WEB DEVELOPMENT WITH PHP

FASTTEMPLATE

By icarus

Table of Contents

Symbiotic Relationships

If you've worked with PHP, JSP, ASP or any other tag−based scripting language, you already know that one of the most powerful features of these languages is the ability to combine regular HTML code with programming constructs like variables and function calls. By substituting variable "placeholders" for actual content in an HTML page, these languages make it easy to construct dynamic Web pages; simply alter the values of the variables embedded within the HTML code, and the content displayed on the page changes appropriately.

As any software designer will tell you, however, this convenience comes at a price − most PHP scripts are so closely interwoven with HTML code that maintaining them is a nightmare. Since both the HTML user interface elements and the program logic are in the same physical file, it becomes difficult for users with no programming experience to modify one without affecting the other. The most common example of this is when interface designers need to alter the look and feel of a particular Web application; typically, the changes they make to the HTML code must be monitored by a developer to ensure the integrity of the remainder of the script.

As you might imagine, having a developer hand−hold an interface designer through an interface re−design process is annoying to both parties concerned, not to mention expensive to the organization as a whole. A simpler alternative would be to somehow separate the interface (HTML code) from the programming routines (PHP, Perl et al), such that each could be modified independent of the other; this would allow designers to do what they do best, and developers to sleep nights without worrying about the likely repercussions on their beautifully−handcrafted software routines.

Which brings us to FastTemplate...

When Time Is Money

FastTemplate − the PHP version − is actually a port of a well−known Perl module. It is designed, in the author's words, to "manage templates and perform variable interpolation", and is packaged as a single PHP class which can be easily included in your application.

FastTemplate uses "templates" to simplify maintenance of PHP code, and to separate data from page elements. It assumes that a single Web application is made up of many smaller pieces − it calls these parts "templates" − and provides an API to link templates together, and to fill them with data.

In FastTemplate lingo, a "template" is simply a text file, typically containing both static elements (HTML code, ASCII text) and FastTemplate variables. When FastTemplate reads a template file, it automatically replaces the variables within it with their values. These values may be defined by the developer at run−time, may be read from another file, or may be extracted from a database.

As you will see, FastTemplate also makes it possible to "nest" one template within another, adding a whole new level of flexibility to this template−based method of doing things. By allowing you to split up a user interface into multiple smaller parts, FastTemplate adds reusability to your Web application (a template can be used again and again, even across different projects) and makes it easier to localize the impact of a change.

Before proceeding further, you should visit the FastTemplate home page at http://www.thewebmasters.net/ and download a copy of the latest version (1.1.0 at the time of writing). The package contains the main class file, a document outlining the exposed methods and variables, and some example scripts.

FastTemplate was originally written for PHP3; if you're using PHP4, you'll need to patch it by making a couple of modifications to the class file. A list of the changes is available on the FastTemplate Web site.

Who Am I?

Let's begin with a simple example of how FastTemplate works. Consider the following HTML page − I've called it "mypage.tpl" − which contains personal information about a specific user.

```
<!-- begin: mypage.tpl -->
<html>
<head>
</head>

<body>

<b>Name</b>: {FNAME} {LNAME}
<p>
<b>Age</b>: {AGE}
<p>
<b>Email address</b>: {EMAIL_ADDRESS}
<p>
<b>Tel</b>: {TEL}


</body>
</html>
<!-- end: mypage.tpl -->
```

As you can see, this page does not contain any data per se; instead, FastTemplate variables (enclosed in curly braces) are used as placeholders in the appropriate locations. Once the FastTemplate engine parses the document, these variables will be replaced with their actual values.

