



## Web Application Server Manual

**Version: 2.0**

Revised: April 1998

Pathfinders Software  
1755 O'Farrell, Suite 905  
San Francisco, CA 94115 USA

Sales: (415) 292-4935  
Fax: (415) 292-4913

sales@pfinders.com  
support@pfinders.com

# Table Of Contents

<b>Overview .....</b>	<b>3</b>
<b>Configuration on the Web Server .....</b>	<b>4</b>
Installation .....	4
Script Parameters .....	4
<b>Configuration on the D3 / AP Side .....</b>	<b>5</b>
Installation .....	5
File Inventory .....	5
The PROPERTIES file .....	5
Starting the Phantom Web Application Servers .....	6
<b>Incoming HTTP Interface .....</b>	<b>7</b>
Example Environment Variables .....	7
A Note About PATH_INFO .....	7
Example Form Data .....	7
<b>Outgoing HTTP Interface .....</b>	<b>8</b>
HTTP Header .....	8
Setting A Cookie .....	8
Retrieving A Cookie .....	9
Other Session Ideas .....	9
<b>Tools .....</b>	<b>10</b>
FORMATHTML .....	10
MAKETABLE .....	11
The Big Demo .....	20
<b>Sample Source Code .....</b>	<b>13</b>
The Standard Subroutine Template .....	13
Your First Subroutine .....	13
The Login Subroutine .....	14
Setting A Cookie .....	16
<b>Appendix A: Debugging .....</b>	<b>17</b>
Server Error Encountered .....	17
The LOGFILE .....	17
The WHERE command .....	17
Script Not Associated With Subroutine .....	18
Could not connect to pick.host.com at this time .....	17
Could not connect to pick.host.com, no available ports. ....	18
<b>Appendix B: Environment Variables .....</b>	<b>20</b>
Standard ENV variables .....	20
<b>Appendix C: The Big Demo .....</b>	<b>21</b>

# Overview

## *The Web Application Server*

Pathfinders Software's Web Application Server allows a web server to talk to Pick Systems D3 or Advanced Pick 6.1 running on a UNIX based platform. Data is transferred using the CGI standard across a TCP/IP socket.

The server program connects to Pick on a high socket usually in the range of 5100 to 5105 and transfers all form data and environment variables to the Web Application Server running as a phantom process. This information is then passed to a subroutine for final processing.

Running on a range of socket ports allows the Pick machine to service many requests simultaneously without having to be a multi-threaded application or using up any Pick licenses. The more phantoms you run, the more simultaneous processing can be accomplished.

This system was tested in Linux and AIX environments running UNIX and NT based Web servers. It has been tested across firewalls as well as running the entire system on a single machine (web server and Pick).

The Web Application Server will attempt to call a subroutine you provide. It expects that you will return either a Pick variable with HTML code in it or template data. Both are passed back through the socket to the web server for display on the users browser. If template data is returned, then the template is expected to be found on the web server. Any template data is "filled in" before being displayed.

# Configuration On The Web Server

## *Installation*

There are two diskettes labeled "Web Application Server 2 of 2". One is in UNIX tar format and the other is in DOS format. They both have the same Perl Script on them. The program is called "server". Copy it into your executable directory (cgi-bin) or rename it with the ".cgi" extension to enable execution.

You may copy it into many different directories and even change its name. It does not matter. Some web masters have renamed it "catalog", or "login" or even "session". The name is only a way to associate what Pick BASIC subroutine to call. Different scripts can call different subroutines on the Pick machine.

The most common place to copy the script into is the "/cgi-bin" directory on most UNIX web servers.

## *Script Parameters*

The script has some user defined parameters you must edit.

```
$pickhostname      = "www.yourhost.com"  
$startingport      = "4100"  
$sendingport       = "4105"  
$sevenload         = "1"
```

Do not change the port numbers unless you make corresponding changes on the Pick side.

## *Evenload*

This parameter controls whether a random socket port is chosen in the range \$startingport to \$sendingport or whether it starts from the first port.

1 = random port in range (normal deployment)

0 = start from \$startingport (usually used for testing purposes)

When testing, set \$sevenload to 0, and you only have to run a single phantom port on the D3 or AP side to interface with the web server.

When you deploy your WEB APPLICATION server set \$sevenload to 1 and startup all D3 or AP phantom processes and they will all be used for service equally, splitting the load.

# Configuration on the D3 / AP Side

## *Installation*

1) Log to the account you wish to install the Web Application Server into.

