dBASE on The Web

The dBASE Web Wizards
The dBASE Web Classes

By A. A. Katz
CEO, dBASE Inc.
Vestal, New York

Printed in the U.S.A.

Table of Contents

Part 4

The dBASE Web Wizards 53

Part 5

The dBASE Web Classes 71

Part 6

Methods and Classes 89

Part 7

Sample Applications 115

Part 1: The Web

Sometimes even those of us who should know better get seduced by the siren song of Conventional Wisdom. So it was, in my case, when I first considered using dBASE for Web applications. After all (I thought), Visual dBASE doesn't create DLLs and it's not a native-code compiler, it ain't C++ and it's not even Java. Thus it was with rather limited expectations that I embarked on creating my first "live" Web page using Visual dBASE.

Fortunately, both Conventional Wisdom and I were really, really wrong. dBASE turns out to be a killer Web database development platform with a flexibility far beyond most of the other, more popular languages, with performance that'll knock your socks off. Ironically, some of the very features for which dBASE has been bashed over the years are the self-same features that make it an ideal engine for live applications over the Web.

The dBASE runtime Virtual Machine, external database engine, built-in report classes - even the humble .DBF - all contribute to making dBASE one of the best platforms for Web e-Commerce and Intranet development.

- **dBASE produces tiny little executables** The common size range for Web-based dBASE Applets is 80-120K. These tiny server-based executables load in milliseconds. Other languages, with their huge native-code executables, require ISAPI, NSAPI or other .DLL-based architectures to pre-load their multi-threaded applets. dBASE does not.
- **dBASE runs code amazingly fast** without the overhead of the Windows graphic interface to hold it back
- **dBASE's Object-Oriented Language** supplies super-high productivity and re-usability across a given Web site, and across all your Web development projects.
- **dBASE is compatible** with almost any major back-end database engine.
- **dBASE's Web Wizards** produce complete, ready-to-run Web applets without writing a single line of code.
- **dBASE's traditional transaction-based bias** (row-oriented instead of set-oriented) is perfectly suited to the requirements of modern transaction-based e-Commerce and Intranet applications (It behooves us to note that the dBASE Object-Oriented Data Classes also make dBASE a terrific RDBMS front-end development tool).

- **dBASE's built-in Report Classes** run lightning fast, and talk directly to your Web Server. No other applications are required.

I'm glad I didn't let my meager expectations deter me or I would have missed out on the opportunity to build killer, high-performance Web applications in record time. In fact, it made my career for a couple of years. Which just goes to show that dBASE has, as always, come through in an unexpected, wonderful way that belies the Conventional Wisdom. Now I know better!

And you will too. I welcome you to dBASE on the Web!

AAK

Intranet and Internet

dBASE Web applications probably shouldn't be called Web applications. The more accurate name might be "http applications" or "CGI applications", since dBASE applets are at least as suitable on a local Intranet as they are out on the World Wide Web. We recommend, in fact, that you develop your applications on a Local Area Network (or even on your own stand-alone machine) before deploying to a site on the Web.

What best defines these "Web" applications is that their user-interface is the Browser, not the operating system. That's an important distinction. Virtually any application that you'd normally write for Windows can be developed to run in the Browser instead. This is the so-called "thin-client" architecture. It offers a number of advantages:

Cross-Platform The dBASE Web Client (the user-interface) is the Browser. That makes dBASE Web applications platform-independent. Any user can enter or retrieve data from a computer running Windows, Unix, MacIntosh, Linux or any other operating system that supports a Browser. Currently, dBASE Web Server-side applets only run on Windows Web Servers, but they support remote data on any operating system and remote clients on any operating system.

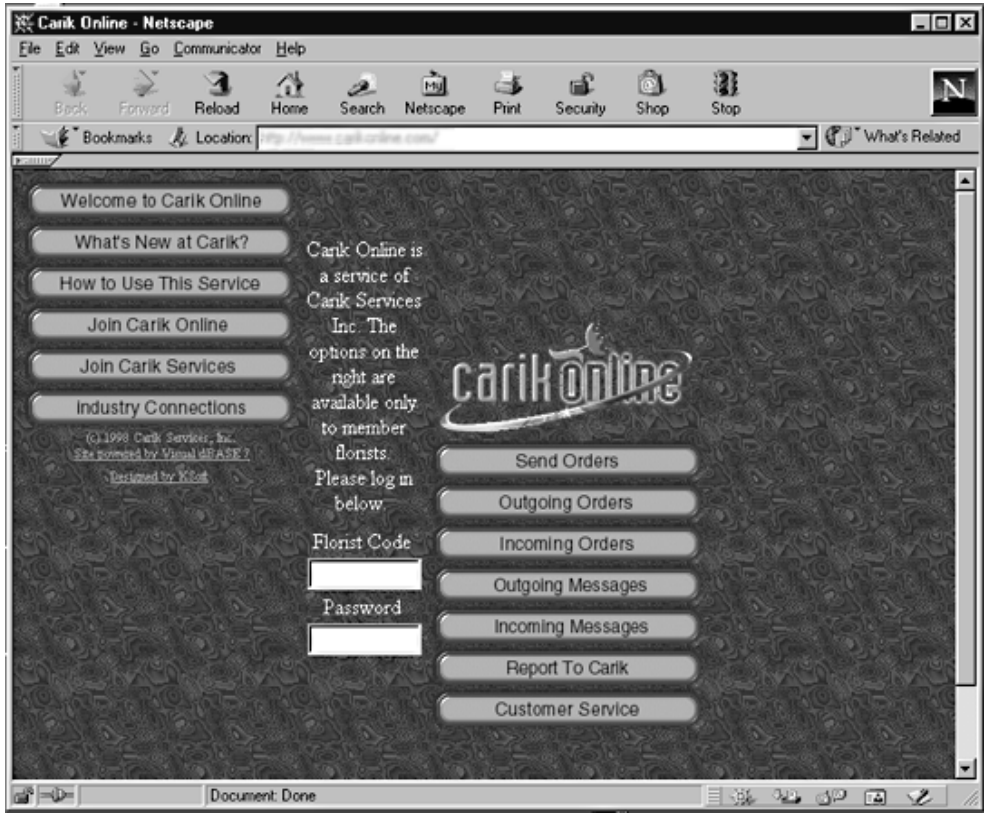
Thin Client Being Browser-based, the dBASE applet user-interface runs fine on older computers with fewer resources. dBASE Web applets allow data to be entered and retrieved on such diverse clients as WebTV®, most PCs, and a growing assortment of portable devices.

Mobile Computing The same dBASE applet can be run on your local Intranet for internal use and surfaced externally to a field sales force or telecommuters, or even made available to your customers.

Remote Computing I often use dBASE Web applets to manage data and applications on my client's networks.

Open System I have one client for whom I developed a complex Visual dBASE 5.6 application. Their company needs to generate new reports without re-compiling the original 16-bit code. Easy solution. Using Visual dBASE, my client designs their own reports against the same tables used in the 5.6 app and delivers them - real-time - to their employees over the Intranet and their customers over the Web.

Low Cost of Ownership Web applets are entirely server-based. Which means you don't have to sit on every workstation to perform upgrades and updates. Simply replace the relevant applet on the Server, and all users get the new features or fixes the next time they run the applet.



On the Carik Web site, above, the options on the right are available to registered members only. The options on the left are available to the general public.

What is a Web Application?

If we start with the assumption that the Internet is just a big network based on a simple client/server architecture (it is) with the ability to hook up to a database and a simple client-side user-interface (the Browser), then we might conclude that Web applications aren't so very different from Windows applications. And we'd be right. That's not to say that there aren't significant areas in which the two diverge: the page-based Browser is nothing like the Windows desktop; the protocols for sending information and instructions back and forth on the Web are designed specifically to meet the challenges of the Internet; and most networks don't require software to "serve up" applications (though we're starting to see similar "application" servers in the LAN environment, too).

Though Web and Windows apps provide the same basic functionality (entering, processing, retrieving and reporting data), they differ in two significant ways:

- **Windows apps track states.** A state is an abstract concept that's easier to describe than define. For example, a Windows app maintains stacks so that the application knows exactly where the user is and where he or she has come from. It keeps variables, property settings, source files loaded into memory - a whole collection of meaningful information about the user and the "state" of the application. Most Web applications don't keep states (some Web development tools do, such as NetObjects™ and Cold Fusion™, but they're notable exceptions and very resource-hungry), since keeping live information and long-term connections for each user on a Web site eats up resources like there's no tomorrow. The ultimate performance goal of a Web site is to connect the user only as long as is necessary to receive a request and return a page. Any longer than that and the server is likely to bog down.
- **The Browser's not the desktop** (regardless of Microsoft's attempts to fuse the two). They use radically different approaches to displaying information. The Browser is page-oriented, most its formatting accomplished automatically and on-the-fly. The Windows desktop is pixel-oriented, redrawing only the section of a screen in which something has been changed. Though, at first glance, this may look like a piddling detail, it is, in fact, the most significant difference between the two: While Windows may repaint one field, the Browser will deliver a whole new page.

Note: *Which brings up an interesting question: why is the Browser interface so much clumsier than the Windows interface? Because the Browser is, at its core, a multi-platform client device. The same page that's viewed in 800/600 on a Windows notebook may be viewed in an entirely different resolution (and possibly an entirely different aspect ratio) on a UNIX PC or MacIntosh. The Browser reformats on-the-fly. Windows is much less flexible (have you tried to scale a Windows graphic application recently for different resolutions?), which is OK, since Windows knows all its parameters as soon as it loads - and they're not supposed to change while the application is running.*

Because the Browser has to paint an entirely new page each time it "refreshes" its data, Web applications are even more form-based than Windows applications. To conserve connections and to achieve acceptable performance, the Web page doesn't communicate with the server at all until the user submits the page. Then, all the information on the page is sent back over the wire to the server in one fell swoop. The Server moves on to the next user and the next request. You're history. If nothing else, that's very economical. In most dBASE server-side applets you're only connected for a fraction of a second. Windows applications are *persistent*. Web applications are not.

In order to maintain "states", that is to remember data that needs to be available throughout an application (such as UserID or Customer Code or any such), Web pages must be chained together, passing the data from page to page. Fortunately that's easy using the dBASE Web classes. The built-in method: PassDataThrough() automatically sends all the data received by the current page out to the next page in the form of "Hiddens", the HTML equivalent of memory variables. It just stores the text in an input control and then hides it. Elegant, no?

How Do Web Applications Work?

Remote Control

A Web Application is a program that runs by remote control. A user invokes an application by typing a URL (Internet address) in the Web Browser, clicking a "Submit" button, or clicking a link on a Web page. The application invoked runs on a remote server and returns a page of results or acknowledgement (in both cases called a Response Page) to the user's Browser. Web applications are truly client/server. The executable part of the application runs on a remote server while the navigation, data-entry and results occur on the local user's machine.

Windows applications give the impression of movement by repainting. A window doesn't really go away, Windows repaints whatever lies beneath the Window being removed. The Browser, on the other hand, doesn't repaint the same page over and over as data changes and results are received. Instead, it paints a new page. Thus, Web applications navigate by going forward from page to page, a process also known as "chaining". Therefore, one of the basic and most important rules of Web application design is that every page submitted returns a Response Page - without fail. Another suggestion is that pages should contain links (such as HOME) that close the loop and get the user back to where he or she started.

Web applications may consist of any number of static pages (pages that are pre-designed and pre-generated) and dynamic pages (pages that are created on-the-fly by the Web application). Your application's home page, for instance, is likely to be static. Data-entry pages may also be static. However, the applications that post the entered data, the pages that return results and pages that acknowledge data entry will all be dynamic.

Persistence And Not

Windows applications are persistent. The operating system and the runtime environment retain all kinds of stacks and pointers and tables, including row pointers, variables, open files and windows. Web applications are not persistent. They are said to be "stateless". The server running the executable only knows who you are and where you are for as long

as it takes the executable to run. After that, you're history. Therefore, a large part of developing Web applications is designing methods to pass data on from page to page. You need to manage any dependencies yourself.

For example, if a customer logs into your site, you've got to ensure that the customer's code is passed along to every subsequent server-side application that requires that information. The dBASE Classes have a special built-in method to pass data from page to page automatically

HTML

The Browser is your application's user-interface. It's also the "Client" that transmits data to the "Server" to be processed. It's based on HTML, a common (supposedly) Browser formatting language which can be augmented with JavaScript, the programming language of HTML. HTML is a *Page Description Language* (PDL), supporting text formatting, image placement, table formatting, data entry forms and, most recently, cascading style sheets (Microsoft IE only). A text-based language, it employs tags to embed instructions in a page. Most tags have an "end" tag and nest the data, control, text or image within a start and end boundary.

```
<HTML>

    <HEAD>
        <TITLE>dBASE Web App</TITLE>
    </HEAD>

    <BODY>
        <B><I>Welcome to a dBASE Web App</I></B>
    </BODY>

</HTML>
```

Note that "end" tags start with a "/" and that tags are nested.

To submit data to a Web application on a remote server, HTML provides a special tag called <FORM>

```
<FORM METHOD="POST" ACTION="MYAPP.EXE">

    <INPUT TYPE="TEXT" NAME="FIRSTNAME" WIDTH="24" SIZE="24>
    <INPUT TYPE="SUBMIT" NAME="SUBMIT DATA">

</FORM>
```

HTML allows you to have multiple forms on any given page, each with its own Submit button, data and target applet. Neat feature. Multiple forms allow you to call more than one application from a single HTML page.

Note: *As I noted earlier, JavaScript is the programming language of HTML. If you're familiar with JavaScript, you may have noticed some similarities to the dBASE OOP language. That's not coincidental. Some of the same people designed both languages.*

Tip! *The supposedly common language, HTML, is anything but. Both Microsoft's Internet Explorer and Netscape's Navigator can make hash out of HTML that runs just fine in the other. Be careful to test on both Browsers before deploying an app. In fact, you might want to be sure to test on some of the older Browsers if you're going out on the Web - you never know what antique your most valuable customer may be running!*

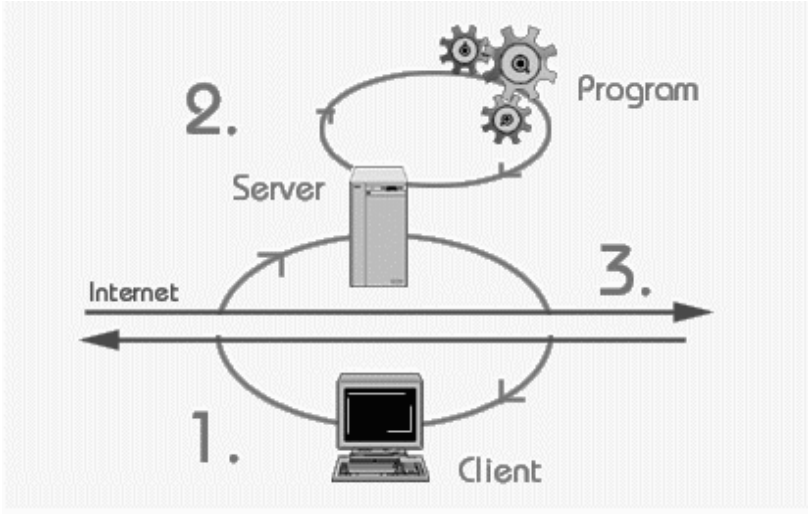
Communication

In order for Web applications to work across many incompatible environments, a common protocol had to be established that supports the same operations on all platforms. CGI (Common Gateway Interface) is that protocol. It consists of both data formats and sequences of operation that are respected by all Web Browsers supporting CGI - which is almost all Web Browsers. Think of CGI as the ASCII or ANSI of the Web.

Under the CGI protocol, a Web application performs the following steps:

1. A URL that calls your server-side application is typed into a Browser, or a URL is sent by clicking a "Submit" button, linked text or graphic on a Web page. The Web Browser contacts the Web Server at the IP address or Domain Name specified, and sends it any data from the "submitting" page.

2. The Web Server recognizes that the requested address is an application, not a page, so it calls the application and waits for it to terminate.
3. Your Visual dBASE application runs on the Server, posting data or retrieving data through reports.
4. When its work is completed, your application sends a dynamic HTML "response page" (Thank you for your order..., Data has been entered successfully, etc.), a formatted report or another interactive page back to the Web Server.
5. The Web Server returns this dynamically-created HTML page to the Browser.



The Response Page

Although the page returned by you to the Web Server (and by the Web Server, in turn to the Browser) looks very much like any other HTML page, it does have one important additional feature: the *CGI header*.

The CGI header is a generally a single line of code that tells the Web Server what to do with the information being fed back to it. For example, to send an HTML page back to the Browser, the very first line of text that you send back to the Web Server will be:

```
Content-type: text/html
```

This line never makes it back to the Browser. The Web Server strips it off. However, without this line, the Web Server will think you're trying to download a file, or, more likely, it will just choke with an "Incomplete CGI Headers" or "Misbehaved Application" error message. The dBASE Web Class streamHeader() method sends this CGI header automatically to the Web Server.

How Does The Server Recognize an Application?

It depends on the Web Server software installed. Microsoft's Internet Information Server (IIS) recognizes an application by its “.exe” extension. Apache and O'Reilly's™ differentiate between a page and an application by the mapped folder in which it sits.

For example, in Apache, you'll find a folder called "CGI-bin" Any files in this folder will be *run* instead of "returned to the Browser". If you check the configuration file for Apache, you'll note that this folder is assigned as the "CGI" remote application folder. Server configuration files and utilities allow you to define the purpose and location of various folders by *mapping*. Mapping is the process of assigning an actual folder to a "virtual" folder using simple substitution.

Example: If I map the following path:

```
c:\MyWebSite\DataGathering\OrderEntry
```

to a new "virtual" folder:

```
/Orders/
```

every time the server sees "/orders/" it will automatically substitute:

```
c:\MyWebSite\DataGathering\OrderEntry
```

Tip! *Remember to use slash (/) instead of backslash (\) when defining or addressing "virtual" folders. Web addresses use the Unix conventions, not the DOS/Windows conventions for folder notation.*

When you set up your Web Server, you'll assign or map a folder to be the “CGI-bin” folder (most Web Servers come pre-setup, so you may opt to use the default mappings). Whenever the Web Server encounters your "CGI-bin" folder in an address (URL), it will know to *execute* the file specified in the URL as an application rather than trying to *return* it as a Web page.

The traditional server mapping for the CGI-Bin folder is either /cgi-bin/ or /cgi/. However, you may use any real or virtual folder you wish as long as it is mapped and accessible to your Web Server. Consult the help files and manuals of whichever Web Server you use.

Calling Your Web Applets

Web applets are launched the same way you used to launch DOS apps, by issuing a path to an executable file (`c:\myfolder\myfile.exe`). The big difference is that Web apps may reside on the local machine, the LAN or across the world. The Domain Name (or IP address) is used to find the right server. Once the server is found, the rest of the "path" is passed along. Of course, it's not called a "path" on the Web, it's a URL. Ultimately, there's not much difference.

A Web-*page* URL may look something like this:

```
http://www.dBASE.com/vdb.htm
```

A Web *application* URL may look something like this:

```
http://www.dBASE.com/cgi-bin/myapplication.exe
```

CGI URLs can also take a command-line parameter (used by the Visual dBASE Drill-down Query and Response Wizard):

```
http://www.dBASE.com/cgi-bin/myapplication.exe?myparam
```

Just to confuse the issue further, you can have your user's HTML form call your application automatically when it's "submitted", call the applet yourself using the "Address" or "Location" field of your Browser, or assign the URL that fires your program to a text link or image link on an HTML page. Which method you use to launch your program depends on the purpose of your application and its relation to other pages and servlets. Again, this is not much different from DOS. You could call an app from the command line, a batch file, a menu shell, or from within another program.

As you gain experience working with Web applications, you'll find the best (or at least the most convenient way) to launch your server-side app. Just remember that what "feels" best to you may leave your users in the dust. I often launch Web applications from the Browser's URL field while building and testing. But I never deploy an application that requires that of the user. I always provide a menu or "starting" page.

