

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

Shopping Cart Application in PHP

Many PHP applications are based around a shopping cart – a user selecting a series of products from around a site, which are then presented as a total order, cost is added up, then is checked out. This module is an example of such a script, and covers issues such as handling user entry errors, selection of products by category where you have too many products to fit on one page, cleaning up aborted order files, and accepting credit card payments on line.

<i>The store front page</i>	4
<i>Under the hood</i>	7
<i>Adding to the shopping cart</i>	9
<i>User Details</i>	10
<i>The final phase</i>	14
<i>Complete carter.php4 file</i>	16
<i>Using a secure server for credit card details</i>	19
<i>Commercial matters</i>	23

This module is a worked example of a simple shopping cart application which uses the power of PHP and its functions to provide session tracking for numerous visitors at the same time. It also performs data validation, templating of HTML to provide a consistent look and feel from page to page and the ability for that look and feel to be maintained through the use of an HTML editor such as Dreamweaver.

The application uses a single URL though which all the stages of the process are accessed. A session variable (`$step`) is used to track how far through the user has been, once it's established that a session is active.

- Step 1 User is selecting products from the store
- Step 2 User is entering his details (e.g. email address)
- Step 3 User is entering secure details (e.g. credit card)
- Step 4 Order is logged. User is thanked.

Each of the batchlets that comprises the process is threaded into a single script that goes through three application phases each time it is run:

- Phase A Handle incoming form information and decide if we can move on to the next step
- Phase B General work such as reading the products
- Phase C Handling outgoing information (working out what has to go onto the output template)

Within each phase, you'll see that we have a **switch** statement to select appropriate code. Note that the step variable changes between the switch at Phase A and the switch at Phase C, which means that a user can enter his details on a form and have them received (Step 2, Phase A), and if they're valid, the program will go on to Step 3/Phase C to generate a form asking for secure details.

2.1 The store front page

Here's the front page of our store:

Figure 1 Our store front, carter.php4

Four elements are used in its creation:

- i) The script `carter.php4` which controls the whole operation
- ii) Functions in the file `getshop.inc`, which is included
- iii) HTML from a template file called `offer.htm`
- iv) Data about what the shop's called and what it offers - file `products`

Let's follow the creation of the front page:

```
session_start();
include_once("getshop.inc");

if (! session_is_registered("cart")) {
    // New session - initialise for order entry
    session_register("cart","currentaisle","step","details");
    $cart = array();           # Current cart contents
    $details = array(email => "", postal => "",
        name => "", card => "");           # Details of person
    $step = 1;                # How far through?
                                # 1 - selecting from aisles
                                # 2 - giving own details

    $currentaisle = "";
}
```

It's a new session, so the shopping cart is created and starts off empty. An array for the user's details is also created with all its elements empty. We also make a note of the current aisle and step, as these are things that we wish to persist. We're not really into Step 1 yet while this code is being run, but nevertheless the common code is run:

```
////////// ANY GENERAL WORK TO BE DONE

$fill = array_merge($details,$alert);
$fill[storename] = readproducts($products,$aisles,$currentaisle);
$fill[step] = $step;
$fill[totsteps] = 3;
```

This code starts to set up the "fill" array which is going to be used to complete the appropriate template just before the end of the batchlet. We actually know which template we'll be completing as our `$step` variable has been set, but some further information may yet have to be worked out.

Much of the work of Phase B is encapsulated in, or hidden within, our `read-products` function. This function sets up arrays of products and aisles from a data file, modifies the `$currentaisle` if necessary¹ and also returns the store title. Although functions usually take inputs from parameters and return values to be assigned to a variable, in this case we've broken from the norm and used parameters for return values. This is a great way of dealing with a whole lot of variables to pass back, but make it clear in your documentation!

Let's now prepare to offer the products to the visitor:

```
$bask = "";
reset ($cart) ;
while ($item = each($cart)) {
    if ($item[value] > 0) {
        $haveitems = 1;
        $blines .= "<tr><td>".$products[$item[key]][title].
```

¹ it is necessary this first time through, as we put out newcomer into the entry aisle

```

        "</td><td>".
        $item[value].
        "</td><td>".
        $products[$item[key]][days].
        "</td><td>".
        ($dc = $item[value]*$products[$item[key]][days]).
        "</td></tr>";
    $totaldays += $dc;
}
}
if ($blines) $bask = "<table border=1><tr><th>Product</th><th>places</th>".
    "<th>duration</th><th>total days</th></tr>$blines</table><br><br>";
$fill["bask"] = $bask;

```

That was pretty pointless the first time through as we have nothing in the cart yet and the loop will have been skipped over completely.

