



Web Application Server Manual

Version: 3.0

Revised: February 14, 2000

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

| | |
|---|-----------|
| Overview | 3 |
| Configuration on the Web Server | 4 |
| Installation of the web script..... | 4 |
| Editing The User Defined Script Parameters | 4 |
| Configuration on the D3 / AP Side | 6 |
| Installation | 6 |
| The PROPERTIES file | 7 |
| Starting the Phantom Servers | 8 |
| Starting From The Coldstart Procedure | 8 |
| Incoming HTTP Interface | 10 |
| A Note About PATH_INFO..... | 10 |
| Outgoing HTTP Interface | 11 |
| HTTP Header | 11 |
| Setting A Cookie | 11 |
| Retrieving A Cookie..... | 12 |
| Other Session Ideas | 12 |
| Tools | 13 |
| FORMATHTML | 13 |
| GETCOOKIE..... | 14 |
| MAKETABLE | 15 |
| FILLTEMPLATE..... | 17 |
| MAIN.SUB | 18 |
| NEWSESSION, READSESSION, UPDATESESSION..... | 19 |
| The Big Demo..... | 20 |
| Sample Source Code..... | 20 |
| Your First Subroutine..... | 20 |
| The Login Subroutine..... | 21 |
| Setting A Cookie | 23 |
| Filling A Template..... | 24 |
| Appendix A: Debugging..... | 25 |
| Server Error Encountered..... | 25 |
| Waiting Forever? Use The WHERE command | 26 |
| Script Not Associated With Subroutine..... | 26 |
| Could not connect to pick.host.com at this time..... | 26 |
| Could not connect to pick.host.com, no available ports..... | 26 |
| Appendix B: Environment Variables | 27 |
| Appendix C: The Big Demo | 28 |

Overview

The Web Application Server

The Web Application Server enables you to click on a web page and have it run a Pick BASIC subroutine on your D3 system. This software provides the connectivity. It easily turns your D3 system into a Web Form Processing Server.

Your Pick database will use one or more phantom ports to process connections from the web server. If you find that you have a lot of web traffic coming in simultaneously, just run more phantom processes to handle the load. The web site will automatically split the processing evenly across all your phantoms. Web jobs may queue up at most 5 deep per phantom port before users start getting rejected.

The Web Application Server requires Pick Systems D3 or Advanced Pick 6.1 running on a UNIX based platform. It also requires the Perl scripting language to be available on the Web Server. Perl is freely available and all reputable Internet Service Providers make its use widely available.

Data is transferred from the web site and back using the standard CGI interface. This allows the use of any Web Server on the market today including Microsoft NT's IIS. The web server and the D3 server may exist on the same machine or on separate machines across the Internet. It is up to you.

We do prefer Unix based web servers such as Apache due to its flexibility and reliability.

On the web server there is a CGI program (provided) that is activated when a click occurs. This program connects to D3 on a high socket usually in the range of 5100 to 5105 and transfers all form data and environment variables to a waiting phantom process. Then it waits for a response. This response is either send back to the user's web browser directly or pre-processed before sending the results to the browser.

The response to an incoming web request is satisfied by Pick BASIC subroutines. There is a demo application with many samples of source code. Also, some standard subroutines are provided for handling logging in, cookies, filling a template, etc. You may even TANDEM to a phantom port and debug your subroutine in real time by inserting a DEBUG statement in it.

Since web designers are usually art oriented and Pick BASIC programmers are usually not, this system allows both to work independently by using web templates. More on this preferred style later in this document.

This system was tested with D3/Linux, AP/RS6000 and D3/RS6000 (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 D3).

Configuration On The Web Server

Installation of the web script

There are 3 diskettes provided in this distribution:

- 1) Web Application Server D3 T-DUMP format
- 2) Web Application Server tar format
- 3) Web Application Server DOS format

Number one is T-LOADed into your D3 account and explained later.

Number two is in UNIX tar format and the third 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.

Personally, I prefer to hide it by creating a special directory that is executable just like cgi-bin, such as "pick". Then I copy the server program into it. This requires access to the web server configuration files.

On Apache add a line such as:

```
ScriptAlias /pick/ /home/pfinders/html/pick/
```

This way I can access it in my browser with:

```
http://www.pfinders.com/pick/server
```

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.

