

Notes from Well House Consultants

These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit

*<http://www.wellho.net/net/whcotnl.html>
for details.*

1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

graham@wellho.net

technical contact

lisa@wellho.net

administration contact

Our full postal address is

404 The Spa
Melksham
Wiltshire
UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

More CGI Programs and Facilities

The extra facilities of CGI, things that you'll need from time to time but probably won't use on a daily basis. Learn how to upload and download files, how to email from the server, how to recognise which browser your client is using, and how to return a page that automatically moves on to a second response page a few seconds later.

<i>Automatic emailing; recognising the browser</i>	4
<i>Uploading and downloading files</i>	9
<i>Multi part documents</i>	12
<i>Server Side Includes</i>	16

Once you get beyond the CGI fundamentals, you'll find that there are an incredible number of tasks you can have performed through relatively simple Perl scripts. In this chapter, we're giving you a few samples to hopefully start you thinking toward your particular goals and objectives, and also to see how these scripts can be adapted to even further functions. We'll be covering major sections such as state and searching in up-coming chapters.

2.1 Automatic emailing; recognising the browser

Having your script send an automatic email can entail more than just "thank you for visiting our site." You can use this feature to actually verify a correct email address, letting you know, for instance, your database of email addresses is filling in with valid entries. Not especially a "ruse" but if you were to respond to their request by automatically sending them their password for future entry, this will make sure that they really have entered a correct address.

Another reason for having your script send an automatic email would be to send their booking or request for brochure from a form to your administrator for processing.

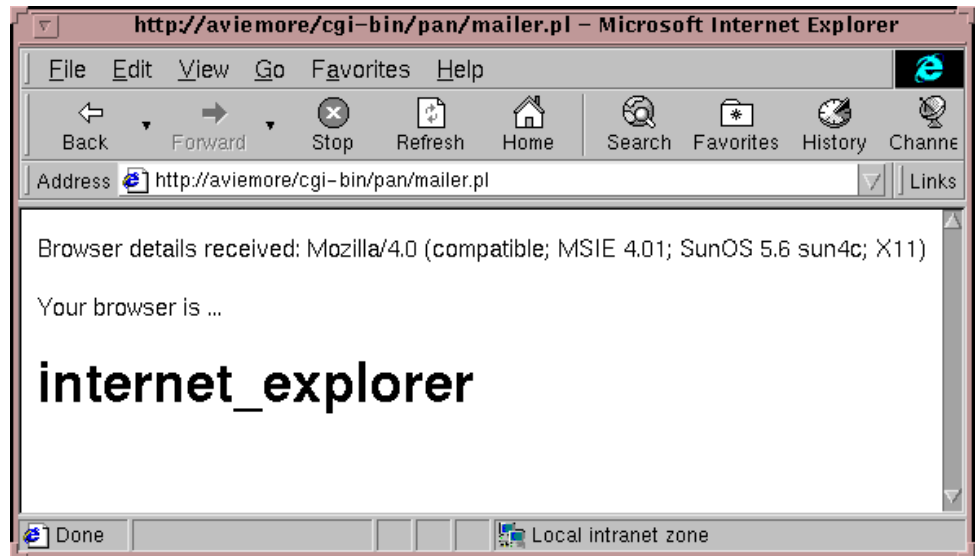
You may even want to send an email to an automatic email handler on another system which, when invoked, will process it automatically. In other words, have a CGI set off a chain of events on other systems on the network. We'll come back to automatic email handling later.

As you have no doubt already encountered, not all browsers, nor versions of browser, view web pages the same. Would it be helpful to you to be able to automatically adjust the HTML that's seen according to the user's browser? Would you like to know the percentage of people visiting your site that use Netscape versus Internet Explorer? How is Swedish-based iCab affecting usage?

Let's view an illustration of an even further function of automatic emailing and incorporating a way of identifying the user's browser.

Figure 1 Identifying the user's browser: Netscape...

Figure 2 ...Internet Explorer



Here's the Perl program behind the page:

```
#!/bin/perl -- -*-perl-*-

# work out which browser is calling

$browser = $ENV{"HTTP_USER_AGENT"};

if ($browser) {
    $br = "not_known";
    $br = "netscape" if ($browser =~ /^Mozilla/i) ;

$br = "mosaic" if ($browser =~ /^NCSA|Spry|Air/i) ;
    $br = "text_only" if ($browser =~ /^Lynx/i) ;
    $br = "text_only" if ($browser =~ /^CERN-LineMode/i) ;
} else {
    $br = "not_specified";
}

if ($br eq "netscape") {
    if ($browser =~ /MSIE/) {
        $br = "internet_explorer" ;
    } else {
        @br_spec = split (/\//,$browser);
        $br_age = "_youthful";
        $br_age = "_midddeaged" if ($br_spec[1]<4.0) ;
        $br_age = "_retired" if ($br_spec[1]<3.0) ;
        $br .= $br_age;
    }
}

##### Extra Code Hidden from Page Visitor #####

$mailprog = '/usr/lib/sendmail';
$regr = 'graham@wellho.net';

open (MAIL, "|$mailprog $regr") || die "Can't open $mailprog!\n";
print MAIL "Reply-to: nobody@dumy.address\n";
print MAIL "Subject: Visitor Alert!\n\n";
```

```
print MAIL "Page has been visited from ", $ENV{"REMOTE_ADDR"}, "\n";
print MAIL "Using Browser ", $browser, "\n";
print MAIL "Host is called ", $ENV{"REMOTE_HOST"}, "\n";
print MAIL "\n";
close (MAIL);
```

```
#####
```

```
print << "REPLYPAGE"
Content-type: text/html

<Body BGCOLOR=white text=black>
Browser details received: $browser <P>
Your browser is ...
<H1>$br</H1>
</Body>
REPLYPAGE
;
```