Proofing The Pudding

Next, we need to tell FastTemplate about the template, assign values to the variables, and put the two together. Here's the script:

```
<?
// mypage.php - generate output using FastTemplate

// include class file
include("class.FastTemplate.php3");

// instantiate new object
$obj = new FastTemplate(".");

// assign names for template files
// "index" now references the template "./mypage.tpl"
$obj->define(array("index" => "mypage.tpl"));

// assign values to FT variables within the template
// this may also be set up as an associative array of
key-value pairs
$obj->assign("FNAME", "John");
$obj->assign("LNAME", "Doe");
$obj->assign("AGE", "36");
$obj->assign("TEL", "(12)-34-567 8912");
$obj->assign("EMAIL_ADDRESS", "jdoe@anonymous.com");

// parse template "index" and store in handler "result"
$obj->parse(result, "index");

// print contents of handler "result"
$obj->FastPrint(result);
?>
```

A quick explanation is in order here.

1. The first step when using the FastTemplate class is to create a new FastTemplate object, and tell it the location of the template files.

```
<?
include("class.FastTemplate.php3");

$obj = new FastTemplate(".");
?>
```

In this case, there's only one template and it's in the current directory.

2. Next, the define() method is used to assign names to the templates you plan to use through the script − these names are defined in an associative array, and will be used throughout the remainder of the script to refer to the respective template files. In this case, I've used the name "index" to refer to the template "mypage.tpl".

```
<?
$obj->define(array("index" => "mypage.tpl"));
?>
```

3. Once FastTemplate knows which templates to use, and has names by which it can refer to them, the next step is to assign values to the variables in the templates. As you can see from "mypage.tpl" above, it contains five variables − the assign() method is used to assign values to these variables

```
<?
$obj->assign("FNAME", "John");
$obj->assign("LNAME", "Doe");
$obj->assign("AGE", "36");
$obj->assign("TEL", "(12)-34-567 8912");
$obj->assign("EMAIL_ADDRESS", "jdoe@anonymous.com");
?>
```

Variables may also be assign()ed values via an associative array containing key-value pairs − you'll see that technique in the next example.

4. With the variables assigned values, it's time to put the two together. This is accomplished via FastTemplate's parse() method, which is easily the most important method in the object collection.

The parse() method accepts two arguments − a variable name, and a template name. In this example, the variable is called RESULT, and the template is named "index" (which references the template "mypage.tpl").

```
<?
$obj->parse(result, "index");
?>
```

In English, the line of code above means "read the template referenced by the name "index", assign values to the variables found within it, and then assign the result, complete with substituted values, to the variable "result"." Whew!

5. At this point, the variable "result" contains the final page output − all that remains is to print it to the browser.

```
<?
$obj->FastPrint(result);
?>
```

The FastPrint() function prints the result of the last parse() function call − or you can specify the name of the variable to be printed, as above.

Here's what it looks like:

Name: John Doe

Age: 36

Email address: jdoe@anonymous.com

Tel: (12)-34-567 8912

Since the HTML interface is in a separate template file, it's easy for designers with no programming experience to radically alter the interface without affecting the program code...so long as they remember to replace the placeholders when they're done. As an illustration, here's a completely reworked version of "mypage.tpl"

```
<!-- begin: mypage.tpl -->
<html>
<head>
</head>

<body>

<table border="1" cellpadding="5">
<tr>
<td bgcolor=black><i><font color=white>First
name</font></i></td>
<td bgcolor=black><i><font color=white>Last
name</font></i></td>
<td bgcolor=black><i><font color=white>Tel</font></i></td>
<td bgcolor=black><i><font color=white>Email
address</font></i></td>
<td bgcolor=black><i><font color=white>Age</font></i></td>
</tr>
<tr>
<td>{FNAME}</td>
<td>{LNAME}</td>
<td>{TEL}</td>
<td><a href="mailto:{EMAIL_ADDRESS}">{EMAIL_ADDRESS}</a></td>
<td>{AGE}</td>
```

```
</tr>
</table>

</body>
</html>
<!-- end: mypage.tpl -->
```

which looks like this:

First name	*Last name*	*Tel*	*Email address*	*Age*
John	Doe	(12)-34-567 8912	jdoe@anonymous.com	36

By using FastTemplate, this interface modification was accomplished solely by modifying the template file, leaving the scripting routines untouched.