You can either create a new account or load the software into your application's account. If you load it into its own account you may have to create Q pointers later to point back to your important database files.

2) Insert the floppy diskette labeled Web Application Server 1 of 2 and SET-DEVICE to the floppy diskette. Then type:

```
T-LOAD MD (O
```

A single program will load called INSTALL.

3) Run this program by typing:

```
INSTALL
```

The rest of the floppy programs will install themselves.

4) Edit the items in the PROPERTIES file.

## *File Inventory*

After installation you will notice that the following files were created:

PS.BP	Basic Program File for Pathfinders Software
PROPERTIES	Where important initialization records are kept
LOGFILE	For debugging and tracking server activity

## *The PROPERTIES file*

You must edit three important records in this file.

### **The FILES record**

This record lists all of YOUR files that you will use during normal operation. The Web Application Server will open all of the files when it starts up. Then when it calls your subroutine it passes all the file descriptors for your use. This saves valuable time as your subroutine is called over and over again. It is passed as a dimensioned array and currently set to a maximum of 30 filenames.

### **The SCRIPTS record**

This record lists all of your web scripts and what Pick BASIC subroutine they should call. Value 1 of each attribute is the path of the script and value 2 is the cataloged subroutine name to call.

```
001 /cgi-bin/server]LOGIN
002 /catalog/search]SEARCHSUB
```

If there is no association between a calling script and a Pick BASIC subroutine an error message is returned to the web browser.

### **The INDEXES record**

This record is similar to the FILES record. It lists all the Pick indexes to open up using the ROOT statement in BASIC during initialization. The list of root descriptors is passed to your Pick BASIC program each time the Web Application Server calls it. This saves you time and you do not have to open up the indexes in your subroutine over and over again. The format of this file is value 1 is the filename and value 2 is the A correlative to open up.

```
001 CUSTOMERS]A0
002 INVOICES]A3
003 INVOICES]A9
```

## *Starting The Phantom Servers*

SERVER {socketport} {(options)}

The only option available is the L option for logging activity to the LOGFILE. The Web Application servers are perfectly happy to run in the background as phantom processes. You can run as many servers as you wish, simultaneously. The only thing to be aware is that the script on the web server must know how many. The default is 1 to 6. The common way to do this is with the following commands:

```
Z SERVER 4100
Z SERVER 4101
Z SERVER 4102
Z SERVER 4103
Z SERVER 4104
Z SERVER 4105
```

In the above example 6 phantom processes are logged on and each will listen on its own socket port from 4100 - 4105.

If you are debugging, then you might want to use the (L option so it writes its activities to the LOGFILE.

```
Z SERVER 4100 (L
```

# Incoming HTTP Interface

When a web server calls a CGI program it passes it Environment variables and form data. This information is captured and passed to your subroutine by the Web Application Server program. Both the Environment variables and Form data are simply name and value pairs. This data is passed as multivalued attributes. Value 1 is the name and value 2 is the value. See Appendix B for full details.

Form data behaves the same way. All form data is name and value pairs also. They are passed to the Pick host and your subroutine the same way as multivalued attributes.

## Example Environment Variables

```
SCRIPT_NAME]/cgi-bin/server  
PATH_INFO]/456776/login  
QUERY_STRING]n=fred&p=xjy5  
REQUEST_METHOD]GET  
REMOTE_HOST]192.116.43.81
```

## Example Form Data

```
NAME]bob  
ADDRESS]100 Main St.  
CITY]New York City  
EMAIL]bob@global.com
```

## *A Note About PATH\_INFO*

This field is additional directory path information that is appended to the calling script. A web server walks down the path looking for the file or the executable program. Once it finds what it is looking for it stops. Then it takes any remaining directory path information and assigns it to the variable PATH\_INFO. In the above example the web page request could have been:

```
http://www.host.com/cgi-bin/server/456776/login?n=fred&p=xjy5
```

You can always tell if there is form data because the REQUEST\_METHOD will always be POST.

# Outgoing HTTP Interface

## *HTTP Header*

When returning HTML back from your subroutine, you are also responsible to prefix it with the proper HTTP header. This gives a truly proper HTTP response.

Example:

```
Content-type: text/html

<html>
<head>
</head>
<body>
Hello World!
</body>
</html>
```

Notice the blank line after the header. This is required.

## *Setting A Cookie*

To set a cookie, you simply put another line in the header with the cookie information.