Once our user starts adding in products we always want to remind him what he has. Lets's **switch** ...

```

switch ($step) {
case 1:
    // Offer Selection of goods from aisles
    // Work out the products on the current Aisle
    $ci = "<table border=1>";
    reset($products);
    while (list($code,$product) = each ($products)) {
        if ($product[aisle] == $currentaisle) {
            $ci .= "<tr>";
            $ci .= "<td>".$product[title]."</td>";
            $ci .= "<td>".$product[days]."</td>";
            $ci .= "<td><input size=3 name=__".$code.
                " value=".$cart[$code].
                "></td>";
            $ci .= "</tr>";
        }
    }
    $ci .= "</table>";
    $fill["ci"] = $ci."<input type=submit value='update cart'><hr>";

```

Going through the array of available products, we're making up a table of all those that are on the current aisle. Each product has a part code which we're pre-pending with `__` to give a field name that will be easily identified as a product code field.

We're now running code that will be common with all the pages that we generate while our user is browsing our shelves. Notice how we pre-fill the user entry box with the number of products he has already selected of each type, allowing him to increase or decrease the number of each product easily.

```

// Work out other departments

$od = "You are currently in ".$aisles[$currentaisle]."<br>";
reset ($aisles);
while ($aisle = each($aisles)) {
    if ($aisle[key] == $currentaisle) continue;
    $od .= 'Go to <input type=submit name=jump value="'.
        $aisle[value].'"><br>';
}
$fill["od"] = $od;

```

That code works out a table of all the other aisles that are available, and writes a series of submit buttons to allow the user to single-click navigate the store. Finally, the Step 1/Phase C code produces a summary line to tell us a total of what's in the basket. As this is the initial entry to a script, the answer will be "nothing".

```
// Summarise what's in the basket so far

$haveitems or $fill[bask] .= "Nothing in the cart yet<br>";
if ($haveitems) {
    $fill["bask"] .= "TOTAL TRAINING IN BASKET - $totaldays days<br><br>";
    $fill["bask"] .= "Select <input type=submit name=go_out value=here> to".
        " checkout the contents of your basket<br>";
}
$nextpage = "offer.htm";
break;
```

You'll have noticed the setting of `$nextpage`; that's the variable that contains the template for shopping around the store. The very end of our *carter.php4* script calls a function to read and complete that template, and then spits it out to the browser.

```
$html = filltemplate($nextpage,$fill);
print ($html);
?>
```

2.2 Under the hood

The storefront generation probably looked shorter than you expected. That's largely because much of the work is done elsewhere.

getshop.inc includes the `readproducts` function that reads in our data file and populates assorted variables and the `filltemplate` function.

There's nothing very special about `readproducts`, although if you're not familiar with arrays of arrays you might find it a bit daunting. Other little things to note in the code:

- Use of `&$` in the parameters to call by name
- Provision of a capability to comment the data file
- Use of `explode` on a tab-separated variable file (very useful if our data is likely to come from Excel)
- Telling our data line types apart by counting the fields.

```
<?php

function readproducts(&$products,&$aisles,&$currentaisle) {

// Data file products has lines as follows:
// Store title line                # no tabs
// P   Perl Courses                 # one tab
// P   PP      4      Perl Programming # three tabs

// Read in the products that we offer!

$fh = fopen("products","r");
while (! feof($fh)) {
    $parts = explode("\t",preg_replace("/#.*\/"," ",fgets($fh,1024)));
    if (count($parts) < 2) {
```

```

        # Title record or nothing
        if (ereg("[:graph:]", $parts[0])) {
            $title = $parts[0];
        }
        continue;
    }
    if (count($parts) < 4) {
        # Aisle Record
        $aisles[$parts[0]] = $parts[1];
        $defaultaisle or $defaultaisle = $parts[0];
        if (trim($_POST[jump]) == trim($parts[1]))
            $currentaisle = $parts[0];
    } else {
        # Product Record
        $products[$parts[1]][aisle] = $parts[0];
        $products[$parts[1]][days] = $parts[2];
        $products[$parts[1]][title] = $parts[3];
    }
}

$currentaisle or $currentaisle = $defaultaisle;

return $title;
}

function filltemplate($file, $filler) {
    $fh = fopen($file, "r");
    $html = fread($fh, filesize($file));
    foreach (array_keys($filler) as $fld) {
        $html = preg_replace("/%$fld%/", $filler[$fld], $html);
    }
    return $html;
}

?>