Tip! *The paragraph above may seem obvious - as it should. But just think of all the Web sites you've been to that require a guidebook to find what you're looking for. Like all good application design, make your Web applications easy to navigate through the generous use of images, links, menus and the like. This could mean quite a bit of extra work on your part, adding all those cool navigation controls to the response pages you return from your dBASE apps. Trust me, it will be well worth it. I don't know about other people, but if I have trouble navigating an eCommerce site, I'm gone. After all, there's usually hundreds of sites around that sell the same thing without the aggravation!*

For more on launching dBASE Web applications, see "How Do Visual dBASE Web Applications Work", "How CGI Works" and "Mapping and URLs"

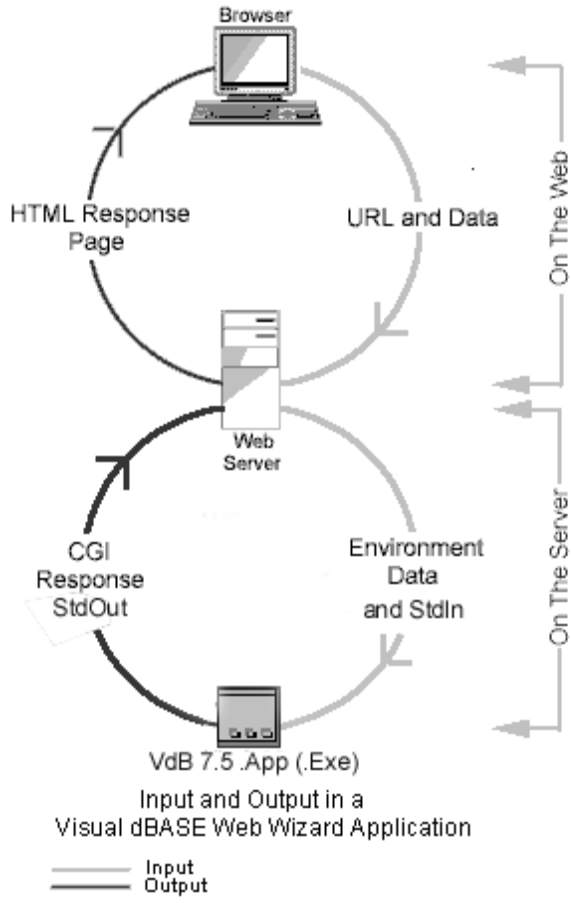
Part 2: dBASE on The Web

How Do Visual dBASE Web Applications Work?

Complexity

On its face, designing and building dBASE Web applications appears to be exceedingly complex. Appearances can be deceiving. The Visual dBASE Web Classes hide most of the complexity beneath their easy-to-use and easy-to-learn methods. The Web Wizards are even simpler - they bury all the difficult stuff beneath a simple step-by-step interface.

Nonetheless, it's important for you to understand exactly how dBASE applications work (and particularly how they interact with the other components involved in Web applications) or you might find yourself without the basic tools required to extract you from sticky situations when things go awry - as they have been known to do on occasion.



The relationship between the Browser, the Web Server and your dBASE applet is very delicate. If any part of the chain is not set up or working right, the entire application fails. Fortunately, once you've got all the pieces up and running, you'll find dBASE Web applications to be remarkably stable over time. That's because dBASE relies entirely on standard Internet protocols and tools: Browsers, Web Servers, HTTP, TCP/IP, StdOut, CGI and HTML, not to mention dBASE code and a wide range of databases. Don't let this plethora of applications, protocols and code daunt you. dBASE makes it all pretty simple.

The Non-dBASE Parts of a dBASE Web Application

Let's start with the outside parts: the Browser and the Web Server. They communicate with each other using *http* (HyperText Transfer Protocol), the protocol that powers your Browser and powers the Web. Fortunately, you don't need to know much about http or *TCP/IP* (the networking protocol of the Internet). Most of your work is on the back side of the server, acting as the interface between the Web Server and the data being queried or updated.

The only notable exceptions are the HTML page that starts your application and the response pages that chain your applets together. Most often, your application will start with a static HTML page. That page has a form on it, and that form has "form components", such as Text controls (similar to dBASE Entryfields), Selects (Comboboxes), Lists (Listboxes), Checkboxes, Radiobuttons, Text Areas (Editors) and Buttons (Pushbuttons). There are no Grid, Notebook or container controls.

These HTML controls have name and value properties, just as dBASE components do.

```
<INPUT TYPE="TEXT" NAME="FirstName" Value="Alan">
```

How fortuitous! Fields have names and values. Field objects have names and values. Associative Arrays have names (keys) and values. This almost-universal implementation of Name/Value pairs allows the Visual dBASE Web Classes to black-box the process of retrieving, processing and storing data - dramatically simplifying the development of Web applications.

Tip! *When you design an HTML startup or data-entry form, apply the same standards you'd apply to a dBASE Windows application: Name your controls appropriately, size them correctly for the data to be entered, default them to obvious values and position them for ease-of-use and clarity of data-entry.*

On The "Back Side" of The Web Server

This is where dBASE shines. As it should. It is, after all, one of the most sophisticated database development tools, and most Web applications are database applications. Once your dBASE applet is launched and the user's data passed along by the Web Server, there are six things you need to do and four of them are things you'd do in any good database app:

1. Retrieve the Incoming Data

When the Web Server launches your server-side dBASE application (sometimes called an *Applet* or *Servlet*), it runs it as a child process, passing along a copy of its environment plus the data it picked up from the user's HTML form. It sends its data either through the environment block, a command-line parameter or as a data stream. A data stream is a relatively new concept to dBASE users. As of Visual dBASE 7.5, dBASE applications can now read or write a stream of bytes to and from another application. A byte stream is just like a text file except that the text flows in real time and doesn't get saved to disk. You can send data through a pipe from parent to child and child to parent without the overhead of creating, finding, reading and writing disk files.

The dBASE Web Classes automatically determine how the data was sent, how it's supposed to be received and then goes out and gets it.

2. Format the Incoming Data

The data that gets read into your application is received in a garbled form of Name/Value pairs similar to “.ini“ files, dBASE "SET" commands or DOS environment commands (PATH=). These pairs must be parsed by your application and converted to dBASE-readable data.

The incoming data stream may look something like this:

```
FIRSTNAME=Alan&LASTNAME=Katz&ADDRESS=102+Main+St%21
```

In the line above, the ampersand is the delimiter between pairs, the plus signs are keyboard "spaces", the “%” indicates punctuation or special characters in Hex format. The left side of the equal sign is the name of the control on the HTML page, the right side is the data the user typed into the control.

Fortunately, this is one more operation that you can safely ignore. A simple dBASE Web Class method, oemFormat(), automatically converts this ANSI string into dBASE OEM data and stores it to an Associative Array:

```
oCGI["FIRSTNAME"]="Alan"
oCGI["LASTNAME"]="Katz"
oCGI["ADDRESS"]="101 Main St."
```

Note: *The Visual dBASE Web Classes consist primarily of the array in which the incoming data is stored. Derived from the built-in dBASE AssocArray class, the Web Classes conveniently allow us to store both the data and the methods that act on that data in the same object!*

3. Validate, Manipulate the Data

OK, now you've got this data retrieved from the Browser, stored in your AssocArray as text, sitting there waiting for you to do something with it. What do you do first? The same thing you do in any decent database application: validate the data!

In the simplest cases, this might mean looking for missing data:

```
If empty(oCGI["LastName"]
   aData.sorryPage("Last name is required!")
endif
```

or performing more elaborate operations, such as checking a user's ID and password, or a customer's CustNo:

```
if not q.rowset.findKey(aData["Custno"])
   oCGI.sorryPage("Customer number is not valid!")
endif
```

or performing calculations or other data manipulation:

```
oCGI["FullName"] = trim(oCGI["LastName"]) + ', ' ;
                        + oCGI["FirstName"]
```

Tip! *One of the nice features of the Visual dBASE Web Classes is that they let you pass an array instead of a string to sorryPage() (the HTML equivalent of MsgBox() in dBASE, or Alert() in other languages). You can batch all the errors instead of requiring your user to submit a page for one error at a time until he or she finally gets the darned thing right!*

Another Tip! *For years I've been trying to convince dBASE developers to do form-level validation instead of field-level validation. The Web has made my case. Although you can do rudimentary validation using client-side JavaScript, all your serious validation happens on the server after the form has been submitted.*

4. Store or Retrieve the Data

Here is where dBASE and the Visual dBASE Web Classes really shine. The methods of the Web Classes let you store your incoming data to a table or send table data back to the user with almost no code. Well, you do have to instantiate a dBASE Query object, but aside from that, reading data and writing data should take only a single line of code each.

This automated, black-box data access only works if you name the form controls on your HTML page with names identical to the table fields they're going to or coming from. And that includes matching case. The AssocArray class is case-sensitive.

Tip! *You'll also be amazed to see the speed of opening queries and saving rows in dBASE when there's no Windows GUI to slow dBASE down.*

5. Build and Send a Response Page

This is the hard part. For virtually every dBASE applet you write, you'll have to write an HTML page as well. This response page gets sent back to the user to either allow forward motion through your application or at least give the courtesy of acknowledging their kindness in sending you data! If you don't send a page, the Web Server chokes. And your user will know that you blew it by virtue of an unkind message sent to the Browser.

Streaming is the process of generating *code* instead of *text*. Just as the dBASE Form and Report Designers stream out dBASE code, you'll be streaming HTML as the last act of your dBASE applet.

Tip! *I usually design my response page using Symantec's Visual Page™ and then use the new Visual dBASE utility HTMLtoPRG to convert my HTML into dBASE code and have it poked right into my applet's source file. Very fast, very helpful.*

Here are some typical Response pages:

- Thank You page
- Confirmation of Order
- Acknowledgement of Data Received
- Another data-entry page (multi-page applications)
- A second copy of the page that launched your applet
- A menu page to continue surfing the site
- The "Checkout" page for an Online Store

Some of the pages above are pretty simple (such as the "Thank You" page). Others require that you build a whole new "launch" page for the next phase of the application.

Confirmation of Order Response Page

Regardless of what you return to the user, it behooves you to personalize your response. And that's very easy to do in dBASE. You can just insert the values from your array into the HTML you stream back to the Web Server:

```
this.fOut.Puts('<B>Thank you, '+oCGI["FullName"] + ;
               ', for your order!')
```

Sending your data back to the Web Server is as simple as using the Puts() method of the built-in dBASE File Class.

6. Clean Up and Quit

Like all good dBASE programmers, I'm sure you're always careful to clean up behind yourself when you close an application. Even though the dBASE Quit command should, theoretically, close down everything and restore your resources, I've never been 100% confident of that. Mostly because this is Windows, and there always seem to be resource problems of one type or another. It's just good form to clean up your classes and queries:

```
q.active = false
q = null
d.active = false
d = null
oCGI = null
Quit
```

Tip! *Cleanup is particularly important in Web applications. Losing a small amount of resources on one workstation run by one user is probably not catastrophic. On the other hand, a Web applet may be run literally thousands of times a day. Chewed-up resources can have a drastic impact on your Web Server.*

Learning To Code dBASE Web Applications.

If you're new to Web development, we strongly recommend that you generate a few Visual dBASE applets using the Web Wizards and inspect the resulting HTML and dBASE code. The code generated by the Wizards can serve as an invaluable guide to creating interactive Web sites using Visual dBASE.

We also recommend that you take some time to look over the Visual dBASE Web Classes, which will help you dramatically improve your productivity in developing hand-coded, superfast Web applets.

How CGI Works

Server-side applications involve more than one simultaneous process. The Web Server software (which is running at all times) calls your dBASE application to perform all the wondrous tasks that you designed it to do and then grabs back control when your program completes execution.

While your dBASE server-side application is loaded, it's running as a child process of the Web Server. Since there's going to be more than one application running at the same time and these applications have to talk back and forth pretty intimately, a set of rules and regulations for interprocess communication is required. Those rules and their implementation on the Web Server go under the acronym of CGI, *Common Gateway Interface*.

The CGI Protocol defines the capability of your server to pass and regain control from a child application, and the circumstances under which it sends data to the child application and retrieves its output. All major Web Servers support CGI in one or more of its flavors. Win-CGI was specifically designed for Windows Web apps, but is supported only in O'Reilly's WebSite on Win 95/98 (and it's terribly slow!).

Most servers support CGI-bin (also known as standard CGI), which uses a direct connection at the operating system level between the Web Server and your server-side applet.

Tip! *Standards exist so that everything will work together smoothly (theoretically). But open standards don't give much of a competitive edge, so both Netscape and Microsoft have come up with proprietary alternatives, ISAPI and NSAPI, respectively. The only advantage of these single-vendor protocols is that they don't require a child process - they use a DLL to put your app in the same process as the Web Server itself. Sounds great, right? No. Our benchmarks show no real-world performance advantage to NSAPI or ISAPI over dBASE running in straight CGI (following the open standards, as it were). Furthermore, proprietary standards are proprietary, not standard, which leaves you at the mercy of the vendor and its own strategic purposes.*

How the Web Server Talks to a Server-Side Application

Because you want your dBASE applets to be able to communicate with all possible servers, you'll use CGI-Bin, the "other" (actually the original) CGI protocol for all your server-side applications. CGI Bin communicates in one or more of three ways:

Command Line Method

In the Command Line method, the Web Server sends the data as a string parameter to the server-side application. The Web address we used earlier in this chapter is an example of Command Line interprocess communication:

```
http://www.ksoftinc.com/cgi-bin/myprog.exe
                          ?firstname=Alan%&lastname=Katz
```

Environment Method

Since your dBASE application is a child process of the Web Server software, the Web Server determines what environment will be available to your application. We're talking a real DOS-style environment block. Name/Value pairs sent back from an HTML form are stored in an environment variable called QUERY_STRING. However, it does so only if the HTML form uses a "GET" method to send the page back to the server. We don't recommend using the GET method, so forget the "environment" option for the moment.

StdIn/StdOut Method

The third method of communication between Web Server and server-side applet is StdIn/StdOut. Until now, StdIn and StdOut have been generally unknown quantities to Visual dBASE developers, but UNIX developers know them well and DOS developers have used these two for years. StdIn and StdOut are streams - a pipeline of moving data that can be identified and grabbed with a file handle. It's not a real file, written to disk, but is treated as such by DOS. You've used them many, many times, though perhaps unwittingly. When you "TYPE" a file in DOS, you're streaming its

contents to the screen using StdOut. Every time you type on a keyboard, a character is sent to the computer through StdIn.

As of Visual dBASE 7.5, dBASE Web applications can, for the first time, access StdIn and StdOut using the built-in File class. Here's some typical syntax for opening StdIn and StdOut (you'll see similar code in the dBASE Web Classes):

```
fIn = new file()
fIn.open('StdIn')
fOut = new file()
fOut.open('StdOut')
```

Note: *You don't create() a file object when using StdIn or StdOut. They are a pipe over which data will be streamed, meaning that the Web Server already opened these pipes from its side. You're the child process - you're just connecting to an existing pipe.*

Another Note: *The surfacing of StdIn and StdOut was quite a coup for dBASE Inc. and its talented engineering team. According to Microsoft's developer-level tech support, it couldn't be done. Well, it certainly can be done in dBASE! The difficulty is that the "conversation" that happens over the StdIn and StdOut pipes happens at the console level (remember SET CONSOLE TO?), meaning that essentially, they are DOS routines. Windows doesn't like to let Windows apps talk to the DOS layer. Something about a loss of control. No matter, we got it working and it dramatically improved the already-fast performance of dBASE on the Web.*

We recommend that you use the read() method of the File class to get your data from the Web Server. The Web Server will conveniently drop some information in your applet's environment block that tells it how the data was submitted and how much there is, as shown in the following code:

```
cPostType = getEnv('REQUEST_METHOD') ==> "GET" or "POST"
nLen = getEnv('CONTENT_LENGTH') ==> length of data stream
if cPostType = 'POST'
   cInputStream = fIn.read(nLen)
endif
```

When you're sending data back to the Web Server, we recommend you use the puts() method of the File class, rather then the write() method. Puts() was designed specifically to send strings rather than streams. It adds carriage returns and line-feeds to the end of every string. Since the data you'll be sending back to the Web Server is in the form of HTML (which is very line-oriented), puts() will make your life much easier. You

won't have to deal with terminating characters, dBASE will take care of that for you.

```
fOut.puts('<HTML>)
fOut.puts('<HEAD>')
fOut.puts('<TITLE>Welcome to dBASE/Web</TITLE>')
fOut.puts('</HEAD>')
etc.
```

(See WebClass.cc for more detailed and documented code).

The best thing about dBASE is that you don't have to know all this stuff. Although it certainly can't hurt to understand what you're doing (especially when something goes wrong), the dBASE Web Class's connect() method does all this stuff for you transparently. Aren't classes great?

HTML and The CGI Header

Whether you plan to use the dBASE Web Classes or the dBASE Web Wizards, you should know at least the basics of HTML. The Wizards will generate HTML for you, but part of the joy of Web development is the high-level of individuality the Web lets you express. The code generated by the Wizards is *meant* to be changed as you gain experience, confidence and push the envelope of Web development. HTML is your opportunity to design your application's user-interface without the rigid limitations of the Windows GUI. Go ahead, get creative!

HTML is used at both ends of a Web application. Home pages and data-entry forms are written in HTML (even if they're written by your dBASE application), and CGI response pages are essentially HTML pages with a minor addition called the CGI Header (more later).

The Language

HTML is a Page Description Language (PDL), not a programming language. It uses tags to define formatting, placement, colors, attributes, to put data-entry controls on a page and to design forms.

Most HTML commands consist of pairs of tags - a start tag and an end tag. At least all tags that cover real estate use start and end tags. For example, text formatting is done with pairs of tags. Sections of the HTML page are defined with pairs of tags. Tables and rows are done with pairs of tags.

```
<FONT FACE="Arial" SIZE="2">Hello dBASE</FONT>
```

But HTML input controls do not use paired tags. They are objects. And, as we know, objects have neither a beginning or an end. They are self-contained.

```
<INPUT TYPE="TEXT" NAME="FirstName" VALUE="Alan">
```

HTML tags are nested, both horizontally and vertically. Let's take the FONT example above and enhance it with Bold and Italic tags:

```
<B><I><FONT FACE="Arial" SIZE="2">Hello dBASE</FONT></B></I>
```

Check out the following code to see how the rows and data of an HTML table are nested vertically:

Tip! *Tables are internal structures used to lay out complex Web pages. Most HTML pages with input controls use them. There doesn't seem to be any other way of getting them under control. Without a table to set proportional or absolute borders (percent or pixels are allowed), your eMail address field may end up five inches beyond the Browser on a Mac, centered on a Win 95 screen and totally off the map in a 640/480 16-color notebook.*

```
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
   <TR>
      <TD WIDTH="83" HEIGHT="77" BGCOLOR="silver"> </TD>
      <TD WIDTH="21" HEIGHT="77" BGCOLOR="silver"> </TD>
      <TD WIDTH="35" HEIGHT="77" BGCOLOR="silver"> </TD>
      <TD WIDTH="70" HEIGHT="77" BGCOLOR="white"> </TD>
      <TD HEIGHT="77" COLSPAN="3" BGCOLOR="white">
         <H1>
            <FONT COLOR="#009999" FACE="Arial, Helvetica">
               Conference Sign-Up</FONT>
         </H1>
      </TD>
   </TR>
</TABLE>
```

As you might have guessed from the previous code snippets, end tags are identified with a slash (/) before the tag name, and all tags are enclosed in angle brackets <>.

Tags may also have attributes (properties). The FONT tag above shows attributes for FACE and SIZE (it also does color and alignment), much like the FontName and FontSize properties of the dBASE Text object. Paragraph tags <P> start new sections of the page on a new line (they have neat attributes not available on any other tag except <DIV>) and they too require a closing tag, but for some unknown reason, nobody seems to use them.

<BLOCKQUOTE> is an important tag because tabs don't work in HTML. Indenting without using a table to define each "tabstop" is almost impossible because HTML abhors spaces. If you leave two spaces between a first name and a last name, the Browser will eat the second space and display only one. So, <BLOCKQUOTE>s are used to emulate tab tops. These definitely need a closing tag at the end of the indented block or your page may list more and more to the right until it runs right off the Browser.

