treehouse.

A Web Development Magazine

November 2005

ISSN :1558-013X

Inside the Treehouse

Cover Notes
by Brian Campbell

I'd found this tree up in North Carolina on my honeymoon, almost completely gone, just this one leaf hanging on. It stood amongst some other trees in reality, but it had this air of defiance and independance to it. Tried to build a scene around that. Something about this little tree trying to grow despite opposition reminded me a bit of some friends I knew. *Smirk.* Lots of acrylic paint and paper mixed in here digitally for the background. Also a great quote that I'd had saved somewhere that seemed to fit aesthetically and thematically. Maybe some tape to hold it all together?

Treehouse Logo
by Julius Santiago

From the Editors — *by Kevin Hale* — 3

To the Editors — *Letters & Comments* — 4

Cartoon — *Robot Tea Party* — 36

Network News — *Around 9rules* — 37

Marketplace — *Classifieds* — 65

Code

Catching Up with addEvent() — *by Ryan Campbell* — 7

Interview — Peter-Paul Koch — 12

Book Review — *Foundations of Ajax* — 17

Best of the Web — *Code Links* — 18

Design

Understanding Section 508 — *by Alex McClung* — 20

Interview — Lisa McMillan — 28

Book Review — *Thinking with Type* — 33

Best of the Web — *Design Links* — 34

Business

Working With Non-Profits — *by Nathan Smith* — 39

Interview — Josh Williams — 43

Book Review — *The World is Flat* — 47

Best of the Web — *Business Links* — 48

Digest

The 4 Layers of Separation — *by Ryan Campbell* — 50

Ten Tips to a Better Form — *by Chris Campbell* — 59

digest.

"As for the future, your task
is not to foresee but to enable it."

Antoine de Saint-Exupery

DIGEST

SEE DEMO

DOWNLOAD FILES

VISIT ORIGINAL

4 LAYERS OF SEPARATION

By Ryan Campbell

Thanks to a lot of progressive education, web developers are starting to regularly practice three layers of separation (structural, presentational, and behavioral) in their projects and applications. Loosely assigned, XHTML builds the structure, CSS defines the presentation and JavaScript (for the most part) creates the behavior. This code segregation allows developers to create web applications that are organized, maintainable and reusable.

I believe, however, that a fourth layer of separation is being neglected: the data layer. This layer is represented by server side scripts that process and retrieve information from a data source. More often than not, we find this layer embedded messily into the structural layer. When the goal is to build modular architectures that are flexible and adaptable, combining structure and data processing is, in the long run, going to be very costly conceptual mistake. Through the use of a very promising XML technology, XSLT, we can free our data processing and retrieval logic from our display and structural logic completely and build web applications that are easier to understand and faster to iterate.

While none of the ideas in this article are particularly new, we believe there's been a lack of attention brought to these topics (especially XSL) and since companies like Google are even using XSLT in their projects, we figure there's a lot of developers out there who could use a look at how all of the puzzle pieces (web technologies) are starting to come together. For your convenience, we've also created a PDF of all the diagrams used in this article.

The Concept Behind 4 Layers

The only thing unique about the 4 Layers of Separation is the division of the Structure Layer into XSL

See It In Action :
In an earlier article on The Hows and Whys of Degradable Ajax, I created a simple Todo List to demonstrate a modern web application that's not dependent upon JavaScript. I've recreated the same application here, but I used the principles underlying the 4 Layers of Separation to build the code. You can see the demo and compare the source code files here:

Todo List Demo

Download 3 Layers Code

Download 4 Layers Code

templates and a set of server-side scripts to process information from the data source. Essentially, we want to allow structures or templates to be created or manipulated regardless of the data source or retrieval logic that will populate it. This is because we want to avoid embedding markup into a PHP script and vice versa. For example, here's a simple script that populates and builds a table based on information in a database:

```
<?php
// Step 1 Get any form fields needed for query
$username = $_REQUEST["username"];

// Step 2 - Build Query
$sql =  "SELECT * FROM tAccounts WHERE
accountName = '".$username."'";

// Step 3 - Execute Query
$rs = $cDB->ExecuteReader($sql);

// Step 4 - Display Results
echo "<table>";
while($row = @mysql_fetch_array($rs, MYSQL_ASSOC)) {
    echo "<tr>";
    echo "<td>".$row["username"]."</td>";
    echo "</tr>";
}
echo "</table>";
?>
```

This is a simple yet typical way of how most scripts are written on the web and the problem here is that we are combining the handling of data with the HTML presentation of it. Changing how this data will be presented requires the server side processing and the HTML wireframes to be manipulated simultaneously because they are chained together in the code. This makes structural changes like showing the same data as an ordered list difficult and slow.