```

```

Scheduled Course Shop
P   Perl Courses
H   PHP Courses
M   MySQL training Courses
J   Java Courses
T   Tcl Courses
P   PP           4   Perl Programming
P   LP           5   Learning to program in Perl
P   PL           3   Perl for larger projects
P   PW           2   Perl for the Web
H   PG           1   Technology for PHP
H   PH           3   PHP Programming
M   MQ           2   MySQL
J   JP           5   Java Programming for the Web
T   TB           3   Tcl and Expect basics
T   TK           2   Tk Programming

```

Figure 2 The products data file that we used for the illustrations in these notes

We won't explain the data file format here, since we've correctly commented it in the code above.

The other function in *getshop.inc* is `filltemplate`. It's just a few lines of code, but a powerful loop and regular expression performs a `mailmerge` type operation, removing special tags from the HTML and replacing them with parameter values. Any similarity to the structure of PHP itself is, of course, totally accidental.

Here's the template before it was filled in:

```
<html>
<head><title>%storename% - Product selection</title></head>
<body bgcolor=white>
<center><h1>%storename%, step %step%</h1></center>
Please feel free to browse around our store and make your selections. You
may update how many of each product in the current aisle you want, and
place them in your cart using either the update button, or a button to
select a different aisle. Once you have completed your selections from all
the aisles, select the checkout button and you'll be asked for your
details. Finally, you'll be asked for your credit card information on our
secure server.
<form method=post>
<table border=1><tr><td>
<h2>Current Aisle selection</h2>
%ci%
<h2>Move to a new aisle</h2>
%od%
</td><td>
%bask%
</td></tr>
</table>
</form>
</html>
```

The layout of pages such as this can be developed through an application such as Dreamweaver. Laying out of tables within tables, conforming exactly to the W3 standards for coding, getting JavaScript exactly right, etc., is more easily done with such tools than writing it yourself on a text editor.

2.3 Adding to the shopping cart

Once the front page has been generated, most of the hard work has really been done. To add products to the shopping cart, the only extra code is Step 1/Phase A:

```
switch ($step) {
// Incorporate selection of goods from aisles
case 1:
    reset($_POST);
    while ($item = each($_POST)) {
        if (ereg("^__(.*)", $item[key], $got)) {
            $cart[$got[1]] = $item[value];
        }
    }
    if ($_POST[go_out]) $step = 2; // Move on to checkout!
    break;
```

This takes any field values starting `__` and adds them to the cart. It also notes a move on to Step 2 (entry of user information) if the submit button inviting a checkout has been selected.

Here's the sort of display you'll be getting as you navigate around the store and add to your cart:

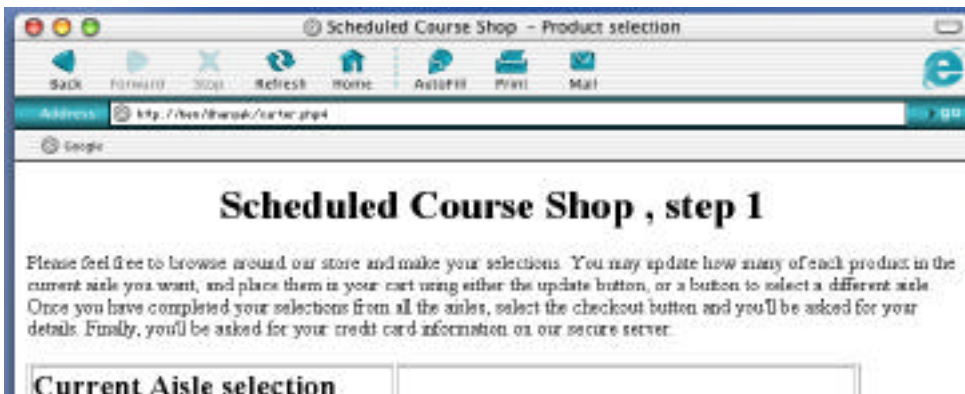


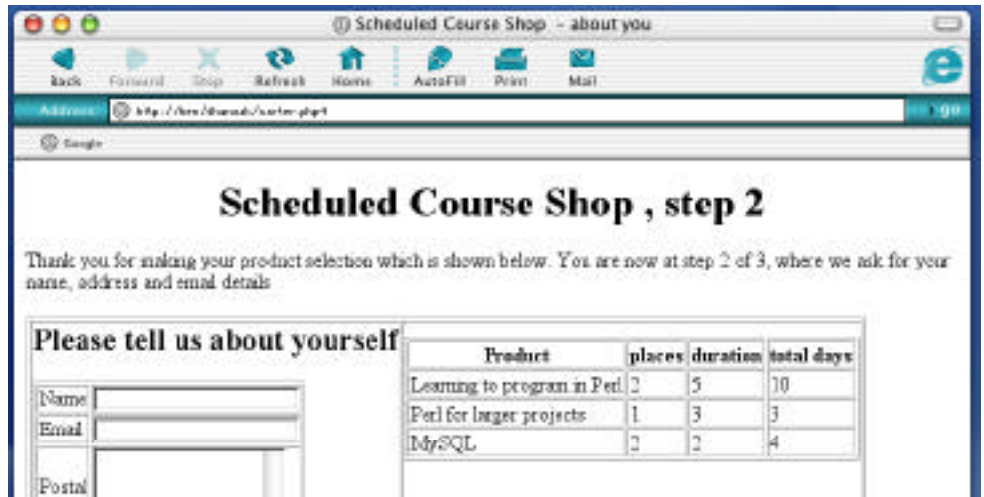
Figure 3 Navigating around the store and entering information

2.4 User Details

User details – Step 2 – brings some new issues. We require the user to enter a number of fields, each of which is to be checked as best as we can to make sure that we have input, and it's sensible. We then want to go back to the Step 2 page, flagging that we can't carry on until it's been filled in properly, or that we want to move on if we can.

The first display in Step 2 will look like:

Figure 4 Initial request for user details



Scheduled Course Shop , step 2

Thank you for making your product selection which is shown below. You are now at step 2 of 3, where we ask for your name, address and email details

Please tell us about yourself

Product	places	duration	total days
Learning to program in Perl	2	5	10
Perl for larger projects	1	3	3
MySQL	2	2	4

Name:

Email:

Postal:

Then we'll validate any data entered with the following code:

```
// Entry of user's name and address
case 2:
    if (ereg("[[:graph:]]@[[:graph:]]",$_POST[email])) {
        $details[email] = $_POST[email];
    } else {
        $alert[xemail] = "<BR>REQUIRED";
    }

    if (ereg("[[:graph:]]",$_POST[name])) {
        $details[name] = $_POST[name];
    } else {
        $alert[xname] = "<BR>REQUIRED";
    }

    if (ereg("[[:graph:]]",$_POST[postal])) {
        $details[postal] = $_POST[postal];
    } else {
        $alert[xpostal] = "<BR>REQUIRED";
    }

    if ($details[name] and $details[email] and $details[postal]) $step=3;
    break;
```

That's a good use of regular expressions. Notice that once we get a valid input, we copy it into the session variables; we don't want to loose it!

If an entry is incorrect or missing, we set up an extra variable with a little error message. When we look at the template for this page in a moment, you'll see that we have provided a whole series of fill-ins so that we can get back to the user. Just concluding the code above, though, if we have valid inputs for all of the fields we need, we can move on to Step 3.

Here's the variable part of the user information form:

```
<h2>Please tell us about yourself</h2>
<table border=1>
<tr><td>Name%xname%</td><td><input name=name value="%name%"></td></tr>
<tr><td>Email%xemail%</td><td><input name=email value="%email%"></td></tr>
<tr><td>Postal%xpostal%</td><td><textarea name=postal cols=25
rows=4>%postal%</textarea></td></tr>
</table>
```

Let's see what it would look like with an error:

Figure 5 The user is asked to fill in his email address (it is "required")

Scheduled Course Shop - about you

Address: http://www.ozonix.com/step2.php

Scheduled Course Shop , step 2

Thank you for making your product selection which is shown below. You are now at step 2 of 3, where we ask for your name, address and email details

Please tell us about yourself

Name	Graham Ellis
Email	REQUIRED
Address	404 The Spa

Product	places	duration	total days
Learning to program in Perl	2	5	10
Perl for larger projects	1	3	3
MySQL	2	2	4

Oops, back to Step 2..
Let's move on to Step 3 with a correct entry:

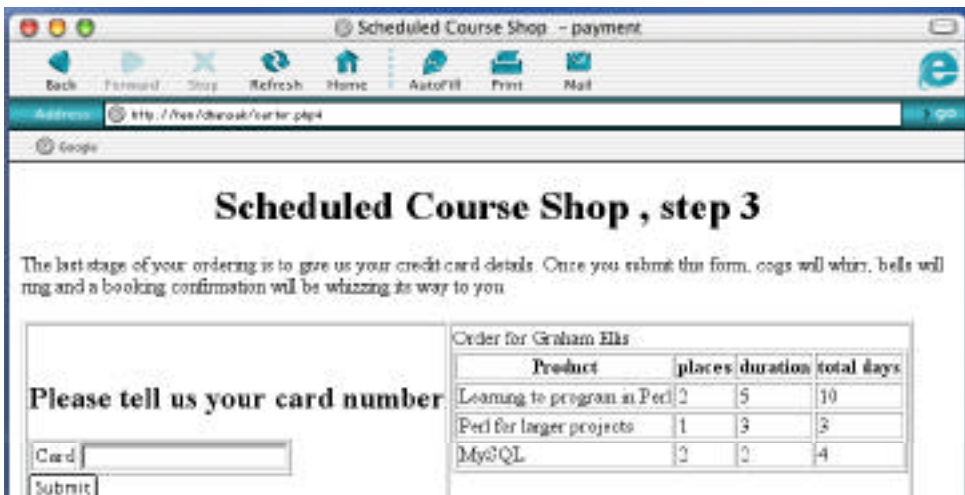


Figure 6 A correct input then took us to the next entry page

That's better.

2.5 The final phase

Once the credit card details have been accepted,¹ our script needs to email the user to confirm payment, place the order on a database, and destroy the session. Here's our sample code:

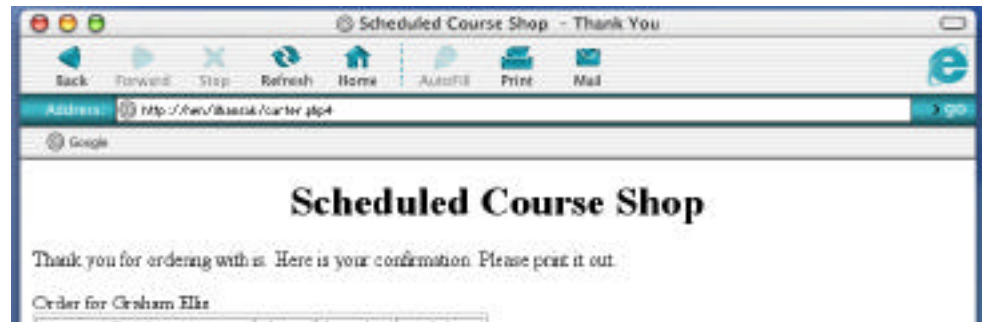
```
case 4:
    // Order placed. Send email. Write to database.
    session_destroy();
    $nextpage = "thankyou.htm";
    $fill["basket"] .= "TOTAL TRAINING IN BASKET - $totaldays days<br><br>";
    break;
```

You'll note that `$step` has increased up to 4, even though we only declare three phases to the user. Ah, and there was a step 0 as well, if you recall, so it's really a five-step process.

¹ we hope you switch to a secure server!

Although there might be substantial work to do at this last stage, the final display from *carter.php4* is a simple screen thanking our user for his order, telling him when and how to accept delivery, how to contact us for any reason should he need to do so, and wishing him well with our product:

Figure 7 Synopsis of the order and "thank you" acknowledgement



Take care to ensure that you don't leave your user stranded when he's finished buying. Offer him an onward-going link. Since our site in this example was offering training courses, shall we send our customer to the joining instructions?



Figure 8 Giving an onward link instead of just leaving the user hanging at the synopsis