Sections of an HTML Page

HTML pages are divided into sections - sections that you must adhere to rigidly. The Browser is looking for them, the Web Server is looking for them, and any omission (or creative formatting) will result in an unreadable or undeliverable Web page.

The main sections of an HTML page are the Header and the Body. They must be separated by a blank line and both enclosed in a pair of start and end HTML tags.

The HTML Header

The HTML Header contains "meta" information about the page, such as page title, margins, etc.

The HTML Body

The HTML Body contains the page layout to be displayed by the Browser, graphic references to images, links, as well as any forms or "hiddens" (hidden controls used to emulate variables). Any given HTML page may contain any number of HTML forms within it.

```
<HTML>        // defines the PDL
   <HEAD>     // start header
      <TITLE>Hello dBASE</TITLE> //title tag
   </HEAD>    // end head
              // must leave blank line!
   <BODY>     // start body
      <FORM> // start form
      </FORM>// end form
   </BODY>    // end body
</HTML>       // end HTML page
```

HTML Forms

There couldn't be Web applications if there were no HTML forms. These forms are sections of an HTML page that provide the ability to enter or

view data, rather than text and images. They are inextricably linked to CGI. The form defines the data that gets sent back to the server, the <FORM> tag defines the action to take when it gets back there.

Forms always appear in the BODY of an HTML page. There may be more than one, each with its own unique "Submit" and "Reset" buttons. These are two stock HTML objects that perform the core operations of the HTML form. The "Submit" fires the form's action and the "Reset" clears the data from an existing HTML page. Let's take another quick look at the FORM tag (for more see How Do Visual dBASE Web Applications Work?)

Forms take two main attributes, ACTION and METHOD.

The Action Attribute

The ACTION attribute tells the Web Server what it's supposed to do when the page is submitted. In almost all cases, this involves calling a CGI script or executable (why submit data if you're not going to do anything with it?).

The Method Attribute

The METHOD attribute tells the Web Server how to handle the data it's sending back. There are only two options, "GET" or "POST". We don't need to go into great detail about these methods, but if you don't want your users to see all their data echoed to the Address field in the Browser, use POST. In fact, you should always use the POST method. I can't think of a single advantage of using GET with a dBASE Web application (though the Web Classes support both).

```
<FORM Action="Myprog.exe"
      Method="POST"
      ENCTYPE="application/x-www-form-urlencoded">
```

Tip! *I prototype almost all of my applications using Symantec's Visual Page™. I let it generate HTML code and then I convert the code to dBASE. That's really easy now with* **HTMLtoPRG.wfm***, the new utility that converts HTML to dBASE output code. That was the most annoying part of writing Web stuff in dBASE: hand-writing all those dBASE "puts" and parens and delimiters. What used to take me hours now takes about ten seconds. A very valuable tool indeed!*

Tip! *Need to edit HTML as source code? If you haven't, you will, once you get up and running using the dBASE Web Classes. I've got a great HTML source editor for you: dBASE. Probably a remnant of the now-defunct IntraBuilder, the Brief editor in dBASE provides full support for HTML editing, including color-coding and error trapping. One more little dBASE goodie.*

The CGI Header

As I mentioned earlier, there is a minor, but very important difference between the HTML pages you'll write to start your Web applications and the HTML you'll write to send back to the Web Server. That difference is the CGI Header.

The CGI Header (which must be the very first line streamed back through StdOut) tells the Web Server what you want it to do. The most common CGI header looks like this:

```
Content-type: text/html
```

This single, all-important line of code tells the Web Server that you're sending back an HTML page that it is to pass along to the Browser. Without that line, the Server will likely throw an error along the line of "Incomplete or missing CGI header".

Tip! *If your Browser tries to save your CGI response page as a file, you know you screwed up the CGI header. Be really careful here! You can spend hours trying to figure out what went wrong! Use the Web Class's streamHeader() method to ensure that it gets sent right every time.*

Other header options, such as "Location" tell the Web Server to go look for an existing page somewhere and send that back to the Browser. This type of header saves you a lot of code if you don't need to customize the response page on-the-fly.

Reading The Data You Get From The Web

Input

Perhaps nothing else in history has reminded us so forcefully that we live in a world, not a country than the Internet has. The Internet is not an English-Language communications medium. It supports all languages. And alphabets. And communicates using a world-standard alpha protocol called ANSI.

dBASE on the other hand, speaks "OEM" (Original Equipment Manufacturer). OEM refers to the internal language which itself supports many languages through its language drivers and resource .DLLs.

In order to use the data sent back over CGI, you need to parse it first and convert it to dBASE-usable code. A typical data string (also known as QUERY_STRING) sent to your dBASE application over StdIn looks something like the following:

```
Name=A%46+.A%46+Katz&Address=101+Main+Street&City=Vestal
```

The string above consists of three Name/Value pairs separated by an ampersand (&):

```
Name =
Address =
City =
```

The period (like most punctuation) is represented by the hexadecimal value of the character preceded by the percent sign (%). That's to prevent conflicts between real characters and delimiters.

Spaces are represented by the plus signs (+).

Once again, you don't really have to worry about this if you're using the dBASE Web Classes or Wizards. The loadArrayFromCGI() method gets the data from the Web Server, parses and reformats it, converts from OEM to ANSI and stores it in the main Web Class AssocArray in a perfect reflection of the Name/Value pairs:

```
this["Name"] = 'A. A. Katz'
```

```
this["Address"] = '101 Main Street'
this["City"] = 'Vestal'
```

Output

Here's another nice touch in dBASE. You don't need to be concerned about OEM, Double-Byte, Unicode, ANSI or any other protocol or standard when you output data through the dBASE File class. It's converted automatically.

Mapping and URLs

You'll note we used the word "mapped" several times in earlier topics. Mapping probably deserves a moment's special attention - especially if you're going to set up your own Web Server. Like drive mapping in Windows, the Web Server maps "real" folders to "virtual" folders. Why bother? Because you don't want the whole world to know the directory structure of your server and because you can move stuff around a lot more easily if your path isn't hard-wired. Like a BDE alias, mapping gives your Web site and Web applications portability.

In all Web Servers, one folder on the Web Server's hard drive is mapped as the *root* of the Web site.

In Web Servers that support server-side applications (most, some call them scripts), the applications reside in a folder mapped as "CGI" or "cgi-bin" or something like that. Even though it may not be a child of the root folder on the hard disk, it is a logical child of the root of the Web site.

Apache's default installation is a good example:

```
Program Files
  Apache Group
    Apache
      htdocs
      cgi-bin
```

You'll note in the list above that "htdocs" and "cgi-bin" are both on the same level of the directory tree. Nonetheless, Apache's configuration file sets up "htdocs" as the root. All other folders are beneath "htdocs" *logically*.

Here's some URL examples:

A Web-*page* URL may look something like this:

```
http://www.dBASE.com/vdb.htm
```

A Web *application* URL may look something like this:

```
http://www.dBASE.com/cgi-bin/myapplication.exe
```

CGI URLs can also take a command-line parameter (as is the case in the Visual dBASE Query and Response Wizard drill-down reports):

```
http://www.dBASE.com/cgi-bin/myapplication.exe?myparam
```

Web Servers support relative URLs, just as Windows and Unix computers support relative paths. Think back to your DOS days. If you were typing a path and you were sitting on drive C:, you didn't have to type the "C:", it was *assumed*. Web Browsers and Servers do the same thing with URLs, based on the current or calling page's address:

For example. Let's assume that, using Apache as your Web Server, you called a static page using the following URL:

```
http://www.dbase.com/mypage.htm
```

The fact that you have no additional folders after "dbase.com" indicates that you are in the "root" of the Web Site (just as "C:\" might be the root of your hard drive). You can call an application that resides in /cgi-bin without specifying a domain if you call it from a "current" page that's already "sitting" on the root:

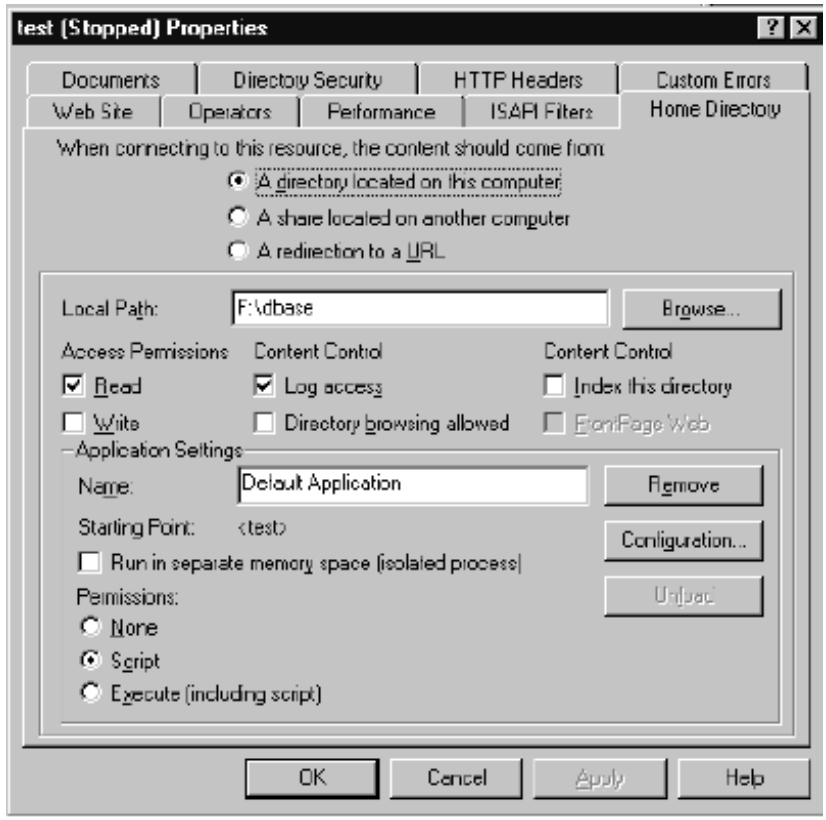
```
<FORM METHOD="POST" ACTION='/CGI/Myapp.exe'>
```

Warning! *If you use http:// in your URL, you must use the entire, fully-qualified URL. That's because http:// reasserts the "root", just as typing "C:\" does in a Windows or DOS path.*

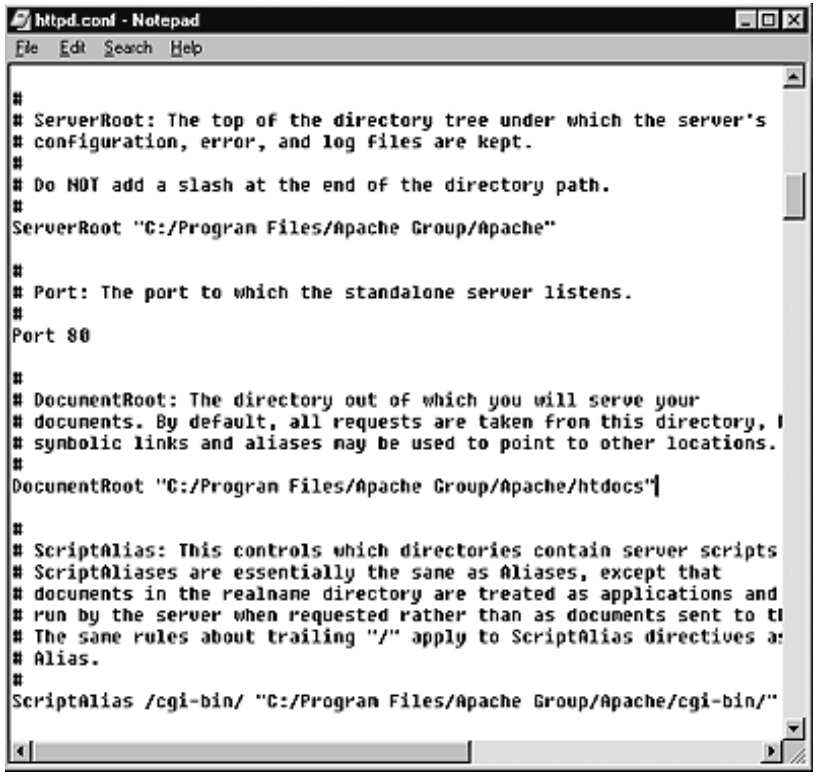
A little bit of experimentation with your own applications will quickly give you a good idea of where you can and cannot use abbreviated relative URLs. On the other hand, if you use the fully-qualified longhand URL, you'll never suffer a pathing error.

Note: *Web Servers use the Unix conventions for drives and folders. "//" follows the drive, "/" sits between folders. Note that we're using "slash", not "backslash". That's also Unix.*

Another Note: *You probably won't be mapping Web site folders with syntax like that shown above. Many of today's Windows Web Servers support mapping through a visual interface (see following figure).*


test (Stopped) Properties
Documents
Directory Security
HTTP Headers
Custom Errors
Web Site
Operators
Performance
ISAPI Filters
Home Directory
When connecting to this resource, the content should come from:
A directory located on this computer
A share located on another computer
A redirection to a URL
Local Path:
F:\dbase
Browse...
Access Permissions
Read
Write
Content Control
Log access
Directory browsing allowed
Content Control
Index this directory
FrontPage Web
Application Settings
Name:
Default Application
Remove
Starting Point:
<test>
Configuration...
Run in separate memory space (isolated process)
Unload
Permissions:
None
Script
Execute (including script)
OK
Cancel
Apply
Help

Apache Web Server, on the other hand, maps folders the old-fashioned (read that as Unix) way, with a text configuration file. Guess which one is more reliable and ultimately easier to configure? Of course, Apache.



VdbFast.exe

dBASE is really, really fast on the Web. Part of the reason that dBASE is so fast is that its engines are external and don't have to load from scratch each time a tiny little executable runs.

Well, that's not quite accurate. The engines will load from scratch each time an applet is called unless you pre-load them. dBASE Inc. has provided a program called VdBFast.exe which does exactly that, and it should be running on your Web Server at all times.

Unfortunately, for a number of architectural reasons, VdBFast.exe is not a Windows NT service, so it won't load automatically at your Web Server's startup (that's coming soon according to current plans). It can be made into a service, but it sometimes causes unexpected problems, so we don't recommend it at this time.

Use either a log-on script or a Startup folder to load VdBFast on your server. You won't believe the improvement in performance. Last time I benchmarked it, with VdBFast loaded, my applets ran more than 1400% faster than they did without.

VdBFast is a must.

Part 3: Installation and Setup

Requirements

1. Web Wizard applications run only on 32-bit Windows operating systems and Web Servers including Windows 95, 98, NT and 2000.
2. You must have Web Server software installed that supports the CGI-Bin protocol. Visual dBASE ships with a copy of Apache Web Server. Apache is an open source Web Server (© Apache Group) that is extremely easy to install. Other Web Servers we've tested these Wizards on are: O'Reilly's WebSite™, Netscape FastTrack™, Microsoft® IIS and Microsoft Personal Web Server.
3. You must install the BDE and the Visual dBASE Runtime on the same server that will run your Web applications.
4. Each simultaneous user requires approximately 10-12 MB of memory on the server. dBASE may grab more if is available. This is not normally a problem since dBASE data-entry and simple query applications only run on the server for a fraction of a second. This depends, of course, on your application design.
5. You do NOT need an Internet connection to build or test dBASE Web applications. They may be written, compiled and run on a stand-alone machine.

Setting Up

Where Do I Put My Files?

One of the challenges of developing Web applications is that you almost never develop where you're going to run. Testing complex database applications on a live Web Server is a bad idea. One crash and your entire Web site may be offline. Also, you may not have your own Web Server and will need to deploy applications to a remote location or ISP Host.

dBASE makes this easy by virtue of its BDE Alias. Once you've installed dBASE, you can run any application, regardless of where the data resides (as long as it's accessible, of course), without rewriting your code. Always use a BDE Alias when developing Web applications.

dBASE ships the Apache Web Server, the most popular (and open-source) Web Server. It runs more than 61% of the Internet's Web sites. It is also extraordinarily easy to install using folder and configuration defaults.

When using Apache:

Your dBASE executable (.EXE) applets deploy to the Apache **\cgi-bin** folder. Static HTML Web pages deploy to the Apache **\htdocs** folder.

When using Microsoft IIS:

Your dBASE executable (.EXE) applets and static HTML pages deploy into any folder you specify. Just make sure, when you set up your Web site that you turn "execute scripts" on or your applets won't run. IIS identifies an app by its extension, not its location.

How To Install dBASE on a Web Server

Servers are not like stand-alone computers or workstations. They boot up and work with no user logged in to the local console. In fact, that's the best and most secure server protocol - don't allow anyone to log on to the server console.

However, any user coming in to your Web or Intranet host machine will be logged on as a remote user. In most cases, this will be an "anonymous" user. The anonymous user allows all those strangers accessing your Web site to access this server with whatever privileges and policies you assign to the Anonymous user.

Because Windows computers support "profiles", you have to be very careful how you install dBASE on your Web Server. If you install in the wrong profile, users may not even see dBASE, let alone be able to run Web applications that require the dBASE login. To help simplify the installation process, I've come up with two basic options:

Anonymous Install

Log onto your server console as the Anonymous User. Install dBASE from the CD-ROM (or install a custom application with a dBASE InstallShield Express deployment).

Warning! *Installing dBASE on your server probably constitutes a new dBASE user, which means you need an additional license to run it both on the server and on your development machine at the same time. You do not need a license to deploy the runtime anywhere. Please respect the terms of the Visual dBASE License.*

Runtime Install

Copy the following files to the same folder from which you plan to run your dBASE executables:

VdB7run.exe

VdB7000n.dll (where "n" is your local dBASE language code)

The BDE must be installed in order for this "runtime" installation to work. One legal way of installing manually is to install dBASE on your server, then do an Uninstall. When asked about removing the shared BDE folders, click No. That will leave a BDE install on your machine without requiring another dBASE license.

Of course, there's a simple and elegant way of deploying the BDE without going through the contortions of installing and uninstalling, and that's writ-

ing your own little InstallShield Express deployment, but don't include program files. Just be aware that it will probably copy VdB7Run.exe into your \Windows\System (or WinNT\System32) folder. You'll want to move it back to your deployment folder before running your Web apps.

Future versions of dBASE will have a Web Runtime installer available on the dBASE CD.

Configuring the Web Server

You'll need to set up a Web Site and a CGI folder in order to deploy and run dBASE Web apps. Consult your Web Server software's manuals or online help for instructions. Setups can vary considerably from server to server.

Deploying Web Applications

Design For Portability

The best way to develop, test, and upgrade Web applications is to build them on one machine (preferably a local machine) and deploy them to their final home on a server when done.

Considering that there are several parts to a Web application, including HTML, dBASE executables, images and data, building and running on two different machines can be a bit of a pain. There's nothing worse than getting an application completely debugged only to find out that drive "H:" doesn't exist on the Web Server. It's mapped to "X:" there.

But there are very simple solutions. dBASE is really excellent at portability. And it will be even better in future versions.

Start with a *BDE Alias*. Use the BDE Administrator to "point" to the location of your data. If you do, you can move the application anywhere, and as long as the server's BDE is configured with that Alias pointing to the right folder, you'll never have a problem finding data.

Note: *Never, ever use an explicit path to images or other external objects in Web applets. If you must, move the stuff you need. On the other hand, the paths you have that point to source code and object code files, such as "set procedure to <path+filename>" don't matter. dBASE strips them out when you build your executables.*

Where Does This Stuff Start?

Almost all Web applications start with an HTML page (Reports alone don't have to. You may call them directly by .EXE filename if you like). The startup page can be an existing page on your Web site, one generated by the dBASE Web Wizards, or one you write yourself. In any case, it goes into either the root folder of your Web site, or a subfolder beneath the root.

If we assume that f:\MySite is the root folder of our Web Site, that's where you put your startup HTML page and run it:

```
http://www.mysite.com/startup.htm
```

If we assume that the HTML startup page is in a subfolder (\mypages) of the root, you put your HTML there and run it from:

```
http://www.mysite.com/mypages/startup.htm
```

If your startup page is also the homepage of your root (It's perfectly OK to use a homepage as a "menu" to various Web apps. Not just OK, it's advisable.), it's probably called Index.htm or Index.html or Homepage.htm or Default.html or something like those. Consult your Web Server's manual or online help for the appropriate page name, or for the place where you can set the default homepage name. If your startup page is the default page in the root:

```
http://www.mysite.com
// everything else is defaults!!
```

We can't (and won't) tell you how to organize your Web site and your server's folders. However, we can (and do) strongly suggest that you never mix executables, data and HTML in the same folder. Each to its purpose.

