Published by JumpX.com © 2005 All Rights Reserved

Reproduction and distribution are forbidden. No part of this publication shall be reproduced, stored in a retrieval system, or transmitted by any other means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher.

This publication is designed to provide accurate and authoritative information with regard to the subject matter covered. It is sold with the understanding that the author and the publisher are not engaged in rendering legal, intellectual property, accounting or other professional advice.

If legal advice or other professional assistance is required, the services of a competent professional should be sought. Robert Plank & JumpX.com individually or corporately, do not accept any responsibility for any liabilities resulting from the actions of any parties involved.

IMPORTANT: **Please Visit JumpX.com to receive your FREE subscription to the JumpX E-Magazine ....**

ROBERT PLANK'S SUPER SIX

TABLE OF CONTENTS

How to Make Your Own Software Generator

Mike Chen is one of many programmers who produce a lot of what's called "simpleware" -- programs that only do one thing, that only take a few days to produce -- including coming up with the idea, making it, and debugging it.

The majority of simpleware doesn't require any sort of data storage so you could get away just making the page with JavaScript, then using an e-book compiler program to make it into an EXE program. That's basically all the original Make Your Own Software did! Some "heavy-hitter" marketers made tons of money from simpleware. Armand Morin is a great example. That guy came out with some really basic stuff. He could have made a little report, saying these are directions on how to get a freeware imaging program, here's a couple of great URLs for clip art... there's how to make some text, and so on. But instead he went ahead and got 1-page logo creator developed that he could sell for $97 instead of $17. If you want to market your own simpleware product, it has to really fulfill a need that people need RIGHT NOW. Your product will really take off if it satisfies a fad. Mike Chen's FlyInAds service owed a lot of success to the pop-up blockers.

For a long time webmasters were using "traditional" pop-ups and delayed pop-ups and pop-unders and the like. Back then I had only seen a couple of sites that used moving pop-ups... now they are everywhere.

Pop-up blocking software started coming out and Mike learned about a little-known way to make a pop-up (that only works in Windows by the way) that couldn't be blocked. The real selling point was the unblockable ad style, not just the traffic exchange. Armand Morin had a product out called FlashPal Generator. This was back before PayPal offered encrypted buttons, so a lot of people were stuck selling products through PayPal while being at risk to thieves. People downloaded Armand's product, filled in a couple of details in a form, and could create as many secure buttons as they wanted. $67 in the bank.

Let's make a simple little program that creates pop-ups (like one of Armand's products does) only instead of making a pop-over we will make a modal dialog box (the kind you see in FlyInAds).

First, create the page that will be your pop-up. This is NOT the page that calls the pop-up, rather it's the page that is contained inside the pop-up.

```
<html>
<head>
<title>Wait a Second!</title>
</head>
<body marginwidth="5" marginheight="5" leftmargin="5" topmargin="5"
rightmargin="5" bottommargin="5">
<font face="Verdana">
<h3 align="center">This HTML file represents the text INSIDE my popup.</h3>
</font>
</body>
```

Save this as: new.html
Now we can make the page that calls the pop-up. I'll put a little bit of filler HTML in for you:

```
<h1 align="center">Simple Site</h1>
<p>Here's a simple site...</p>
<p><b>Lots of stuff to see here.</p>
Here's the most bare bones kind of modal dialog box:
<script language="JavaScript">
<!--
window.onload = function() {
window.showModalDialog("new.html");
}
// -->
</script>
```

This code is placed inside the window.onload function because we want to wait for the page to load first before showing the pop-up. If the dialog box was just placed at the top of the page, nothing would load until the pop-up showed.

Save this file as "pop.html" and try it out.
But if you create a pop-up you want to be able to specify a size and position. This is a modal dialog box with many of the customizations made:

```
<script language="JavaScript">
<!--
window.onload = function() {
window.showModalDialog('new.html', '', 'dialogLeft: 200px; dialogTop:
200px; dialogWidth: 200px; dialogHeight: 150px; help: No; resizable: No;
status: No;');
};
// -->
</script>
```

This sample creates a new pop-up that loads the URL "new.html", in a box sized 150 pixels high and 200 pixels wide. It's placed 200 pixels from the top and 200 pixels from the left of the screen. The rest of the parameters make the pop-up more minimalist... taking out things that aren't needed such as a "help" button.

We can set-up this JavaScript code so that any attribute that should be modified, can be changed easily:

```
<script language="JavaScript">
<!--
window.onload = function() {
var url = "new.html";
var x = "200";
var y = "200";
var width = "200";
var height = "150";
window.showModalDialog(url, '', 'dialogWidth: ' + width + 'px;
dialogHeight: ' + height + 'px; dialogLeft: ' + x + 'px; dialogTop: ' + y +
'px; help: No; resizable: No; status: No;');
};
// -->
</script>
```

Or, even better, make the pop-over creation into its own function... and just call that function from the onload, so anything can be changed easily.

```
<script language="JavaScript">
<!--
window.onload = function() {
popOver("new.html", 150, 200, 250, 150);
};
function popOver(url, x, y, width, height) {
window.showModalDialog(url, '', 'dialogWidth: ' + width + 'px;
dialogHeight: ' + height + 'px; dialogLeft: ' + x + 'px; dialogTop: ' + y +
'px; help: No; resizable: No; status: No;');
}
// -->
</script>
```

Now the 150, 200, 250, etc. are the parameters for x, y, width, and so on. Now you have that pop-over code and the next step is to write some code to CUSTOMIZE that pop-over code.

First lets create the HTML code that a person would fill out.

```
<h1 align="center">Pop-Over Generator</h1>
<form action="#">
URL: <input type="text" id="url" name="url" size="35" value="new.html"><br>
X: <input type="text" id="x" name="x" size="3" value="150"> Y: <input type=
"text" id="y" name="y" size="3" value="200"><br>
Width: <input type="text" id="width" name="width" size="3" value="250">
Height: <input type="text" id="height" name="height" size="3" value="150"><br
><br>
Your Code:<br>
<textarea id="code" name="code" cols="50" rows="5"></textarea><br>
<input type="button" value="Generate Code">
</form>
```

Oh yeah, and don't forget to remove that window.onload function in the JavaScript... we don't want a pop-over coming up when the page loads.