2.6 Complete carter.php4 file

Here's a complete listing of the *carter.php4* file that you've seen in sections throughout this module. You'll find it ties together quite easily once you're happy with the batchlet concept. Programming is an art as well as a science, and in the case of a PHP application like this, the artist needs to come to the fore.

```
<?php

/* Session Demonstration Program that uses text files to note what's on which aisle of a store.
Users may browse around the aisles, and later may check out via a secure server */

session_start();
include_once("getshop.inc");
$trace = ""; # Logging variable
$alert = array(xpostal => "", xemail => "", xname => "", xcard => "");

////////// THINGS TO BE DONE ON ENTRY TO A PAGE

if (! session_is_registered("cart")) {
    // New session - initialise for order entry
    session_register("cart","currentaisle","step","details");
    $cart = array(); # Current cart contents
    $details = array(email => "", postal => "",
        name => "", card => ""); # Details of person
```

```

$step = 1;                                # How far through?
                                           # 1 - selecting from aisles
                                           # 2 - giving own details

$currentaisle = "";
} else {
switch ($step) {
// Incorporate selection of goods from aisles
case 1:
    reset($_POST);
    while ($item = each($_POST)) {
        if (ereg("^__(.*)", $item[key], $got)) {
            $cart[$got[1]] = $item[value];
        }
    }
    if ($_POST[go_out]) $step = 2; // Move on to checkout!
    break;
// Entry of user's name and address
case 2:
    if (ereg("[:graph:]"@[:graph:]", $_POST[email])) {
        $details[email] = $_POST[email];
    } else {
        $alert[xemail] = "<BR>REQUIRED";
    }

    if (ereg("[:graph:]", $_POST[name])) {
        $details[name] = $_POST[name];
    } else {
        $alert[xname] = "<BR>REQUIRED";
    }

    if (ereg("[:graph:]", $_POST[postal])) {
        $details[postal] = $_POST[postal];
    } else {
        $alert[xpostal] = "<BR>REQUIRED";
    }

    if ($details[name] and $details[email] and $details[postal]) $step=3;
    break;
case 3:
    if (ereg("[:digit:]{16}", $_POST[card])) {
        $details[card] = $_POST[card];
    } else {
        $alert[xcard] = "<BR>Just 16 digits<BR>REQUIRED";
    }
    if ($details[card] ) $step=4;
    break;
}
}

```

```

////////// ANY GENERAL WORK TO BE DONE

```

```

$fill = array_merge($details, $alert);
$fill[storename] = readproducts($products, $aisles, $currentaisle);
$fill[step] = $step;
$fill[totsteps] = 3;

```

```

////////// THINGS TO PREPARE FOR NEXT PAGE

// Always need to evaluate what's in the cart!

$bask = "";
reset ($cart) ;
while ($item = each($cart)) {
    if ($item[value] > 0) {
        $haveitems = 1;
        $blines .= "<tr><td>".$products[$item[key]][title].
            "</td><td>".
            $item[value].
            "</td><td>".
            $products[$item[key]][days].
            "</td><td>".
            ($dc = $item[value]*$products[$item[key]][days]).
            "</td></tr>";
        $totaldays += $dc;
    }
}
if ($blines) $bask = "<table border=1><tr><th>Product</th><th>places</th>".
    "<th>duration</th><th>total days</th></tr>$blines</table><br><br>";
$fill["bask"] = $bask;

///// Following depend on what next page is to be!

switch ($step) {
case 1:
    // Offer Selection of goods from aisles
    // Work out the products on the current Aisle
    $ci = "<table border=1>";
    reset($products);
    while (list($code,$product) = each ($products)) {
        if ($product[aisle] == $currentaisle) {
            $ci .= "<tr>";
            $ci .= "<td>".$product[title]."</td>";
            $ci .= "<td>".$product[days]."</td>";
            $ci .= "<td><input size=3 name=__".$code.
                " value=".$cart[$code].
                "></td>";
            $ci .= "</tr>";
        }
    }
    $ci .= "</table>";
    $fill["ci"] = $ci."<input type=submit value='update cart'><hr>";

    // Work out other departments

    $od = "You are currently in ".$aisles[$currentaisle]."<br>";
    reset ($aisles);
    while ($aisle = each($aisles)) {
        if ($aisle[key] == $currentaisle) continue;
        $od .= 'Go to <input type=submit name=jump value="'.
            $aisle[value].'"><br>';
    }
    $fill["od"] = $od;

    // Summarise what's in the basket so far

```

```

$haveitems or $fill[bask] .= "Nothing in the cart yet<br>";
if ($haveitems) {
    $fill["bask"] .= "TOTAL TRAINING IN BASKET - $totaldays days<br><br>";
    $fill["bask"] .= "Select <input type=submit name=go_out value=here> to".
        " checkout the contents of your basket<br>";
}
$nextpage = "offer.htp";
break;

case 2:
    $nextpage = "getaddy.htp";
    $fill["bask"] .= "TOTAL TRAINING IN BASKET - $totaldays days<br><br>";
    break;

case 3:
    $nextpage = "getccard.htp";
    $fill["bask"] .= "TOTAL TRAINING IN BASKET - $totaldays days<br><br>";
    break;

case 4:
    // Order placed. Send email. Write to database.
    session_destroy();
    $nextpage = "thankyou.htp";
    $fill["bask"] .= "TOTAL TRAINING IN BASKET - $totaldays days<br><br>";
    break;
}

$html = filltemplate($nextpage,$fill);
print ($html);
?>

```

2.7 Using a secure server for credit card details

What is SSL and https?