And the designated email recipient received:

```
From nobody Tue Apr 20 22:18:53 1999
Date: Tue, 20 Apr 1999 22:18:52 +0100
From: nobody@wellho.co.uk (Nobody)
Subject: Visitor Alert!
```

```
Page has been visited from 192.168.200.127
UsingBrowserMozilla/4.5
[en](X11; I; SunOS5.6 sun4c)
Host is called sealion
```

And also:

```
From nobody Tue Apr 20 22:32:39 1999
Date: Tue, 20 Apr 1999 22:32:38 +0100
From: nobody@wellho.co.uk (Nobody)
Subject: Visitor Alert!
```

```
Page has been visited from192.168.200.125
Using Browser Mozilla/4.0
(compatible; MSIE 4.01; SunOS 5.6 sun4c; X11)
Host is called walrus
```

Notes – browser:

When working out the browser type, both Internet Explorer and Netscape declare themselves as "Mozilla" but you can tell that you have Internet Explorer because it also includes the string "MSIE". The **HTTP_USER_AGENT** environment variable is useful in helping you identify which facilities you can use in pages returned to a user. In an extreme case, you may have several versions of a site and return different pages to the user depending on which browser he's using. The visitor to the site is completely unaware that this email was sent.

Remember too, although our sample web server is set up with a distinctive cgi-bin directory, it is possible for every page (including even a home page */index.html*) to call a CGI script. Who said that a server needs to have any document tree at all?

Notes – email:

Here's the code we used:

```
$mailprog = '/usr/lib/sendmail';
$regr = 'graham@wellho.co.uk';

open (MAIL, "|$mailprog $regr") || die "Can't open $mailprog!\n";
print MAIL "Reply-to: nobody\@dummy.address\n";
print MAIL "Subject: Visitor Alert!\n\n";
print MAIL "Page has been visited from ", $ENV{"REMOTE_ADDR"}, "\n";
print MAIL "Using Browser ", $browser, "\n";
print MAIL "Host is called ", $ENV{"REMOTE_HOST"}, "\n";
print MAIL "\n";
close (MAIL);
```

We had a choice of talking directly to sendmail, which is what we did, or we could also have talked to a mail client, or port 25 (the SMTP port) on a server.

In our example, we started a new copy of sendmail, and we piped information to it. The information (rather like http!) is in the form of a header, followed by a blank line, followed by a body. As author of a CGI script you have only limited control over the email headers. After all, the email is being sent by the web server and the administrator won't want to let his users override this. It would just be too tempting for people to hijack his system and use it as the source of junk emails. But you do have some control over the headers.

Advantage of using a user mail tool: It's easy to understand, quick to code and will generate emails with accurate headers. Disadvantage: It's likely to be very system-dependent – which particular mail tools will you have on your development server? Will they also be available on the public server? Etc.

Here we assume that **mailx** is available:

```
#!/bin/perl -- *-perl-*

$mailer = 'mailx -s "Sample using mailx" ' .
          '-c lisa@wellho.com ' .
          '-r 100650.2335@Compuserve.com ' .
          'graham@wellho.co.uk';

open (MAIL, "|$mailer");

print MAIL <<"FFF";
This is the body of the email which is sent to let
folks know this script has been run.

FFF

foreach (sort keys %ENV) {
    print MAIL;
    print MAIL "    $ENV{$_}\n";
}
close (MAIL);

print
"Location: http://www.wellho.co.uk/index.html\n\n";
```

The email we get looks much cleaner:

```
N105 100650.2335@Compuserve Tue Apr 20 23:01 38/1217 Sample using mailx
? 105
```

Message 105:

```
From 100650.2335@Compuserve.com Tue Apr 20 23:01:08 1999
Date: Tue, 20 Apr 1999 23:01:06 +0100
From: 100650.2335@Compuserve.com@wellho.co.uk
To: graham@wellho.co.uk
Cc: lisa@wellho.co.uk
Subject: Sample using mailx
```