Editing The User Defined Script Parameters

The script has some user defined parameters you must edit.

```
$basewebdir = "/home/pfinders/html";  
$pickhostname = "d3.pfinders.com";  
$startingport = 5100;  
$endingport = 5105;  
$evenload = 1;  
$mailprog = "mail.pfinders.com";
```

Basewebdir

This means the directory to find your web page templates. You can store your web templates anywhere, really. But its easy to find them if you leave them in your web tree of HTML pages.

Pickhostname

This is the hostname of the machine running D3 or its IP address.

Startingport and Endingport

These are socket numbers. They represent a range of phantoms to try and connect on. You can leave them as is. The default gives you up to 6 phantom processes to split the load across. If you change these numbers, make sure you start more phantoms on the D3 host.

Evenload

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

- 1 = random port in range (normal use)
- 0 = start from \$startingport (used for testing purposes)

When testing, set \$evenload to 0, then only run a single phantom port on the D3 side.

This will guarantee that you know what phantom Pick port will satisfy the web request. This way you can TANDEM to that Pick port and step through your BASIC program to debug it.

You may place DEBUG statements and PRINT statements in your subroutines to watch activity. The web server will never see them or get confused because it is communicating across a socket, not the terminal!

When you deploy your Web Application server set \$evenload to 1 and startup all D3 or AP phantom processes.

Mailprog

Future feature. This parameter can be set to a program such as: `/usr/lib/sendmail` or it can be set to the name of your SMTP mail server. It is part of a future feature that will allow you to send an email reply to a user and show him a resulting HTML page at the same time. But, for now its not used.

Configuration on the D3 / AP Side

Installation

1) Logto 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. When prompted to load the **Demo**, enter Y.

4) Edit the items in the PROPERTIES file.

New Files Created

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

| | |
|------------|---|
| PS.BP | Basic Program File for Web Application Server. Some programs have the same name as the DEMO.BP programs. |
| PROPERTIES | Where important initialization records are kept, especially the mapping of click URLs to D3 subroutines to run. |
| LOGFILE | For debugging and tracking server activity. Each phantom can record its activity, if desired. |
| DEMO.BP | BASIC source code for demo application with user login |
| HTML | Demo HTML template records (not web templates) |
| SESSION | File to store active session records with item-id stored on users web browser as a cookie. |
| USER | Add users to this file for the demo application. Passwords are attribute one. |
| TABLES | Empty file upon installation. Can be used with MAKETABLE subroutine. |

The PROPERTIES file

You must edit three important records in this file.

1) The FILES record

This record lists all of the D3 files that you want each phantom process to open up and hold open during its lifetime. Then, when it calls your subroutine it passes all the file descriptors to it as a dimensioned array (FDLIST). The current maximum is 30 open files using this method. Remember, opening and closing files with each web request is time consuming, so this is why we do this.

```
:ED PROPERTIES FILES
top
.P
001 PROPERTIES
002 PS.BP
003 HTML
004 SESSION
005 USER
eoi 005
```

2) The SCRIPTS record

This record lists all of your web scripts and what Pick BASIC subroutine they should call. The first value of each attribute is the URL path of the script and the second value is the cataloged subroutine name to call.

```
001 /cgi-bin/server]LOGIN
002 /session/user]MAIN.SUB
```

If there is no association between a URL path (CGI script) and a Pick BASIC subroutine, then an error message is returned to the web browser. The common thing to do here is to put a big case statement in the BASIC program called so it can handle many different types of web requests. It is better than constantly changing this record and restarting all the phantoms. You will see in the example programs.

3) 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 subroutine each time the phantom (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 record is value 1 = filename and value 2 = 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 5100
Z SERVER 5101
Z SERVER 5102
Z SERVER 5103
Z SERVER 5104
Z SERVER 5105
```

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
```

Starting From The Coldstart Procedure

You should start the phantom processes from the coldstart procedure. But, DO NOT place the above commands directly into the user-coldstart. They will start in the DM account and not in the account you loaded this software into.