The example that we've seen in this module goes through a series of pages (steps) on our regular server. We have, though, carefully kept apart the page that requested credit card details which users will expect to provide via a secure server.

If you use an "https" rather than an "http" URL, the communications that your browser sends are encoded, to be decoded by the server. Using the facility ensures that your packets of data cannot be read at any intermediate system; they will just appear to contain garbage.

How can you use https?

There are a number of ways that this secure server requirement (and the services behind it) can be met:

1. You can register your domain and server for secure service with *www.thawte.com*, and run your own secure server. Apache includes support for "SSL" (Secure Sockets Layer), so that part is no cost, but the annual fee to Thawte to have them check out your authenticity is US\$159, after a higher charge in the first year. Without this authority, you can run SSL but all your visitors will be told that the certificate is not recognised, which will probably frighten them more than an untrusted page!



Figure 9 The Thawte page – a good source for certificates

Figure 10 If you try to use https without a certificate...

2. You can purchase space with an ISP that already has a certificate for the machine hosting your web site, and then use his certificate. This will mean that the URL of your secure pages will not be your own domain, but will be ISP-based. This is usually more than acceptable to the knowledgeable customer, as he figures out that you purchased your secure work from an expert.
3. You can purchase secure space elsewhere, away from your main server machine. Since there are not many ISPs offering same-system security,¹ this is a frequently used solution.
4. You can purchase an online payment facility from a company like WorldPay, or Barclay's (their epdq service). You don't need any security certificate or SSL facilities, as your credit card transactions will be performed on your supplier's system.

¹ they mostly have just one or two machines secure licensed

Programming implications for using a secure service

Whichever method you're using to obtain your secure service, you'll want to:

- a) Ensure that the form that asks for secure input is also secure, so that your user knows he's okay. Technically, it doesn't give you extra security. Psychologically it's everything.
- b) Ensure that the form that sends the secure information is also secure. This is why you're using the security, after all, and if you turn it off after the initial form your user will be told that you've gone insecure.

Whichever method you use, you'll also be expected to look after the security of the confidential data entered once it's got to the server, and ensure that any onwards transmission is also secure.

If you have your own certificate, there's very little extra you need to do in order to offer a secure data entry page to your user beyond the items that we've just mentioned.

If you're using your ISP's certificate on the same server that hosts your site, you'll use a hidden field to chain through to the secure page, as cookies apply to particular domains and so the session cookie that you've been using up to this point won't provide the connection. You'll find that the secure document directory is a different directory to the insecure, so you'll need PHP scripts in both,¹ but the same data files can be accessed from both locations.

Here's the chaining URL link from our online shop:

```
<FORM method=post action="https://pine.he.net/~wellho/regex/mousemat.php4">
<input type=hidden name=tag value=<?php echo ($tag); ?>>
```

And here's the part of the shop that sends a confirmation email to you that we have your order, and alerts our order processing department too:

```
mail("lj7@wellho.net", "Mouse and Pen order", $emailbcc,
     "From: WHC Online Ordering system <admin@wellho.net>\n");
mail($cart["email"], "Order Confirmation from Well House Consultants", $emailtext,
     "Reply-to: lj7@wellho.net\nFrom: Lisa Ellis<lj7@wellho.net>\n");
```

Going a step further, if your secure space is on a different computer to your regular space, you'll once again chain via a hidden field. The receiving secure script can access any information it needs from the insecure server by using

```
fopen("http://xxxxxx....", "r");
```

to read from a URL on the original server. It's probably going to be a special URL that you've provided for the purpose that responds only to the IP address of the secure server. Replies and control can be passed back once you're back to the insecure server via a similar process. On some occasions you may be using one language on the secure server and a different language on the regular one – that can work fine² – it's just data being transferred. Perl has the **LWP::UserAgent** modules that can read across a network, Java can open a URL and so on.

