.

Drawbacks and Disadvantages

Don't get me wrong, ActiveX is not the answer to every problem. In fact, this technology serves to create a fair number of problems. The first major problem with this technology is that every time a user travels to a page that uses a new ActiveX component, that component is downloaded and registered on the user's system. Although this will make the loading of

these components very fast, it also requires that the user is constantly consuming his or her hard disk's free space with new components.

The second problem with ActiveX technology is that not every browser is able to use it. Remember, the industry's browsers have been fed a steady diet of HTML for years. The weaning process may be slow. However, the two largest browsers already support this technology: Microsoft's Internet Explorer was built for this technology and the Netscape Navigator can support these components through the use of plug-ins.

The third and scariest problem is that using ActiveX components means that there is often somebody that you don't know installing software on your machine. It goes without saying that this is probably one of the best situations for spreading a virus that has ever been conceived. However, be sure that Microsoft is working very hard to come up with solutions for this problem. At present, most creators of ActiveX components can at least be held accountable for their work.

What is important to keep in mind about these drawbacks is that they are all problems that occur only on the *Inter*net (the mean, dangerous world) and a large portion of what you may be developing in the future may be applications for your company's internal Web. When ActiveX components are used on an intranet, most of these problems are eliminated. Your IS division can monitor exactly what is downloaded onto the desktop and the number of components that are used. So, as ActiveX still has many issues for its implementation on the World Wide Web, this technology already provides a powerful and elegant solution for the internal Web.

Versioning Concerns

A real concern with the use of any software is the distribution of the most up-to-date, bug-free version available. This is especially true of ActiveX controls as they may be used in any number of applications and Web pages. For this reason, not only does the Internet Download Service look to see if a component is installed on the system, but it also checks to ensure that the most recent version of the control is present.

When a new version is created for a control, you can include the version information directly in the CODEBASE attribute. As you see in the following code, the version number is simply placed in a position after the URL of the CODEBASE attribute:

```
<OBJECT
CODEBASE="www.wingspan-t.com/controls/nboard.cab#version=5,0,0,5"
WIDTH=460
```

```
HEIGHT=60
DATA=" www.wingspan-t.com /AdvWorks/Controls/billboard.ods"
CLASSID="clsid:6059B947-EC52-11CF-B509-00A024488F73">
</OBJECT>
```

So, the browser will check that the class associated with the GUID is on the system and has a version number that is greater than or equal to the version number specified in the object's CODEBASE attribute. If the GUID specified is not already in the registry, or if the type library of the object describes an object of an earlier version, then the system will employ the Internet Download Service.

There may be some occasions when you wish to download no matter what version is currently loaded on the machine. You are able to achieve this type of behavior by simply making the version number equal to -1,-1,-1,-1. Although you are able to do this, you should not do it often. Remember that one of the real advantages of ActiveX over rivaling technologies (Java Appletts) is that it needs to be downloaded only once.

Using ActiveX Components with Various Browsers

One of the weaknesses of using ActiveX components in your Web development is the fact that not every browser can use them. Until recently, only the Microsoft Internet Explorer was able to make use of these components. As was previously stated, this is a concern only when developing content for the World Wide Web, not when you are developing internal applications using this technology. When you are developing Web-based applications for internal use, you can decide what web client to use and tailor your code accordingly.

Unfortunately, you cannot choose the client for every user that might use your application. For this reason, you may have some concerns about including ActiveX components in World Wide Web applications. Even though the large proportion of your users will view your page through a browser that can use ActiveX components, some users may not see the same content and others may not see any content at all.

At the time of this writing, Microsoft is the only company that supports ActiveX natively in its browser. However, Netscape can use ActiveX components through the use of plug-ins. So, if you want to include ActiveX content for the Netscape Navigator, you can simply make use of the **<Embed>** tag instead of the Microsoft Internet Explorer's **<Object>** tag.

Unfortunately, other browsers have no means by which to provide ActiveX content. This is unfortunate because when you are developing content for the World Wide Web, one of

your goals is to communicate with the largest number of people possible. So, it is important not to exclude anyone looking at your content with a browser that doesn't support ActiveX technology.

Luckily, we can fix this because one of the really nice features of HTML is that it will never "blow up." That is, if a browser doesn't understand tags in HTML, it won't stop processing and crash the browser. Instead, browsers often simply ignore tags that they don't recognize. It is this feature of the browser that can be used to provide a number of different options to users depending on the client application they are using to view your page.