Where Does This Stuff Run?

Good question. So much depends on how you set up your Web Server and how you design your URLs (relative or fully qualified) that it's hard to say where your applets should reside in order to be found at runtime.

In the default installation of Apache, it's pretty simple: executables are deployed to and run from Apache's \cgi-bin folder. Apache has explicit mappings that tell it to look there for executables. You can place your applets in that folder along with tables and even the dBASE runtime with a pretty good confidence that the applet will find all of its pieces.

That doesn't necessarily apply to Microsoft Internet Information Server. I have most of my applets sitting in a \CGI folder under the Web Site root - but a lot depends on whether you're using a virtual folder, a share or a real folder. Unfortunately, this is not the right place for a Windows NT tutorial (that's a book of its own), so your best bet is to experiment. All I can vouch for is that my dBASE applets run beautifully from a real subfolder of the Web root.

Note: *If you can't get an applet to run after copying it to your Web Server - especially if you get a message saying it can't find something, delete the executable's .INI file and let it create a new one the next time it runs. There is*

both the possibility and likelihood that it contains some explicit paths that may not work in the deployment environment. In fact, I recommend that you don't copy .INI files back and forth unless they're .INI files that you created explicitly to configure your applet. In that case, make sure no paths are inadvertently written into it.

Build in Portable URLs

Because every page submitted requires a response, almost every page you stream out of the dBASE Web Wizards and Classes will contain at least one URL embedded in a form or link. These URLs can also be explicit or portable.

An explicit URL starts with "http:", which sets the 'path' back to the Web site root.

However, once you've arrived somewhere on your Web site, all further URLs can, and probably should be relative.

For instance, if you have an HTML page running in your Web Site root and you want to specify where it should find images, and those images are in the same folder, all you need is:

```
<IMG SOURCE="myimage.jpg:">
```

The same applies for running an applet from an HTML page, though it's highly unlikely that an HTML page will be in the same folder as the applet (it shouldn't be!).

Therefore, you might want to use only one level of nesting:

```
<FORM ACTION="/cgi-bin/myapp.exe" METHOD="POST">
```

Now the example above applies only to the first HTML page, which is launched from the HTML folder, not the CGI applet folder.

If we assume that the applet we just called (myapp.exe) includes a URL to call another applet, then both the calling page and the applet will be running from the *same* folder! In that case, all you need is:

```
<FORM ACTION="myapp.exe" METHOD="POST">
```

This stuff is sometimes more confusing than helpful, but you simply will not achieve portability of your dBASE Web code without it. If you use relative URLs in all your applets and your deployment environment is logically similar to your development environment, your code should port from "Localhost" to a real Internet or Intranet host without a problem.

Tip! *The key word here is **logical**. The relationship between folders is not necessarily physical. In fact, in Apache it's not. The htdocs (HTML) and cgi-bin (executables) folders are exactly at the same level. Neither one is physically nested or contained within the other. However, logically, htdocs is the root and cgi-bin is a subdirectory of that Web Site root. So remember, think logical, not physical!*

Another Tip! *Can't figure out exactly how to path out correct relative URLs? No big deal, it happens to me all the time. A few years ago when I was delivering a paper on dBASE at a conference, Ken Chan (one of the truly long-time and expert dBASE developers in the world) confessed that he uses his fingers to count how many ".parents" to type in an OODML expression. Follow his lead. If fingers don't work, just experiment. There are bound to be quirks from server to server and mapping to mapping. Try the darned things out until they work. I usually build a tiny HTML file and use dBASE to type in experimental URLs. Works great for me.*

Deploying dBASE

There is great news about deploying the dBASE runtime, especially for those of you who have ISPs who are reticent to install dBASE on their servers. You don't have to tell them dBASE is there!! Well, you wouldn't except for the BDE. If you use Advantage Database Server™ when we release our native dBASE/Advantage driver, you won't even need that. And almost any ISP will install the BDE for you.

Why don't they need to know you've installed dBASE? Because the dBASE runtime can be installed without any registry entries whatever. There is a search path that dBASE goes through in looking for its parts as follows:

1. Look in the current folder
2. Look in the executable's home folder
3. Look in the Windows System (WinNT System32) folder
   and only if all three of those fail:
4. Look in the Windows registry.

According to my tests on a virgin machine, you only need three files to run dBASE Web applications:

VdB7run.exe
VdB7000n.dll (where "n" is your local dBASE language code)

You can copy these files right into your runtime (CGI) folder and never install dBASE itself on your server. You'll also save the cost of one additional license.

Hosting

Check the dBASE newsgroups and Web Site for the contact information on ISPs who host dBASE Web applications. In addition, dBASE Inc. will be sponsoring low-cost application hosting and maintenance for dBASE applications only. We will host for both you and your clients. Check the dBASE Inc. Web Site for details at:

```
www.dbase2000.com
```

Performance

Web applications are not generally noted for their performance. After all, not only is there a lot of work to be done, most of that work is communicated over relatively slow connections. Even a T1 Internet connection is dramatically slower than even the most modest LANs.

Nonetheless, Visual dBASE Web applications can be surprisingly quick. In a recent test using Microsoft's Internet Information Server on Windows NT 4 and a 56K dialup connection over the Internet, a report consisting of 80 rows was returned to the Browser in less than a second and a half.

This was accomplished by running **VdBFast.exe** on the Web Server. VdBFast is a minimized form whose only purpose is to load the dBASE engines into memory. Any subsequent Visual dBASE applets loaded will use the already-open copies. This results in dramatic performance improvements on the Web Server.

The Visual dBASE Web Wizards automatically copy VdBFast.exe to your server's CGI folder when you generate your first Wizard application. We highly recommend that you launch VdBFast.exe on your server and leave it running as long as you offer users access to a Visual dBASE Web application.

There is virtually no overhead penalty for VdBFast.exe since the engines would be loaded, in any case, whenever your application is called.

Reports

Reports require some common sense. There are limitations to what the Web Browser will handle realistically, and even greater limitations to what a user will sit still for. Reports that return 20,000 rows definitely fall into the category of unrealistic. In fact, anything that takes more than about 45 seconds will send the user clicking and re-clicking.

The same is true of Queries. If a SQL query takes twenty minutes to run, don't even think of putting it out on the Web. Perhaps on the Intranet (with sufficient warning), but you simply can't expect the user to take coffee breaks while surfing through your site.

Redesign for performance!

Run the First Applet of the Day Yourself

There is this marvelous performance-enhancement built into every operating system, Browser and dBASE itself, called cacheing. Try this experiment. Open dBASE on your Win 95/98 machine for the first time after a boot or reboot. Close it. Open it again. Incredible, isn't it, how much faster it loads the second time? That's OS cacheing. If you've just rebooted your Web or Intranet server, run the first dBASE applet yourself and spare your users the single instance in which performance might be less than stellar.

Part 4: The dBASE Web Wizards

The three Visual dBASE Web Wizards, packaged as a single multi-option Wizard-style program **(WebWizard.prg)**, help you write Web applications using the powerful new capabilities of Visual dBASE. The Wizards walk you, step-by-step, through the process of selecting the data, folders and design elements you'll need to create highly interactive Web applications. When you click "finish", the Wizards automatically generate the source code, HTML pages and compiled.EXEs required to complete your project. The only items you need to provide outside of the Wizards are the tables, reports, images or datamodules that are to be included in your final application.

What The Wizards Do

There are three Visual dBASE Web Wizards:

- **The Data-Entry Web Wizard**

  Helps you create an application in which a remote user can enter data into your tables using a Browser.

  Some possible uses of the Data Entry Wizard:

  Salespersons reporting from the field
  Order Gathering
  Name and Address Gathering
  Help Desk
  Online Store
  Bug Reporting

- **The Query and Response Web Wizard**

  Helps you create an application wherein the user enters or selects criteria from a Web Page and receives a report in response. The Visual dBASE Web Wizards support a drill-down query - the user may refine his or her query by selecting an item on the first report, which "drills down" into detail in a second report. You may, for example, start with a

report of categories. The user clicks on the category, which is displayed as a link (automatically embedded by the dBASE Web Wizards), which in turn calls a new report that lists the inventory items in that category.

Some possible uses of the Query and Response Wizard:

Customer Account Queries
Inventory Lookups
Product Line Lookups
Remote Sales Queries
Shipping Status Queries

- **The Publish Web Wizard**

  Helps you post static and live Visual dBASE reports on your Web site or Intranet. Static reports are run and saved each time you run the Wizard. An HTML page is generated on your Web site that doesn't change until you run the Wizard again. Live reports, on the other hand, are run on your Web Server each time the user accesses the report, returning live, up-to-the-minute data to the Browser.

  Typical uses for the Publish Wizard might be:

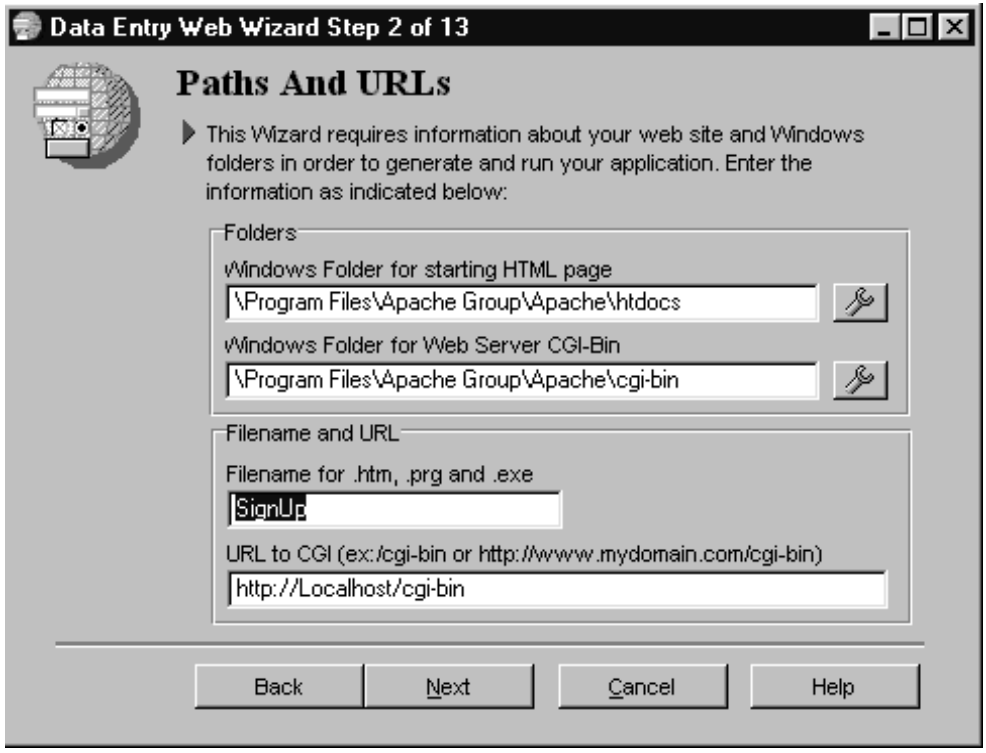
  Price Lists
  Product Lists
  Customer Lists
  Schedules of Events
  To-Do Lists

Using The Web Wizards

Paths and URLs

When you use any of the Web Wizards that generate either HTML pages or live Web applications (they all do except for the Publish Wizard in "static" mode), you'll be asked to enter a varying number of paths and URLs.

Before you start using the Wizards, you'll need to gather the information required below:



Windows Folder for Starting HTML page

Where do you want the Wizards to put your starting HTML page? This is usually in the root directory of your Web site or in a subdirectory dedicated to this particular application. All data-entry and query Wizards produce a starting HTML page to gather data required by the Web application. This page is generated by dBASE.

Windows Folder for CGI-Bin

Where do you want to put executable applications generated by the Web Wizards? This is the physical location from which you want to run your application, not its Web address. This is usually the folder set aside when you configured your Web Server to run scripts and programs. The dBASE Web Wizards will also copy your source code (.prgs) to this location.

Filename for .htm, .prg and .exe

The dBASE Web Wizards use a single name to identify all the parts of any given Web application it generates. Using the example above, the Wizards will generate SignUp.htm, SignUp.prg and SignUp.exe

Warning! *The dBASE Web Wizards "remember" many of your settings from session to session so you won't have to enter the same information over and over as you regenerate applets to improve and test them. Please be careful to change the Filename field to a new name each time you start a new applet. If you don't, the new applet will overwrite your previous one. It will ask. But you will probably click "Yes" and there goes last week's work!.*

URL to CGI

The dBASE Wizards generate pages that call applications and, in the case of the drill-down query, applications that return pages that call applications. Therefore, the Wizards must know, up front, what address to use to call your applications. In most cases, just use the fully-qualified URL you'd use to call the application (not the HTML page) from the Browser.

In the example above, "Localhost" is a standard Windows domain that represents the same machine the Browser is on. That's only used for testing (and to view the dBASE Web samples). You'll want to use your own computername, IP address or Domain name:

```
http://myWebServer/cgi-bin

http://203.55.67.10/cgi-bin

http://mydomain.com/cgi-bin
```

You may also use relative URLs, but we suggest you hold off on that until you become really familiar with all the subtleties of calling complex Web applications.

DataModules and Queries

Paths

The Visual dBASE Data-Entry Web Wizards are based, in large part, on DataModule and Query objects. If you use a table as a datasource, the Data-Entry Wizard creates its own Query object for opening and updating the table. The Visual dBASE Query-and-Response and Publish Wizards use reports (.rep files), which also contain either Query or DataModule objects.

The Web Wizards will automatically compile and build your Query and DataModule objects into the generated application. However, you should keep in mind that the finished application may not be running from the same drive, folder, or even computer on which they were developed. In which case, you'll have to pay special attention to the paths used in the SQL statements of your Query objects. *The Web application generated by the Visual dBASE Web Wizards must be able to find all data sources from the Web Server's CGI folder!*

For example, assume you develop your Web app on a workstation. The data is on the same Server from which the application will run.

From your workstation, the appropriate SQL statement may be something like:

```
Select * from "F:\Mydata\Mytable.dbf"
```

However, drive "F" on your workstation may actually be drive "C" on the Server, in which case the SQL statement needs to be:

```
Select * from "C:\Mydata\Mytable.dbf"
```

There are three ways in which you can ensure the correct paths:

- Set up a temporary folder on your development machine with the same tables you'll be using on the Server and the same path. Using our example above, you'll set up a C:\Mydata\Mytable folder on your workstation.

- Hand recode your .Rep or .DMD files after the wizard is done and then recompile and regenerate the application.

- ***Use only DataModules for all reports and Wizards and include a Database object in each that points to an Alias in the Borland Database Engine***. You won't have to be concerned about paths either in development or at runtime.

Tip! *The current version of the Visual dBASE 7.5 Web Wizards does not support subclassed DataModules (DataModules which are inherited from other data modules).*

Reports

The Visual dBASE Web Wizards use Visual dBASE report classes (.rep files) in the Query and Response Wizard and the Publish Wizard. The fact that these reports will be run remotely and automatically presents certain challenges

SQL Select Statements with Queries

Reports used with the Query and Response Report Wizard must have special SQL Select statements in order to respond to the search criteria typed in or selected by your users.

The Visual dBASE Query Web Wizard creates a public variable "cSearch" at runtime in which it stores the query request typed into the Browser by your user. Therefore, reports that respond to queries need a SQL statement along the lines of:

```
Query1.SQL = 'Select * from "c:/Orders/Customer.dbf" ;
             WHERE Customer."CustName" = ' + '"' + cSearch + '"'
```

You are responsible for the output and formatting of all reports used with the Web Wizards. The Visual dBASE Web Wizards will not alter your reports in any way with the exception of drill-down reports. In that singular case, the first report generated automatically adds an <A HREF> HTML tag to one of your report's text objects. The "HREF" allows the user to click on a row and launch the second, drill-down report.

Reusing Existing Reports.

It is probably advisable to create new reports from scratch for use in the Web Wizards. Reports created for use in applications may have dependencies including (among others) relative paths, parameters, shared queries or global variables. These may present problems to your server-side application since the report will be run from the Server's CGI-Bin folder and the CGI application knows nothing about any dependencies you may have designed into your application.

Using Paths in Reports

Your Visual dBASE Web application will most likely be addressing tables located in some folder other than the one in which it's running. To ensure that the server-side executable application can find your data, make sure you do not include paths to all databases and tables when creating your Data Modules and Queries. Use a BDE Alias instead.

Superclasses in Reports

The Visual dBASE Web Wizards will respect a single level of superclass for reports. If your report is based on more than one parent report class, it will not be rendered properly and may crash your server-side CGI application.

Report Layout

The Browser's window has entirely different metrics from your printer, which can result in unexpected and unaesthetic results when your report is rendered as HTML. It is recommended that you try outputting your report as HTML from Visual dBASE and test it in the Browser before including it in a Web Wizard application. Don't concern yourself with setting the proper Output type for CGI Response (5). The Visual dBASE Web Wizards will reset the Output property automatically at runtime.

Report Length

Design your queries carefully. Performance on a Browser over the wire is very different from performance to a text file, window or printer. In fact, each Browser its own unique limitation on data. Currently, the Visual dBASE Report Wizards generates each report as a single HTML page. If your report yields too many records, it could, conceivably, generate a Browser error instead of a page of query results. Keep your queries limited to the number of rows that can be comfortably accommodated by a single HTML page.

DataModules in Reports

Your Web application may run from within your CGI-Bin folder (on most Servers), not the native folder in which the report or Data Module was created. On other Web Servers, applications may be run from the Web Site root. Wherever it runs, it probably will not be running from where you developed it (most of us do not develop on a live Web Server!). That being the case, it stands to reason that you should always, without fail, use a BDE Alias with datamodules. In fact, use one whether your data source is tables or dataModules!

Note: The Visual dBASE 7.5 Web Wizards do not currently support subclassed data modules.

The Data Entry Wizard

The Data-Entry Web Wizard helps you build information-gathering sites.

It automatically generates a data-entry HTML page that's posted to your Web Site or Intranet and called from your Home Page or any other page you designate. It may also be called by typing the page's URL called directly into the Browser's Address (or Location) field.

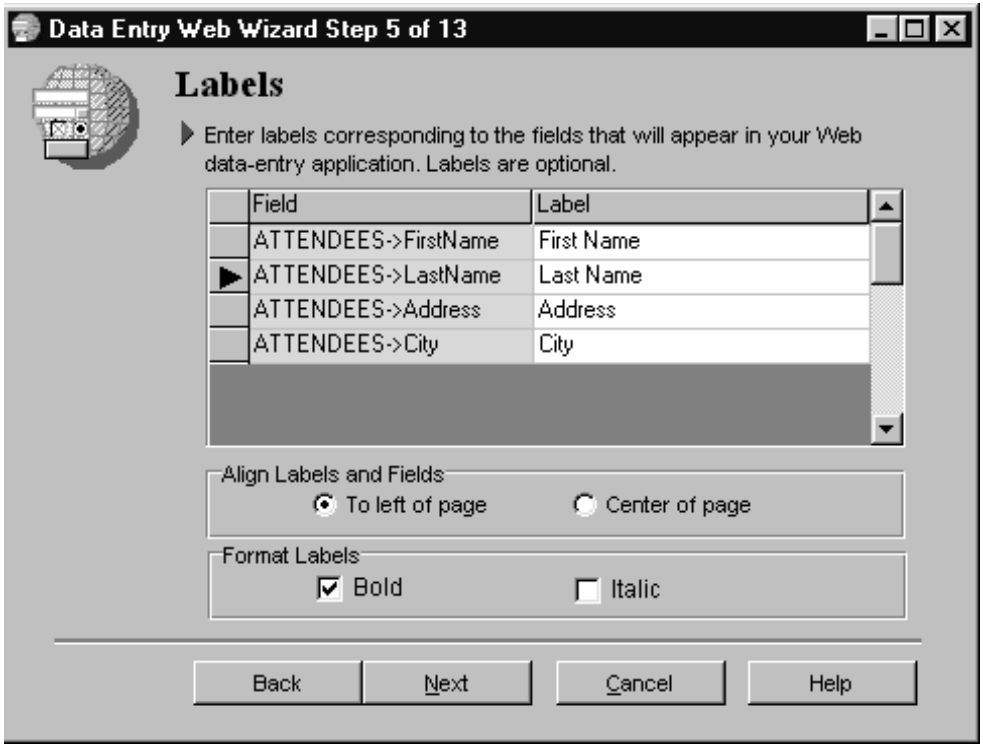
In addition, the Data Entry Wizard creates an executable (.EXE) Visual dBASE application that's called from your HTML page using CGI-BIN.