Getting from there to a software generator takes a few more steps, but let's just go crazy and skip ahead to the end:

```
<script language="JavaScript">
<!--
function writeCode(preview) {
finalCode = (popOver(
getObject("url").value,
getObject("x").value,
getObject("y").value,
getObject("width").value,
getObject("height").value
));
if (preview) eval(finalCode);
else getObject("code").value = finalCode;
}
function popOver(url, x, y, width, height) {
return "window.showModalDialog('" + url + "', '', 'dialogWidth: " + width
+ "px; dialogHeight: " + height + "px; dialogLeft: " + x + "px; dialogTop: "
+ y + "px; help: No; resizable: No; status: No;');";
}
function getObject(name) {
var ns4 = (document.layers) ? true : false;
var w3c = (document.getElementById) ? true : false;
var ie4 = (document.all) ? true : false;
if (ns4) return eval('document.' + name);
if (w3c) return document.getElementById(name);
if (ie4) return eval('document.all.' + name);
return false;
}
// -->
</script>
<h1 align="center">Pop-Over Generator</h1>
<form action="#">

URL: <input type="text" id="url" name="url" size="35" value="new.html"><br>
X: <input type="text" id="x" name="x" size="3" value="150"> Y: <input
type="text" id="y" name="y" size="3" value="200"><br>
Width: <input type="text" id="width" name="width" size="3" value="250">
Height: <input type="text" id="height" name="height" size="3"
value="150"><br><br>
Your Code:<br>
<textarea id="code" name="code" cols="50" rows="5"></textarea><br>
<input type="button" value="Generate Code" onclick="writeCode(false)"><input
type="button" value="Preview" onclick="writeCode(true)">
</form>
```

First of all, I added in the getObject() function so that we could read and write form variables. Remember how within a DIV or SPAN tag we used the "innerHTML" property to read text? Well in form objects we use "value" instead. The next step was to change that popOver function to return a long string of text representing the pop-over.

A good example is:

```
window.showModalDialog('new.html', '', 'dialogWidth: 250px; dialogHeight:
150px; dialogLeft: 150px; dialogTop: 200px; help: No; resizable: No; status:
No;');
```

You have to change the double quotes in that function around a bit to get the variables to get placed in the string correctly.

If you compare both popOver functions you'll see what I had to do.

Then, look at that writeCode() function:

```
function writeCode(preview) {
finalCode = (popOver(
getObject("url").value,
getObject("x").value,
getObject("y").value,
getObject("width").value,
getObject("height").value
));
if (preview) eval(finalCode);
else getObject("code").value = "window.onload = function() {\n" +
finalCode + "\n}";
}
```

That function is triggered by the Preview and Generate Code functions. When "Generate Code" is pressed, the function writeCode is called with the "preview" parameter set to false. When "Preview" is pressed, writeCode is called with "preview" set to true. The first thing happens in that function: the values from the form elements of "url", "x", "y", and the rest are gathered up and put into that popOver function, which creates the pop-over code. This string is stored into the variable "finalCode."

Next... if the preview button was pressed, we evaluate, or execute, that pop-over code. This means the pop-over is actually created. Otherwise, we take that code and paste it into the generate code textarea box. (Remember, the "\n" within a string means add a new line.)

Try it out yourself. Play around with the values, try the preview button, and try the generate code button. The generated code is supposed to be copied and pasted into an HTML document to make a pop-over appear out of thin air.

Those are basically the steps to make a generator of your own, where you have a given piece of code, and want to make it customizable. Like I said earlier, all you need to transform that HTML page into an EXE is to use an e-book compiler. I recommend Activ Ebook (Google for it).

And one last thing... if possible, put a simple opt-in form to your newsletter at the bottom of the page. If they like your program, they'll subscribe themselves.

How to Get An E-Book Boost

It's a common strategy to take a simple product someone else already has, add a clever twist to it, and have a new unique product of your own. Jimmy Brown did this by taking the Opt-In Lightning "single click subscribe" idea, and turning it into two different products.

The first was called Opt-In Boost. It was just like Opt-In Lightning but without the autoresponder. Its big selling point was that you could continue to use the autoresponder program you were already using, but you could add those one click JavaScript opt-in boxes for only $20 instead of paying the full $89.95 for Opt-In Lightning.

Jimmy's second offshoot script was called E-Book Boost, which was almost identical to Opt-In Boost. The user is given a prompt, but if they choose "yes" they're sent to an e-book download instead of opting in to a newsletter. So, if you remember, a simple on-exit Opt-In Lightning style pop-up was made using something like this:

```
<script language="JavaScript">
<!--
var message = "Hey you, I really want you to get in on my newsletter.\n"
+ "Here are the two vital things you're missing out on:\n\n"
+ "1.) Reason #1\n"
+ "2.) Reason #2\n\n"
+ "Click 'OK' below and get your self signed up!";
var exit = true;
window.onunload = function() {
if (exit && confirm(message)) {
getObject("subscribe").click();
}
}
function getObject(name) {
var ns4 = (document.layers) ? true : false;
var w3c = (document.getElementById) ? true : false;
var ie4 = (document.all) ? true : false;
if (ns4) return eval('document.' + name);
if (w3c) return document.getElementById(name);
if (ie4) return eval('document.all.' + name);
return false;
}
// -->
</script>
<form action="mailto:email@example.com" method="POST">
<input type="submit" id="subscribe" name="subscribe" value="Subscribe Me">
</form>
<a href="http://www.google.com">Leave</a><br>
<a href="http://www.google.com" onclick="exit=false;">Don't leave</a>
```

Using this for e-book downloads is a lot easier. First of all, we won't be dependant on any HTML for the pop-up to work. Since we won't be writing to any DHTML layers, the getObject() function won't be needed either.

Forwarding a person to a new URL in JavaScript is easy. A line like this will do: window.location = "http://www.google.com";

Instead of pushing the submit button on a form like we used to, send the user to the URL to download your e-book. I'm going to put it into a separate variable called "url" like this:

```
<script language="JavaScript">
<!--
// Enter your e-book download URL here
var url = "http://www.example.com/book.exe";
var message = "Check out my new book, Some E-Book.\n"
+ "With this free resource, I will teach you:\n\n"
+ "1.) Benefit One\n"
+ "2.) Benefit Two\n"
+ "3.) Benefit Three\n\n"
+ "Clicking 'OK' will start the download!";
var exit = true;
window.onunload = function() {
if (exit && confirm(message)) {
window.location = url;
}
}
// -->
</script>
```

If all you have are EXE e-books, you're all done. But if you want to share non-EXE books, like PDFs or HTML files (if you've got some sort of special report), most browsers are going to try to open those right in the window. Don't expect your free e-book (or preview for a full paid-for book) to become viral if people can't save it.

In a perfect world you could accomplish a forced download with one line of HTACCESS. But this isn't a perfect world, so we have to bring in a PHP script that sends three different headers to the user for the browser to understand this file needs to be saved instead of opened.

Here's my PHP script, save it as the filename "download.php":

```
<?php
$file = $_GET["file"];
$extension = preg_replace("/(.*)\./", "", $file);
$fileTypes = array("pdf", "zip", "exe", "txt", "htm", "html");
if (in_array($extension, $fileTypes)) forceDownload($file);
function forceDownload($fileName) {
$size = filesize($fileName);
$shortName = basename($fileName);
// IE browsers only
if (strstr($_SERVER["HTTP_USER_AGENT"], "compatible; MSIE ") !== false &&
strstr($_SERVER["HTTP_USER_AGENT"], "Opera") === false) {
header("Content-Disposition: inline; filename=\"$shortName\"");
header("Content-Type: application/octetstream; name=\"$shortName\"");
header("Content-length: $size");
}
// Non-IE Browsers
else {
header("Content-Disposition: attachment; filename=\"$shortName\"");
header("Content-Type: application/octetstream; name=\"$shortName\"");
header("Content-length: $size");
}
readfile($fileName);
}
?>
```

Let's take it apart. First, that function forceDownload() is based on some public-domain code I found on PHP.net. It just outputs the chosen file in a way that the download is forced. The code above that function is the stuff I added in. First of all, it takes the GET variable "file" and puts it into $file. So if you called the script like: download.php?file=contents.txt then it would put the string "contents.txt" into $file.