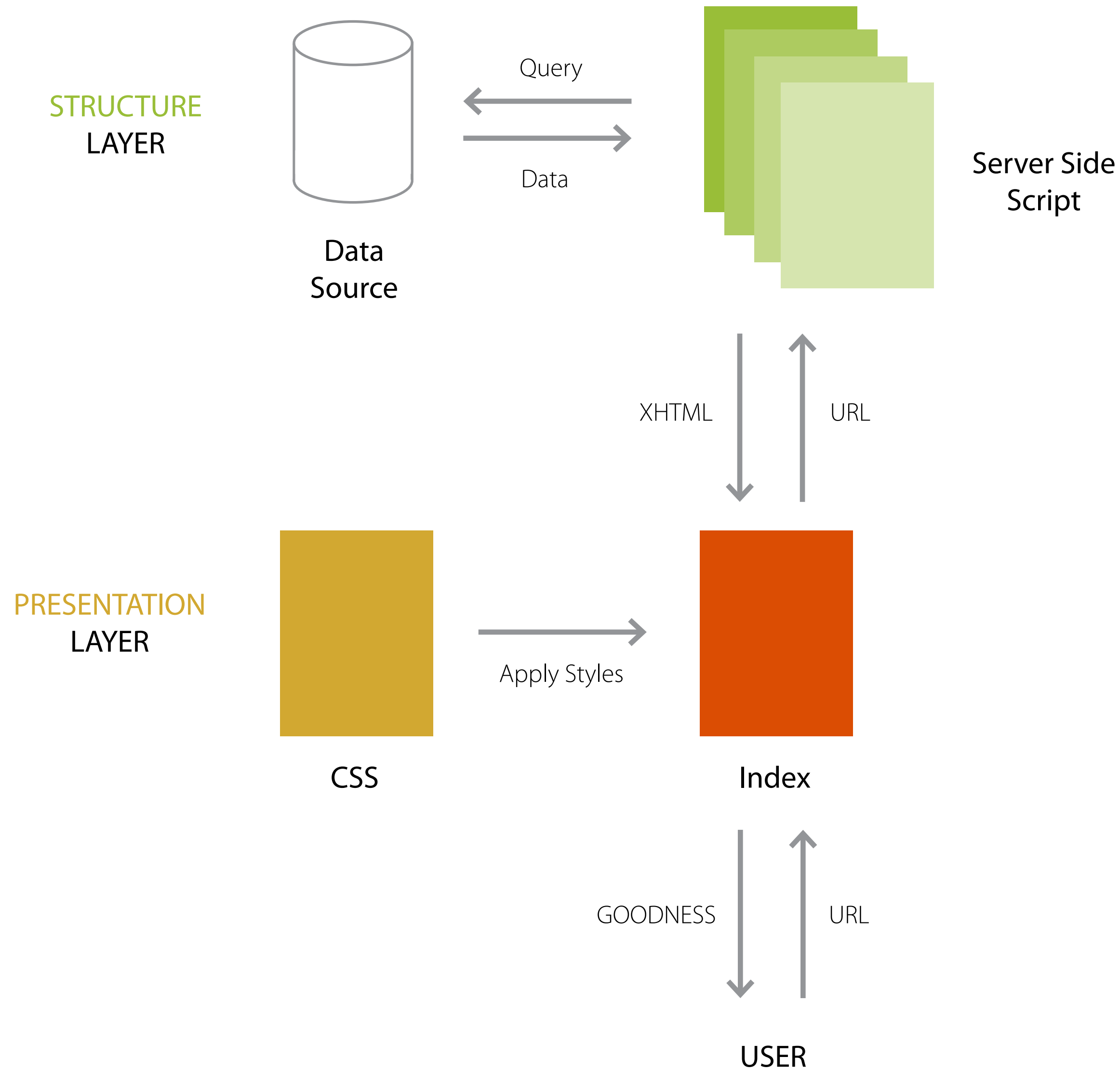
In addition to that, steps 1-3 could easily become 50+ lines of code when you consider additional data processing tasks such as validation, security, etc. Now, let's take a look at the same code written in XSLT to separate out the data layer:

```
<table>
  <xsl:for-each select="Accounts/Account">
    <tr>
      <td><xsl:value-of select="username" /></td>
    </tr>
  </xsl:for-each>
</table>
```

Functionally, everything is the same. We are creating a loop and going through it to display the rows of a table. The difference here is that the processing and retrieval of data is nowhere to be found. This is the power of XSLT. It doesn't care where the data comes from or how it is created. All XSLT requires is a properly formatted XML file (which would be created by your data layer scripts) to do its work. To learn more about XSLT, I recommend checking out Kevin's excellent roundup on the subject. For the visually inclined, Kevin whipped up a little diagram to show how all of these layers come together.