You've Got Mail

A great amount of FastTemplate's power, however, lies in its ability to manage more than one template simultaneously. Consider the following HTML page:

Now, suppose I were to split this up into the following sections,

and assign each section to a template. This would mean that the header, the main form, and the footer at the bottom could be modified independently of each other − a useful capability to have.

Here are my templates:

```
<!-- begin: header.tpl -->
<html>
<head>
<basefont face=Verdana>
</head>
<body>
<table width=100% cellpadding=10 bgcolor="Black">
<tr><td height=30><b><font
color=white>{TITLE}</font></b></td></tr>
```

```
</table>
<!-- end: header.tpl -->

<!-- begin: form.tpl -->
<p>
<i>{INSTRUCTIONS}</i>
<p>
<div align=center>
<table border="0" cellspacing="5" cellpadding="5">
<form action="mailer.php" method="post">
<tr>
<td>Name</td>
<td><input type="Text" name="name" size="15"></td>
</tr>
<tr>
<td>Email address</td>
<td><input type="Text" name="email" size="25"></td>
</tr>
<tr>
<td>Subject</td>
<td><input type="Text" name="subj" size="25"></td>
</tr>
<tr>
<td>Comments</td>
<td><textarea name="comments" cols="35"
rows="8"></textarea></td>
</tr>
<tr>
<td colspan=2 align=center><input type="Submit" value="Send
Feedback"></td>
</tr>
</form>
</table>
</div>
<!-- end: form.tpl -->


<!-- begin: footer.tpl -->
<div align=center><font size=-2>
Everything here is © <a
href="http://www.melonfire.com/">Melonfire</a>, 2001. All
rights
reserved.<br>
Read our <a href="tac.html">legal notices</a>, and our <a
href="privacy.html">privacy policy</a>
</font></div>
<br>
<table width=100% align=center cellpadding=0 bgcolor="Black">
<tr><td> </td></tr>
```

```
</table>
</body>
</html>
<!-- end: footer.tpl -->
```

And here's the script which puts them together with FastTemplate.

```
<?
// feedback.php - generate a feedback form using multiple
templates

// include class file
include("class.FastTemplate.php3");

// instantiate new object
$obj = new FastTemplate("./tmpl/");

// assign names for template files
$obj->define(array(
"header" => "header.tpl",
"form" => "form.tpl",
"footer" => "footer.tpl"
));

// assign values to FT variables within the template
// as an associative array of key-value pairs
$obj->assign(array(
"TITLE" => "Feedback Form",
"INSTRUCTIONS" => "Please use the following form to send us
your feedback
on this Web site"
));

// parse template "feedback" and store in handler "result"
$obj->parse(ft_header, "header");
$obj->parse(ft_form, "form");
$obj->parse(ft_footer, "footer");

// print contents of handler "result"
$obj->FastPrint(ft_header);
$obj->FastPrint(ft_form);
$obj->FastPrint(ft_footer);
?>
```

Note that, in this case, FastTemplate is parsing and printing more than one template to create a composite HTML document. Each template may be modified independently of the others, making it easier to alter just

the top bar or the form fields, for example.

Repeat Customers

Another nifty little feature you'll find in FastTemplate is the ability to "nest" one template within another − in the following example, a welcome message template is nested within the main index page template.

```
<!-- begin: message.tpl -->
<div align=center style="font-family: Verdana; font-size:
10pt">
{MESSAGE}
</div>
<p>
<!-- end: message.tpl -->

<!-- begin: welcome.tpl -->
<html>
<head>
</head>
<body>
{CONTENT}
<hr>
</body>
</html>
<!-- end: welcome.tpl -->
```