Example:

```
Content-type: text/html
Set-cookie: Session=456783

<html>
<head>
</head>
<body>
Hello World!
</body>
</html>
```

## *Retrieving A Cookie*

All cookies are stored on the users web browser. Any valid cookies that are part of the current session are passed back to the web server during the web page request. This cookie information is stored in an environment variable and available in your Pick BASIC program.

**Example:**

```
HTTP_COOKIE]Session=456783
```

## *Other Session Ideas*

Since maintaining state is the most important idea in an application you may wish to use cookies, however if you wish to do it another way, you can hide the state or session id in a form by using hidden fields.

**Example:**

```
<FORM METHOD="POST" ACTION="http://www.host.com/cgi-bin/server">  
<INPUT TYPE=HIDDEN NAME=SESSION VALUE=4453215>  
</FORM>
```

Now, when the form is submitted, you will have SESSION]4453215 as additional information.

Another way to accomplish this is to use the PATH\_INFO environment variable

**Example:**

```
<FORM METHOD="POST" ACTION="http://www.host.com/cgi-bin/server/4453215">  
</FORM>
```

Now, when the form is submitted you will have PATH\_INFO]/4453215 and you simply strip off the leading "/".

The final way to maintain state is with command line arguments.

**Example:**

```
<FORM METHOD="POST" ACTION="http://www.host.com/cgi-bin/server?4453215">  
</FORM>
```

Now, when the form is submitted, you will have QUERY\_STRING]4453215

## Tools

### *FORMATHTML(REPLY)*

This subroutine takes a Pick BASIC variable and strips out the attribute marks and replaces them with CRLF sequences. Char(13), Char(10). It is the final preparation before sending back the HTML code to the web server. This subroutine should be called after your HTML is ready to go.

#### Example

```
SUBROUTINE WELCOME(FILELIST, ROOTLIST, FORM, ENV, REPLY)
*
REPLY = "Content-type: text/html"
REPLY<-1> = "; * blank line required by HTTP"
REPLY<-1> = "<html>"
REPLY<-1> = "<HEAD></HEAD>"
REPLY<-1> = '<BODY BGCOLOR="FFFFFF" TEXT="000000">'
REPLY<-1> = "<H1>Welcome</H1>"
REPLY<-1> = "</BODY></HTML>"
CALL FORMATHTML(REPLY)
RETURN
```

## *MAKETABLE(FILEDESCR, ITEMID, REPLY)*

This subroutine reads a control record (item) and uses it to generate an HTML table. This HTML code is returned back in the variable REPLY. It is not a complete web page, just a table.

The format of the control record is:

001 Heading HTML to appear before the table  
002 Parameters for Table Tag, such as BORDER=0 CELLSPACING=5 WIDTH=85%  
003 col1 width]col2 width] col3 width, etc.  
004 table data  
005 table data  
006 table data  
007 etc.....

Note: Attribute 3 is in pixels, blank values will automatically adjust the table around the data.

### *Example Control Record for Table Generation*

```
:CT TABLES MEDIA  
  
    MEDIA  
001 <h3>Media Catalog</h3>  
002 border=0  
003 120]120]120  
004 Film - Motion]/session/catalog/filmmotion.html  
005 Film - Still]/session/catalog/filmstill.html  
006 Video]/session/catalog/videotape.html  
007 Audio]/session/catalog/audiotape.html  
008 Data Cartridge]/session/catalog/data.html  
009 Accessories]/session/catalog/media.html
```

The above control record will generate a 3 by 2 cell table. Each column will be 120 pixels wide.

Below is a sample program using this subroutine. The program is also a subroutine that is normally called automatically by the Web Application Server as needed.

Notice in line 11, it gets the table HTML from the subroutine. Then in lines 13 to 21 it wraps this HTML with the full HTML web page syntax before returning it back to the browser.

```

001 SUBROUTINE CATALOG(FDLIST ,ROOTLIST, FORM, ENV, REPLY)
002 *-----*
003 * Return the media catalog in HTML format *
004 *-----*
005 MAXFILES = 30
006 DIM FDLIST(MAXFILES)
007 DIM ROOTLIST(MAXFILES)
008 AM = CHAR(254)
009 CR = CHAR(13)
010 TABLESFILE = 3
011 CALL MAKETABLE(FDLIST(TABLESFILE),"MEDIA", MEDIAHTML)
012 REPLY = "Content-type: text/html"
013 REPLY<-1> = ""
014 REPLY<-1> = "<HTML>"
015 REPLY<-1> = "<HEAD>"
016 REPLY<-1> = "<TITLE>On-Line Catalog</title>"
017 REPLY<-1> = "</HEAD>"
018 REPLY<-1> = '<BODY BGCOLOR="FFFFFF">'
019 REPLY = REPLY:AM:MEDIAHTML
020 REPLY<-1> = "</BODY>"
021 REPLY<-1> = "</HTML>"
022 CALL FORMATHTML(REPLY)
023 RETURN