The next line takes out just the file extension (like "txt") and puts it into the variable $extension.

Then, the array $fileTypes contains a list of appropriate file extensions, things like "pdf," "zip," and so on. Then in that last line before the function, it figures out if the file requested is one of these types. You should have some sort of safeguard like this in because if you allowed any type of file to be read, someone could use this script to read the source code of your PHP files, even your HTACCESS file (yes, I've tried this and it works!)

If someone can read your HTACCESS file in a password-protected directory, they can figure out the path of the HTPASSWD file (the password list). Even though those passwords are encrypted, there are many programs available that can reverse-engineer those passwords in a matter of minutes, sometimes in seconds. So that's your little lesson on being careful which files you allow the user access to. PHP doesn't follow the HTACCESS rules since it opens files directly and doesn't go through the web server to do it.

Now, to force a downloaded file with this script, just change your "url" variable to something like: http://www.example.com/download.php?file=book.pdf

With HTACCESS, it's possible to do some crazy stuff like force *all* PDFs to download, but that's pretty much overkill.

How to Add Interactivity to Your Sales Letter

Every once in a while I see a tool that's selling like crazy on the Internet. It's such a ridiculously simple product from a programmer's point of view it should be going at a price range between $0 and $20. But it does something so cool and useful that people are willing to pay through the nose for it.

If you haven't heard about interactive sales letters, they're pretty cool. What you have is a regular sales page with a set of options in the letter every once in a while. The user can pick one of the choices, and parts of the sales letter change based on what the user has chosen.

First off, to have an INTERACTIVE sales page, you first have to have a sales page. To save space I'm just going to give you the whole interactive sales letter now and explain parts of it as we go along.

```
<html>
<head>
<style type="text/css">
h1 {
font-family: Tahoma;
color:#63ABE5;
font-size:32px;
line-height:32px;
}
h2 {
font-family: Tahoma;
color:#2EBF57;
font-size:24px;
line-height:24px;
}
body, td {
font-family:Verdana;
font-size:12px;
}
</style>
<script language="JavaScript" type="text/javascript">
<!--
data = {
ebooks: "<h2 align=\"center\">Sell 217% More E-Books Next Year</h2>"
+"<p>Here is the part of my sales copy that gets really
specific towards a certain audience...</p>",
software: "<h2 align=\"center\">Killer Add-on To Any Software
Bundle!</h2>"
+"<p>Here I'm talking about how this product helps those
people selling software products.</p>"
+"<p>I'm talking a little more here...</p>",
copywriting: "<h2 align=\"center\">Never Run Out Of Copywriting
Clients...</h2>"
+"<p>Here's why:</p>"
+"<ul>"
+"<li>Reason number one.</li>"
+"<li>Reason number two.</li>"
```

```
+"<li>Reason number three.</li>"
+"<li><b>The final reason.</b></li>"
+"</ul>",
graphics: "<h2 align=\"center\">$1 Million Web Design Secret
Recipe</h2>"
+"<p>I can talk for as long as I want.</p>"
+"<p>I can keep talking...</p>"
+"<p><b>And talking...</b></p>"
+"<p>And talking and talking and talking.</p>",
advertising: "<h2 align=\"center\">In Search Of That Ultimate
Headline</h2>"
+"<p>More sales copy here...</p>"
};
function place(area, choice) {
var contents = data[choice];
document.getElementById(area).innerHTML = contents;
}
// -->
</script>
<title>Your Site's Title</title>
</head>
<body>
<div align="center">
<table border="0" cellspacing="0" cellpadding="0" width="475">
<tr><td>
<h1 align="center">Your Introductory Headline</h1>
<h2 align="center">Sub-Headline Goes Here</h2>
<p>This might be a starting paragraph.</p>
<p>Here is another paragraph.</p>
<p>Tell me about what you do the <b>best</b>:</p>
<form method="post" action="#">
<p>
<input type="radio" name="nicheChooser" value="ebooks"
onclick="place('nicheArea', this.value)" checked>Selling E-Books<br>
<input type="radio" name="nicheChooser" value="software"
HOW TO ADD INTERACTIVITY TO YOUR SALES LETTER
45 SECTION 10
SALES PAGE TACTICS ROBERT PLANK
onclick="place('nicheArea', this.value)">Selling Software<br>
<input type="radio" name="nicheChooser" value="copywriting"
onclick="place('nicheArea', this.value)">Writing &amp;amp; Editing Sales Copy<br>
<input type="radio" name="nicheChooser" value="graphics"
onclick="place('nicheArea', this.value)">Graphic Design (site headers, e-book
covers)<br>
<input type="radio" name="nicheChooser" value="advertising"
onclick="place('nicheArea', this.value)">Advertising (Co-ops, exchanges, PPC
networks)</p>
</form>
<p> </p>
<div id="nicheArea" name="nicheArea">
<h2 align="center">Sell 217% More E-Books Next Year</h2>
<p>Here is the part of my sales copy that gets really specific towards a
certain audience...</p>
```

```
</div>
<p> </p>
<h2 align="center">The Rest of Your Sales Copy Here</h2>
<p>Here I go on and on about something else...</p>
</td></tr></table>
</div>
</body>
</html>
```

Copy all that into a new text file and save it as something like "sales.html." It doesn't really matter. What's important is that you take a look at it. Heck, you don't even have to upload it to your web server... just load it in your browser locally if you have to.

If you understand HTML and CSS then most of this source code is easy to understand. The JavaScript stuff is almost as easy once you take a second to look at it.

First, the "data" array:

```
data = {
ebooks: "<h2 align=\"center\">Sell 217% More E-Books Next Year</h2>"
+"<p>Here is the part of my sales copy that gets really
specific towards a certain audience...</p>",
software: "<h2 align=\"center\">Killer Add-on To Any Software
Bundle!</h2>"
+"<p>Here I'm talking about how this product helps those
people selling software products.</p>"
+"<p>I'm talking a little more here...</p>",
copywriting: "<h2 align=\"center\">Never Run Out Of Copywriting
Clients...</h2>"
+"<p>Here's why:</p>"
+"<ul>"
+"<li>Reason number one.</li>"
+"<li>Reason number two.</li>"
+"<li>Reason number three.</li>"
+"<li><b>The final reason.</b></li>"
+"</ul>",
graphics: "<h2 align=\"center\">$1 Million Web Design Secret
Recipe</h2>"
+"<p>I can talk for as long as I want.</p>"
+"<p>I can keep talking...</p>"
+"<p><b>And talking...</b></p>"
+"<p>And talking and talking and talking.</p>",
advertising: "<h2 align=\"center\">In Search Of That Ultimate
Headline</h2>"
+"<p>More sales copy here...</p>"
};
```

This is what's called an associative array, or an object. Each element (ebooks, software and so on) is a part of that array. So if you went to data["graphics"] you would get all the HTML code I have in there for graphics.