First create an item in the Master Dictionary. This way when you logto STARTWEB, you will fire off 6 phantoms and then logoff automatically.

```
:ED MD STARTWEB
new item
top
.I
001+N
002+Z SERVER 5100
003+Z SERVER 5101
004+Z SERVER 5102
005+Z SERVER 5103
006+Z SERVER 5104
007+Z SERVER 5105
008+OFF
009+
top
.FI
[221] 'STARTWEB' filed.
```

Next go to the DM account and create a Master Dictionary Q pointer.

```
:LOGTO DM
< Connect time= 46 Mins.; CPU= 46000 Units; LPTR pages= 0 >

:ED MDS STARTWEB
new item
top
.I
001+Q
002+MYACCOUNT
003+
top
.FI
[221] 'STARTWEB' filed.
```

Don't forget to change MYACCOUNT to the real name of your account.

Finally, place a LOGTO statement at the end of the USER-COLDSTART record.

```
:ED MD USER-COLDSTART
top
.P
001 N
002 startshp 1,1,0,s8,lp.unix,lpr (s
003 LOGTO STARTWEB
eoi 003
```

Incoming HTTP Interface

When a web server calls a CGI program to process a web request, 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
```

Anything after the question mark “?” is considered part of a form to the Web Application Server.

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. Even when specifying template data, you still need to specify the HTTP Header.

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. The provided subroutines can set cookies for you. See the `NEWSSESSION` subroutine for details.

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. See the GETCOOKIE subroutines. Multiple cookies are separated by semi-colons.

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. Both the DEMO and MAIN.SUB program use cookies (so you don't have to learn how!).

Example Hiding Cookies:

```
<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 URL Cookies:

```
<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 GET Form Cookies:

```
<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 has been built, just before the final RETURN back to the web server. Remember web servers don't know what attribute marks are!

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
```

In the above example, we generate all the required HTML directly, including the required header. Afterwards we call FORMATHTML to strip the attribute marks. Then the final RETURN line sends the REPLY variable back to the web server for sending directly to the browser.

GETCOOKIE(ENV, COOKIENAME, COOKIEVALUE)

When passed a cookie name, this subroutine returns the cookie value. Cookies are great for storing session record item-ids or personal settings the user likes.

Example

```
CALL GETCOOKIE(ENV, "session", COOKIEVALUE)
IF COOKIEVALUE = "" THEN
  *****
  * No session in progress, start new one *
  *****
  CALL NEWSESSION(FDLIST, FORM, SESSREC)
END ELSE
  *****
  * Show Main Page *
  *****
  CALL FORMATHTML(MAINPAGE)
RETURN
END
```

See the source code of READSESSION for a complete example.

Notice that you must pass ENV to the GETCOOKIE subroutine because that is where it is looking for the cookie information.

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

FILLTEMPLATE

This subroutine allows you to take your D3 data and fill in fields in an existing HTML file that resides on the web server. You do not have to generate an entire HTML web page from D3. Let the web designer do it on the web server. Also, you do not have to call FORMATHTML at the end.

Example

```
001 SUBROUTINE LOOKUPPART(FDLIST ,ROOTLIST, FORM, ENV, REPLY)
002 *-----*
003 * Get information about part the user requested*
004 *-----*
005 INCLUDE HDR.INCLUDE
006 CODE = ""; FIELDS = ""
007 FCOUNT = DCOUNT(FORM,AM)
008 FOR I = 1 TO FCOUNT
009   IF FORM<I,1> = "code" THEN
010     CODE = FORM<I,2>; * sample info only
011   END
012 NEXT I
013 READ PARTREC FROM FDLIST(7),CODE ELSE PARTREC = ""
014 FIELDS<-1> = "heading":VM:"Found Your Part"
015 FIELDS<-1> = "partname":VM:PARTREC<1>
016 FIELDS<-1> = "partnumber":VM:PARTREC<2>
017 FIELDS<-1> = "template":VM:"showpart.html"
018 HEADERS = "Content-type: text/html"
019 CALL FILLTEMPLATE(HEADERS,FIELDS,REPLY)
020 RETURN
```

Notice we initialize FIELDS on line 6. This is very important to all subroutines. Since the Web Application Server never stops running, we don't want any variables to grow forever. Also, notice that we expect a web form that has a field called code to be submitted and this code is the item-id. It can be passed in a regular POST form or as a GET such as:

<http://www.host.com/cgi-bin/server?code=abc123>

MAIN.SUB

This subroutine is what you should base a full blown application on. This subroutine can drive session tracking with cookies and user logins with passwords. The case statement for you to insert your subroutines is already to go.

Don't confuse PS.BP MAIN.SUB (this program) with DEMO.BP MAIN.SUB. Because it's a very different program. Recatalog all the PS.BP programs if you are done with the demo.

Set Your Parameters