If the program executes properly and the data is entered, a response page (which you also design within the Wizard') is returned to the Browser.

If the program encounters any problem while executing, it returns an Error page to the Browser.

How is the Data-Entry Application Built?

The application is built in the Wizard from a table or data module you specify. The Wizard walks you through the selection of fields, field order, labels (to display next to the entry controls), text, logo, mail and homepage links.



When you click "Finish", the Visual dBASE Data-Entry Wizard will generate:

- An HTML starting page in a folder on your Web site. (X.htm)
- A modifiable source code file in your CGI-Bin folder. (X.prg)
- An Executable file in your CGI-Bin folder (X.EXE)

Tip! *This application will run from your Web Site and your CGI-Bin folder. Make absolutely sure that any tables referenced in a SQL Select statement are located in the same folder or use a BDE Alias to find them. The lack of a path can cause your Web application to crash, or worse, hang up.*

This Wizard is useful for basic e-Commerce. You may want to enhance the Wizard-generated files to accommodate secure transactions.

The Query and Response Web Wizard

The Visual dBASE *Query and Response* Web Wizard helps you develop a Web or Intranet applet wherein your user may enter or select a search criteria and obtain a listing of the results in the Browser.

This Wizard supports two-level drill-down reports. For example, you might let the user select "category" from the first query page. Then click on any item in the resultant report to get descriptions and prices.

The drill-down capability is provided by two separate report files running from within two separate server-side executable programs. Each one streams a Visual dBASE report back to the user.

Query Input

The Visual dBASE Query and Response Wizard provides two options for gathering search criteria from the user:

1. An HTML Text object (similar to an Entryfield) in which the user can type his or her search criteria.
2. An HTML Select object (similar to a Combobox) in which the user may click on one of your pre-defined search strings.

The Wizard generates:

- A customized HTML query page in your specified Web Site folder.
- A modifiable .PRG for each level of query in your CGI-Bin folder.
- One executable (.EXE) for each level of query in your CGI-Bin folder.

Response Reports

The Query and Response Wizard gathers the query criteria typed in or selected by your user and stores it in a public variable (cSearch) for use in your query's SQL statement, findKey() or setRange() methods. For more information on setting up the SQL statement, see Reports. You are responsible for designing your SQL statements to respond appropriately to the user's search criteria.

This Wizard is used for creating:

- Inventory Lists
- Price Lists
- Customer Lists
- Remote Data Lookup of any kind.

The Publish Web Wizard

The dBASE Publish Web Wizard helps you create applications that will display Visual dBASE reports (.rep) on the Browser.

Static Reports

Static reports are run from within the Wizard. The resultant HTML output is saved in a folder on your Web site or Intranet. To access the report, just put a link to the generated page in any other page on your site. Static reports remain the same until you run the Wizard again.

Live Reports

Live reports are generated at runtime using a CGI executable application on your Web Server. Unlike static reports, these reports are as up-to-date as the moment they're run.

Advantages and Disadvantages

Static reports yield very high Browser performance. No application needs to be run, no data generated. However, they maybe inappropriate on a site where live data is required or where data changes rapidly.

Live reports always show the most current data. If you're posting a price list and you have daily or hourly changes, a live report is undoubtedly preferable. If, however, your price list changes only once a month or once a quarter, you can save traffic on your site by using static reports.

For static reports, the Visual dBASE Publish Wizard generates only a single HTML file in a folder on your Web site.

For live reports, the Visual dBASE Publish Wizard generates:

- A modifiable .prg source code file in your CGI-Bin folder
- An executable (.EXE) file in your CGI-Bin folder

There is no calling page for the Publish Wizard. You call the wizard by embedding a link in one of your own Web site HTML pages. An example of the URL you'll need to use:

```
http://www.mydomain.com/cgi-bin/myreport.exe
```

When you click "finish" on the last page of the Wizard, you will be given a list of the paths and files used and created in generating your application.

Images

The Visual dBASE Web Wizards support a wide variety of graphic types, including animated GIFs.

However, the most common Web graphic formats are .GIF and .JPG.

You may use any of the images shipped with the Wizards freely.

Warning! *The Browser Palette and the Visual dBASE Palette use different methods of reconciling palettes when displaying 256 color images. To get the best idea of what your colors, backgrounds and images will look like on the Browser,* ***you should set your Windows Display for 65,000 colors or higher.*** *Otherwise, colors and images may appear distorted when working in the Web Wizards.*

On the other hand, it makes sense to occasionally test your HTML output in a Browser while your video is set to 256 colors if you're going to publish your applets on the Web. Remember, the users out there have a wide variety of hardware and software. Not everyone is set up to view more than 256 colors (and .GIFs are only 256 colors, anyway).

As for people using 16 color video cards. Well, it's time to upgrade. That's certainly below the threshold of minimum standards you should support.

Part 5: The dBASE Web Classes

What Are The dBASE Web Classes?

The dBASE Web Classes are a collection of classes written in the dBASE language and based on the dBASE AssocArray class that:

- Connect to the Web Server
- Retrieve data from the Web Server
- Format the data received and store it in the AssocArray
- Read data from tables into the AssocArray
- Save data from the AssocArray back to tables or DataModules
- Stream the response page back to the Web Server
- Pass through data for "chained applications"
- Report back user and data errors (Sorry!) to the Web Server
- Report back system errors (An Error Occurred...) to the Web Server
- Clear UserID and Password access.
- Send Mail Through Windows NT

By employing and subclassing the Web Classes, you can build hand-coded Web sites, including e-Commerce sites, in record time. The errorPage() and sorryPage() methods provide serious Browser-based debugging aids.

The Web Classes are located in the following folders:

```
\Program Files
   dBASE
      Visual dBASE 75
         Web
            Classes      // the class code is here
            Samples
               Source   // sample source is here
               cgi-bin  // sample executables are here
               htdocs   // sample HTML is here
               data     // sample data is here
```

The dBASE Web Wizards include three main source files:

WebClass.cc	Main Class
WebPWClass.cc	Password-Enabled Subclass
WebIISMailClass.cc	Send Mail Through Windows NT

They also include a utility that converts HTML code generated by the popular HTML authoring tools to dBASE source code:

HTMLtoPRG.wfm

Tip! *The newly updated dBASE Web Wizards employ the dBASE Web Classes to generate data-entry, query and publishing applications without writing code. We recommend that you build a few apps with the Wizards. They generate HTML, dBASE source code and dBASE executables. They provide an excellent example of how to use the dBASE Web Classes with various kinds of applications.*

How Do The dBASE Web Classes Work?

AssocArray

Very few languages have array classes. Normally, an array is a primitive - a basic data type, not a class. The disadvantage of a primitive is that you can't define the properties and behaviors of a primitive internally. You have to act on it from outside. Conversely, the dBASE array classes offer enormous object-oriented power for the asking. Any array class created in dBASE may contain all the data and all the methods it needs to achieve its purpose.

The AssocArray class is wonderfully powerful. Instead of accessing its elements with a numeric index (as you do in a normal Array class), you access its elements using a text string.

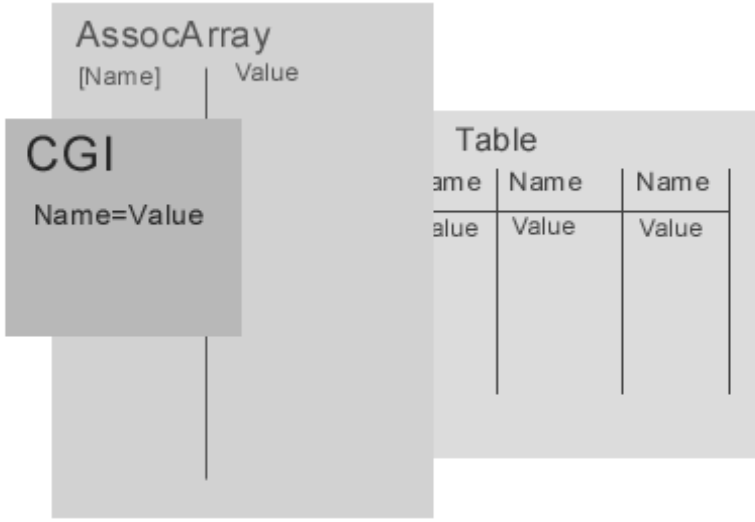
Remember the Name/Value pairs we mentioned in earlier topics?

Think about the following for a minute:

```
CGI:        FirstName = "Alan"
Table:      fields["FirstName"] = "Alan"
AssocArray: this["FirstName"] = "Alan"
```

Lo and behold, we have exactly the same pairs and exactly the same behavior in CGI, dBASE tables and the remarkable AssocArray class.

This fortunate congruence of Name/Value pairs plus the ability of an AssocArray to embed its data and manage itself, led to the dBASE Web Classes. To make it a bit simpler, the fact that all three elements (CGI, tables and AssocArrays) are essentially different implementations of the same basic data structure inspired the design of totally black-box Web Classes wrapped up in a subclass of AssocArray. That custom AssocArray subclass is named *CGISession*. It is the heart of the dBASE Web Classes.



The Rule

Here's how it works. If you name your HTML controls exactly the same as the fieldnames of your table (including case, please!) the dBASE CGISession AssocArray will import your data from the Web Server, import your data from a database table, export your data to a database table, and export your data to a CGI response page. In short, the CGISession will act as an automatic intermediary between the Web and your tables without your interference.

And it does a lot more than data manipulation. It manages the connection to the Web Server, handles both input and output data streams, clears passwords, sends mail, streams HTML and passes data through from page to page. Not bad for a humble AssocArray!

That doesn't leave a lot for you to do. Validate the incoming data, specify the HTML you want sent back in response. That's it. The dBASE Web Classes handle all the rest for you.

Validating And Manipulating Data

The only real work you have to do is to play with the data. And there's really no limit to what you can do when you get accustomed to working with the AssocArray.

Here's a few examples:

```
// Instantiate CGISession Object

Set procedure to WebClass.cc additive
oCGI = new CGISession()

oCGI.connect() // Connect to Web Server and get data.

// Check for missing LastName
if empty(oCGI["LastName"]) // See if last name is empty
   oCGI.sorryPage("Last name required!")  // error and quit
endif

// See if already registered

d = new Database()  // Instantiate database object
with (d)
   databaseName = 'Signup'
   active = true
endwith

c = new Query()         // Instantiate new query object
with (c)
   database = d
    sql = 'select * from "attendees.dbf"'
    active = true
endwith

c.rowset.indexName = Name

// see if that name is already registered
if c.rowset.findKey(oCGI["LastName"]
   // if it is, say sorry and quit
   oCGI.SorryPage('Name already on file!')
endif

// Append row and store data to row from array
```

```
oCGI.LoadFieldsFromArray(c.rowset.fields,true)

// Send CGI response page
oCGI.streamHeader('Thank You') // send CGI header back
oCGI.streamBody('Data updated successfully!') // Stream body
oCGI.streamFooter()            // close HTML

Quit
```

Of course, in your code, you'll make sure to wrap everything in a "Try...Catch" to trap errors and you'll want to release all your objects before quitting. That code was omitted here, but after all, this is just an example. To see the real thing, generate an application with the Data-Entry Web Wizard or view the various source code files for the Signup sample Web application.

Subclassing For Fun And Profit

The single most powerful feature of dBASE, without any doubt whatever, is its fully Object-Oriented Language. Classes deliver such amazing productivity and quality improvements that you're spinning your wheels (and reinventing a few) if you don't take full advantage of what they offer.

The most productive feature of classes is *inheritance*. Inheritance gives you all the tested and debugged functionality of the base class - plus the improvements and customizations you add to your own subclass. dBASE has delivered a lot of useful and clean functionality in the core Web Classes. Now it's up to you to make them your own through subclassing.

Tip! *Don't touch the original classes shipped with dBASE! One of the other advantages of inheritance is the ability to inherit fixes and improvements from the base class. dBASE Inc. has promised to enhance these classes over the next few years of the dBASE Subscription. In fact, it's likely that these classes will become open-source, updated by the very developers using them. If you fix, modify or delete code from the base class, you'll have to do the same to each new version shipped with dBASE. Of course, if you should find a really nasty bug...*

In my humble opinion, you should always subclass the Web Classes when implementing them in your Web applications. Certainly, each of you will at least want to customize the response page. For that reason, the default streamBody() method has been designed without bells, whistles or notable graphic design. It's decidedly vanilla. The way to customize your response pages is to declare a subclass of CGISession in your source file and override the streamBody() method with your own.

The dBASE "Conference Signup" Web sample actually inherits three classes deep:

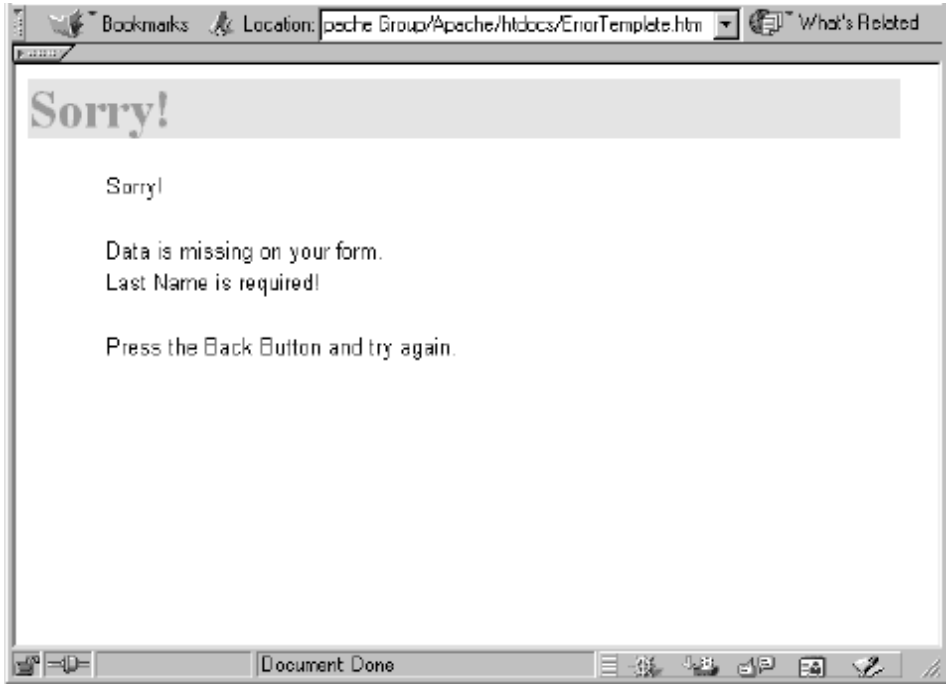
Class 1	CGISession	The default Web Class
Class 2	signupCGISession	Customizes errorPage()/sorryPage()
Class 3	signup	Customizes streamBody()

Class CGISession

This is the base dBASE Web Class. It's in WebClass.cc and it is the superclass from which all the other Web Classes are inherited.

Class signupCGISession

This is a custom class built specifically to be used to derive every applet in this application. It includes general customizations that apply to all applets, such as errorPage() and sorryPage(). This first-level subclass of CGISession gives a uniform "look and feel" to all the applets using it. Think of it as a template or schema, if you'd like. Just don't forget it's truly an inherited and inheritable class.



Base Class sorryPage()

Subclass signupCGISession sorryPage()

Class Signup

This is the lowest-level class in the Conference Signup sample. Inherited from signUpCGISession, it's used strictly to customize the response page for each individual applet. This one never gets subclassed, so feel free to declare a new subclass with the same name at the bottom of each and every applet source file.

The "Signup" class returns the CGI response page for the particular applet. That's accomplished by overriding the streamBody() method inherited from signUpCGISession. Just declare a method (function) with the same name as the original, and your new one will automatically replace the inherited version. Using this three-level hierarchy will allow you to deliver consistent apps without writing a heck of a lot of hand-code.

The code that generates this response page was created visually, using Symantec's Visual Page 2™ (one of the best visual HTML editors). Then I used the dBASE utility, HTMLtoPRG.wfm to poke the code generated by Visual Page into my dBASE app.

If you subclass some generation of CGISession somewhere in your applet source, **HTMLtoPRG** will produce the exact syntax required when you override streamBody().

Use it. It will save you tons of time!

The dBASE Web Classes and Reports

dBASE sports a set of classes unknown in any other development package: the Report Classes. Though they may seem clumsy to work with at times (they're very powerful), they are superb for the Web.

A dBASE report can be run directly from a Browser! You don't have to generate a single line of hand-code to put your data on the web - literally in seconds. That's due to a new option in the Output property of the Report Class:

```
output = 5    // CGI Response
```

Set the response property as indicated in the line of dBASE code above (or just set it in the Inspector) to run directly on the Web. To test the report in Windows, just set the FileName property to whatever you like and the report will be streamed out to a file of that name in HTML format. There is no StdIn pipe set up when you're running in the dBASE design environment, so dBASE knows to output to a file instead.

Compile and build the report:

```
Compile myReport.rep
Build myReport.reo to \Webserver\cgi-bin\myReport.exe
```

That's all. Its ready to call from the Browser:

```
http://www.mydomain.com/cgi-bin/myReport.exe
```

When the report renders, it will connect via StdOut to the Web Server, stream out its own CGI header, output its text objects as HTML and quit when done. Instant report, just add Web!

There are some cautions, however, when running reports:

- Reports must find their data. Use a BDE alias and a Database object with every report.
- Reports must find their Datamodules and Superclasses (if you subclassed the report). Make sure to build them in or the Web report may hold up your Browser forever.

- Images must be available to the Report. Make sure there are no explicit paths to images if you use them in reports. Copy them to the cgi-bin folder before running the report in the Browser.
- Even though you don't need it, you might want to write a WebClass program (.PRG) to serve as a wrapper for your report. The Report class does everything you need to run the report over the wire except for one thing: recover from errors. There is no "sorryPage", "errorPage" or Try...Catch built into the report classes. You might want to make them available by calling the report from a startup program that renders the report from within a Try... Catch. Or, to make matters even simpler, you might want to embed code above the "End Header-Do Not Remove" line of the Report file.

In the following example, let's assume a report called "Inventory.rep". You'll want to insert code something like the following before the header of the file:

```
////// Instantiate and connect the Web Class
Set procedure to WebClass.cc additive
oCGI = new CGISession()
oCGI.connect()

////// Instantiate and run report from within a
//////  Try...Catch
Try
    r = new InventoryReport()
    r.render()

Catch (exception e)
    oCGI.errorPage(e) // Throws error page to Browser

endTry

////// Clean up and get off
oCGI = null
Quit

// End Header - Do not remove this line
```

Don't forget to include WebClass.co when you build your executable:

```
Build Inventory.reo, WebClass.co to;
          \Webserver\cgi-bin\Inventory.exe
```

Warning! *When "wrapping" a report, make sure you don't call any of the other Web-Class methods, such as streamHeader() or streamBody(). The Report Class already does that automatically. You really don't want to see how ugly a response page looks when there are two headers or two bodies!*

Sending Mail

There are any number of ways to send mail from dBASE applets. Commercial OCXs, DLLs and the ubiquitous SendMail are available for free (or almost free) download from the Web. Any of these that work with dBASE and don't require or spawn any GUI screens is perfectly usable.

Most of them, however, generate consequences that can be unfortunate, impacting the performance or reliability of your dBASE application. It should be obvious that the less you do during the execution of an applet, the faster it finishes. As a corollary to that, the sooner your applet finishes, the less connection time per user, the greater number of users can access your site before exhausting your resources. To state it simply, the tinier and more efficient your dBASE applets are, the better your performance and scalability will be.