Everything has to be enclosed in quotes, to let JavaScript know that this text is assigned to that variable. Notice that when I wanted to tell JS that I want the text itself to have quotes, I have to put in backslashes. (The <h2 align=\"center\"> part.)

The pluses are used to put the strings of text together. You don't have to do it that way but I like to so that everything isn't crammed together on one line, and I can go back and make changes easily.

Also, notice how outside of a string of text you can space stuff out as much as you want to. I just like to line everything up so it's more readable.

So that's the array, where all the little snippets of HTML come from. If you play around with the actual page in your browser, you'll see that these pieces are what get substituted in when the user makes one of the choices.

Now let's look down at the actual choice list:

```
<form method="post" action="#">
<p>
<input type="radio" name="nicheChooser" value="ebooks" onclick=
"place('nicheArea', this.value)" checked>Selling E-Books<br>
<input type="radio" name="nicheChooser" value="software" onclick=
"place('nicheArea', this.value)">Selling Software<br>
<input type="radio" name="nicheChooser" value="copywriting" onclick=
"place('nicheArea', this.value)">Writing &amp;amp; Editing Sales Copy<br>
<input type="radio" name="nicheChooser" value="graphics" onclick=
"place('nicheArea', this.value)">Graphic Design (site headers, e-book covers
)<br>
<input type="radio" name="nicheChooser" value="advertising" onclick=
"place('nicheArea', this.value)">Advertising (Co-ops, exchanges, PPC networks
)</p>
</form>
```

First of all, when having stuff like checkboxes or radio buttons on a page, even if you don't intend to submit a form, make sure to put <form> </form> tags around them since most non-IE browsers won't even bother showing those things unless a form tag exists.

These are pretty straightforward radio buttons, the form field is called "nicheChooser." Each has values like "graphics", "advertising" and so on... the same names as those array elements from above. (NOT a coincidence.) Now there's that onclick part. "Onclick" means, when this object is clicked, run this little piece of JavaScript code.

In this case we're calling the JavaScript function called "place" and putting in two parameters. The first is just a string called "nicheArea." Notice that this is the same ID as the part of the HTML where that custom code is substituted in. Look here:

```
<div id="nicheArea" name="nicheArea">
<h2 align="center">Sell 217% More E-Books Next Year</h2>
<p>Here is the part of my sales copy that gets really specific towards a
certain audience...</p>
</div>
```

See? "nicheArea." Now, the first parameter is "nicheArea," and the second is the value of that form field. So if you clicked on the radio button with the value "copywriting", it would run the code: place('nicheArea', 'copywriting');

Go back up near the top and find the place() function. It looks like this:

```
function place(area, choice) {
var contents = data[choice];
document.getElementById(area).innerHTML = contents;
}
```

When the copywriting button is clicked, this function is run with the variable "area" set to "nicheArea" and the variable "choice" set to "copywriting." The first thing it does is look at that array called "data" and find the correct element, in this case, data["copywriting"]. If copywriting really was chosen, the variable "contents" would contain:

```
<h2 align="center">Never Run Out Of Copywriting Clients...</h2>
<p>Here is why:</p>
<ul>
<li>Reason number one.</li>
<li>Reason number two.</li>
<li>Reason number three.</li>
<li><b>The final reason.</b></li>
</ul>
```

The new HTML is now chosen, all that's left is to put it into the proper layer.
document.getElementById(area).innerHTML = contents;

This does a couple of things. First, document.getElementById gives you an object that represents a given layer. Our layer's ID is nicheArea, so this code used document.getElementById("nicheArea") as an object. This object has a property called "innerHTML", and when you set a value to it, whatever you set becomes the actual HTML in that layer.

Think of it as dumping the raw HTML in that box.

So, remember... if you want to add more sections of interactivity to your sales letter... first, add the elements to that "data" array just as it's shown above. Remember the last element in the array doesn't need a comma after it. Then create the "div" layer where you want the text to be changed. Don't forget to put some default text in there.

Then, anywhere before that layer, put in a set of radio buttons (change the name of that set from "nicheChooser" to something else). The value of each one should correspond to an element name in the data array. Finally add the onclick code to each radio button and have it call:

place('layer', this.value) But instead of 'layer' it should be the ID of the div box you want to fill in. Make sure to keep the quotes in there.

To make sure this works on non-IE browsers, I like to use my own function for editing div layers, which I call getObject() which I use instead of document.getElementById(). Paste this function in your JavaScript code:

```
function getObject(name) {
var ns4 = (document.layers) ? true : false;
var w3c = (document.getElementById) ? true : false;
var ie4 = (document.all) ? true : false;
if (ns4) return eval('document.' + name);
```

```
if (w3c) return document.getElementById(name);
if (ie4) return eval('document.all.' + name);
return false;
}
```

And then inside the place() function, replace document.getElementById with getObject. It's that simple.

Finally, for search engine optimization purposes, you might want to include this CSS and JavaScript in their own separate files. Search engines tend to rate text higher if it's positioned at a higher point on your page.

Copy your stylesheet code into a new text file (you don't have to include the <style> </style> tags).

```
h1 {
font-family: Tahoma;
color:#63ABE5;
font-size:32px;
line-height:32px;
}
h2 {
font-family: Tahoma;
color:#2EBF57;
font-size:24px;
line-height:24px;
}
body, td {
font-family:Verdana;
font-size:12px;
}
```

Save this in a file called "style.css." Then in your HTML document just put:

```
<link rel="stylesheet" href="style.css" type="text/css">
```

Then create a new blank text file, and copy the JavaScript stuff in it... just as before you don't need to copy the SCRIPT tags.

Copy this:

```
data = {
ebooks: "<h2 align=\"center\">Sell 217% More E-Books Next Year</h2>"
+"<p>Here is the part of my sales copy that gets really
specific towards a certain audience...</p>",
software: "<h2 align=\"center\">Killer Add-on To Any Software
Bundle!</h2>"
+"<p>Here I'm talking about how this product helps those
people selling software products.</p>"
+"<p>I'm talking a little more here...</p>",
copywriting: "<h2 align=\"center\">Never Run Out Of Copywriting
Clients...</h2>"
```

```
+"<p>Here's why:</p>"
+"<ul>"
+"<li>Reason number one.</li>"
+"<li>Reason number two.</li>"
+"<li>Reason number three.</li>"
+"<li><b>The final reason.</b></li>"
+"</ul>",
graphics: "<h2 align=\"center\">$1 Million Web Design Secret
Recipe</h2>"
+"<p>I can talk for as long as I want.</p>"
+"<p>I can keep talking...</p>"
+"<p><b>And talking...</b></p>"
+"<p>And talking and talking and talking.</p>",
advertising: "<h2 align=\"center\">In Search Of That Ultimate
Headline</h2>"
+"<p>More sales copy here...</p>"
};
function place(area, choice) {
var contents = data[choice];
getObject(area).innerHTML = contents;
}
function getObject(name) {
var ns4 = (document.layers) ? true : false;
var w3c = (document.getElementById) ? true : false;
HOW TO ADD INTERACTIVITY TO YOUR SALES LETTER
52 SECTION 10
SALES PAGE TACTICS ROBERT PLANK
var ie4 = (document.all) ? true : false;
if (ns4) return eval('document.' + name);
if (w3c) return document.getElementById(name);
if (ie4) return eval('document.all.' + name);
return false;
}
```

And save it as "functions.js." Now all you need is:

```
<script language="JavaScript" src="functions.js"></script>
```

And you're all set. I know that in my example, the changes were very noticeable on the screen, but those few times I've seen an interactive sales letter done, the choices were at least a page or two above the place where the actual changes took place. This way you couldn't really see the text change. You would make a choice and then scroll down and it would be different.