```
017 *-----*
018 * User Defined Parameters *
019 *-----*
020 SESSIONTRACKING = 1; * If zero, no login required
021 WEBSERVERNAME = "www.pfinders.com"
022 COOKIENAME = "Session"
023 LOGINPAGE = "Location: http://":WEBSERVERNAME:"/pages/index.html"
024 REDIRECT = "Location: http://":WEBSERVERNAME:"/pages/session.html"
025 ERRORFILE = "error.html"
```

Line 20 shows SESSIONTRACKING. If zero then no cookies are set and no session record is written.

Line 23 and 24 shows pointers to the web site. LOGINPAGE has your standard userid and password fields. But, REDIRECT is the next page after a valid password is entered. This page should be a quick intermediate page that takes you to the real main page all by itself.

For example:

```
<html>
<head>
<meta http-equiv="refresh" content="0; url=http://www.pfinders.com/cgi-bin/server/main">
<title>Petrolis Customer Center</title>
</head>
<body bgcolor="ffffff">
<h1>Starting Session</h1>
<h2>Please wait</h2>
</body>
</html>
```

This method allows for the cookie to be "set" on all browser types that support cookies.

NEWSESSION, READSESSION, UPDATESESSION

These subroutines are called from MAIN.SUB but you can call them directly to manage sessions.

NEWSESSION can validate a userid and password, then create a session record.

READSESSION can see if a cookie exists on the browser and read it as an item-id in the SESSION file. If it finds the record, it returns it else its an expired or non-existing session. The user could have logged off, too. It will also read the USER record that is associated with this session.

UPDATESESSION can be called to change info in a session, like date and time of last access. Its optional.

Example

```
009 COOKIENAME = "mysettings"
010 CALL READSESSION(FDLIST, ENV, FORM, COOKIENAME, COOKIEVALUE,
SESSION, USER)
011 IF SESSION = "" THEN
012   CALL ERROR.SUB
013 END ELSE
014   CALL SHOWPAGE.SUB
015 END
```

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. Of course you could use EXAMPLE.SUB or MAIN.SUB to give you a head start on developing your application.

```
SUBROUTINE TEMPLATE(FDLIST, ROOTLIST, FORM, ENV, REPLY)
INCLUDE HDR.INCLUDE
*
* 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(FDLIST, ROOTLIST, FORM, ENV, REPLY)
INCLUDE HDR.INCLUDE
*
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 make sure to ED PROPERTIES SCRIPTS and make sure there is a line about cgi-bin/server or 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(FDLIST, ROOTLIST, FORM, ENV, REPLY)
02 *-----*
03 * Stand alone Login script example *
04 * *
05 * Assumes a form sent us here with NAME and PASSWORD *
06 * If valid then it reads the reply web page from the HTML *
07 * file else it reads the failure page from the HTML file also*
08 *-----*
09 INCLUDE HDR.INCLUDE
10 *-----*
11 * Extract FORM data *
12 *-----*
13 NAME = ""
14 PASSWORD = ""
15 FCOUNT = DCOUNT(FORMDATA,AM)
16 FOR I = 1 TO FCOUNT
17     BEGIN CASE
18         CASE FORMDATA<I,1> = "NAME"
19             NAME = FORMDATA<I,2>
20         CASE FORMDATA<I,1> = "PASSWORD"
21             PASSWORD = FORMDATA<I,2>
22     END CASE
23 NEXT I
24 *-----*
25 * Validate LOGIN form from a PASSWORD file *
26 * The PASSWORD file was opened by the Web *
27 * Server when it first started as file # 2 *
28 *-----*
29 READ USERINFO FROM FDLIST(5),NAME ELSE USERINFO = ""
30 IF USERINFO = "" THEN GOTO 100; * Bad login
31 IF PASSWORD NE USERINFO<2> THEN GOTO 100
32 READ REPLY FROM FDLIST(3),"MAIN" ELSE GOTO 100; * Bad login
33 CALL FORMATHTML(REPLY); * Convert AM's to CRLF's
34 RETURN
35 *
36 100 * Bad Login
37 READ REPLY FROM FDLIST(3),"FAILEDLOGIN" THEN
38     *-----*
39     * Get web page from an item in the HTML file *
40     *-----*
41     CALL FORMATHTML(REPLY); * Convert AM's to CRLF's
42     RETURN
43 END ELSE
44     *****
45     * Couldn't read web page as an item, so send *
46     * back hard coded HTML below *
47     *****
48     REPLY = "Content-type: text/html"
49     REPLY<-1> = ""; * Must have a blank line
50     REPLY<-1> = "<HTML><HEAD><TITLE>Bad Login</TITLE></HEAD>"
```