```

*Sample source code using MAKETABLE subroutine*

# Sample Source Code

## *The Standard Subroutine Template*

The following piece of Pick BASIC source code is the way you should interface with the Web Application Server. The parameter list should be used EXACTLY as shown and the final parameter REPLY should be returned with HTML code in it.

```
SUBROUTINE TEMPLATE(FILELIST, ROOTLIST, FORMDATA, ENVDATA, REPLY)
MAXFILES = 30
DIM FILELIST(MAXFILES)
DIM ROOTLIST(MAXFILES)
*
* Your code here !!!
*
CALL FORMATHTML(REPLY); * Convert AM's to CRLF's
RETURN
```

Use the above template for all of your interfacing with the Web Application Server and you can't go wrong.

## *Your First Subroutine*

```
SUBROUTINE TEMPLATE(FILELIST, ROOTLIST, FORMDATA, ENVDATA, REPLY)
MAXFILES = 30
DIM FILELIST(MAXFILES)
DIM ROOTLIST(MAXFILES)
*
REPLY = "Content-type: text/html"
REPLY<-1> = ""
REPLY<-1> = "<HTML>"
REPLY<-1> = "<HEAD>"
REPLY<-1> = "<TITLE>My First Subroutine</TITLE>"
REPLY<-1> = "</HEAD>"
REPLY<-1> = "<BODY>"
REPLY<-1> = "<H1>Hello World</H1>"
REPLY<-1> = "</BODY>"
REPLY<-1> = "</HTML>"
*
CALL FORMATHTML(REPLY); * Convert AM's to CRLF's
RETURN
```

Now on your web server, make a simple hyperlink to this subroutine, for example:

```
<A HREF="http://www.hostname.com/cgi-bin/server">My First Pick Web
Application</A>
```

If you get an error message check Appendix A for the reason.

## The LOGIN Subroutine

This subroutine is included in the software distribution as an example of a web based login script. It requires that the FILES item in the PROPERTIES file has been edited and attribute 2 has the name of the password file. It also assumes that the password file has user information in it where attribute 2 has the unencrypted password.

```
01 SUBROUTINE LOGIN(FILELIST, ROOTLIST, FORMDATA, ENVDATA, REPLY)
02 *-----*
03 * Login example script *
04 * *
05 * Assumes a form sent us here with NAME and PASSWORD *
06 * If valid then it reads the HTML from a file. *
07 *-----*
08 MAXFILES = 30
09 DIM FILELIST(MAXFILES)
10 DIM ROOTLIST(MAXFILES)
11 AM = CHAR(254)
12 *-----*
13 * Extract FORM data *
14 *-----*
15 NAME = ""
16 PASSWORD = ""
17 FCOUNT = DCOUNT(FORMDATA,AM)
18 FOR I = 1 TO FCOUNT
19   BEGIN CASE
20     CASE FORMDATA<I,1> = "NAME"
21       NAME = FORMDATA<I,2>
22     CASE FORMDATA<I,1> = "PASSWORD"
23       PASSWORD = FORMDATA<I,2>
24   END CASE
25 NEXT I
26 *-----*
27 * Validate LOGIN form from a PASSWORD file *
28 * The PASSWORD file was opened by the CGI *
29 * Server when it initialized as file # 2 *
30 *-----*
31 READ USERINFO FROM FILELIST(2),NAME ELSE USERINFO = ""
32 IF USERINFO = "" THEN GOTO 100; * Bad login
33 IF PASSWORD NE USERINFO<2> THEN GOTO 100
34 READ REPLY FROM FILELIST(3),"MAINPAGE" ELSE GOTO 100; * Bad login
35 CALL FORMATHTML(REPLY); * Convert AM's to CRLF's
36 RETURN
37 *
38 100 * Bad Login
39 REPLY = "Content-type: text/html"
40 REPLY<-1> = ""
41 REPLY<-1> = "<HTML><HEAD><TITLE>Bad Login</TITLE></HEAD>"
42 REPLY<-1> = '<BODY BGCOLOR="FFFFFF">'
43 REPLY<-1> = "<H1>Bad Login</H1>"
44 REPLY<-1> = "</BODY></HTML>"
45 CALL FORMATHTML(REPLY); * Convert AM's to CRLF's
46 RETURN
```