I imagine this is to keep the user from being disoriented or from feeling manipulated. Years and years ago when I was first learning about selling and marketing I remember reading an article that said something like... when you're trying to sell something to a person, take notice if that person lights up. If you're explaining something to them and they sit up and really pay attention, if you've got them hooked, keep on that subject even if it doesn't make sense to you at the moment WHY you should stay on that subject.

On the Internet, you aren't really speaking with the person one on one, but with something adjustable like this, you could do something similar. For one you have to emphasize in the sales

letter that the person follow along and actually choose an option along the way for their own benefit.

If you laid out a couple of choices along the way, you could do the same amount of selling as you'd do in a 10-20 page sales letter in 5 pages or so, since you are only sticking to the things the user wants to hear about. If you were really ambitious, you could even create choices within choices in your ISL... have a 5 or so page sales letter balloon out to 10 or more pages as the user makes his or her interests more and more specific to you.

Price Negotiator

Price testing is a tricky thing.

First we have the sales page where all the sales copy is shown and instead of a regular order button, the user is asked to provide a price they think is fair.

Save this code as index.php and work your sales page into it:

```
<div align="center">
<h1>Your Headline</h1>
<h2>Your Subheadline</h2>
<p align="left">Dear Marketer,</p>
<p align="left">The rest of your sales copy here...</p>
<?
if ($_COOKIE["price"]) {
// If a price has already been chosen, just show the PayPal button.
include("button.php");
}
else {
// A price hasn't been given yet, so ask.
include("ask.php");
}
?>
```

Next we need to put in the page where we ask the user for the right price. Again, not much to see here, just a form explaining you need their input for the best price and a little bit of JavaScript to make sure an honest price is given.

Save this code as ask.php:

```
<script language="JavaScript">
<!--
// Keep a user from giving a blank or $0.00 price in the price check form
function priceCheck(obj) {
var input = obj.value;
input = input.replace(/\$/, '');
var value = parseFloat(input);
if (isNaN(value) || value <= 0.00) {
alert("Please tell me what you think my product is worth!");
obj.value = "";
obj.focus();
return false;
}
return true;
}
// --></script>
<form action="order.php" method="POST" onsubmit="return
priceCheck(this.price);">
<div style="background-color:#EEEEEE; border:black solid 1px; padding:10px;">
<p>As part of a limited marketing test, I'm going to allow you the
opportunity to set your own price for this amazing software! This is simply
```

```
to determine the best price I should be charging and won't last long. Grab it
while you can at whatever price YOU decide!</p>
<h2 style="color:blue;">What's A Fair Price For This Product?</h2>
<p align="center">I would be willing to pay: $<input type="text" id="price"
name="price" size="7"></p>
<p align="center"><input type="submit" value="Order Now!"></p>
</div>
</form>
```

Once the user fills out this form they will be taken to the actual order page, where you do a little bit of talking... you chose this price but my price is this price. Here's how much we differed (if they overshot, they get a huge discount -- if they undershot... they didn't give you a fair price!). Save this as order.php:

```
<?php
// The price you are selling this product for
$targetPrice = "$7.50";
// Your e-mail address
$you = "you@example.com";
if (!$_COOKIE["price"]) {
$price = '$' . number_format($_POST["price"], 2);
// Read the e-mail template
$contents = file_get_contents("message.txt");
// Apply IP and price variables
$contents = str_replace('{price}', $price, $contents);
$contents = str_replace('{ip}', $_SERVER["REMOTE_ADDR"], $contents);
$ip = $_SERVER["REMOTE_ADDR"];
$subject = "New Estimate ($ip)";
// Send e-mail to the site owner
if ($you) {
$mail = new Mailer("Pricing Suggestion", $you);
$mail->send($you, $subject, $contents);
}
setcookie("price", $price, strtotime("+1 year"), "/");
}
else {
$price = $_COOKIE["price"];
}
// Figure out how many dollars and how much of a percentage is saved
$discount = parsePrice($price) - parsePrice($targetPrice);
if (parsePrice($price) > 0) {
$ratio = $discount / parsePrice($price);
}
else {
$ratio = 0;
}
$discount = '$' . number_format($discount, 2);
$ratio = number_format($ratio * 100, 0) . '%';
function parsePrice($input) {
$input = str_replace('$', '', $input);
return floatval($input);
}
/**
```

```
Library for sending e-mails in different formats.
*/
class Mailer {
var $myname, $mymail, $headers, $headerList, $type;
function Mailer($myname, $mymail) {
$this->headerList = array(
"text" => "Content-Type: text/plain; charset=us-ascii\nFrom:
{myname} <{mymail}>\nReply-To: <{mymail}>\nReturn-Path:
<{mymail}>\nX-Originating-IP: ".$_SERVER["REMOTE_ADDR"]."\nX-Mailer: PHP"
);
$this->type = "text";
$this->myname = $myname;
$this->mymail = $mymail;
$this->refresh();
}
function refresh() {
$newType = $this->headerList[$this->type];
$newType = str_replace('{mymail}', $this->mymail, $newType);
$newType = str_replace('{myname}', $this->myname, $newType);
$this->headers = $newType;
}
/**
Sends a text e-mail message containing the appropriate headers.
@param myname the sender name
@param mymail the sender e-mail address
@param email the recipient e-mail address
@param subject the subject line
@param body the actual message
@return true if the e-mail was accepted for delivery
false if something went wrong
*/
function send($email, $subject, $body) {
@ini_set("sendmail_from", $this->mymail);
$this->refresh();
if ($email != "") { $result = mail($email,stripslashes($subject),
stripslashes($body),$this->headers); }
else $result = false;
@ini_restore("sendmail_from");
return $result;
}
}
?>
<h1>You're Willing to Pay <?=$price?>?</h1>
<? if (parsePrice($price) < parsePrice($targetPrice)) { ?>
<p>Let's be fair. Didn't I explain all the time-saving features of this
script, isn't the time saved from ... worth more than <?=$price?>?</p>
<p>Does <?=$targetPrice?> sound like a more fair price to you?</p>
<? }
elseif (parsePrice($price) > parsePrice($targetPrice)) { ?>
<p>So you're telling me I should sell this software for <?=$price?>...</p>
<p><b>Wow... that is a generous offer.</b></p>
<p>What about this:</p>
<p align="center">I'll give you a discount of <?=$discount?>, making the
```

```
price just <span style="color:blue;"><b><?=$targetPrice?></b></span>.<br>
(That's a <?=$ratio?> savings!!)</p>
<? }
else { ?>
<p align="center">You and I think alike. <?=$targetPrice?> was the EXACT
price I had in mind!</p>
<? } ?>
<p align="center">You can order below.</p>
<?include("button.php")?>
```

When the user gives you their price on ask.php, their guess will be e-mailed to you. I know, you COULD have it so it saves all the results, and averages them out, but you will have a lot of jackasses pricing it at $0.01 and $1.00 in the hopes of getting the product for that price.