We all know from personal experience that the performance of eMail servers is normally less than instantaneous. That's almost entirely the result of traffic, and therefore unpredictable and beyond your control.

I found it unbearable to build superfast dBASE applets that slowed to a crawl when they had to access a busy eMail server. As a result, all my mail solutions are designed to run outside of the applet. That way, the application still gives the performance and resulting scalability that makes dBASE so good on the Web. The eMail operation can crawl at its own pace, entirely separate from anything going on between the Browser, the Web Server and the dBASE applet.

There are two solutions that we've found work really well:

Text File Interface

Bowen Moursund, the dBASE Inc. Online Manager wrote CGIIISMailClass.cc, a file housing a custom class that uses the built-in NT mail capabilities to send mail from dBASE apps. That's the interface he's currently using to send eMail confirmations to customers and orders back to Corporate Headquarters from our Online Store.

It's by far the easiest way to send mail from a dBASE mail applet. Just drop a properly formatted text file in the appropriate folder and Internet Information Server sends the mail for you.

Bowen's class, mailCGISession subclasses CGISession and adds Windows NT eMail capabilities.

Usage:

```
Try // Always use a try...catch

   // Define Carriage Return/linefeed
   #define CRLF chr(13)+chr(10)

   // Create new mailCGISession object
   // Pass the full Windows path to the pickup
   // folder as a parameter

   oCGI = new MailCGISession('c:\mymailfolder')

   // Send sender address
   oCGI.From := "webmaster@mycompany.com"

   // Send recipient information
   oCGI.To.add(cEmailAddress) // Add data to oCGI "to" array
   oCGI.Cc.add(cCCAddress1)
   oCGI.Cc.add(cCCAddress2)

   // Send subject
   oCGI.Subject := "Thank You!"  // Set Subject

   // The Body property contains the entire text
   // of the message, and must be pre-formatted
   // You'll need to control the line length and add
   // end-of-line markers as appropriate

   oCGI.Body = "Dear Mr. " + cLastName + CRLF + CRLF
   oCGI.Body += "Thank you very much for your comments." + CRLF
   oCGI.Body += "We will take the appropriate action." + CRLF + CRLF
   oCGI.Body += "Jane Doe" + CRLF
   oCGI.Body += "webmaster@mycompany.com"

   oCGI.postForPickup()  // Send mail message to text file

catch (exception e)

   oCGI.errorPage(e)

endtry
```

There are two drawbacks to this method of posting mail. First, you must be using Windows NT Server and running Microsoft Internet Information Server as your Web Server.

Second, there is no feedback (except, of course for the NT logs) to confirm or report an error in the mail send. You don't know if the mail has or has not been sent until you check the logs.

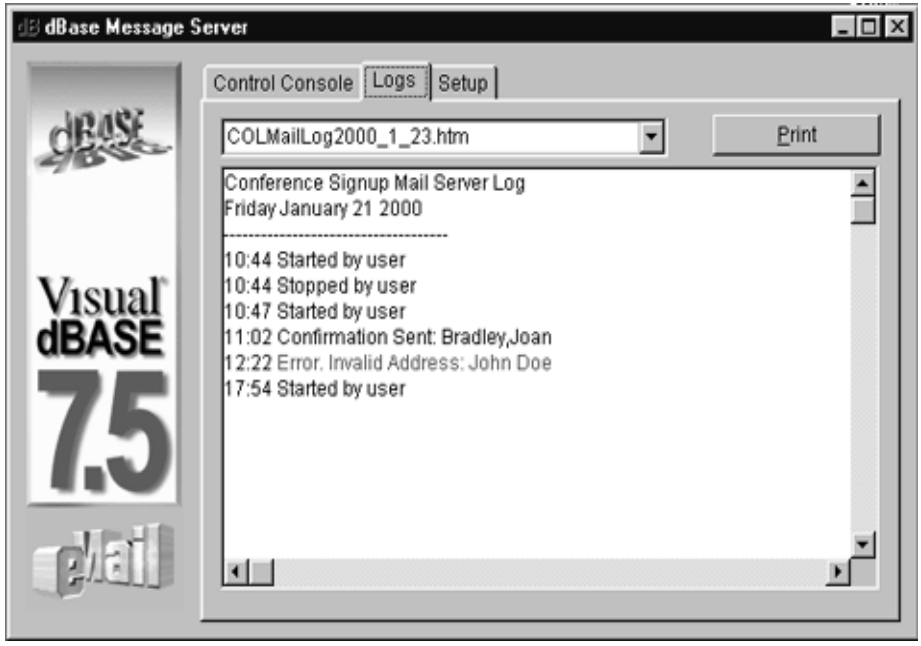
Database Interface

The other alternative is the one that I used in the Message Server included in the Web Samples on the dBASE CD. This is a full-time, 24/7 dBASE application that runs on either your Web Server, or preferably, on another dedicated computer. It runs just fine even as a background task on a Windows 98 computer as long as it has access to your tables over the network.

This is a polling application that continually re-queries your table to see if any new orders, attendees, or requests for information have been added. It then uses the information in the new row to compose and send an eMail wherever you want it to go. The Message Server shipped with dBASE is set up to use the Attendees table used in the Signup Web sample. Each time a new attendee registers, the Message Server sends out an eMail confirming the registration.

This application uses the wonderful Mail library from MarshallSoft. This powerful DLL adds direct mail-server connection, control, download and mailbox management to your dBASE applications. I've had it in production on a high-volume site for more than a year and it has performed beautifully.

This is a much more complex solution than Bowen's, but it generates its own HTML logs that can be accessed remotely, is almost infinitely scalable (high transaction volume? Just add one more computer on the network running the Message Server), and it updates your table to confirm when eMail was sent. And it doesn't require Windows NT or IIS.



For more detail on how the Message Server works, see the Readme.txt in the Visual dBASE 75\Web\MessageServer folder. dBASE ships only the demo version of the MarshallSoft library. It works, but pops up an "alert" that requires a mouse click to clear each time it sends mail. Check the dBASE Online store for information on ordering a fully-licensed copy.

Part 6: Methods and Classes

The Methods of The dBASE Web Classes

Input Methods

Connect()	WebClass.cc	Connects to Web Server and gets data
LoadArrayFromCGI()	WebClass.cc	Internal - Loads CGI data, converts to Name/Value pairs
OEMFormat()	WebClass.cc	Internal - Converts ANSI to OEM

Data Methods

LoadArrayFromFields()	WebClass.cc	Loads table data into array
LoadFieldsFromArray()	WebClass.cc	Loads array data into table row
LoadDataModuleFrom Array()	WebClass.cc	Loads array data into dataModule

Output methods

StreamHeader()	WebClass.cc	Streams out CGIHeader code
StreamBody()	WebClass.cc	Streams out HTML body code
StreamFooter()	WebClass.cc	Streams out HTML closing
PassDataThrough()	WebClass.cc	Embeds all data in new page as Hiddens

Error Recovery

SorryPage()	WebClass.cc	Streams out user error response page
ErrorPage()	WebClass.cc	Streams out system error response page
SetWebMasterAddress()	WebClass.cc	Sets a mailto: address used in error response page

Password Clearing

ValidatePassword()	WebPWClass.cc	Validates UserID and Password

eMail for Microsoft IIS

PostForPickup()	WebIISMailClass.cc	Sends mail via Windows NT

The connect() Method

Method: connect()

Class: CGISession

File: WebClass.cc

Params: None

Description: The *connect()* method establishes a StdIn and StdOut connection with the Web Server. It uses a dBASE file object for each, named fIn and fOut respectively. This method also calls the *loadArrayFromCGI()* internal method that downloads the data stream from the server and parses it out into name/value pairs. Once parsed, the name/value pairs are added to the main CGISession() array object.

Usage:

```
set procedure to WebClass.cc additive
oCGI = new CGISession()

// Connect, get data, parse and convert it
// and store it in the CGISession array

oCGI.connect()
```

Errors: If you can't connect, there's no way to report any errors, so the *connect()* method just gives up and your applet just goes away. Therefore, there is no recovery from *connect()* errors.

The loadArrayFromCGI() Method

Method: loadArrayFromCGI()

Class: CGISession

File: WebClass.cc

Params: None

Description: *LoadArrayFromCGI()* performs most of the core operations that acquire the data entered into an HTML form from your Web server.

Usage: None. This is a protected method, called by *connect()*.

Implementation: To call *loadArrayFromCGI,* invoke the *connect()* method of CGISession, which establishes two-way communication between your dBASE applet and the Web Server.

When called by *connect()*, *loadArrayFromCGI()* will:

- Gather the input stream from StdIn
- Parse the data received into name/value pairs
- Convert the data received from CGI ANSI to dBASE strings
- Add elements to the CGISession array for each pair it finds

Errors: *LoadArrayFromCGI* may encounter a number of errors, which provoke different responses depending on their nature. Most internal errors will throw an error to a try...catch, if one exists.

The loadArrayFromFields() Method

Method: loadArrayFromFields()

Class: CGISession

File: WebClass.cc

Params: (rowsetFields) a Fields Array Object

Description: *LoadArrayFromFields()* adds one element to the CGISession array for each field in a specified rowset. The data in the array can then be used for validation, calculation, inclusion in a response page, or chained to the next page using the *passDataThrough()* method.

Usage:

```
// Create CGISession object
set procedure to WebClass.cc additive
oCGI = new CGISession()
oCGI.connect()

// Create Database object
d = new DATABASE()
d.databasename = 'SIGNUP'
d.active = true

// Create Query object
q = new QUERY()
q.database = d
q.sql = 'select * from "attendees.dbf"'
q.active = true
q.rowset.indexname = 'Name'

// Find row requested in
// calling HTML form. If
// not found, send back a "sorry" page.
if not q.rowset.findKey(oCGI["LastName"])
   oCGI.sorryPage('Sorry, query not found')
endif
```

```
// Copy row field data into
// this CGISession Array
oCGI.loadArrayFromFields(q.rowset.fields)
```

Implementation Notes: Field types not supported by HTML are ignored by *loadArrayFromFields()*. Types excluded are:

- Binary
- OLE
- TimeStamp

All other types (including Autoincrement and Date) are converted to string format when stored in the CGISession array. HTML treats all data as strings. Therefore, you'll particularly want to use this when echoing data back in a CGI response page. It saves all that conversion code and time.

Errors: All errors encountered by *loadArrayFromFields()* should throw an exception to Try...Catch, which you should report using the *errorPage()* method of CGISession.

The LoadFieldsFromArray() Method

Method: loadFieldsFromArray()

Class: CGISession

File: WebClass.cc

Params: (rowsetFields, bAppend)

Description: *LoadFieldsFromArray()* updates the fields of a table to match the data received from an HTML form. It will update either existing rows or new rows, similar to "Replace Automem" and "Append Automem" in earlier versions of dBASE. It 's entirely based on the assumption that the array element "Names" (indexes, actually) match the table's field "Names". You ensure this by naming the controls on your HTML forms with the exact same names as the table fields they represent. Think of this as "datalinking" your HTML forms to your tables.

Usage:

```
// Create CGISession object
set procedure to WebClass.cc additive
oCGI = new CGISession()
oCGI.connect()

// Create Database object
d = new DATABASE()
d.databasename = 'SIGNUP'
d.active = true

// Create Query object
q = new QUERY()
q.database = d
q.sql = 'select * from "attendees.dbf"'
q.active = true

// Load the CGI data into your table
oCGI.loadFieldsFromArray(q.rowset.fields, true)
```

Implementation Notes:

To specify the row to be updated, pass a reference to its fields[ ] array as the first argument: *queryRowset.*

If you wish to append a new row, pass "true" as the second argument: *bAppend*.

All CGI data is received as strings and stored in elements of the CGISession array. You should do all your validation, calculation and reformatting using these elements. You won't go too far wrong if you treat them as string variables for purposes of data manipulation. *LoadFieldsFromArray()* will convert them back to the appropriate type, length and decimals when it updates the value of your table's fields.

You can also create your own "calculated fields" in the array before storing your CGI data to a table: In the example below, let's assume that you have "LastName", "FirstName" and "FullName" fields in your table. "FullName" is used for alphabetic lookups, so you want to store its data in the format: "LastName, FirstName":

```
// Concatenate the data received from the HTML form
oCGI["FullName"] = trim(oCGI["LastName"]+', '+;
                        oCGI["FirstName"])

// Update table from CGISession array
oCGI.loadFieldsFromArray(q.rowset.fields, true)
```

Warning!

This is a very elegant, black-box method requiring little input or code on your part. However, it does require some care in managing the data in the CGISession array. In particular, you must be very careful about case. This method is not as sensitive as some, but you must remember at all times that Associative Arrays are case sensitive. If you query oCGI["firstname"] in the example above, you'll throw an error message. I make it my policy to always match case on fieldnames and HTML control names.

You may call *loadFieldsFromArray()* over and over again, passing references to different queries and rows. Only fieldnames that match the CGISession array indexes will be updated. For example, if you have a CustNo field in both your invoice and customer

tables and a CustNo control on your HTML form, you may call *loadFieldsFromArray()* for each of them:

```
oCGI.loadFieldsFromArray(Customer1.rowset.fields, true)
oCGI.loadFieldsFromArray(Invoice1.rowset.fields, true)
```

LoadFieldsFromArray() will update the value of CustNo in both of the tables.

However, if you need to update more than one table and don't want fields of the same name updated in both, use *loadDataModuleFromArray()* instead.

Since they are not compatible with HTML, *loadFieldsFromArray()* does not support the following types:

- Binary
- OLE
- TimeStamp
- Autoincrement is automatically updated when a new row is appended.

Errors:

You don't need to be concerned about accidentally trying to update a field that doesn't exist. *loadFieldsFromArray()* works by matching. It doesn't attempt to update unless a fieldname matches an index in the CGISession array. If either one is missing, no update is attempted.

The most common error you're likely to experience using this method is a BDE database engine error. These are all trapped by Try...Catch, which you can report back to the user by calling the *errorPage()* method.

The loadDataModuleFromArray() Method

Method: loadDataModuleFromArray()

Class: CGISession

File: WebClass.cc

Params: (oDataMod, bAppend)

Description: *LoadDataModuleFromArray()* allows you to "datalink" individual controls on your HTML form to specific fields in specific queries. Pass a reference to a query's rowset fields array as the first argument: *oDataMod*. If you wish to append a new row to any rowset that gets updated, pass True as the second argument: *bAppend*.

LoadDataModuleFromArray() requires a dBASE datamodule that contains the queries you want to update with incoming CGI data. This method is similar to *loadFieldsFromArray()*, but is more closely targeted. It updates only specific fields in specific queries, regardless of how many times it's called. Therefore it's unnecessary to call it more than once.

LoadDataModuleFromArray() links to your table fields by virtue of matching names. Your HTML form controls must be named using the following convention in order for this method to work:

```
<INPUT TYPE="TEXT" NAME="Customer1*@CustNo">
```

Where Customer1 is the query name and CustNo is the fieldname. The "*@" acts as an "alias" operator, similar to:

```
Customer1->CustNo
```

LoadDataModuleFromArray() will only update exact name matches, and it's case sensitive, so be sure to take great care in naming your HTML controls.

Usage:

```
// Create CGISession object
set procedure to WebClass.cc additive
oCGI = new CGISession()
oCGI.connect()

// Create dataModule object
set procedure to MyCustomers.dmd additive
d = new MyCustomersDataModule()

oCGI.loadDataModuleFromArray(d,true)
```

Implementation Notes: Because they are not supported in HTML,the following types are also not supported by *loadDataModuleFromArray()*:

- Binary
- OLE
- TimeStamp

Errors: Don't forget to include your datamodule object code (.DMO) when you build your executable.

Datamodule errors and *loadDataModuleFromArray()* errors should all throw exceptions to your Try...Catch, which you can then return to the user by invoking *errorPage().*

The passDataThrough() Method

Method: passDataThrough()

Class: CGISession

File: WebClass.cc

Params: None

Description: *PassDataThrough()* allows you to automatically "chain" pages together. There is no "return" in the Web Browser, so you have to emulate Windows repaints by generating brand new pages in sequence that reflect the changes that have occurred along the way.

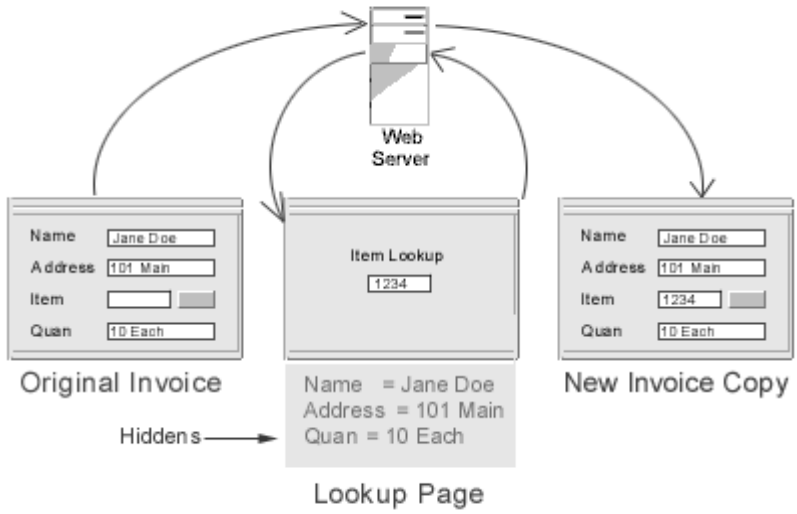
For example, let's assume we plan to deploy an e-Commerce sales site that requires the user to fill out some kind of an invoice. Let's further assume that somewhere along the way, you'll want to allow them to click an image or link and do an inventory lookup. This can only be done in HTML, at this moment, with chaining.

The process is relatively simple:

1. User clicks "Lookup" button. Page comes back to the server along with all the data the user has typed in.
2. Your dBASE applet reads the data into its CGISession array, streams out the HTML code for a lookup page and passes along the original data received from the user in the form of "hiddens" on the lookup page.
3. The user "Submits" the lookup page, which brings with it their inventory selection.
4. Your second dBASE applet (the one that's called from the lookup page), reads in the CGI data which now includes the hidden data we stored on the lookup page plus the lookup selection.

5. Your dBASE applet streams out a brand new copy of the original Invoice page, only now the data controls are filled in with the original typed-in data plus the lookup selection.

The *passDataThrough()* method makes it easy and automatic to pass the data along from page to page as required in step 2 above. You could certainly hand-code all the HTML required to pass the data through, but why bother? Instead, call *passDataThrough()*, which simply streams all the data in your CGISession array out to the new form as HTML.



Usage: *PassDataThrough()* is always used in a subclass of CGISession or one of its descendants. That's because the "hiddens" it streams must be part of the BODY of the HTML page.

```
// Create CGISession Subclass object

oCGI = new SignUpCGISession()
oCGI.connect()

// Do whatever you need to do here

// Start streaming response page
oCGI.streamHeader('Lookup')
oCGI.streamBody() // see code below
oCGI.streamFooter()

quit

////// Subclass CGISession to customize
////// streamBody() method

Class SignUpCGISession of CGISession from "WebClass.cc"

// Method to customize HTML response page

function StreamBody()

   with (this.fOut)

    // Stream out lookup form
    puts('<BODY>')
    puts('   <FORM> METHOD="POST" ACTION="lookup.exe">
    puts('   Enter Lookup')
    puts('   <INPUT TYPE="TEXT" NAME="ItemNo" SIZE=10>')
    puts('   <INPUT TYPE="SUBMIT" NAME="SUBMIT">
    puts('   </FORM>')

    // embed all data as Hiddens in this page
    // This must be called after the start BODY tag and
    // before the closing BODY tag.
    oCGI.passDataThrough()
```

```
            // Close body tag
            puts('/BODY')

         endwith

         return

      endclass
```

Errors: There should be no errors using the *passDataThrough()* method. It reads the array and streams out code. However, if something should break down, Try...Catch will trap it.

The sorryPage() Method

Method: sorryPage()

Class: CGISession

File: WebClass.cc

Params: (cMsg, cSubTtl, cRcvr)

Description: *SorryPage()* streams out a highly customizable HTML error reporting page. *SorryPage()* should be used to trap user-errors such as:

Sorry, Last Name is required!