This is the body of the email which is sent to let folks know this script has been run.

```
DOCUMENT_ROOT /extra/disc0.slice5/www
GATEWAY_INTERFACE CGI/1.1
HTTP_ACCEPT image/pjpeg, image/jpeg, image/x-xbitmap, image/gif, */*
HTTP_ACCEPT_ENCODING gzip, deflate
HTTP_ACCEPT_LANGUAGE en-us
HTTP_CONNECTION Keep-Alive
HTTP_HOST aviemore
HTTP_REFERER http://aviemore/pan/
HTTP_USER_AGENT Mozilla/4.0 (compatible; MSIE 4.01; SunOS 5.6 sun4c; X11)
PATH /usr/sbin:/usr/bin
QUERY_STRING
REMOTE_ADDR 192.168.200.225
REMOTE_HOST magnet
REQUEST_METHOD GET
SCRIPT_FILENAME /export/home/cgi-bin/pan/mailx.pl
SCRIPT_NAME /cgi-bin/pan/mailx.pl
SERVER_ADMIN graham@wellho.demon.co.uk
SERVER_NAME aviemore
SERVER_PORT 80
SERVER_PROTOCOL HTTP/1.1
SERVER_SOFTWARE Apache/1.0.3
TZ GB
```

From the Perl program:

```
$mailer = 'mailx -s "Sample using mailx" ' .
    '-c lisa@wellho.com ' .
    '-r 100650.2335@Compuserve.com ' .
    'graham@wellho.co.uk';
open (MAIL, "|$mailer");
```

Much simpler than sendmail; just set up the email subject, copy list, reply address and destination, and start the command, running a pipe to it.

There's now no need to generate a header, just send the content:

```
print MAIL <<"FFF";
This is the body of the email which is sent to let
folks know this script has been run.

FFF

foreach (sort keys %ENV) {
```

```

        print MAIL;
        print MAIL "    $ENV{$_}\n";
    }
close (MAIL);

```

Remember that if you fail to specify a loop variable for a `foreach` loop, Perl uses `$_`, and if you don't tell `print` what to use, it uses `$_` as well.

Finally, the reply sent to the browser was a location; we have decided that we want to send just a document back. Although the document we chose in this case was our home page, at the end of a sequence of entering data onto a form it could well be a "Thank you for your submission, we'll be in touch" page.

2.2 Uploading and downloading files

Here's a form that will allow us to access upload and download capabilities, particularly useful if you want to use the web as a method of file transfer. There is no need for the page to be displayed, and there is no restriction on size or contents in our example.

Figure 3 *Uploading and downloading files*

```

<BODY bgcolor=white text=black>
<H2>File upload </H2>
Enter details of a *file* you want to send to the server
<P>
<Form method=POST ENCTYPE="multipart/form-data"

```

```
action="http://cgi-bin/pan/up.pl">
Please enter the name of the file to upload<P>
<Input type="file" name="filedata"><P>
<Input Type="submit" Value="go!">
</Form>
<HR>
<H2>File Download</H2>
Enter details of a *file* you want to get from the server
<P>
<Form method=POST action="http://cgi-bin/pan/down.pl">
Please enter the name of the file to retrieve<P>
<Input name="filedata"><P>
<Input Type="submit" Value="go!">
</Form>
</BODY>
</HTML>
```

You'll notice that for uploading, we're using a different form of encoding, and that means that our Perl program will look somewhat different:

Figure 4 What you'll see is a short report on the screen

```
#!/bin/perl -- *-perl-*

# upload a file

read (STDIN,$buffer,$ENV{'CONTENT_LENGTH'});

($mime_sep) = split(/\s+/, $buffer);
@fields = split(/$mime_sep/, $buffer);
foreach $input(@fields){
($headline,$data) = split(/\r\n\r\n/, $input, 2);
$filedata = $data if ($headline =~ /filedata/)
}

$fnsave = "/tmp/".$$;
chop $filedata;
chop $filedata;
open (HERE,">$fnsave");
print HERE $filedata;
```

```

# Generate the output page

$fsz = length ($filedata);
$now = localtime();

print << "REPLYPAGE"
Content-type: text/html

<BODY BGCOLOR=white text=black>
<H2>Your page has been uploaded and saved
    as $fnsave</H2>
File size is $fsz bytes<BR>
$now
</BODY>
REPLYPAGE
;

```

For downloading, here's the Perl script:

```

#!/bin/perl -- -*perl*-

# Download a file

read (STDIN,$buffer,$ENV{'CONTENT_LENGTH'});

@pairs = split(/&/,$buffer);
foreach $entry (@pairs)
{
    ($name,$value) = split(/=/,$entry);
    $value=~ tr/+// ;
    $value=~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C",hex($1))/eg;
    $value=", ".$value if ($input{$name});
    $input{$name} .= $value;
}

$fnget = $input{"filedata"};
$size = -s $fnget;
if (open (SEND,$fnget) ) {
read (SEND,$content,$size);

print "Content-type: application/octet-stream\n\n";
print $content;

} else {

print "Content-type: text/html\n\n";
print "File $fnget on the server is not available";

}

```

Note:

The following may be concerns if you are uploading and downloading files and your tutor will talk you through them as necessary.

- Selection of file names
- Permission issues on server
- Getting a directory of the server - how about using "location" and no index.