Notice lines 18 - 25 there is a FOR / NEXT loop used to extract the form data. Also notice in line 31 we read from a previously opened file. This file was opened by the Web Application Server not this subroutine. If the user enters a valid password then the proper HTML code is read from a file in line 34.

The web page that could have called this subroutine is as follows:

```
<html>
<head>
<title>Login Page</title>
</head>
<body>
<form method="post" action="http://www.hostname.com/cgi-bin/server">
<input type=text name=name size=20>
<input type=password name=password size=20>
</form>
</body>
</html>
```

Also the SCRIPTS item in the PROPERTIES file must be updated to call the proper Pick BASIC subroutine.

```
CT PROPERTIES SCRIPTS
001 /cgi-bin/server]LOGIN
```

Also, the FILES item in the PROPERTIES file must be updated to open our file used to validate users. In this example I created a password file.

```
CT PROPERTIES FILES
001 PASSWORD
```

## Setting A Cookie

This code creates a unique session id from a 3 digit port number, 5 digit internal date, 5 digit internal time and part of the IP address of the web browser. All cookies are written to a file.

```
01 SUBROUTINE SETCOOKIE(FILELIST, ROOTLIST, FORMDATA, ENVDATA, REPLY)
02 *-----*
03 * Send back a cookie to browser ONLY if one does not exist *
04 *-----*
05 MAXFILES = 30
06 DIM FILELIST(MAXFILES)
07 DIM ROOTLIST(MAXFILES)
08 AM = CHAR(254)
09 ECOUNT = DCOUNT(ENVDATA,AM)
10 FOR I = 1 TO ECOUNT
11   BEGIN CASE
12     CASE ENVDATA<I,1> = "HTTP_COOKIE"
13       ALLCOOKIES = ENVDATA<I,2>
14       CONVERT " " TO "" IN ALLCOOKIES
15       CONVERT ";" TO AM IN ALLCOOKIES
16   END CASE
17 NEXT I
18 *
19 HTMLHEADER = "Content-type: text/html"
20 IF INDEX(ALLCOOKIES,"SESSION",1) ELSE
21   * make a cookie for this user's session
22   EXECUTE "WHO" CAPTURING WHO
23   PORT = FIELD(WHO," ",1)
24   PORT = "000":PORT
25   PORT = PORT"R#3"; * force 3 digits
26   RAWDATE = DATE()
27   RAWDATE = "00000":RAWDATE
28   RAWDATE = RAWDATE"R#5"; * force 5 digits
29   RAWTIME = TIME()
30   RAWTIME = "00000":RAWTIME
31   RAWTIME = RAWTIME"R#5"; * force 5 digits
32   COOKIE = PORT:RAWDATE:RAWTIME:FIELD(REMOTEADDR,".",4)
33   HTMLHEADER<-1> = "Set-cookie: SESSION=:COOKIE
34   WRITE USERID ON FILELIST(1),COOKIE
35 END
36 HTMLHEADER<-1> = ""; * must have blank line separator
37 READ MENU FROM FILELIST(4),MENU.HTML ELSE GOTO HTMLERROR
38 REPLY = HTMLHEADER:AM:MENU
39 CALL FORMATHTML(REPLY)
40 RETURN
```

## Appendix A: Debugging

### *Server Error Encountered*

If you get a server error when it appears like all is working, you probably forgot to pass back the HTTP header before the actual HTML. Also, don't forget the blank line between the HTTP header and the HTML code.

```
Content-type: text/html

<html>
```

### *The LOGFILE file*

View the log of the Web Application server to see if there is any information to help you track down your problem. Use the (L option when starting the Server to force it to log its activities. There is a separate record for each Server running on its own Pick port.

The associations between the script and the subroutine to call is kept in the PROPERTIES SCRIPTS item.