Here's the script which puts them together:

```
<?
// index.php - welcome page

// include class file
include("class.FastTemplate.php3");

// instantiate new object
$obj = new FastTemplate("./tmpl/");

// assign names for template files
$obj->define(array(
"welcome" => "welcome.tpl",
"message" => "message.tpl"
));

// normally, this variable might be set from a cookie
// uncomment this to see how the message changes
// $repeat_visitor = 1;
```

```
// assign values to FT variable within the template
if ($repeat_visitor == 1)
{
$obj->assign("MESSAGE", "Welcome back! We've updated our
catalog since
your last visit - click here to see the new arrivals.");
}
else
{
$obj->assign("MESSAGE", "You're visiting our site for the very
first time,
so you might like to take our New User Tour.");
}

// parse templates
// in this case, "message" is parsed first
// the resulting output is assigned to the FT variable CONTENT
// the next template "welcome" is parsed
$obj->parse(CONTENT, array("message", "welcome"));

// and print
$obj->FastPrint(CONTENT);
?>
```

When the parse() method is assigned a series of templates to parse via an array, FastTemplate will proceed through the array in a sequential manner, assigning the result of each successive parse() operation to the variable specified.

In this case, FastTemplate will first parse the template "message", assign a value to the "MESSAGE" variable, and then assign the result to the variable "CONTENT". At this stage, the variable "CONTENT" contains:

```
<!-- begin: message.tpl -->
<div align=center style="font-family: Verdana; font-size:
10pt">
You're visiting our site for the very first time, so you might
like to take
our New User Tour.
</div>
<p>
<!-- end: message.tpl -->
```

Next, it will proceed to parse "welcome", assign the value of the newly-created variable "CONTENT" to the template, and again store the result in "CONTENT". At this stage, the variable "CONTENT" contains:

```
<!-- begin: welcome.tpl -->
<html>
<head>
</head>
<body>
<!-- begin: message.tpl -->
<div align=center style="font-family: Verdana; font-size:
10pt">
You're visiting our site for the very first time, so you might
like to take
our New User Tour.
</div>
<p>
<!-- end: message.tpl -->
<hr>
</body>
</html>
<!-- end: welcome.tpl -->
```

This is what finally gets printed to the browser via FastPrint().

Note also that you can assign values to FastTemplate variables on the basis of conditional tests; in the example above, the message changes depending on whether or not the user is a repeat visitor.

Flavour Of The Month

Another useful FastTemplate feature involves creating a sequence of items by appending to a single variable; this comes in particularly handy when creating HTML constructs like lists and table rows. Consider the following templates:

```
<!-- begin: list.tpl -->
<html>
<head>
</head>
<body>
<ul>
{LISTCONTENT}
</ul>
</body>
</html>
<!-- end: list.tpl -->

<!-- begin: list_item.tpl -->
<li>{ITEM}
<!-- end: list_item.tpl -->
```

In this case, I plan to build a list (from a database) by repeatedly calling the "list_item.tpl" template; this list will then be assigned to a FastTemplate variable named "LISTCONTENT", and substituted in the main page, "list.tpl". Here's the script:

```
<?
// list.php - item list

// include class file
include("class.FastTemplate.php3");

// instantiate new object
$obj = new FastTemplate("./tmpl/");

// assign names for template files
$obj->define(array(
"list" => "list.tpl",
"list_item" => "list_item.tpl"
));

// get item list
/*
// open connection to database
$connection = mysql_connect($hostname, $user, $pass) or die
```

```
("Unable to
connect!");

// query
$query = "SELECT item from itemtable";
$result = mysql_db_query($database, $query, $connection);

// build $items array
$items = array();
while(list($item) = mysql_fetch_row($result))
{
$items[$count] = $item;
$count++;
}
*/
// assume it looks like this..
$items = array("vanilla", "pineapple", "strawberry",
"chocolate chip",
"peach", "banana", "grape");

// build LISTCONTENT variable by concatenation of multiple
instances of
"list_item" template
for ($x=0; $x<sizeof($items); $x++)
{
$obj->assign("ITEM", $items[$x]);

// append the result of parsing the template to LISTCONTENT
$obj->parse(LISTCONTENT, ".list_item");
}

// parse templates
$obj->parse(RESULT, "list");

// and print
$obj->FastPrint(RESULT);
?>
```

Each time the loop iterates through the $items array, a new value is assigned to the "ITEM" variable. FastTemplate then parses the "list_item" template and replaces the placeholder with its actual value. The "." operator is used to append the result of each iteration to the "LISTCONTENT" variable.