In the following HTML sample, you can see how this is done. Both the **<EMBED>** and the **<IMG>** tags are used inside of the **<OBJECT>** tag. At the highest level in this nesting structure, the **<OBJECT>** tag is used so that if the client is using Microsoft Internet Explorer, this will identify an ActiveX object. If this HTML is viewed by a browser that cannot use the **<OBJECT>** tag, then the browser being used may be the Netscape Navigator and Navigator can make use of the **<EMBED>** tag. If the user is viewing the page with a browser that supports neither the **<OBJECT>** nor the **<EMBED>** tag, you can provide the user with an image of what he or she is missing through the use of an **<IMG>** tag.

```
<OBJECT
     classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
     codebase="http://www.futurewave.com/fsplash.cab"
     width=350 height=150>
     <PARAM NAME="Movie" VALUE="wing.spl">
     <PARAM NAME="Loop" Value="False">
     <PARAM NAME="Play" Value="True">
     <PARAM NAME="BGColor" Value="ffffff">
     <PARAM NAME="Quality" Value="Autohigh">
     <PARAM NAME="Scale" Value="Showall">
     <PARAM NAME="SAlign" Value="">

<!-- The EMBED tag is used for the Netscape Navigator Plug-in -->
     <EMBED src="wing.spl" width=350 height=150
     Loop="False" Play="True" BGColor="ffffff" Quality="Autohigh"
Scale="showall" SAlign=""
     pluginspage="http://www.futurewave.com/downloadfs.htm">

<!-- The IMG tag is used for all other browsers -->
```

```
    <NOEMBED>

    <IMG src="wing.gif" width=100 height=100 ALT="Hey, it's time to upgrade or
get a faster connection.">
    </NOEMBED>
</OBJECT>
```

Of course, it is conceivable that someone is using a browser that cannot even accept graphical information. For this reason, the **<IMG>** tag is used with the ALT attribute to define the text that will be viewed if a really, really old browser is being used to view your content or if the user disabled graphics on his or her browser in order to avoid the overhead of downloading graphic material.

Security

Okay, so this ActiveX technology is powerful, but is it safe? Just as much of ActiveX's power comes from the fact that it can work with all of the client's resources, it is also its largest liability. Since any component is able to manipulate the client's file system and memory, ActiveX is the perfect vehicle for both revolutionizing the Internet and destroying it with computer virus.

As discussed, when you are implementing ActiveX components on an internal web, it is certainly safe. However, on the Internet, Microsoft has had to make some decisions about this technology and implement a special model for its safe use. Under the ActiveX model, the component is able to use the full resources of the system, but it must first be validated as safe before it is used by the system. This model is in contrast to the *Java Sandbox* model.

Under the later model, the safety of downloaded code is achieved by limiting its abilities to working with only a portion of system resources—a sandbox where downloaded code can make a mess without affecting the rest of the system. Although sandboxing is included in the Microsoft Internet Explorer, the decision has been made not to use it with ActiveX components, thereby enabling the creation of a more powerful component model for the World Wide Web.

Of course, that's all fine until somebody gets hurt. So, what can be done to protect the masses while not hobbling the technology out of paranoia? Well, under the model that Microsoft has developed, people are made to be responsible for their actions through the use of certificates and the digital signing of the component. (This sort of registration seems to work with handguns most of the time. Right?)

Protecting Yourself

The way that Microsoft has decided to implement this type of Active content has taken the onus of safety away from the technology and placed it on the implementers of the technology and those who use it. To do this, Microsoft has come up with the scheme of *Authenticode*, a system by which code must be *signed* by the author and *validated* by the user. The system of Authenticode demands that the author of the ActiveX component identify her- or himself and verify that the control has not been modified since it was signed. By doing this, something akin to a tamper-proof cap is placed on the component.

The Microsoft Internet Explorer has been fully integrated with the use of Authenticode. So, Microsoft has allowed its user base to decide for themselves how safe they want to be. Figure 4.8 shows the Microsoft Internet Explorer's **Options** dialog's **Security** tab. This is where the user can choose exactly how to use the Authenticode system to protect his or her system.