- Allowing upload may spread viruses
- Passing a name back on download
- ENCTYPE. IE and Netscape - delimiters
- Use of two forms on the same web page
- File selection menus presented in each direction.
- Screen display does not change in download mode; see multi part document download in NPH section.

2.3 Multi part documents

Perhaps you wanted the example in the previous section to download the file and also to move on to a new page having several response pages in a single enquiry.

Server Push

Let's see an example that sends out several pages in a single enquiry. This one will report on the system load by using the Unix `uptime` command, but it could be anything from that to a clock to a moving stock quote to reloading a picture from a videocam...

Two response pages:

Figure 5 Several pages in a single enquiry

H

ere's the HTML that links to the script:

```
<HTML>
<BODY BGCOLOR=white TEXT=black>
Select
<A HREF="http://cgi-bin/pan/nph-clock.pl">here
</A>
to run a CGI script which demonstrates
"Server Push"
</BODY>
</HTML>
```

You'll see that we have a very odd script name. It starts with the characters **n p h** - which stands for "non-parsed header". It's the way that we can instruct the web server that the page being returned by the CGI script is not to be parsed by the server.

Thus the server will not add the "200" correct code or anything else, so we've taken over the responsibility to do so. Although we have extra work to do, it means that we can be extra flexible.

Here's the Perl:

```
#!/bin/perl -- -*-perl-*-

# Generate a page "On the fly".
#
# Non Parse Header for server push!

$tnow = time();
($sec,$min,$hour,$mday,$mon,$year,$yday) = localtime($tnow);

# Set up stuff

print <<NPH
HTTP/1.0 200 OK
Server: $ENV{'SERVER-SOFTWARE'}
MIME-Version 1.0
Content-type: multipart/x-mixed-replace;boundary=qwertyuiop

qwertyuiop
NPH
;
$|=1;

# Loop to generate pages

while (1) {
$tnow = time();
($sec,$min,$hour,$mday,$mon,$year,$yday) = localtime($tnow);
$eomminute = 60-$sec ;
$report[$nrep++] = `uptime`;

print << "TOPPER"
Content-type: text/html

<BODY BGCOLOR=WHITE TEXT=BLACK>
<CENTER>
TOPPER
;
}
```

```
printf ("<H1>UK Date and time: %d-%d-%d %02d:%02d</H1>",
        $mday,$mon+1,$year,$hour,$min);
print "</CENTER><PRE> @report </PRE>\n";
print "</BODY>\n\n";
print "qwertyuiop\n";
sleep $eomminute;
if ($nrep > 10) {
    shift @report;
    $nrep --;
}
}
```

Because our script is **nph**, we have to start it with a complete set of headers. What we're printing here is just a minimum and things like **content-length** and **last-modified** would be a good idea!

```
print <<NPH
HTTP/1.0 200 OK
Server: $ENV{'SERVER-SOFTWARE'}
MIME-Version 1.0
Content-type: multipart/x-mixed-replace;boundary=qwertyuiop

qwertyuiop
NPH
;
```

We've defined this as a multi part document so we have to tell it what the separator (boundary) between sections is, and we've selected "qwertyuiop".

We want to ensure that this information, and any subsequent information, really is sent; we don't want it sitting in the buffers of our CGI program, so we turn buffering off:

```
$|=1;
```

And we now want to feed a series of pages to the browser – every minute – reporting on the uptime.

```
while (1) {
```

Get the current time and find how much longer it is to the end of the minute:

```
$tnow = time();
($sec,$min,$hour,$mday,$mon,$year,$wday) = localtime($tnow);
$eomminute = 60-$sec ;
```

Run an uptime report and store it in an array of such reports for inclusion on the page:

```
$report[$nrep++] = `uptime`;
```

and generate the (next) page.

Note that each page has three components:

- Header, terminated by a new-line character.
- Page (could include a **<HEAD>** as well as a **<BODY>** section)
- The "qwertyuiop" separator, after a blank line, to separate the sections.

```
print << "TOPPER"
Content-type: text/html

<BODY BGCOLOR=WHITE TEXT=BLACK>
<CENTER>
TOPPER
;
```

```
printf ("<H1>UK Date and time: %d-%d-%d %02d:%02d</H1>", $mday,$mon+1,$year,$hour,$min);
```

```
print "</CENTER><PRE> @report </PRE>\n";
print "</BODY>\n\n";
print "qwertyuiop\n";
```

Having output the page (and it has been flushed because of `$|`, and the user's browser does complete the drawing of it because of the separator), the server-side application sleeps until the minute ends. If there are more than 10 reports, it scraps the oldest, and then it loops around to generate the next page.

```
sleep $eomminute;
if ($nrep > 10) {
  shift @report;
  $nrep --;
}
```

The special naming of files to start `nph-` is a convention shared by many web servers, and isn't even a configuration option on many. If you are writing your own specialist server you could, of course, set up your own standard if you wished.