This technique comes in very handy when building repetitive sequences − all you need to do is build a template containing one item of the sequence, and let FastTemplate generate as many copies of it as you need.

- vanilla
- pineapple
- strawberry
- chocolate chip
- peach
- banana
- grape

A Strict() Master

While the methods discussed above will suffice for most of your FastTemplate requirements, the class also comes with a bunch of ancillary capabilities.

The strict() method is used to display an error if FastTemplate finds template variables without any values assigned to them; these undefined variables will also appear as is in the final output.

```
<?
// strict error checking
$obj->strict();
?>
```

The opposite of this is the no_strict() method, which replaces these unassigned variables with empty strings.

```
<?
// turn off error checking
$obj->no_strict();
?>
```

The fetch() method returns the raw data which results from a parse() operation. Consider the following:

```
<?
// parse templates
$obj->parse(RESULT, "list");

// print
echo $obj->fetch("RESULT");
?>
```

The clear() method is used to clear variables.

```
<?
// parse templates
$obj->parse(RESULT, "list");

// clear
$obj->clear("RESULT");

// prints nothing
$obj->FastPrint("RESULT");
```

```
?>
```

The get_assigned() method is used to obtain the value of any FastTemplate variable.

```
<?
$obj->assign("EMAIL", "jdoe@somewhere.com");

// returns "jdoe@somewhere.com"
echo $obj->get_assigned("EMAIL");
?>
```

And finally, the utime() function comes in handy when you need to measure script execution time.

```
<?
// list.php - item list

// include class file
include("class.FastTemplate.php3");

// instantiate new object
$obj = new FastTemplate("./tmpl/");

// get start time
$begin = $obj->utime();

// assign names for template files
$obj->define(array(
"list" => "list.tpl",
"list_item" => "list_item.tpl"
));

// get item list
// assume it looks like this..
$items = array("vanilla", "pineapple", "strawberry",
"chocolate chip",
"peach", "banana", "grape");

// build LISTCONTENT variable by concatenation of multiple
instances of
"list_item" template
for ($x=0; $x<sizeof($items); $x++)
{
$obj->assign("ITEM", $items[$x]);
$obj->parse(LISTCONTENT, ".list_item");
}
```

```
// parse templates
$obj->parse(RESULT, "list");

// and print
$obj->FastPrint(RESULT);

// get end time
$end = $obj->utime();

// print script execution time
echo "Done in " . sprintf("%01.3f ", ($end - $begin)) ."
seconds.";
?>
```

Musical Chairs

I'd like to wrap up this article with a comprehensive example, which demonstrates how easy it is to use FastTemplate to quickly build different types of page layouts.

Let's suppose I wanted to generate a Web page containing music news and reviews, and let's further suppose that it looks like this:

Here are the templates I plan to use:

```
<!-- begin: main.tpl -->
<html>
<head>
<basefont face="Verdana">
</head>

<body link="Navy" vlink="Navy" alink="Navy">

<!-- standard header -->
<table width="100%">
<tr>
<td bgcolor=navy height=50> </td>
</tr>
</table>

<!-- table for quick links -->
<table width="100%" border="0" cellspacing="10"
cellpadding="10">
<tr>
{LINKS}
```

```
</tr>
</table>

<p>

<!-- main story -->
{ARTICLE}

<!-- standard footer -->
<hr>
<center><font size=-2>All content copyright and proprietary
Melonfire,
2001. All rights reserved.</font></center>

</body>
</html>
<!-- end: main.tpl -->

<!-- begin: list.tpl -->
<!-- this generates the quick link boxes -->
<td valign=top>
<table border=1 cellspacing=0 cellpadding=5>
<tr><td align=center bgcolor="black">
<font color=white><b>{SECTION_TITLE}</b></font>
</td></tr>
<tr><td>
<ul>
{LIST}
</ul>
</td></tr>
</table>
</td>
<!-- end: list.tpl -->

<!-- begin: listitem.tpl -->
<!-- individual list items -->
<li><a href="story.php?id={ID}"><font
size=-1>{ITEM}</font></a>
<!-- end: listitem.tpl -->

<!-- begin: article.tpl -->
<!-- story area -->
<table border="0" cellspacing="0" cellpadding="0">
<tr><td valign=top>
<img src="{IMAGE}" height=100 width=100 align="left" border=0
hspace=5
alt="">
<h3>{SLUG}</h3>
<p>
```

```
{STORY}
</td></tr>
</table>
<!-- end: article.tpl -->
```

I'm going to use these four templates to generate the layout illustrated above.

```
<?

/*** this entire section would come from a database ***/

// article details - title, content, poster
$slug = "Yellow Gold";
$story = "Coldplay's debut album, <i>Parachutes</i>, has won
them a bunch
of awards, and the first single, <i>Yellow</i>, has been
receiving a fair
amount of airplay in both Europe and America. But if you want
to understand
why this band is regarded as one of best new acts to have
emerged from
England, you need to dig a little deeper...<p> Let's repeat
that once
again, shall we? <p> Coldplay's debut album,
<i>Parachutes</i>, has won
them a bunch of awards, and the first single, <i>Yellow</i>,
has been
receiving a fair amount of airplay in both Europe and America.
But if you
want to understand why this band is regarded as one of best
new acts to
have emerged from England, you need to dig a little
deeper...<p>";
$image = "poster.gif";

// list of sections
$sections = array("Pop", "Rock", "Dance");

// list of titles for quick links
// set as a 2D array
$items = array();
// pop links
$items[0][0] = "All For You - Janet Jackson";
$items[0][1] = "Room Service - Roxette";
$items[0][2] = "No Strings Attached - N-Sync";

// rock links
```

```
$items[1][0] = "Jaded - Aerosmith";
$items[1][1] = "All That You Can't Leave Behind - U2";
$items[1][2] = "Parachutes - Coldplay";
$items[1][3] = "Everything You Want - Vertical Horizon";

// dance links
$items[2][0] = "Spin - Collection";
$items[2][1] = "Buddha Bar - Collection";

// corresponding story ids
$ids = array();
$ids[0][0] = 23;
$ids[0][1] = 124;
$ids[0][2] = 65;

$ids[1][0] = 63;
$ids[1][1] = 234;
$ids[1][2] = 43;
$ids[1][3] = 533;

$ids[2][0] = 12;
$ids[2][1] = 239;

/*** database action ends **/

// include class file
include("class.FastTemplate.php3");

// instantiate new object
$obj = new FastTemplate("./tmpl/");

// assign names for template files
$obj->define(array(
"main" => "main.tpl",
"list" => "list.tpl",
"listitem" => "listitem.tpl",
"article" => "article.tpl"
));

// assign variables
$obj->assign(array(
"SLUG" => $slug,
"STORY" => $story,
"IMAGE" => $image
));

// this section builds the list items, and then the different
list boxes
for ($x=0; $x<sizeof($sections); $x++)
```