Usage:

```
// Create CGISession object
set procedure to WebClass.cc additive
oCGI = new CGISession()
oCGI.connect()

if empty(oCGI["LastName"])
   oCGI.sorryPage('Last Name is required!','Data is missing',;
                  'Please press the Back button and try again,')
endif

...
```

Implementation: The *sorryPage()* method returns either one or a list of errors. It operates in single or batch mode. I generally prefer batch mode for validating data. That way, I can give the user a list of all the missing or incorrect data, rather than having them resubmit each time, which tries their patience and puts an unnecessary load on my server.

The *cMsg* argument may take either a character string or array of character strings. These strings represent the error messages themselves.

The *cSubTtl* argument allows you to send a subtitle to the Sorry page, such as: "Data Missing".

The *cRcvr* argument passes along instructions as to what the user should do to recover from the problem. It could be "Press the back button" or "Try again later", or even the message supported in *errorPage()*, which includes a link to the Webmaster's eMail address.

SorryPage() also serves as the base for *errorPage()*, which handles system errors.

Note: *This method just begs for subclassing! You'll certainly want to use your own colors, logo, background and layout for your own error and sorry pages. Therefore, we recommend highly that you subclass CGISession() for each application and use it to ensure a consistent look and feel across applets.*

Errors: This is a simple HTML streaming method. Any errors should be trapped by Try...Catch.

The errorPage() Method

Method: errorPage()

Class: CGISession

File: WebClass.cc

Params: (e) An error object

Description: This method is a preparation method, used to format dBASE error messages before calling *sorryPage()* to return them to the user. Unlike *sorryPage()*, it's not intended to report user errors, but rather, to report system errors. It takes as its only parameter the error object thrown by a Try...Catch...EndTry structure. It interprets that message and streams it out, along with the filename and line number on which the error ocurred. It is an invaluable aid in debugging Web applications developed under dBASE.

Usage:

```
// Create CGISession Subclass object
Try  // Always wrap your applet in a Try...Catch
   oCGI = new SignUpCGISession()
   oCGI.connect()
   ...
   ...

Catch (exception e) // if an error is thrown

   oCGI.errorPage(e) // call errorPage()

endtry
```

Implementation: If you subclass CGISession to customize your sorry page, *errorPage()* will automatically use the identical HTML response page layout.

Errors: There is one potentially huge error problem associated with *errorPage()*. If there is an error in *errorPage()*, your applet will go into an infinite loop. Each time it breaks, it will throw an error, which will call *errorPage()*, which will throw an error... well, you can take it from there. I suggest you test *sorryPage()* and *errorPage()* thoroughly before you link it into a real-live application so you don't get any ugly surprises.

The setWebMasterAddress() Method

Method: setWebMasterAddress()

Class: CGISession

File: WebClass.cc

Params: (cWebMastereMailAddress)

Description: This is a trivial method which, when used in conjunction with the default *errorPage()* method of CGISession, provides an eMail link at the bottom of the response page similar to:

Contact WebMaster for assistance.

Usage:

```
// Create CGISession object
set procedure to WebClass.cc additive
oCGI = new CGISession()
oCGI.connect()

oCGI.setWebMasterAddress(janedoe@mydomain.com)
```

Errors: There should be no errors. *SetWebMasterAddress()* simply stores a string to a property of the CGISession array object. However, make sure you send a string, not any other data type, or the default *errorPage()* may choke at runtime.

The Streaming Methods

The streaming methods of the CGISession class and its descendants ensure correct CGI and HTML formatting of your CGI response pages. They're also used internally by *errorPage()* and *sorryPage()*.

streamHeader()

Description: *StreamHeader()* sends the CGI header, the opening <HTML> tag and the <HEAD> tags of your response page out to the Web server over StdOut. Always use this method to guarantee that your CGI header is sent correctly. If you want to send a CGI header other than the standard "text/HTML", use a subclass of CGISession.

streamBody(cText)

Description: This method streams out the <BODY> tags and the core HTML code of your response page. There are two ways it can be utilized:

1. Send the body text you wish to display as the argument *cText* when calling *streamBody()*. If you use this approach, you'll need to preformat your entire HTML page as a single string with explicit line breaks (chr(13)+chr(10)). This approach is useful for small amounts of text. "Thank you" or other simple responses will give you no difficulty.
   However, for more complex HTML, such as a lookup, query or data-entry response page, we recommend that you:
2. Subclass CGISession and override the *streamBody()* method. Embed your HTML code after the puts('<BODY>') line. The *streamFooter()* method closes the <BODY> tag. One advantage of this method is that you can use **HTMLtoPRG.wfm**, the dBASE conversion applet to bring in complex HTML from visual design tools or other sources.

streamFooter()

Description: This is a relatively trivial method that simply closes your open <HTML> and <BODY> tags. Trivial, but absolutely required. If you forget to call *streamFooter()*, you may get strange results when the page is viewed in the browser.

Usage: (Includes all three Streaming Methods)

```
Try  // always use a try...catch
    // Create CGISession object
   set procedure to WebClass.cc additive
   oCGI = new CGISession()

   ....   // do processing stuff here
   ....

   // Send response page
   oCGI.streamHeader()
   oCGI.streamBody("Thank you, your profile has been updated!")
   oCGI.streamFooter()

catch(exception e) // trap errors

   oCGI.errorPage(e)

endtry

// Clean up and quit
oCGI = null
Quit
```

Implementation Notes: The streaming methods should probably be the last calls you make before terminating your applet.

The WebPWClass

Class: CGIPWSession

File: WebPWClass.cc

Description: The WebPWClass is a subclass of WebClass.cc that adds password validation to the core class functionality. This is particularly useful for private Internet/Intranet sites, membership sites or business-to-business e-Commerce sites.

Usage:

```
// Create CGIPWSession Object
set procedure to WebPWClass.cc additive
oCGI = new CGIPWSession()

// Connect to Web Server and get data
oCGI.connect()

// Confirm UserID and password. If
// It fails, it returns a sorryPage()
// and quits.
oCGI.validatePassword(cMyAlias)

/* Important Note: This method will only work if
used in an applet called from an HTML page with
two TEXT controls or HIDDENS using the following
naming convention:

<INPUT TYPE="TEXT" NAME="XPD"> // password
<INPUT TYPE="TEXT" NAME="UserID"> // User ID

Both are case sensitive.
However, this method will not crash if the data
missing. The XPD and UserID elements are pre-
defined as "blank" in the constructor of the
class. */
```

Implementation Notes: This class requires a table: users.dbf
With two fields:

UserID	Char	Len: 30
Password	Char	Len: 30

and one index:

UserKey: upper(UserID)

Upper() is used to ensure that the UserID is not case-sensitive (at the explicit request of users, who are often annoyed at having to remember case along with password). In this custom subclass, neither UserID nor Password is case-sensitive.

Where to Put the Data

You have two options for placing the Users table or database:

1. In the cgi-bin folder so that it can be accessed by your applet without the need of a path, or;
2. Anywhere on the network where it can be accessed by way of a BDE Alias. This class supports password lookups with and without a database object.

The *validatePassword()* method's sole argument (*cDatabase-Name*) is the name of a BDE Alias. If the method receives the parameter, it creates both Database and Query objects, otherwise, it creates just the query.

Security Warning: *The Password and UserID validated in this class are not in any way secure. If you're chaining the password from applet to applet, unless you're using https:// (Secure Sockets Layer) security, your passwords are being sent in plain text to the server and echoed back in the source of your HTML pages. That is not recommended.*

The HTML text control for password is named "XPD" for no particular reason other than making it just slightly harder for anyone to figure out what the data represents. At least it's better than naming it "PASSWORD".

Don't forget to include WebClass.co when you build your executable.

Part 7: Sample Applications

The dBASE Web Class Samples

The dBASE "SignUp" Sample application is only a very basic implementation of the dBASE Web Classes. Use them as a starting place and learning tool. Once you've mastered them, your own Web and Intranet applications will be limited only by your imagination.

The Three Applets:

Signup.prg	Data-Entry Processing
SignupBrowse.prg	Data Browse emulation
SignupReport.prg	dBASE Report

The Source

The source code for the dBASE Web Class samples may all be found in:

```
Visual dBASE 75\Web\Samples\Source
```

You'll find six dBASE and HTML source files in this folder that are included in the Signup application:

SignupCGIClass.cc

A Subclass of CGISession, the primary purpose of this custom class is to provide a uniform look-and-feel to Error and Sorry pages throughout the application.

Signup.htm

This is the starting HTML page for the registration applet.

Signup.prg

This is the main data-processing applet. It receives the data from the browser, validates it, appends a new row to the table and updates its fields.

SignupBrowse.prg

This is the applet that provides a rudimentary "Listbox" browse capability for viewing the Conference Attendee data.

SignupReport.prg

This is a "wrapper" to manage signupReport.rep. Its primary purpose is to provide error recovery to the dBASE report.

SignupReport.rep

This is the dBASE report that runs real-time in the Browser.

The Signup application also uses WebClass.cc, the core dBASE Web Class, which it pulls from the \Web\Classes folder at build time.

The Data

There is only one table used in the Signup application: Attendees.dbf. It normally resides in the Visual dBASE 75\Web\Samples\Source folder. When Visual dBASE 7.5 ships, it should merge a new BDE Alias, SIGNUP, which points to this folder on your machine. You should not have to adjust this Alias if you've done a "typical" install of dBASE into its default folders.

Building

There are two .prg files in the Samples\Source folder designed strictly to automate the compile and build process:

1. **BuildAllToApache.prg** compiles, builds and deploys executables directly to Apache's cgi-bin folder.
2. **BuildAllToSamples.prg** compiles, builds and deploys executables to the \Visual dBASE 75\Samples\cgi-bin folder.

To build new versions of the sample applets, just click on either of these program files in the dBASE Navigator.

Installing The dBASE Web Class Samples

The SignUp Application

The dBASE Web Class samples consist of a small suite of three applets designed for use as a remote registration application for some kind of conference.

This application (called SignUp), consists of three applets:

The SignUp Page (Data Entry)
The Browse Page (HTML List)
The List Page (dBASE report)

that demonstrate the basic capabilities of the dBASE Web Classes. They are shipped as both source and binary files. We encourage you to both run the applets and browse the source code. The SignUp application graphically demonstrates how simple it is to build impressive Web applications with dBASE.

Quick Install

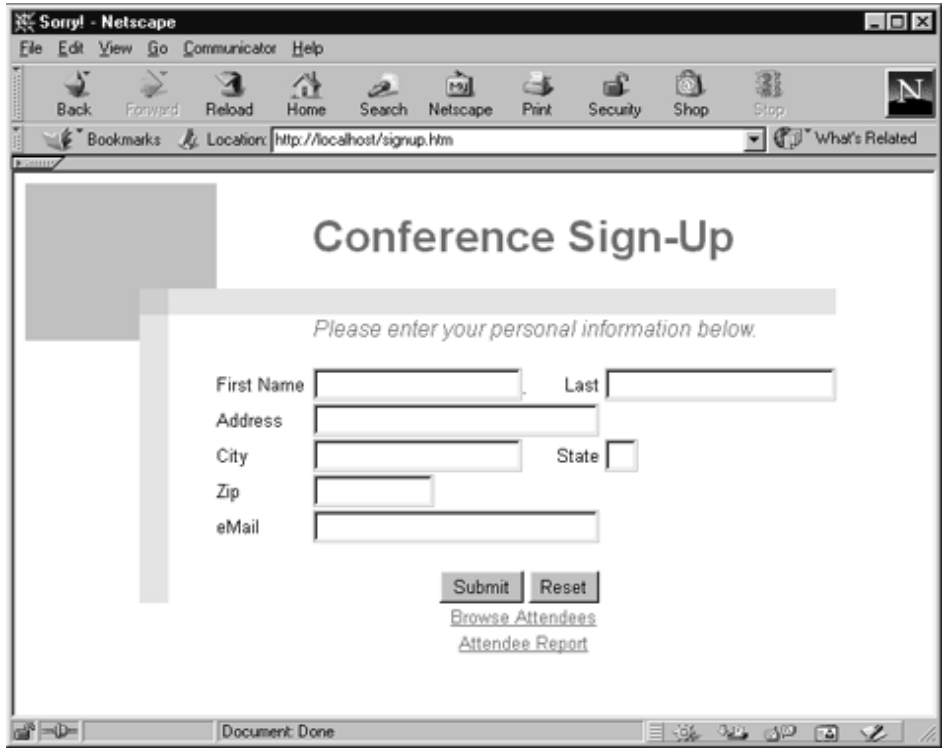
To view the SignUp application on your local computer, you need to first install the Apache Web Server that comes on the dBASE CD. That's assuming, of course, that you don't already have one installed on your local machine.

Then, go to Windows Explorer and:

1. Copy the contents of the Visual dBASE 75\Web\Samples\htdocs folder to Apache's \htdocs folder
2. Copy the contents of the Visual dBASE 75\Web\Samples\cgi-bin folder to Apache's \cgi-bin folder
3. Run VDBFast.exe from either \cgi-bin or the Web Wizards folder
4. Start Apache
5. Start up your Web browser and type the following URL:

```
http://Localhost/signup.htm
```

The SignUp data-entry page should pop up in your browser. Try submitting without filling out the fields and see how the ErrorPage reports back in batch mode. Then enter your contact information into the main form and Submit it.



To confirm your data, go back to the SignUp page and click on the "Browse Attendees" link. After viewing your data, press the browser's Back button, return to the SignUp page and click on the "Attendee Report" link to spawn a real-time dBASE report.

I trust you'll be impressed with the speed, crispness, richness and simplicity of the dBASE Web Classes and the Sample applets derived from them. If you're running on Windows 95/98, you can look forward to a performance improvement of 200-500% when you run this application on a Windows NT Server.

The dBASE Message Server Application

This dBASE application was originally written for Carik Services Inc. of Denver Colorado. Carik is a floral wire service, connecting florist-to-florist across the country and the world. When you place an order with your local florist for delivery in another state, there's a good possibility that your order is being handled by Carik Services.

Carik is also the first floral wire-service with a Web-based transmission network. Its competitors are still on older technologies based on direct dial-up modems or outdated terminals. The entire Carik Web site (www.carikonline.com) is powered by Visual dBASE.

This Message Server application polls incoming orders in the database and sends mail notifications whenever it encounters a new row. This is used by Carik to forward orders. Think of it not just as a mail/message system, but as a data-transfer system.

Do you have a remote server hosted outside of your network? Use the dBASE Message server to send each transaction to a dedicated mail account. Then, write a quick-and dirty (Carik's is a long-and fancy one) mail client application that checks for keyworded incoming mail, parses out the text and stores it in your local tables. Almost instant, hands-free long-distance data transfer. We're even using the Carik Mail Client to import data from florists' Web sites.

The dBASE Message server is offered only as a sample of where you might go in a combination of dBASE/Web and dBASE/Windows. You could always just use Windows NT mail or another simple messaging solution, but this one is tied to your tables, updates a field when sent (if you desire) and keeps a full HTML log of its activities, including any error messages sent through Try...Catch.

The dBASE Message Server is powered by the MarshallSoft® SMTP/POP3 library and handles its own Mail Server connection, queries and downloads. It does not require a Windows NT mail server. In fact, it runs just fine on any Win 95/98 machine. You can scale this application easily just by adding new installations on different computers, each of them querying the same database.

Appendix A: Source Code

In the following source code, the ↵ symbol appears wherever a line is broken to fit on the printed page. In programs, these lines would not be broken.

Signup.prg

```
/* Signup.prg    A sample program that demonstrates how to build
                 dBASE Web applications.

Author:          A. A. Katz

Version:         1.0  01/12/2000

Description:     This .prg receives input from "Signup.htm" over the
                 Web Browser and updates a table with rudimentary
                 validation. It subclasses the Visual dBASE
                 CGISession (WebClass.cc) to customize the response
                 page sent back to the user's Browser.

                 Copyright 2000, dBASE Inc.
                 You have the right to use, modify, and freely
                 redistribute this source code and any binaries
                 associated with it or derived from it.

Build:           Build signup.pro, Webclass.co to Signup.exe

*/


   Try  // Always wrap application in a Try/Catch/Endtry

      oCGI = new signUp() // create instance of Web subclass.

      oCGI.Connect()      // connect to the Web Server and
                          // load in the CGI data received.

      oCGI.setWebMasterAddress('webmaster@mysite.com')
                          // Set the Web Master eMail address in
                          // case of error
```

```
                    // Validate data received  in batch mode
aErrorList = new array() // create an array to store possible
                    // errors

//Validate required fields


if empty(oCGI["FirstName"])
   aErrorList.Add('First name is missing!')
endif

if empty(oCGI["LastName"])
   aErrorList.Add('Last name is missing!')
endif

if empty(oCGI["Address"])
   aErrorList.Add('Address is missing!')
endif

if empty(oCGI["City"])
   aErrorList.Add('City is missing!')
endif

if empty(oCGI["State"])
   aErrorList.add('State is missing')
endif

if empty(oCGI["Zip"])
   aErrorList.add('Zip code is missing!')
endif



if aErrorList.size > 0 // if there is an error
   oCGI.sorryPage(aErrorList) // Send error page and quit.
endif

// Create database object to point to tables

d = new database()
d.databaseName = 'Signup'
d.active = true
```

```
   // Create query to open Attendees.dbf

   /* Important note:  in real life, always use a BDE Aliasand a
                       database object to ensure portability!
   */

   Attendees1 = new query()
   with (Attendees1)

      database = d
      sql = 'Select * from "Attendees.dbf"'
      active = true

   endWith



   // Copy data from the oCGI array into the table row.
   // Second param tells it to create a new row.
   // Changes are saved automatically.

   oCGI.LoadFieldsFromArray(Attendees1.rowset.fields, true)


   // all data is done, now stream back a response to the user

   oCGI.StreamHeader() // Send out CGI and HTML Headers
   oCGI.StreamBody()   // send the page itself (see below)
   oCGI.StreamFooter() // Close HMTL tags


catch(Exception e)

   oCGI.Errorpage(e) // send error page if program fails

endTry


Attendees1.active = false  // clean up the leftovers
Attendees1 = null          // just as insurance to guarantee
oCGI = null                // resources are never lost!


Quit                       // All done, close down.
```

The SignUp Sample HTML Page

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>

<HEAD>
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html; ↵
                                    CHARSET=iso-8859-1">
   <META NAME="GENERATOR" Content="Visual dBASE">
   <TITLE>Conference Signup</TITLE>
</HEAD>

<BODY LINK="#009999" VLINK="#009999">

<FORM ACTION="/cgi-bin/Signup.exe" ↵
      METHOD="POST" ENCTYPE="application/x-www-form-urlencoded">
<P>
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
   <TR>
      <TD WIDTH="83" HEIGHT="77" BGCOLOR="silver"> </TD>
      <TD WIDTH="21" HEIGHT="77" BGCOLOR="silver"> </TD>
      <TD WIDTH="35" HEIGHT="77" BGCOLOR="silver"> </TD>
      <TD WIDTH="70" HEIGHT="77" BGCOLOR="white"> </TD>
      <TD HEIGHT="77" COLSPAN="3" BGCOLOR="white">
         <H1><FONT COLOR="#009999" FACE="Arial, Helvetica"> ↵
         Conference Sign-Up</FONT></TD>
   </TR>
   <TR>
      <TD WIDTH="83" BGCOLOR="silver"> </TD>
      <TD WIDTH="21" BGCOLOR="#CECECE"> </TD>
      <TD WIDTH="35" BGCOLOR="#E4E4E4"> </TD>
      <TD WIDTH="70" BGCOLOR="#E4E4E4"> </TD>
      <TD COLSPAN="3" BGCOLOR="#E4E4E4"> </TD>
   </TR>
   <TR>
      <TD WIDTH="83" BGCOLOR="silver"> </TD>
      <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
      <TD WIDTH="35"> </TD>
      <TD WIDTH="70"> </TD>
      <TD COLSPAN="3"><I>
```