Client pull

In the example we've looked at, the server keeps the connection to the browser alive and multiple documents can be "fed". An alternative approach is for the server to send a single document which includes an instruction to the browser telling it to move on automatically to another page after a certain time. The two methods are known as "server push" and "client pull".

Client pull is frequently used, even in documents without any CGI content. Have you ever visited a site and had it return a "this page has moved" message, followed by another page a few seconds later? Or have you ever visited a home page to get a "welcome" page, followed by a "real" front page a few seconds later?

Let's use client pull to put up a pair of pages.

The first page:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Refresh" Content="6; URL=/cgi-bin/pan/passing.pl">
</HEAD>
<BODY BGCOLOR=white TEXT=black>
<H2>This page tells you something for a few seconds and then
you'll get taken on to the next place ...</H2>
</BODY>
</HTML>
```

which takes us to the second page:

```
#!/bin/perl -- -*-perl-*-

open (LOG,">> datafiles/logs");

print LOG "##### ",`date`;
foreach (sort keys %ENV) {
  print LOG ("$_ $ENV{$_}\n");
}

print "Location: http://aviemore/pan/index.html\n\n";
```

That second page will write to a logfile and will pass the user back to the stated URL. He won't even know he's run a CGI script!

But, looking on the server, you'll find in the data file:

```
##### Thu Apr 22 14:00:40 BST 1999
DOCUMENT_ROOT /extra/disc0.slice5/www
GATEWAY_INTERFACE CGI/1.1
HTTP_ACCEPT image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/
png, */*
HTTP_ACCEPT_CHARSET iso-8859-1,*,utf-8
HTTP_ACCEPT_ENCODING gzip
HTTP_ACCEPT_LANGUAGE en
HTTP_CONNECTION Keep-Alive
```

etc.

2.4 Server Side Includes

Rather than using the Common Gateway Interface and running documents in a different subtree on the server, the system administrator on the server can select two other options:

- It's possible to place the CGI scripts in the same tree as the documents, but then mark them as programs to be executed by their name, i.e. because the name of this page ends in *.html*, it's a document, and because this one ends in *.pl*, it's a program. This approach is considered insecure and is not often followed. It makes no difference to Perl or CGI work, except in that the URLs don't include a script path.
- Server Side Includes

For some straightforward tasks, the user may ask the server to parse documents before they're delivered to the browser. Directives can be included within such documents, which are then acted upon by the server; these range from inserting a date through to running an application.

Just as with CGI, it's up to the server administrator to decide whether or not to allow Server Side Includes (SSIs), and he can allow them in a variety of ways:

- For all files
- For files with specific extensions
- Not at all

Our course server is set up to parse any *.shtml* files, so is an example of method 2, and is perhaps the most common. For a low-traffic, feature-rich intranet site on a powerful server, you would probably choose to parse all files. "Free Web Space" servers will probably not support the facility at all!

Here's an HTML file as held on the server:

```
<HTML>
<BODY BGCOLOR=white text=black>
This is a dynamically changing page which uses Server Side Includes.<P>
There is no need to write a CGI script in some of the simpler cases
making the provision of "slightly dynamic" pages easier and quicker
even (!) than using CGI. Let me show you:<P>

<H2>The current date and time is
<!--#echo var="DATE_LOCAL"--><P></H2>

See - each time you reload the page, that changes, doesn't it?
</BODY>
</HTML>
```

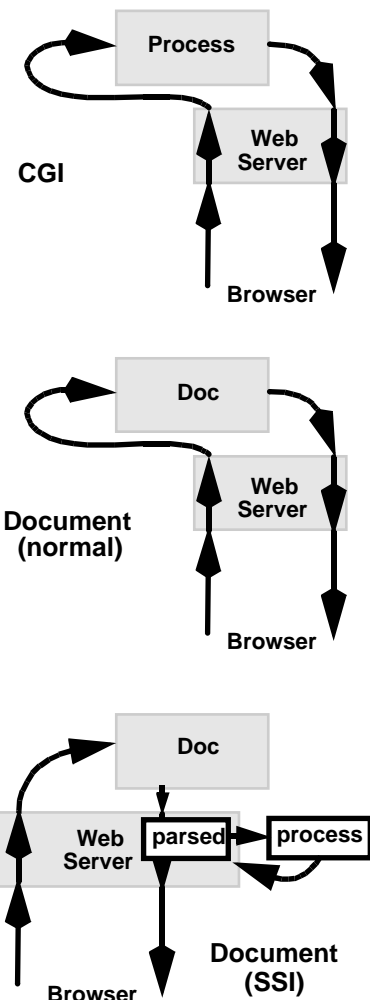
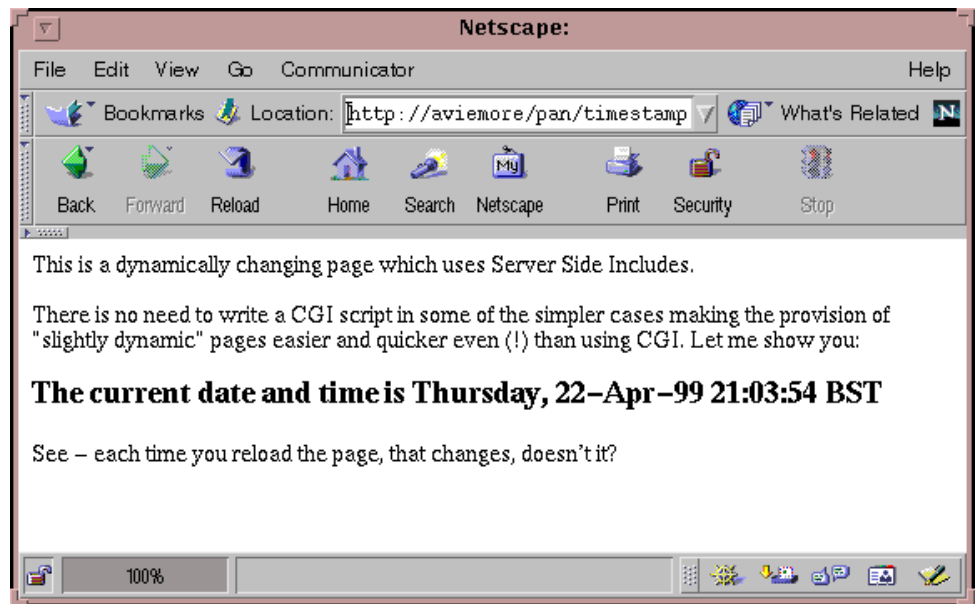


Figure 6 Three ways of running documents in a different subtree on the server

And here's the same page saved on a browser:

Figure 7 A dynamically changing page using SSI