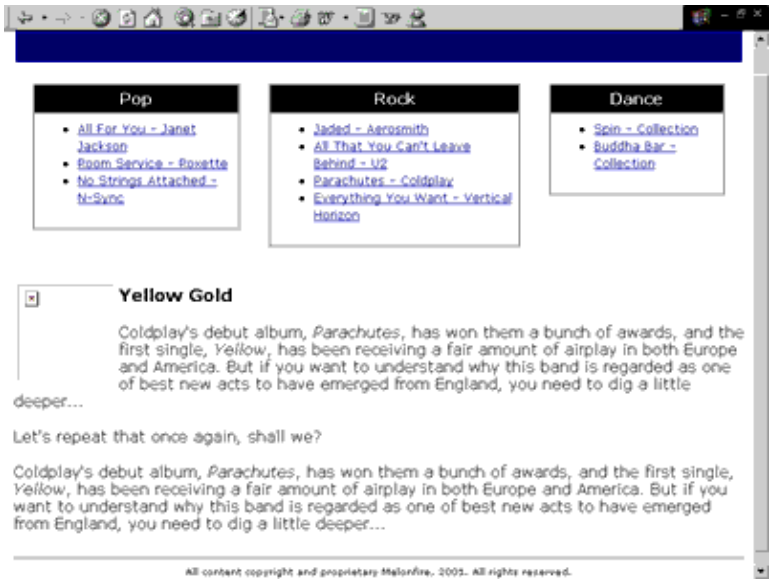
```
{
// first loop through section list
// get the name of this section
$obj->assign("SECTION_TITLE", $sections[$x]);

// this loop is to build the <li> items
for ($y=0; $y<sizeof($items[$x]); $y++)
{
$obj->assign(array(
"ITEM" => $items[$x][$y],
"ID" => $ids[$x][$y]
));

// each item is added to the previous
$obj->parse("LIST", ".listitem");
}

// at this stage, the list is complete
// the complete list is assigned (appended) to the end of a
new variable
$obj->parse("LINKS", ".list");

// clear the LIST variable for the next series
$obj->clear("LIST");
}

// parse templates
$obj->parse(ARTICLE, "article");
$obj->parse(RESULT, "main");

// and print
$obj->FastPrint(RESULT);
?>
```

The first part of this script is focused solely on extracting information to display from a database − I've hard−coded the values here for demonstration purposes. Once the variables are set, the script initializes a FastTemplate object and defines names for the four templates I plan to use.

Since the article template is the simplest, the script assigns values to the "STORY", "SLUG" and "IMAGE" variables first. Once that's done, the various link boxes are built up and appended to the "LINKS" variable using the "." operator.

At the end of this process, the "LINKS" variable stores all the HTML code required to generate the three boxes at the top of the page. Next, the "article.tpl" template is parsed and values assigned to its variables; the result is then stored in the "ARTICLE" variable. Finally, the values of both the "ARTICLE" and "LINKS" variables are transposed in the "main.tpl" template, and the result printed to the browser.

Here's what the output looks like:


Pop
All For You - Janet Jackson
Room Service - Roxette
No Strings Attached - N-Sync
Rock
Jaded - Aerosmith
All That You Can't Leave Behind - U2
Parachutes - Coldplay
Everything You Want - Vertical Horizon
Dance
Spin - Collection
Buddha Bar - Collection
Yellow Gold
Coldplay's debut album, Parachutes, has won them a bunch of awards, and the first single, Yellow, has been receiving a fair amount of airplay in both Europe and America. But if you want to understand why this band is regarded as one of best new acts to have emerged from England, you need to dig a little deeper...
Let's repeat that once again, shall we?
Coldplay's debut album, Parachutes, has won them a bunch of awards, and the first single, Yellow, has been receiving a fair amount of airplay in both Europe and America. But if you want to understand why this band is regarded as one of best new acts to have emerged from England, you need to dig a little deeper...
All content copyright and proprietary Melonfire, 2005. All rights reserved.

A Rose By Any Other Name...

So that's one look − but now how about changing it so that the quick link boxes appear in a column on the left, with the main story taking up the rest of the page?