What I like to do whenever an e-mail is sent out is have a plain-text template so the script will just fill in the details.

This is message.txt:

```
You have a new pricing suggestion for Your Product.
The recommended price they gave was: {price}
IP Address: {ip}
```

Oh yeah, and finally, I made the order button on it's own separate include page, that way the button could be included in the sales page (if the user has revisited without giving you a price) or on the order page after submitting a price.

It's done this way so if you need to change the order button it only needs to be done in one place.

This is button.php:

```
<div align="center">
(Place your payment button here)
</div>
```

Item-Based Sale Counter

An alternative to running a time limited special offer would be a quantity-based one. This is the sort of thing where you offer a script up for sale and offer, say, only 25 copies. Once one copy is sold that number changes to 24, then 23, and so on until they're all gone.

This is a little better than the timed countdown since:

1. Your offer is never going to end prematurely, it will run until your quota runs out.

2. You know how much you're going to make on this special (eventually).

3. Theoretically anyway, there should be more orders as that number drops.

4. People are assured there are going to be a limited number of these out in the wild (great for resale rights).

5. If someone misses out on this special offer they should be more likely to pounce on the next once if it's something they're interested in.

Run your special offer in PayPal because:

- It's easier to do this weird shit with it (IPN, automate refunds, and MassPay craziness on multiple tiers)

- no affiliates (that's a good thing this time mainly because affiliates won't get mad when you change the price)

- People know it's an exclusive special offer as opposed to Clickbank where any affiliate can sign up

I hate Clickbank because of all the refunds. Basically the only things that keep me with Clickbank are the affiliates and the constant promotion I get from being listed in the Clickbank Marketplace.

TECHNICALLY it would be possible to have a limited time sale offer through Clickbank, but their order links are pretty easy to guess. I just don't like it.

Anyway, this sale counter will be really simple. We will take a look at the PayPal receipt and write it to a text file. (We will make sure no receipt is written to the text file twice). Each receipt will be written one per line.

Then, all we have to do to figure out the number of sales is count the number of lines (not "lies", silly) in that text file.

From there, the number of products remaining is just the total you offered minus the amount already sold. Let's say we were capping the sales at 25 and there were 10 receipts in our file. This would mean 10 sales were made, and 25 minus 10 means 15 slots are left. This is third-grader stuff.

Just to keep things exciting let's go backwards on this one. First worry about reading the license file. We will work on the code to write to it later.

Start off by creating a new text file, let's call it "receipt.txt" and place a couple of sample lines in it:

license1
license2
license3

Save and upload to your web server and all that. Now create a new text file, let's name this one "read.php" and don't forget to save and upload to your server as we go along. First, we want to read the whole file into an array.

```
$list = file("receipt.txt");
This places each line in its own element of the array. If we count the number of elements in
this array it will give us the number of lines in the file.
$sales = count($list);
Let's output that $sales variable to see what we've got:
echo $sales;
So this is all of our PHP code in that new file now:
<?php
$list = file("receipt.txt");
$sales = count($list);
echo $sales;
?>
```

Save this as "read.php" and upload, then load it in your browser. You should see the number "3" since there are 3 entries in the receipt.txt file. We said the number of units remaining was the total number we were going to sell minus the number already sold.

Let's set a variable $total to 25 at the beginning, then at the set $left to $total - $sales.

```
<?php
$total = 25;
$list = file("receipt.txt");
$sales = count($list);
$left = $total - $sales;
echo $left;
?>
```

Refresh. You should see 22 and that's just what we want. If that file filled up to 25 lines the $left variable would go down to 0 and on into negative numbers. So, if we want to use the number of units remaining in the sales copy, just output the variable $left like this:

```
<?=$left?>
```

And on deciding whether or not to show the live sales page or tell them, "You missed out," we only need to check to make sure that $left is greater than 0. That means there are MORE than zero units still available.

Your read.php file should look like this now:

```
<?php
$total = 25;
$list = file("receipt.txt");
$sales = count($list);
$left = $total - $sales;
?>
<? if ($left > 0) { ?>
<h1>Hurry Up - Only <?=$left?> Units Left</h1>
(the rest of the sales page here)
<? } else { ?>
<h1>Sorry, All Out!</h1>
(but check out such and such other special offer)
<? } ?>
```

Try setting $total to 4 and then 3 to see what happens. We have 3 units in the receipt file, and 4 units are being offered, so there's 1 left. If our limit is 3 units for sale and 3 units have been purchased... the offer is all over.

*** STOP ***

Before you continue, make SURE receipts.txt is chmoded to 777 in your FTP program so that the script can write to the file.

Payment Data Transfer function:

```
function pdt($token) {
$input = $_POST;
// read the post from PayPal system and add 'cmd'
$req = 'cmd=_notify-synch';
$tx_token = $_GET['tx'];
$auth_token = $token;
$req .= "&tx=$tx_token&at=$auth_token";
// post back to PayPal system to validate
$header .= "POST /cgi-bin/webscr HTTP/1.0\r\n";
$header .= "Content-Type: application/x-www-form-urlencoded\r\n";
$header .= "Content-Length: " . strlen($req) . "\r\n\r\n";
// Try SSL first
$fp = @fsockopen ('ssl://www.paypal.com', 443, $errno, $errstr, 30);
if (!$fp) {
// SSL doesn't work, just use HTTP
$fp = fsockopen ('www.paypal.com', 80, $errno, $errstr, 30);
}
if (!$fp) {
// HTTP ERROR
return false;
}
fputs ($fp, $header . $req);
// read the body data
$res = '';
$headerdone = false;
while (!feof($fp)) {
```

```
$line = fgets ($fp, 1024);
if (strcmp($line, "\r\n") == 0) {
// read the header
$headerdone = true;
}
else if ($headerdone) {
// header has been read. now read the contents
$res .= $line;
}
}
// parse the data
$lines = explode("\n", $res);
$lines = array_map("trim", $lines);
$keyarray = array();
if ($lines[0] != "SUCCESS") {
return false;
}
for ($i=1; $i<count($lines);$i++){
list($key,$val) = explode("=", $lines[$i]);
$keyarray[urldecode($key)] = urldecode($val);
}
return $keyarray;
}
```