The simplest level of security can be implemented by clicking on the **Safety Level** button on this dialog and setting the security level to either **High**, **Medium**, or **None**. If your security is set on **High** and the component to be viewed has not been signed, then the browser will not display the control. If the user has set the security of the system to **Medium**, then the

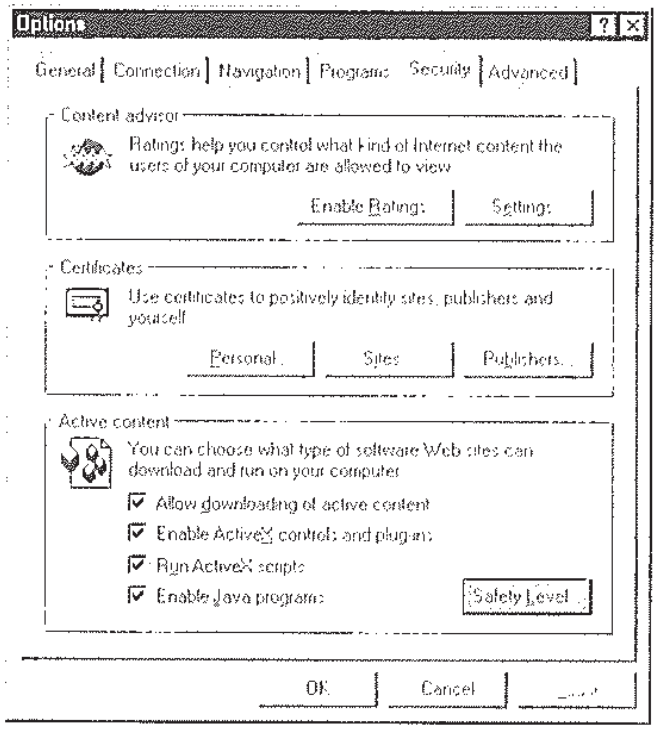

Figure 4.8 The Microsoft Internet Explorer's Safety Settings tab.

Microsoft Internet Explorer will first display a dialog to the user explaining that the control may be dangerous and giving the user the option whether to download the control or to pass it by for safety's sake. Finally, if the security of the browser has been set to **None**, then the ActiveX component will be slapped right onto the user's system without a complaint.

Once a control has been installed on the system, it will not be checked again to see if it has been *signed*. However, when the client instances the component, it will be checked to see if it has been *marked* as safe for *initializing* and *scripting*. Once again, the security levels set in the **Safety Settings** tab have impact when the control is to be instanced.

If an unmarked component is to be instanced by a client with security set on **High**, then the component will not be instanced and any scripting that refers to the control will fail. Likewise, an unmarked control being viewed by a client with a security setting of **Medium** will prompt the Microsoft Internet Explorer to first warn of possible danger, then allow the user to make a decision. Finally, if the user has elected to have a security setting of **None**, then there is no protection provided to the client.

So, if a component is to be viewed and scripted by clients, then it must be both *signed* and *marked* as safe for use. Once again, all that this does is to demand that the developers identify themselves and be held accountable for any damage that they cause. However, it does in no way guarantee that the authors of the controls are not complete incompetents who have somehow bumbled their way through the ActiveX SDK to sign and mark their control, in error, as safe. Still, since controls are signed by their authors, at least you now know exactly who you can tell your lawyer to contact.

So, you can think of the security that is employed for ActiveX as something like trick-or-treating. That is, your users scurry around the Web, picking up goodies and dropping them into their baskets (Registry). The only real protection that they have is that if one of the goodies that they have picked up has a razor blade (virus) in it, then at least you can find out where they got it.

The Authenticode Model

Much has been said about the relatively unsecured nature of ActiveX components. As was previously stated, ActiveX is less secure than the "Secure Sandbox" model employed in Java (even though Java security is far from absolute). Rather than preaching that ActiveX is secure, I would rather expose you to the model used for the secure transmission of code from the software manufacturer to the client machine and let you draw your own conclusions.

The First Step: Getting a Certificate Getting the code to be distributed from the software publisher to the secure client system is a tree step process. The first step is getting a

certificate from a Certification Authority (CA). A CA is simply a trustworthy person or organization that issues certificates to software publishers. Each certificate is linked to the CA that issued it. In this way, CAs act like notary publics for ActiveX components.

The primary responsibilities of a CA include publishing their criteria for certification and granting certificates. CAs store information about the publisher of the software and will provide it to you if needed. Therefore, if an ActiveX component causes damage to your system, at least you can identify the liable party.