```
<HTML>
<BODY BGCOLOR=white text=black>
This is a dynamically changing page
which uses Server Side Includes.<P>
There is no need to write a CGI script
in some of the simpler cases making
the provision of "slightly dynamic"
pages easier and quicker even (!)
than using CGI. Let me show you:<P>

<H2>The current date and time is
Thursday, 22-Apr-99 21:03:54 BST<P>
</H2>
See - each time you reload the page,
that changes, doesn't it?
</BODY>
</HTML>
```

Note that there are no special facilities required in the browser to handle this (or any other SSI features) as the dynamic data in the page is "cut in" by the server.

SSI is suitable for pages that alter dynamically and are called up by hypertext links. Because the SSI program runs purely on the output side of the server, SSI is not suitable for handling form inputs or URLs that include data added via the get method in other ways.

Server Side Include Syntax

Server Side Includes take the form:

```
<!--#
command
parameter(s)
-->
```

The `<!--#` sequence is taken as a comment by browsers which should never see it, of course, but will if you run the code on a server which has SSI disabled.

There is no space before the command ... thus

```
<!--#echo var="DATE_LOCAL"-->
```

There are six server side commands. In the next example, we have used:

echo (to include a system variable at this point)
include (to include a text file at this point)
fsize (to include a file size at this point)
flastmod (to include a file's change date at this point)

and there's also

config (to alter how error messages, file sizes and dates are displayed)
exec (to run another program and insert results)

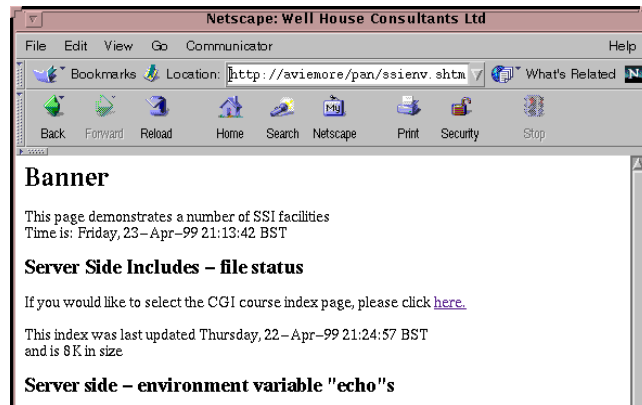


Figure 8 Using six server side includes commands

Here's the source code of the page:

```
<HTML>
<!--#include file="webtop"-->
<BODY BGCOLOR=WHITE text=black>
<!--#include virtual="/pan/letterhead"-->
Time is: <!--#echo var="DATE_LOCAL"--><P>
<H2>Server Side Includes - file status</H2>
If you would like to select the CGI course index page, please click <A
HREF="index.html">here.
</A><P>
This index was last updated
<!--#flastmod file="index.html"--><BR>
and is <!--#fsize file="index.html"--> in size
<H2>
Server side - environment variable "echo"s</H2>
SSI Environment variables:
<UL>
<LI>DATE_LOCAL<BR><!--#echo var="DATE_LOCAL"-->
<LI>DATE_GMT<BR><!--#echo var="DATE_GMT"-->
<LI>LAST_MODIFIED<BR>
<!--#echo var="LAST_MODIFIED"-->
<LI>DOCUMENT_NAME<BR>
<!--#echo var="DOCUMENT_NAME"-->
<LI>DOCUMENT_URI<BR>
<!--#echo var="DOCUMENT_URI"-->
<LI>QUERY_STRING_UNESCAPED<BR>
<!--#echo var="QUERY_STRING_UNESCAPED"-->
</UL><P>
Some other useful environment variables:
<UL><LI>SERVER_NAME<BR>
<!--#echo var="SERVER_NAME"-->
<LI>SERVER_PORT<BR>
<!--#echo var="SERVER_PORT"-->
<LI>SERVER_SOFTWARE<BR>
<!--#echo var="SERVER_SOFTWARE"-->
<LI>SERVER_ADMIN<BR>
<!--#echo var="SERVER_ADMIN"--><P>
<LI>REMOTE_HOST<BR>
<!--#echo var="REMOTE_HOST"-->
<LI>REMOTE_ADDR<BR>
<!--#echo var="REMOTE_ADDR"-->
<LI>REMOTE_USER<BR>
<!--#echo var="REMOTE_USER"--><P>
<LI>HTTP_USER_AGENT<BR>
<!--#echo var="HTTP_USER_AGENT"-->
<LI>PATH<BR><!--#echo var="PATH"-->
</UL>
<!--#include file="signature"-->
</BODY>
</HTML>
```

Which the server turns into

```

<HTML>
<HEAD>
<META HTTP-EQUIV="keywords" CONTENT="training perl java tcl">
<META HTTP-EQUIV="reply-to" CONTENT="lisa@wellho.net">
<META NAME="author" CONTENT="Lisa Ellis">
<TITLE>
Well House Consultants Ltd
</TITLE>
</HEAD>
<BODY BGCOLOR=WHITE text=black>
<H1>Banner</H1>
This page demonstrates a number of SSI facilities<BR>
Time is: Friday, 23-Apr-99 21:13:42 BST<P>
<H2>Server Side Includes - file status</H2>
If you would like to select the CGI course index page, please
click <A HREF="index.html">here.</A><P>
This index was last updated
Thursday, 22-Apr-99 21:24:57 BST<BR>and is 8K in size
<H2>Server side - environment variable "echo"s</H2>
SSI Environment variables:
<UL><LI>DATE_LOCAL<BR>Friday, 23-Apr-99 21:13:42 BST
<LI>DATE_GMT<BR>Friday, 23-Apr-99 20:13:42 GMT
<LI>LAST_MODIFIED<BR>Friday, 23-Apr-99 11:33:10 BST
<LI>DOCUMENT_NAME<BR>ssienv.shtml
<LI>DOCUMENT_URI<BR>/pan/ssienv.shtml
<LI>QUERY_STRING_UNESCAPED<BR>(none)
</UL><P>
Some other useful environment variables:
<UL><LI>SERVER_NAME<BR>aviemore
<LI>SERVER_PORT<BR>80
<LI>SERVER_SOFTWARE<BR>Apache/1.0.3
<LI>SERVER_ADMIN<BR>graham@wellho.demon.co.uk<P>
<LI>REMOTE_HOST<BR>magnet
<LI>REMOTE_ADDR<BR>192.168.200.225
<LI>REMOTE_USER<BR>(none)<P>
<LI>HTTP_USER_AGENT<BR>Mozilla/4.5 [en] (X11; I; SunOS 5.6 sun4c)
<LI>PATH<BR>/usr/sbin:/usr/bin
</UL>
<H4>Well House Consultants<BR>
404, The Spa<BR>
Melksham<BR>
Wiltshire<BR>
SN12 6QL<P>
graham@wellho.net<P>
01225 708225 (phone)<BR>
01225 707126 (fax)<BR></H4>
</BODY>
</HTML>

```

The environment variables listed (shown through the "echo" commands) are not the only ones available; the full set of CGI variables should be available in addition to the extras shown at the top of the list.

Include commands

There's often a desire to include one item within another, and indeed this can be

done using HTML where a text file can include an image, (using ``), and applet (`<Applet ...>`) or some other content (`embed`, `object`, `script`, etc ...). But, amazingly, there's no way of including one text (HTML) file within another in HTML.

Why would you want to? Probably to share a set of standard headers and footers between a whole number of pages. Yes, you might use cascaded style sheets, but is everyone's browser going to support them? What if you want to change the header file regularly to include a message of the day?

Use SSI and the `include` command. This allows you to bolt the contents of one file ("boiler plate") into another. The parameter you can give is:

`file=` to include a file relative to current directory
`virtual=` to include a file giving a path on the document path.

Thus:

```
<!--#include file="webtop"-->
<!--#include virtual="/pan/letterhead"-->
<!--#include file="signature"-->
```

Include commands have added in the following files:

- signature

```
<H4>Well House Consultants<BR>
404, The Spa<BR>
Melksham<BR>
Wiltshire<BR>
SN12 6QL<P>
```

```
graham@wellho.net<P>
```

```
01225 708225 (phone)<BR>
01225 707126 (fax)<BR></H4>
```

- webtop

```
<HEAD>
<META HTTP-EQUIV="keywords" CONTENT="training perl java
linux">
<META HTTP-EQUIV="reply-to" CONTENT="lisa@wellho.net">
<META NAME="author" CONTENT="Lisa Ellis">
<TITLE>
Well House Consultants Ltd
</TITLE>
</HEAD>
```

- letterhead