And here's the code to check the rest of the PayPal stuff:

```
// Product name (it must match part of the item name on the PayPal button)
$productName = "Test";
// Your PayPal PDT identity token here
$token = "identity-token";
// Your PayPal email address
$yourmail = "you@example.com";
// The price of this product
$price = 0.01;
// Receipt filename, do not change
$filename = "receipt.txt";
// Make sure the transaction is valid
if ($info = pdt($token)) {
if (
$info["payment_gross"] == $price and
$info["business"] == $yourmail and
$info["payment_status"] == "Completed" and
eregi($productName, $info["item_name"])
) {
// Everything went through, so open the read the list of receipts
$list = file($filename);
$list = array_map("trim", $list);
// Add this transaction ID to the end
$list[] = $info["txn_id"];
// Make sure there are no duplicates
$list = array_values(array_unique($list));
// Change from an array into a string
$list = implode("\n", $list);
```

```
// Write back to the file
$fp = fopen($filename, "w");
fwrite($fp, $list);
fclose($fp);
?>
(your thank you page here)
<?
}
}
```

So here's all of write.php:

```
<?php
// Product name (it must match part of the item name on the PayPal button)
$productName = "Test";
// Your PayPal PDT identity token here
$token = "identity-token";
// Your PayPal email address
$yourmail = "you@example.com";
// The price of this product
$price = 0.01;
// Receipt filename, do not change
$filename = "receipt.txt";
// Make sure the transaction is valid
if ($info = pdt($token)) {
if (
$info["payment_gross"] == $price and
$info["business"] == $yourmail and
$info["payment_status"] == "Completed" and
eregi($productName, $info["item_name"])
) {
// Everything went through, so open the read the list of receipts
$list = file($filename);
$list = array_map("trim", $list);
// Add this transaction ID to the end
$list[] = $info["txn_id"];
// Make sure there are no duplicates
$list = array_values(array_unique($list));
// Change from an array into a string
$list = implode("\n", $list);
// Write back to the file
$fp = fopen($filename, "w");
fwrite($fp, $list);
fclose($fp);
?>
(your thank you page here)
<?
}
}
function pdt($token) {
$input = $_POST;
// read the post from PayPal system and add 'cmd'
$req = 'cmd=_notify-synch';
$tx_token = $_GET['tx'];
$auth_token = $token;
```

```
$req .= "&tx=$tx_token&at=$auth_token";
// post back to PayPal system to validate
$header .= "POST /cgi-bin/webscr HTTP/1.0\r\n";
$header .= "Content-Type: application/x-www-form-urlencoded\r\n";
$header .= "Content-Length: " . strlen($req) . "\r\n\r\n";
// Try SSL first
$fp = @fsockopen ('ssl://www.paypal.com', 443, $errno, $errstr, 30);
if (!$fp) {
// SSL doesn't work, just use HTTP
$fp = fsockopen ('www.paypal.com', 80, $errno, $errstr, 30);
}
if (!$fp) {
// HTTP ERROR
return false;
}
fputs ($fp, $header . $req);
// read the body data
$res = '';
$headerdone = false;
while (!feof($fp)) {
$line = fgets ($fp, 1024);
if (strcmp($line, "\r\n") == 0) {
// read the header
$headerdone = true;
}
else if ($headerdone) {
// header has been read. now read the contents
$res .= $line;
}
}
// parse the data
$lines = explode("\n", $res);
$lines = array_map("trim", $lines);
$keyarray = array();
if ($lines[0] != "SUCCESS") {
return false;
}
for ($i=1; $i<count($lines);$i++){
list($key,$val) = explode("=", $lines[$i]);
$keyarray[urldecode($key)] = urldecode($val);
}
return $keyarray;
}
?>
```

Split Tester

Split testing is a way to fine tune your sales copy in the hope of improving your conversion rate. It's a very scientific process -- you make only small methodical tweaks, just a few things at a time so you only focus on a few variables.

For example, you might try one font against another on a site. 50% of the visitors will see your site in a particular font, the other 50% in some alternate font. Or you might want to rotate different headlines, long copy vs. short copy, the background color... all sorts of stuff. Over time with a lot of patient work you can increase the chance that a visitor to your site will buy from you.

Right now, you and I could make a really simple split tester that would be so easy to use... the script could read off a very simple template (we wouldn't even need to bring FastTemplate into it this time) where we defined custom tags representing different splits.

Let's make our template the way we want it. Call this file template.html:

This is {an amazing:a great:a super:a powerful} test site...
{Try me:Do it now:Go here}
{single test}
bye.

So here we might split different adjectives in our headline, such as "an amazing", "a great" site and so on. (Each split is separated by a colon ":" in this case.) Every time our script reads off this template it will choose a random cycle and display the words accordingly.

So if it chose the 1st cycle, this would be an "amazing" site with the words "Try me" below. If it was the third, this would be a "super" site with the words "Go here" below. Get it? The words "single test" on the bottom should remain "single test" throughout because there's only 1 version of it... we don't have a 2nd or 3rd or 4th.

Read template.html and output its contents:

```
$contents = file_get_contents("template.html");
Now grab everything inside the brackets:
preg_match_all("/{(.*?)}/si", $contents, $matches);
print_r($matches);
"s" means treat everything as if it's on a single line, so if there are line breaks in there it will still catch them. "i" means case-insensitive.
Result:
Array
(
[0] => Array
(
[0] => {an amazing:a great:a super:a powerful}
[1] => {Try me:Do it now:Go here}
[2] => {single test}
)
[1] => Array
(
[0] => an amazing:a great:a super:a powerful
[1] => Try me:Do it now:Go here
[2] => single test
)
)
```

```
We only want to save the 1st set of matches (not the 0th which contains the entire string)
$matches = $matches[1];
Now make a function that will split up the string by ":", using explode.
function splitOptions($input) {
return explode(":", $input);
}
Run an array_map to apply this function to every element in the array.
$matches = array_map("splitOptions", $matches);
Result:
Array
(
[0] => Array
(
[0] => an amazing
[1] => a great
[2] => a super
[3] => a powerful
)
[1] => Array
(
[0] => Try me
[1] => Do it now
[2] => Go here
)
[2] => Array
(
[0] => single test
)
)
```

But, if you think about it, we actually will need to save the 0th element of that $matches array so we can easily get back to the original string to replace.

Take out this line:

```
$matches = $matches[1];
All of our segmented matches will be in the 1st element of $matches, so that array_map has to take place with just that element, not the whole array:
$matches[1] = array_map("splitOptions", $matches[1]);
```

Here's a tip. If you're as sick of constantly viewing the source code like I am, you can put <xmp> and </xmp> around your print_r() statement. The <xmp> tag in HTML says to treat whatever is inside that tag as text.

```
echo "<xmp>";
print_r($matches);
echo "</xmp>";
```

So now our array contains both the original string, and split pieces of each one.

```
Array
(
[0] => Array
(
[0] => {an amazing:a great:a super:a powerful}
[1] => {Try me:Do it now:Go here}
```

```
[2] => {single test}
)
[1] => Array
(
[0] => Array
(
[0] => an amazing
[1] => a great
[2] => a super
[3] => a powerful
)
[1] => Array
(
[0] => Try me
[1] => Do it now
[2] => Go here
)
[2] => Array
(
[0] => single test
)
)
)
```

The next steps are as follows: first, loop through $matches[1]. Every element in $matches[1] is an array, right? Let's take the 0th element within $matches[1] as an example. Array

```
(
[0] => an amazing
[1] => a great
[2] => a super
[3] => a powerful
)
```

Now we need to figure out how many splits are in this array. Every element in $matches[1] is an array, right? Within $matches[1], array #0 has 4 elements, array #1 has 3 elements, and array #2 has only 1 element.