With FastTemplate, altering the layout is a snap − it merely involves editing the "main.tpl" and "list.tpl" templates.

```
<!-- begin: main.tpl -->
<html>
<head>
<basefont face="Verdana">
</head>

<body link="Navy" vlink="Navy" alink="Navy">

<!-- standard header -->
<table width="100%">
<tr>
<td bgcolor=navy height=50> </td>
</tr>
</table>

<table width="100%" border="0" cellspacing="10"
cellpadding="10">
<tr>
<td valign=top>
{LINKS}
</td>
<td valign=top>
{ARTICLE}
</td>
</tr>
</table>

<!-- standard footer -->
<hr>
<center><font size=-2>All content copyright and proprietary
Melonfire,
2001. All rights reserved.</font></center>

</body>
</html>
<!-- end: main.tpl -->

<!-- begin: list.tpl -->
<table border=1 cellspacing=0 cellpadding=5>
```

```
<tr><td align=center bgcolor="black">
<font color=white><b>{SECTION_TITLE}</b></font>
</td></tr>
<tr><td>
<ul>
{LIST}
</ul>
</td></tr>
</table>
<p>
<!-- end: list.tpl -->
```

In this case, I've simply set up a new table structure for the main page to accommodate the new layout. And when I run the PHP script (without changing a single line of code), here's what I get:

Why stop there? Let's do away with the link boxes altogether, and have the links appear in neat rows at the bottom…

```
<!-- begin: main.tpl -->
<html>
<head>
<basefont face="Verdana">
</head>

<body link="Navy" vlink="Navy" alink="Navy">

<!-- standard header -->
<table width="100%">
<tr>
<td bgcolor=navy height=50> </td>
```

```
</tr>
</table>

<p>

{ARTICLE}

<p>

{LINKS}

<!-- standard footer -->
<hr>
<center><font size=-2>All content copyright and proprietary
Melonfire,
2001. All rights reserved.</font></center>

</body>
</html>
<!-- end: main.tpl -->


<!-- begin: list.tpl -->
<font size=-1>
New in <b>{SECTION_TITLE}</b>...
<br>
{LIST}
</font>
<p>
<!-- end: list.tpl -->

<!-- begin: listitem.tpl -->
<a href="story.php?id={ID}"><font
size=-1>{ITEM}</font></a>
<!-- end: listitem.tpl -->
```

In this case, I've altered three of the templates to remove the tables and list constructs, so that I'm left with a very simple and elegant layout.

Yellow Gold

Coldplay's debut album, *Parachutes*, has won them a bunch of awards, and the first single, *Yellow*, has been receiving a fair amount of airplay in both Europe and America. But if you want to understand why this band is regarded as one of best new acts to have emerged from England, you need to dig a little deeper...

Let's repeat that once again, shall we?

Coldplay's debut album, *Parachutes*, has won them a bunch of awards, and the first single, *Yellow*, has been receiving a fair amount of airplay in both Europe and America. But if you want to understand why this band is regarded as one of best new acts to have emerged from England, you need to dig a little deeper...

New in **Pop**...
All For You - Janet Jackson Room Service - Roxette No Strings Attached - N-Sync

New in **Rock**...
Jaded - Aerosmith All That You Can't Leave Behind - U2 Parachutes - Coldplay Everything You Want - Vertical Horizon

New in **Dance**...
Spin - Collection Buddha Bar - Collection

As you can see, FastTemplate makes it possible to separate the user interface from the program logic, thereby allowing designers with little or no programming knowledge to alter Web pages quickly and easily. Further − with its ability to nest and repeat chunks of HTML code, it can speed up development time significantly, and also reduce the effort involved in maintaining and modifying a Web application.

That's about it from me. In case you're looking for more information, you should consider visiting the following links:

The FastTemplate home page: http://www.thewebmasters.net/

PHPBuilder's FastTemplate tutorial: http://www.phpbuilder.com/columns/sascha19990316.php3

The original (Perl) FastTemplate module: http://www.sober.com/

Till next time...stay healthy!