```
<H1>Banner</H1>
```

```
This page demonstrates a number of SSI facilities<BR>
```

Notes:

- Formatting capability available
- You can include formatting tags into the included files, but just be certain that you balance them, otherwise they'll become a nightmare to maintain.
- No recursion / further SSI
- You can't include a file within an included file. Indeed, the SSI processor does not parse the included file so you can't include any of the other directives either.
- Includes can go into header too
- An "Include" doesn't have to add anything visible to the page – it can be used for headers, or even to pick up JavaScript functions from a central library. In our example, we've put all the header tags that the search engines like into the header ;) and changed the title.
- `file` = is relative only

- From the description, you might hope that `file=` allowed you to put in an absolute-pathed, operating system-level file name, but it does not. You can do such using `exec`.

Executable content in CGI

The most powerful SSI command is `exec` which allows you to run an external program, as shown on the diagram in the introduction to this SSI section.

From this web page:

```
<HTML>
<BODY BGCOLOR=WHITE text=black>
<H2>The Server's message of the day is:</H2>
<PRE>
<!--#exec cmd="/usr/bin/cat /etc/motd"-->
</PRE>
<H2>Here is a map of the files in this web directory</H2><PRE>
<!--#exec cmd="ls -laR $DOCUMENT_ROOT/pan |/usr/local/bin/perl -p
/export/home/cgi-bin/pan/linkadd.pl"--><BR>

</PRE>
</BODY>
</HTML>
```

We displayed these results:

Figure 9 Executable content

The first `exec`:

```
<!--#exec cmd="/usr/bin/cat /etc/motd"-->
```

is simply a way of including a file using an absolute operating system path – you recall that `include` didn't let us do that. The second is more interesting:

```
<!--#exec cmd="ls -laR $DOCUMENT_ROOT/pan
|/usr/local/bin/perl -p /export/home/cgi-bin/pan/linkadd.pl"--><BR>
```

as we run a more complex operating system command. All highly system dependent, with hard wired paths and the like, but very powerful (and at long last, after the SSI excursion, we're using Perl again).

The Perl program used was as follows:

```
# This script called directly from an "shtml" page
# linkadd.pl

# Directory lines - Hold on to directory name

if (m#^/#) {
    $current_dir = $_;
    $current_dir =~ s/\extra\disc0.slice5\www//;
    $current_dir =~ tr/:\//;
    chop ($current_dir);
}

# File lines - add an HREF'd link

if (/^-/) {
    chop;
    s/ (\S+)$/ <A HREF=http:$current_dir$1>$1<\A>\n/;
}
```

Notes:

There's no **#!** line on the top of the Perl program, because we're running it explicitly from the **exec** statement.

Recall that the **-p** command-line option to Perl adds an implicit loop around the code in the file, so:

```
#!/bin/perl -p
    s/one/two/
```

is the same as

```
#!/bin/perl
while (<>) {
    s/one/two/
    print ;
}
```

The **<>** file handle, in our program by implication, reads from files named on the command line (there aren't any in our example) or STDIN. And within a **while** statement, if the only operator in the condition is a read from, the data read is placed in the variable **\$_**.

\$_ in Perl is used as the default variable in many operations, for example in **s/.../.../** operations, in matching operations, and in print operations if no parameters are given. Thus:

```
if (/^-/) {
    "if the contents of $_ starts with a minus sign"
    chop;
    "remove the last character from the string in $_"
```

With matching, you can change the delimiter by using the letter "m"; we chose to do so in one case here to avoid having to protect the **/**:

```
if (m#^/#) {
```

and again this is a match against **\$_**.

If brackets are used in a pattern match (or substitute) then the string that matches that part of the regular expression is saved into **\$1**; subsequent bracket pairs use **\$2**, **\$3**, etc. This allows us within a substitute to re-specify parts of the matched string in the output:

```
s/ (\S+)$/ <A HREF=http:$current_dir$1>$1<\A>\n/;
```

replaces the characters from the last space in **\$_** through to the end of **\$_** with the same text within an anchor or hypertext reference surrounding it.

Thus:

```
-rw-r--r-- 1 graham wellho 378 Apr 17 08:40 checkdform.html
```

becomes

```
-rw-r--r-- 1 graham wellho 378 Apr 17 08:40  
    <A HREF=http://pan/checkdform.html> checkdform.html</A>
```

To conclude on server side includes ... they are a quick and easy way of building simple changes into a document on your server without going the whole way through a CGI script. You can execute programs and save the results into the page, and you can collect the query string – thus use the `get method` – so it's surprising how powerful they could become.

On the other hand, you're probably better off using CGI for the more complex work; our last "exec" example could have been done just as easily that way, and that would have been more conventional for sure.

Exercise

License

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log
Original Version, Well House Consultants, 2004

Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____

License Ends.