Example of LOGFILE TRANS.LOG.17

```
18:07:32 03 Jul 1997 Web Application Server started on socket 4100
18:07:47 03 Jul 1997 Connection #1 accepted
18:07:48 03 Jul 1997 Received message of 783 bytes, 1 packets
18:07:48 03 Jul 1997 GET: script = /cgi-bin/server, subroutine = 'LOGIN'
```

### *The WHERE command*

If the Web Application Server is misconfigured or your subroutine is thrown into the debugger then your web browser will appear to wait forever for the web page request. To check on this situation perform a WHERE on the port that the Server is running on.

If the R1 and Return Stack Contents shows "DB.GETBUF" or anything starting with "DB." then the process is in the debugger.

On many systems you will be able to TANDEM to that port and see an asterisk "\*". This will verify that you are indeed in the debugger. Type "L" to find out what line you are on to further debug the program.

Eventually you will have to log it off and start the Server again.

## *Script Not Associated With Subroutine*

This error message is generated by the Web Application Server on the Pick host machine. It looks up in a table what subroutine to call for each script from the web server. This table is kept in the PROPERTIES file in the SCRIPTS item. If you modify this item, you must logoff and restart the Server(s) because they only read this item once when they start up.

To help debug this problem, start the Web Application Server(s) with the L option for activity logging. It appends attributes to items in the LOGFILE file.

## *Could not connect to pick.host.com at this time.*

## *Could not connect to pick.host.com, no available ports.*

These two messages are generated by the Perl script on the web server. They are not generated by the Pick side Servers. There are several reasons why a user would see these messages in their web browser.

- 1) The Pick host is down or unreachable in the network.
- 2) The Pick Server(s) have not been started. This could have been caused by a shutdown and reboot. If so, you might consider putting the starting commands in the USER-COLDSTART. Just be careful they are not running out of the DM account rather than your application account.
- 2) The Pick Servers have become so busy that no more socket connection requests can be placed in the queue.
- 3) The Pick Web Application Servers are stuck in the debugger. Perform a where command on your phantom ports to verify this.

## Appendix B: Environment Variables

### *Standard ENV variables*

Below are a list of the standard environment variables from a web server. This information will be passed to your Pick BASIC subroutine. This particular list of values are from your company web server. To see this exact display go to [www.pfinders.com/cgi-bin/diagnose.sh](http://www.pfinders.com/cgi-bin/diagnose.sh) and see our web server's environment variables running on a Sun Sparc server.

```
SERVER_SOFTWARE = NCSA/1.5.2
SERVER_NAME = www.pfinders.com
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
REQUEST_METHOD = GET
HTTP_USER_AGENT = Mozilla/4.0 [en] (Win95; I)
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, */*
HTTP_REFERER =
HTTP_INFO =
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = /cgi-bin/diagnose.sh
QUERY_STRING =
REMOTE_HOST = 206.99.109.30
REMOTE_ADDR = 206.99.109.30
REMOTE_USER =
REMOTE_IDENT =
AUTH_TYPE =
CONTENT_TYPE =
CONTENT_LENGTH =
HTTP_COOKIE = session1=4323453; session2=abcdefg
```

The reason that REMOTE\_HOST and REMOTE\_ADDR are the same in the above example is because reverse hostname lookup is turned off at the server for speed. Normally, you would see the full domain name of the location of the users web browser.

To set some cookies and view the ENV variables like above, try:

```
www.pfinders.com/cgi-bin/setcookies.sh
```

```
www.pfinders.com/cgi-bin/diagnose.sh?a=1&b=2&c=3
```

and see QUERY\_STRING

```
www.pfinders.com/cgi-bin/diagnose.sh/more/pathnames
```

and notice PATH\_INFO

# Appendix C: The Big Demo

## *A Session Tracking System With Login*

The following section describes an optional demo application. This application is included in the distribution. This application shows the following features:

- HTML Templates
- Use of Cookies
- Use of Userid's and Passwords To Login
- Session Tracking
- Easy To Implement New Functionality

### *Source Code List*

<b>SERVER</b>	Main Engine. Not specifically part of the Demo.
<b>MAIN.SUB</b>	Called by main engine. It calls the subroutines to drive the application.
<b>NEWSESSION</b>	Called for a new login.
<b>VALIDATESESSION</b>	Called for existing session verification.
<b>UPDATESESSION</b>	Updates statistics for session tracking.
<b>GETCOOKIE</b>	Isolates a specific cookie from web browser.
<b>REMOVECOOKIE</b>	Deletes or nulls a specific cookie.
<b>FORMATHTML</b>	Removes Attribute marks from HTML templates before returning it to the web browser.