```
      <FONT COLOR="#009999" FACE="Arial, Helvetica">
      Please enter your personal information below.
      </FONT></I></TD>
</TR>
<TR>
   <TD WIDTH="83"> </TD>
   <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
   <TD WIDTH="35"> </TD>
   <TD WIDTH="70"> </TD>
   <TD COLSPAN="2"> </TD>
   <TD> </TD>
</TR>
<TR>
   <TD WIDTH="83"> </TD>
   <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
   <TD WIDTH="35"> </TD>
   <TD WIDTH="70">
      <FONT SIZE="2" FACE="Arial, Helvetica">
      First Name</FONT></TD>
   <TD WIDTH="164">
      <INPUT TYPE="TEXT" NAME="FirstName" SIZE="18"
            MAXLENGTH="15"></TD>
   <TD WIDTH="48">
      <P ALIGN="RIGHT">
         <FONT SIZE="2" FACE="Arial, Helvetica">
         Last</FONT></TD>
   <TD>
      <INPUT TYPE="TEXT" NAME="LastName" SIZE="20"
            MAXLENGTH="20"></TD>
</TR>
<TR>
   <TD WIDTH="83">  </TD>
   <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
   <TD WIDTH="35"> </TD>
   <TD WIDTH="70">
      <FONT SIZE="2" FACE="Arial, Helvetica">Address</FONT></TD>
   <TD COLSPAN="2">
      <INPUT TYPE="TEXT" NAME="Address" SIZE="25"
            MAXLENGTH="35"></TD>
   <TD> </TD>
</TR>
<TR>
   <TD WIDTH="83"> </TD>
   <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
   <TD WIDTH="35"> </TD>
```

```
      <TD WIDTH="70">
         <FONT SIZE="2" FACE="Arial, Helvetica">City</FONT></TD>
      <TD WIDTH="164">
         <INPUT TYPE="TEXT" NAME="City" SIZE="18" MAXLENGTH="20"></TD>
      <TD WIDTH="48">
         <P ALIGN="RIGHT">
         <FONT SIZE="2" FACE="Arial, Helvetica">State</FONT></TD>
      <TD>
         <INPUT TYPE="TEXT" NAME="State" SIZE="2" MAXLENGTH="2"></TD>
   </TR>
   <TR>
      <TD WIDTH="83"> </TD>
      <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
      <TD WIDTH="35"> </TD>
      <TD WIDTH="70">
         <FONT SIZE="2" FACE="Arial, Helvetica">Zip</FONT></TD>
      <TD WIDTH="164">
         <INPUT TYPE="TEXT" NAME="Zip" SIZE="10" MAXLENGTH="10"></TD>
      <TD WIDTH="48"> </TD>
      <TD> </TD>
   <TR>
      <TD WIDTH="83">  </TD>
      <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
      <TD WIDTH="35"> </TD>
      <TD WIDTH="70">
         <FONT SIZE="2" FACE="Arial, Helvetica">eMail</FONT></TD>
      <TD COLSPAN="2">
         <INPUT TYPE="TEXT" NAME="eMailAddress" SIZE="25" ↵
                MAXLENGTH="35"></TD>
      <TD> </TD>
   </TR>
   <TR>
      <TD WIDTH="83"> </TD>
      <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
      <TD WIDTH="35"> </TD>
      <TD WIDTH="70"> </TD>
      <TD WIDTH="164"> </TD>
      <TD WIDTH="48"> </TD>
      <TD> </TD>
   </TR>
   <TR>
      <TD WIDTH="83"> </TD>
      <TD WIDTH="21" BGCOLOR="#E4E4E4"> </TD>
      <TD WIDTH="35"> </TD>
      <TD WIDTH="70"> </TD>
```

```
        <TD COLSPAN="2">
            <DIV ALIGN="RIGHT">
                <P> 
                <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit">
                <INPUT TYPE="RESET" NAME="Reset" VALUE="Reset">
            </DIV>
        </TD>
        <TD> </TD>
    </TR>
    <TR>
        <TD WIDTH="83"> </TD>
        <TD WIDTH="21"> </TD>
        <TD WIDTH="35"> </TD>
        <TD COLSPAN="4">
            <P ALIGN="CENTER">
            <A HREF="http://localhost/cgi-bin/SignupBrowse.exe">
            <FONT SIZE="2" FACE="Arial, Helvetica">↵
            Browse Attendees</FONT></A>
        </TD>
    </TR>
    <TR>
        <TD WIDTH="83"> </TD>
        <TD WIDTH="21"> </TD>
        <TD WIDTH="35"> </TD>
        <TD COLSPAN="4">
            <P ALIGN="CENTER">
            <A HREF="http://localhost/cgi-bin/SignupReport.exe">
            <FONT SIZE="2" FACE="Arial, Helvetica">↵
            Attendee Report</FONT></A>
        </TD>
    </TR>
</TABLE>

</FORM>

</BODY>

</HTML>
```

Signup Sample CGI Response Code

```
////// Class:    SignUp ////////////////////////////////////
////// Purpose:  Customize class CGISession (Webclass.cc) ////
//////           by subclassing ////////////////////////////

/* Note:      the only change in this subclass is the following method,
              StreamBody, which streams out the bulk of the code that
              is returned to the user.
*/

Class signUp of signUpCGISession from "signUpCGI.cc"

   // This HTML was generated using Symantec's Visual Page™
   // and then copied and pasted. Use the VdB untility
   // "HTMLtoPRG.wfm" in the \Web Classes folder to copy generated
   // pages in automatically with parens and delimiters.
   // Symantec and Visual Page are trademarks of Symantec Corp.

   Function streamBody

      this.fOut.Puts('<BODY LINK="#009999" VLINK="#009999">')
      this.fOut.Puts('<P>')
      this.fOut.Puts('<TABLE BORDER="0" CELLPADDING="0" ↵
                             CELLSPACING="0" WIDTH="100%">')
      this.fOut.Puts('<TR>')
      this.fOut.Puts('   <TD WIDTH="13%" HEIGHT="77" ↵
                             BGCOLOR="silver"> </TD>')
      this.fOut.Puts('   <TD WIDTH="3%" HEIGHT="77" ↵
                             BGCOLOR="silver"> </TD>')
      this.fOut.Puts('   <TD WIDTH="4%" HEIGHT="77" ↵
                             BGCOLOR="silver"> </TD>')
      this.fOut.Puts('   <TD WIDTH="7%" HEIGHT="77" ↵
                             BGCOLOR="white"> </TD>')
     this.fOut.Puts('   <TD WIDTH="73%" HEIGHT="77" BGCOLOR="white">')
      this.fOut.Puts('   <H1><FONT COLOR="#009999" ↵
                               FACE="Arial, Helvetica"> ↵
                               Thank You</FONT>')
      this.fOut.Puts('</TD>')
      this.fOut.Puts('</TR>')
      this.fOut.Puts('<TR>')
```

```
this.fOut.Puts('<TD WIDTH="13%" BGCOLOR="silver"> </TD>')
this.fOut.Puts('<TD WIDTH="3%" BGCOLOR="#CECECE"> </TD>')
this.fOut.Puts('<TD WIDTH="4%" BGCOLOR="#E4E4E4"> </TD>')
this.fOut.Puts('<TD WIDTH="7%" BGCOLOR="#E4E4E4"> </TD>')
this.fOut.Puts('<TD WIDTH="73%" BGCOLOR="#E4E4E4"> </TD>')
this.fOut.Puts('</TR>')
this.fOut.Puts('<TR>')
this.fOut.Puts('<TD WIDTH="13%" BGCOLOR="silver"> </TD>')
this.fOut.Puts('<TD WIDTH="3%" BGCOLOR="#E4E4E4"> </TD>')
this.fOut.Puts('<TD WIDTH="4%"> </TD>')
this.fOut.Puts('<TD WIDTH="7%"> </TD>')

// Here I've inserted live data into the response page...
this.fOut.Puts('<TD WIDTH="73%"><I><FONT COLOR="#009999" ↵
                FACE="Arial, Helvetica">' + ;
                this["FirstName"] + ' ' + ;
                this["LastName"] + '</FONT></I></TD>')
this.fOut.Puts('</TR>')
this.fOut.Puts('<TR>')
this.fOut.Puts('<TD WIDTH="13%"> </TD>')
this.fOut.Puts('<TD WIDTH="3%" BGCOLOR="#E4E4E4"> </TD>')
this.fOut.Puts('<TD WIDTH="4%"> </TD>')
this.fOut.Puts('<TD WIDTH="7%"> </TD>')
this.fOut.Puts('<TD WIDTH="73%"><I><FONT COLOR="#009999" ↵
                FACE="Arial, Helvetica"> ↵
                You are now registered.</FONT></I> </TD>')
this.fOut.Puts('</TR>')

for n = 1 to 7 // repeat blank lines
   this.fOut.Puts('<TR>')
   this.fOut.Puts('<TD WIDTH="13%"> </TD>')
   this.fOut.Puts('<TD WIDTH="3%" ↵
                       BGCOLOR="#E4E4E4"> </TD>')
   this.fOut.Puts('<TD WIDTH="4%"> </TD>')
   this.fOut.Puts('<TD WIDTH="7%"> </TD>')
   this.fOut.Puts('<TD WIDTH="73%"> </TD>')
   this.fOut.Puts('</TR>')
next

this.fOut.Puts('<TR>')
this.fOut.Puts('<TD WIDTH="13%"> </TD>')
this.fOut.Puts('<TD WIDTH="3%" BGCOLOR="#E4E4E4"> </TD>')
this.fOut.Puts('<TD WIDTH="4%"> </TD>')
this.fOut.Puts('<TD WIDTH="7%"> </TD>')
```

```
      this.fOut.Puts('<TD WIDTH="73%"><A HREF="/signup.htm">
                          <FONT COLOR="#009999"
                           FACE="Arial, Helvetica" SIZE="2">
                          Home</FONT></A></TD>')

      this.fOut.Puts('</TR>')

      this.fOut.Puts('</TABLE>')
      this.fOut.Puts('</BODY>')

return true

endClass
```

SignupCGISession Subclass

```
/* signupCGI.cc  This is a subclass of CGISession.cc, the dBASE Web
                 Class. It adds custom formatting to the "Sorry"
                 page to match the Visual dBASE 7.5 Conference
                 Signup sample application.

Usage            oCGI = new sampleCGISession()

Author:          A. A. Katz  01/17/2000  dBASE Inc.

*/


Class SignupCGISession of CGISession from "..\..\Webclass.cc"

////// Method:   SorryPage ////////////////////////////////////
////// Purpose:  Returns Sorry page in response to user ///////////
////// Param:    cMsg = Error Message ////////////////////////////
//////                 Accepts a char string or array of messages
//////           cSubTtl = Subtitle, such as "An error has ocurred" //
////                 or "data missing" ////////////////////////////////
cRecover = Suggestion for recovery ////////////////////such as "Press
the browser's back button" ///////or contact Web master..

Function SorryPage(cMsg,cSubTtl,cRcvr)

   // This code is the same as CGISession
   cMess = iif(empty(cMsg),'',cMsg)

   if type('cMess') = 'C'   // if message is not array,
      cMess = new array()   // convert to a one-element array
      cMess.add(cMsg)
   endif
                            // Make private versions for type()
   cRecover =  iif(empty(cRcvr),'',cRcvr)
   cSubtitle = iif(empty(cSubTtl),'',cSubTtl)


   /////// Stream out header
   this.StreamHeader('Sorry!')
```

```
//------------- This is the customized code ---------\\

with (this.fOut)

   ////// Body tag (starts body of page)
   puts('<BODY LINK="#009999" VLINK="#009999">')

   ////// Table is used to format color and text

   ////// First row of table with "Sorry" display.
   puts('<P>')
   puts('<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" ↵
             WIDTH="100%">')
   puts('   <TR>')
   puts('   <TD WIDTH="13%" HEIGHT="77" ↵
             BGCOLOR="silver"> </TD>')
   puts('   <TD WIDTH="3%" HEIGHT="77" ↵
             BGCOLOR="silver"> </TD>')
   puts('   <TD WIDTH="4%" HEIGHT="77" ↵
             BGCOLOR="silver"> </TD>')
   puts('   <TD WIDTH="7%" HEIGHT="77" ↵
             BGCOLOR="white"> </TD>')
   puts('   <TD WIDTH="73%" HEIGHT="77" BGCOLOR="white">')
   puts('      <H1><FONT COLOR="#009999" ↵
                    FACE="Arial, Helvetica">Sorry!</FONT>')
   puts('   </TD>')
   puts('</TR>')

   ////// Narrow gray horizontal stripe
   puts('<TR>')
   puts('   <TD WIDTH="13%" BGCOLOR="silver"> </TD>')
   puts('   <TD WIDTH="3%" BGCOLOR="#CECECE"> </TD>')
   puts('   <TD WIDTH="4%" BGCOLOR="#E4E4E4"> </TD>')
   puts('   <TD WIDTH="7%" BGCOLOR="#E4E4E4"> </TD>')
   puts('   <TD WIDTH="73%" BGCOLOR="#E4E4E4"> </TD>')
   puts('</TR>')

   ////// If there is a "Subtitle", such as
   ////// "An error has ocurred on the Server!"
   if len(cSubtitle) > 0
      ///// Page subtitle
      puts('<TR>')
      puts('   <TD WIDTH="13%" BGCOLOR="silver"> </TD>')
      puts('   <TD WIDTH="3%" BGCOLOR="#E4E4E4"> </TD>')
      puts('   <TD WIDTH="4%"> </TD>')
```

```
   puts('   <TD WIDTH="7%"> </TD>')
   puts('   <TD WIDTH="73%"><I><FONT COLOR="#009999" ↵
               FACE="Arial, Helvetica">' ;
                + cSubtitle + '</FONT></I></TD>')
   puts('</TR>')
endif

////// Empty row for spacing
puts('<TR>')
puts('   <TD WIDTH="13%" BGCOLOR="silver"> </TD>')
puts('   <TD WIDTH="3%" BGCOLOR="#E4E4E4"> </TD>')
puts('   <TD WIDTH="4%"> </TD>')
puts('   <TD WIDTH="7%"> </TD>')
puts('  <TD WIDTH="73%"> </TD>')
puts('</TR>')

for n = 1 to cMess.size
   puts('<TR>')
   puts('   <TD WIDTH="13%"> </TD>')
   puts('   <TD WIDTH="3%" BGCOLOR="#E4E4E4"> </TD>')
   puts('   <TD WIDTH="4%"> </TD>')
   puts('   <TD WIDTH="7%"> </TD>')
   puts('  <TD WIDTH="73%">'+cMess[n]+'</TD>')
   puts('</TR>')
next

////// Empty row for spacing
puts('<TR>')
puts('   <TD WIDTH="13%"> </TD>')
puts('   <TD WIDTH="3%" BGCOLOR="#E4E4E4"> </TD>')
puts('   <TD WIDTH="4%"> </TD>')
puts('   <TD WIDTH="7%"> </TD>')
puts('  <TD WIDTH="73%"> </TD>')
puts('</TR>')

////// Stream out recovery instructions
cRecover = iif(empty(cRecover), ;
             "Press Browser's Back Button and try again.", ;
               cRecover)
puts('<TR>')
puts('   <TD WIDTH="13%"> </TD>')
puts('   <TD WIDTH="3%" BGCOLOR="#E4E4E4"> </TD>')
puts('   <TD WIDTH="4%"> </TD>')
puts('   <TD WIDTH="7%"> </TD>')
```

```
        puts('   <TD WIDTH="73%"><I><FONT COLOR="#009999"
                    FACE="Arial, Helvetica">' ;
                    + cRecover + '</FONT></I></TD>')
        puts('</TR>')

        ////// Close table
        puts('</Table>')


    endwith

    ////// Close out HTML structure
    this.StreamFooter()

    quit

endClass
```

Appendix B: Equivalents

HTML Controls and dBASE Equivalents

HTML	dBASE
TEXT	Entryfield
TEXT AREA	Editor
<SELECT> <Option >One</Option> <Option>Two</Option> </SELECT>	Combobox
<LIST> <Option> One </Option > <Option> Two </Option> </LIST>	Listbox
RADIOBUTTON	Radiobutton
BUTTON	PushButton
SUBMIT	Default Pushbutton
RESET	Clear Pushbutton
PASSWORD	Asterisk Entryfield

Appendix C: Glossary

The following is a short glossary of terms that will be useful to know when developing Web applications with the Visual dBASE Web Wizards.

Applet

A single-purpose small Web executable. Similar to a single form or module in Windows applications. Web applications are developed using small programs that generate and read single HTML pages at a time.

Application

A collection of HTML pages, applets, data and images that are chained together to perform complex business tasks.

CGI

Standards had to be established In order for various Web Servers to be able to respond to requests from Browsers by running an application. CGI is the protocol that describes how the Web Server will receive data from the Browser, call the application, send the data to the application and then return an HTML response page to the Browser. There are two major implementations of CGI: CGI-Win and CGI-Bin. CGI-Win is optimized for Windows but is not available on all servers and is very small. CGI-Bin is the more general standard and is supported by most popular Web Servers. dBASE supports only CGI-Bin.

CGI Header

The CGI header is normally a single line of instructions returned from your Web application that tells the server what to do when the application is complete:

This is the CGI header used when an application returns a dynamically generated HTML page:

```
Content-type: text/HTML
```

This is the CGI header used when an application wants the server to return an already-existing page:

```
Location: Myfile.htm
```

HTML

HTML (Hypertext Markup Language) is the page-description language used by most Web Servers and Browsers. It's an international standard for commands (called tags), symbols and conventions used to create and display Web pages. HTML is designed specifically to generate readable formatted text and graphics regardless of the platform on which the Browser is run. As such, HTML is resolution-independent (will reformat text to fit any size Browser running on any video resolution) and platform-independent (it runs on any computer that runs a Browser).

HTTP

HTTP (Hypertext Transfer Protocol) is the protocol used for communicating HTML Web pages over a TCP/IP network. Just as Xmodem and Zmodem are protocols used for moving data across phone lines, HTTP is a set of data-transfer conventions (protocols) dedicated to moving HTML pages across the Internet or Intranet.

Mapping

Web Servers use virtual folders (as opposed to real subdirectories) in order to locate pages and programs. By using virtual folders, locations in a Web site can be relative to the home folder instead of hard-wired to the directories on your hard disk. That way, you can have any number of sites, each with its own home folder on a single server, or across multiple servers. When you *map* a folder, you're just telling the Web Server: this virtual folder is assigned to this hard-disk directory on this LAN server.

Let's use our CGI folder as an example. On your hard drive, that folder may be:

```
C:\WebStuff\Home\Applications\Vdb\CGI\myprog.exe
```

The path above is much to complex to use in a URL. So, instead, we map this path to a virtual folder: /CGI/

So, instead of using the real address:

```
http://www.dBASE.com/C://webstuff/home/applications/vdb/cgi/myprog.exe
```

we can use:

```
http://www.dBASE.com/cgi/myprog.exe
```

Not only is the above address infinitely simpler, it is also *portable*. You can move the files that comprise this address to another folder or another server without altering the address just by changing the mapping in the Web Server to point to the new location.

One other purpose of mapping is that Web Servers allow you to define functionality when you create a new virtual folder. That's important to our purposes. When you map your CGI folder for your Visual dBASE Web Wizard applications, you will tell it to "run" rather than "read" the files in this folder. Files called from your CGI folder will be executed instead of down-loaded.

Response Page

The response page (or CGI response page) is HTML that your dBASE applet sends back to the Web Server for display in the Browser. It may be as simple as a "Thank you" page or as complex as a chained data-entry page. Every call to a Web application requires that a CGI response page be returned to the Browser. If you omit the CGI response page, the Browser will display an error.

URL

A URL (Universal Resource Locator) is the Internet address of a specific page or folder on a Web Site. It consists of the following parts:

Example:

Protocol	Host	Path
http:	www.dBASE2000.com	/vdb/vdb7.htm

The URL: `http://www.dBASE2000.com/vdb/vdb7.htm`

The Visual dBASE Web Wizards will ask for both a CGI folder and a CGI URL. The folder is the actual location on your network to which the Web Wizards will copy your files. The URL is the Internet address that should be used to call the Web application.

Example:

Folder: `F:\Site\CGI`

URL: `http://www.=dBASE.com/cgi`

Note that URLs use the "/" convention of UNIX rather than the "\" symbol used in DOS and Windows paths.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W