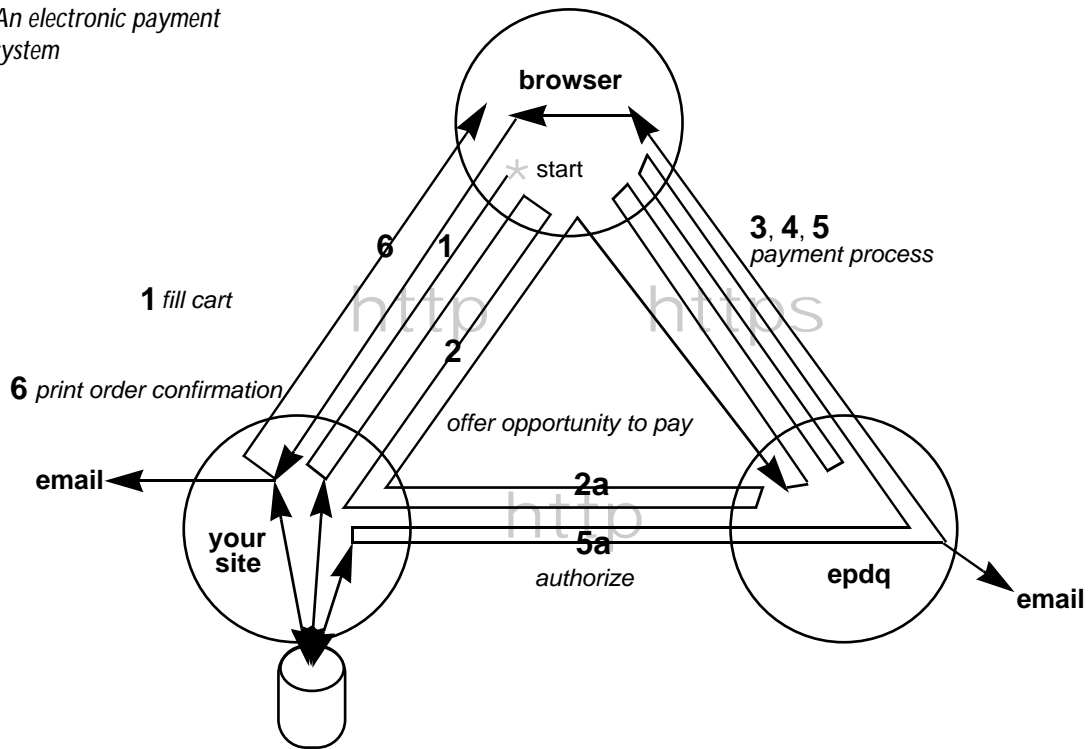
Programming to use an electronic payment system

If you interface to Barclay's epdq, or to other such systems, you don't need your own certificate, nor do you need your own secure space. There's quite a number of steps involved and you'll be encouraged to become an authorized developer. But once you get into it, it's more long-winded than difficult.

¹ yes, the include files could be in a common location

² we have written such a system

Figure 11 An electronic payment system



Your customer shops on your site, building up his order (1)

Your page calculates the customer's bill and offers him payment(2), you run some code¹ that includes an amount, an order ID, and a security string that also identifies the transaction to your store (2a).

Your customer then visits Barclay's by following the link you provide, and fills in a series of forms (3), (4) and (5). When complete (5), he'll be given a link back to your site. Barclay's also contacts a script at your site and gives you an authorization, via a script in a directory, that requires a login and is reserved for the Barclay's computer (5a).

When your visitor returns to your site (6), you can marry up the returned record from Barclay's with the data supplied in the user's HTTP request so that you know which order is which, and the payment status of it.

¹ sample provided by Barclay's

2.8 Commercial matters

Credit card agreements

Your credit card agreement will require that you don't take money unless you can ship within a certain period. Automated payment is only practical, then, if you have practically limitless stock (e.g. you are selling information) or if you have a reliable stock level system that your web site can check with or at least be reasonably up-to-date.

The automated payment system can also be used for other transaction handling too. Authorization and payment can be separated, and indeed a mail order business can replace its credit card machine with an appropriate web site. Even regular monthly payments can be set up.

Data Protection

In the UK, the obtaining, supplying and retaining of personal data is regulated under the Data Protection Act, and it's probable that you'll need to be registered if you're taking a serious interest in an electronic payment system.

The act requires that you don't abuse data nor retain it for too long, that you operate reasonable security in how you handle it, and that the subjects can have access to the data about themselves and can have it corrected if it is in error.

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.