This means there are 4 splits, as follows:

Split #1: 0th value in array 0, 0th value in array 1, 0th value in array 2

Split #2: 1st value in array 0, 1st value in array 1, 0th value in array 2 (in array 2, we're going back to the 0th value because there is no 1st or 2nd or 3rd value, this is being treated as the default)

Split #3: 2nd value in array 0, 2nd value in array 1, 0th value in array 2 (still at 0 for the default)

Split #4: 3rd value in array 0, 0th value in array 2 (because array 2 has no 3rd value), 0th value in array 2

Get it? We find the array with the largest number of elements and that equals the number of splits. Since this is just a simple script (meaning no storage of data) we'll just pick a random split, figure out which values to use, replace those brackets and then output it all. To figure out the size of the largest array, we start off with a variable (let's call it $max) and set it to 0. We loop through

the array and keep comparing $max to the size of the element we're looking at. If the element is larger than $max, update $max to that new value. In the end we'll be left with the value of the largest element.

```
$max = 0;
foreach ($matches[1] as $value) {
if (count($value) > $max) $max = count($value);
}
```

Output $max and halt the script so we know we got the value we wanted (4).

```
echo $max;
die();
```

Instead of having to do...

```
if (count($value) > $max) $max = count($value);
```

... Within the foreach loop, we can use the max() function, which will compare two values and return the largest.

```
$max = max($max, count($value));
```

Choose our random number between 1 and $max. (Call this value $split).

```
$split = mt_rand(1, $max);
```

This will be a number between 1 and $max, but remember, arrays start counting at zero, so really, $split should be:

```
$split = mt_rand(1, $max) - 1;
```

CHAPTER 13: SPLIT TESTER
BANANA SPLIT 133
SIMPLE PHP WITH ROBERT PLANK VOLUME THREE

Now that we've chosen a split, we'll loop back through $matches[0] (the array containing those original strings). Then, we choose the matching value in the second array.

```
for ($i=0;$i<count($matches[0];$i++) {
$originalValue = $matches[0][$i];
$newArray = $matches[1][$i];
}
```

Each iteration of the loop will replace a different {tag}.
Keep adding the following inside that for-loop: If that split (i.e. split #3) exists, choose the appropriate element.

```
// Choose the split in the replacement array
if (array_key_exists($split, $newArray)) {
$newValue = $newArray[$split];
}
```

If the element doesn't exist, like in that example where there were 4 splits but a certain variable had less than 4 options, default to 0.

```
// Or the 0th element if the chosen split does not exist
else {
$newValue = $newArray[0];
}
```

Now $originalValue is the string we want to replace (like "{an amazing:a great:a super:a powerful}") and $newValue is the single string we want to put in that place (like "a great"). Whenever you want to do replacements in a string when you don't need any sort of logic -- just replace this text with this text -- use str_replace().

```
$contents = str_replace($originalValue, $newValue, $contents);
```

Now you can stop working inside that for loop, and finally:

```
echo $contents;
```

Take out that die statement and the other echo statement while you're at it, too. Here's the index.php script inside our "split" folder:

```
<?php
$contents = file_get_contents("template.html");
preg_match_all("/\{(.*?)\}/si", $contents, $matches);
$matches[1] = array_map("splitOptions", $matches[1]);
// Find the split count
$max = 0;
foreach ($matches[1] as $value) {
$max = max($max, count($value));
}
// Choose a random split
$split = mt_rand(1, $max) - 1;
for ($i=0;$i<count($matches[0]);$i++) {
$originalValue = $matches[0][$i];
$newArray = $matches[1][$i];
// Choose the split in the replacement array
if (array_key_exists($split, $newArray)) {
$newValue = $newArray[$split];
}
// Or the 0th element if the chosen split does not exist
else {
$newValue = $newArray[0];
}
$contents = str_replace($originalValue, $newValue, $contents);
}
echo $contents;
function splitOptions($input) {
return explode(":", $input);
}
?>
```

This whole script is made so you can change the format easily. For example, let's say that within a variable bracket, I wanted to separate the split values by a pipe "|" instead of a colon, since we'll be using this for HTML and CSS uses colons. Now that I think about it, CSS uses curly braces as well, so let's change those into something we would otherwise never accidentally do in HTML. Like this: <%tag%>

Make the change from a colon to a pipe in the splitOptions() function...

```
function splitOptions($input) {
return explode("|", $input);
}
```

Then change the search string in the preg_match_all statement to <% and %>...
preg_match_all("/<%(.*?)%>/si", $contents, $matches);

And change it in the template.html file too (a simple search and replace will do). This is <%an amazing|a great|a super|a powerful%> test site...

```
<%Try me|Do it now|Go here%>
<%single test%>
bye.
```

Now that we have the splits working the way we want it, let's actually put some tags in the HTML so they're readable.

```
<html>
<head>
<style type="text/css">
body { font-family: Tahoma; font-size:12px; }
h1 { font-size:19px; }
</style>
</head>
<title>Split Test Page</title>
<body>
<h1 align="center">This is <font color="#0000FF"><%an amazing|a great|a super
|a powerful%> test site...</font></h1>
<div align="center"><%Try me|Do it now|Go here%></div>
<p><%single test%></p>
bye.
</body>
</html>
```

Of course, if you're doing any sort of split testing, you need to know how each cycle performed. For that, you'll need to bring in some sort of link tracker. Instead of linking directly to your order link, you could setup your link tracking script for the first split, the second split, and so on.

If the tracker redirect was something like go.php/split1 you could put in your template:

```
<a href="go.php/<%split1:split2:split3:split4%>">Order now</a>
```

This simple script won't measure a conversion rate, but you could compare the click count on each split and see which has the highest.

To really measure the clickthru rate, you'd have to use an image tracker (re-read chapter 3 of Simple PHP Volume 2 to remember how to do this) to count the number of impressions. You'd have to pass the split number to the image counter also, to figure out how many impressions that given split had, then divide the number of clicks by the impressions.

About the Author

I'm Robert Plank, coder of all sorts of PHP scripts... Redirect Pro, Lightning Track, HyperSplitter, ClickSensor, RotatorBlaze, Ezine Rocket, Ultimate Website Personalizer, Keyword Mixer, Codewarden, and Turbo Thanks.

Plus a ton of custom private-label scripts involving database repair, content seeding and management, wikis, PDF generation, pop-up creation and who knows what else.

Robert Plank is the author of:

Simple PHP Volumes 1 through 3: http://www.simplephp.com

As well as the manual **Sales Page Tactics**: http://www.salespagetactics.com

And **Affiliate Battle Plan**: http://www.affiliatebattleplan.com/

Don't Forget To Visit http://www.JumpX.com and receive your FREE subscription to the JumpX E-Magazine