```
51 REPLY<-1> = '<BODY BGCOLOR="FFFFFF">'
52 REPLY<-1> = "<H1>Bad Login</H1>"
53 REPLY<-1> = "</BODY></HTML>"
54 CALL FORMATHTML(REPLY); * Convert AM's to CRLF's
55 RETURN
56 END
```

Notice lines 16 - 23 there is a FOR / NEXT loop used to extract the form data. Also notice in line 29 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 32. In this program it read the MAIN record from the HTML file.

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">
name <input type="text" name="name" size="20">
password <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
```

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(FDLIST, ROOTLIST, FORM, ENV, REPLY)
02 *-----*
03 * Send back a cookie to browser ONLY if one does not exist *
04 *-----*
05 MAXFILES = 30
06 DIM FDLIST(MAXFILES)
07 DIM ROOTLIST(MAXFILES)
08 AM = CHAR(254)
09 ECOUNT = DCOUNT(ENV,AM)
10 FOR I = 1 TO ECOUNT
11   BEGIN CASE
12     CASE ENV<I,1> = "HTTP_COOKIE"
13       ALLCOOKIES = ENV<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 FDLIST(1),COOKIE
35 END
36 HTMLHEADER<-1> = "; * must have blank line separator
37 READ MENU FROM FDLIST(4),MENU.HTML ELSE GOTO HTMLERROR
38 REPLY = HTMLHEADER:AM:MENU
39 CALL FORMATHTML(REPLY)
40 RETURN
```

Filling A Template

This code fills an HTML page with data generated from D3. This subroutine specifies the name of the template as well as the names and values of the fields to be filled in. The template in this example is called **listu.html** and resides on the web server, not the D3 machine.

The fields that are filled in always look like this: `<!--%name-->` inside the HTML template. In this example, we use 2 fields **heading** and **data**.

This is a great way to allow web page designers to work independently of the Pick Basic subroutine writers.

```
SUBROUTINE LISTU.SUB(FDLIST,ROOTLIST,FORM,ENV,REPLY)
```

```
*-----*
* Fill an HTML template called listu.html *
* with LISTU data. On the web server, *
* <!--%data--> will be replaced with *
* the LISTU ascii text. *
* Also, <!--%heading--> will be filled *
* in with our desired heading here. *
*-----*
MAXFILES = 30
DIM FDLIST(MAXFILES)
DIM ROOTLIST(MAXFILES)
AM = CHAR(254)
VM = CHAR(253)
WHOINFO = ""
HEADERS = ""
FIELDS = ""
*
HEADERS = "Content-type: text/html"; * Standard stuff
FIELDS<-1> = "template":VM:"listu.html"
FIELDS<-1> = "heading":VM:"Who's Logged Into D3"
*
EXECUTE "LISTU" CAPTURING WHOINFO
CALL FORMATHTML(WHOINFO); * convert AM to CRLF
*
FIELDS<-1> = "data":VM:WHOINFO
CALL FILLTEMPLATE(HEADERS,FIELDS,REPLY)
RETURN
```

Appendix A: Debugging

General Notes

When troubleshooting your application, problems usually arise from 2 basic situations. Either your BASIC subroutine didn't generate the correct HTML or it has a more serious problem such as:

Getting thrown into the BASIC debugger

This can be verified by doing a WHERE and checking if any port is in the debugger. This is common to occur if you forget to CATALOG your subroutine.

Variable not assigned, zero used

This problem can be difficult to nail down. Try a TANDEM to the phantom process and leave it that way for a while. See if the error message appears on the screen while you use your web browser to exercise your application's subroutines.

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. Web servers do not like that. 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'
```

Waiting Forever? Use 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 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 stored in DEMO.BP. Some of the names of the programs are also found in PS.BP. Don't get them confused, they are different. Simply CATALOG all the programs to switch back and forth between the different set of applications.

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

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