Class 2 Certification There are two basic levels of certification. The first and lowest certification is the Individual Certificate (Class 2 certification). This type of certification will cost the owner around $20 a year. Applicants for this type of certificate only need to meet the following criteria:

- **Identification:** The publishers of the ActiveX component must submit their name, address, and other personally identifying information that will be checked against an independent consumer database for validity.
- **Pledge:** The publishers must pledge that they are not distributing a component that can, in any way, harm the user's system.

Class 3 Certification The other level of certification is for commercial software publishers. The Commercial Certificate (Class 3) is needed for the distribution of components to even the most secure clients. This type of certification allows for tight verification of the safety of a component and the verification of sufficient financial standing of the component publisher. Also, at about $400 a year, it is significantly more expensive than a Class 2 certificate. In order for a CA to grant a Class 3 certificate, the following criteria must be met:

- **Identification:** Applicants must provide their name, address, and materials that establish them as representatives of the applying organization. This proof of identity requires that the person applying for the certification is physically present or registered materials verifying the identity of the applying party be sent to the CA.
- **Pledge:** Like with a Class 2 certificate, the applicants must pledge that they know that the component is not capable of doing anything that would damage the user's system.
- **Dun and Bradstreet Rating:** This is used to verify that the publisher has achieved a sufficient level of financial standing and is still in business.
- **Private Key Protection:** The publisher must verify the security of its cryptographic key to its signature.

The Second Step: Signing Code Once a software publisher has been granted a certificate, it is the publisher's job to sign the code. This can be accomplished through the ActiveX

SDK and results in a *Signature Block* being inserted into the original file that will be checked when the client downloads the component and runs the **WinVerifyTrust** function to ensure that it is safe.

This *Signature Block* is the key to verifying that the code has been generated by a defined publisher that currently holds a certificate and that the code has not been altered since the time of the signing. Since all of the security of the Authenticode model rests with the security of the Signature Block, its creation is both complicated and secure.

The first step of the signing process is running the code through an algorithm that produces a fixed-length digest of the code. This digest will be the essential key for the verification of the code. If the digest cannot be recreated at the client by running the algorithm on the code, then the code is not matched to the certificate or the code has changed since the time that the certificate was created.

Next, this digest is encrypted through the use of *public* and *private* keys. These are matched sets of keys created by the software publisher and used expressly for the encryption and decryption of the digest into the signature block. The private key is used to encrypt the code and is never exposed to an outside party. In fact, some CAs require, or strongly suggest, that this private key be kept on a dedicated and isolated hardware solution.

This encrypted digest of the original code is now combined into a signature block with the name of the algorithm used to digest the code and the certificate granted from the CA. The certificate (also known as the X.509 certificate) contains the following information.

The version of the certificate format

The serial number used by the CA to identify the certificate being used

The identification of the algorithm used to sign the certificate, along with any needed parameters

The name of the CA

A pair of dates signifying the time during which the certificate is valid

The name of the software publisher

The publisher's public key name, any necessary parameters, and the public key

The signature of the CA

After the X.509 Certificate is added to the encrypted digest of the code, it is inserted back into the original code in a special reserved section.

The Third Step: Verifying the Code at the Client The third and last step of the secure transmission of code from software publisher to client machine is the verification of code on the client machine. This is done at the client through the use of the **WinVerifyTrust** API. This API function first extracts the signature from the code being downloaded and determines the CA that authenticated the certificate. Then, it obtains the software publisher's public key as it is distributed by the CA.

After the public key is gained, the function uses this key to decrypt the digest of the original code. The next step is to run the algorithm used to create the original digest on the code that was downloaded to the system. Last, this new digest is compared against the decrypted digest sent in the signature block.

If these two digests are not an exact match, then either the code has been modified since the time it was signed, or the public and private keys are not a match. In either circumstance, the downloaded code becomes suspect and will either not be installed or the user will be warned.

Summary

This is not the last that we will see of ActiveX technologies. In fact, this book is very much about the use of ActiveX and it will be looking at this technology from a number of different standpoints. In Chapter 5 these controls will be used in conjunction with VBScript. Chapter 6 deals with the creation of these controls using Visual Basic 5. Finally, Chapter 7 deals with a newer feature of VB called an ActiveX document.

In each of these chapters you will be dealing with the same basic phenomena; the movement of the Windows operating system and ActiveX (OLE) onto the Internet. As you see this type of technology implemented, you can begin to understand what an important development this really is.