2 Layers of Separation

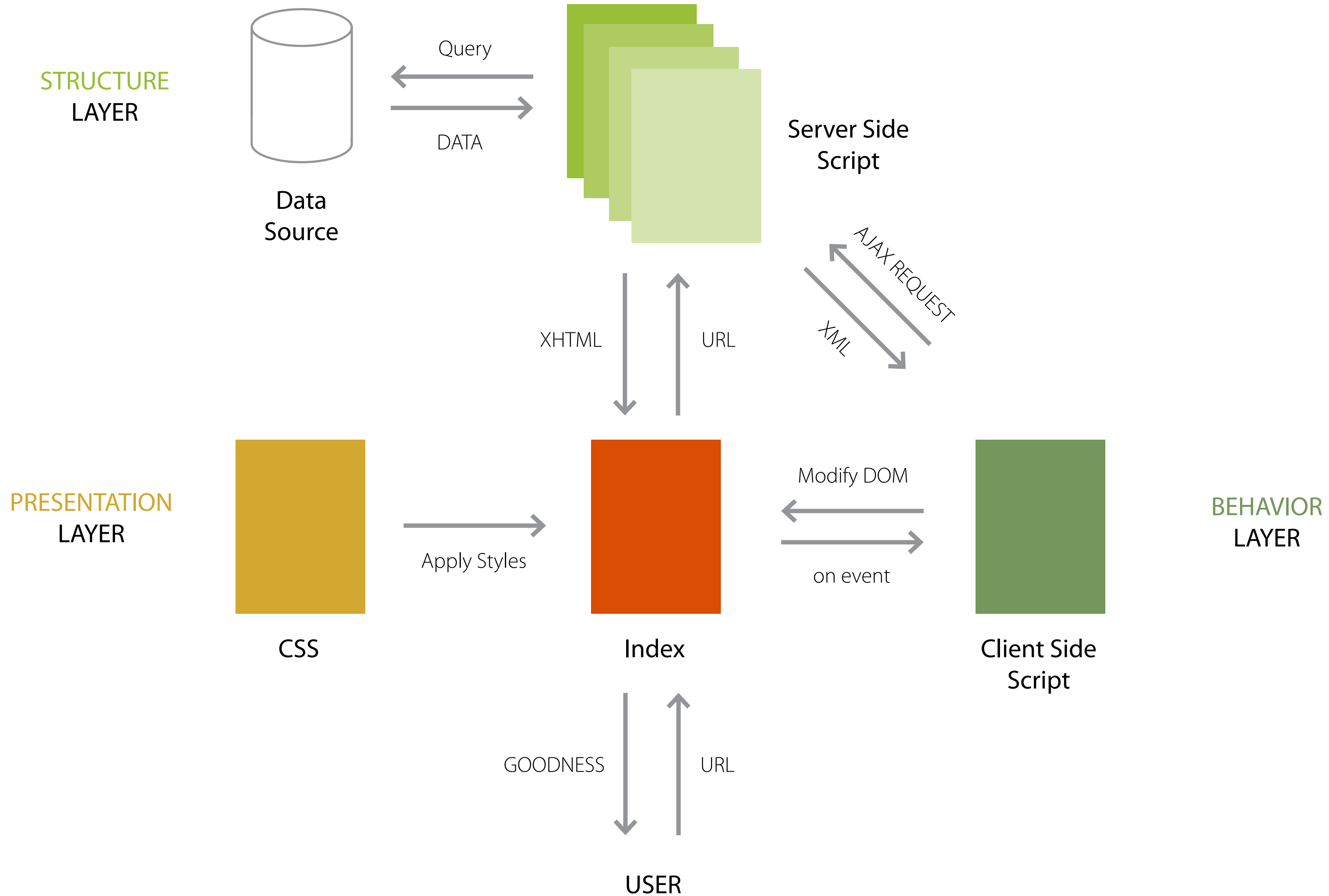
Typical Way of Doing Things



Copyright © 2005
Particletree Inc.

3 Layers of Separation

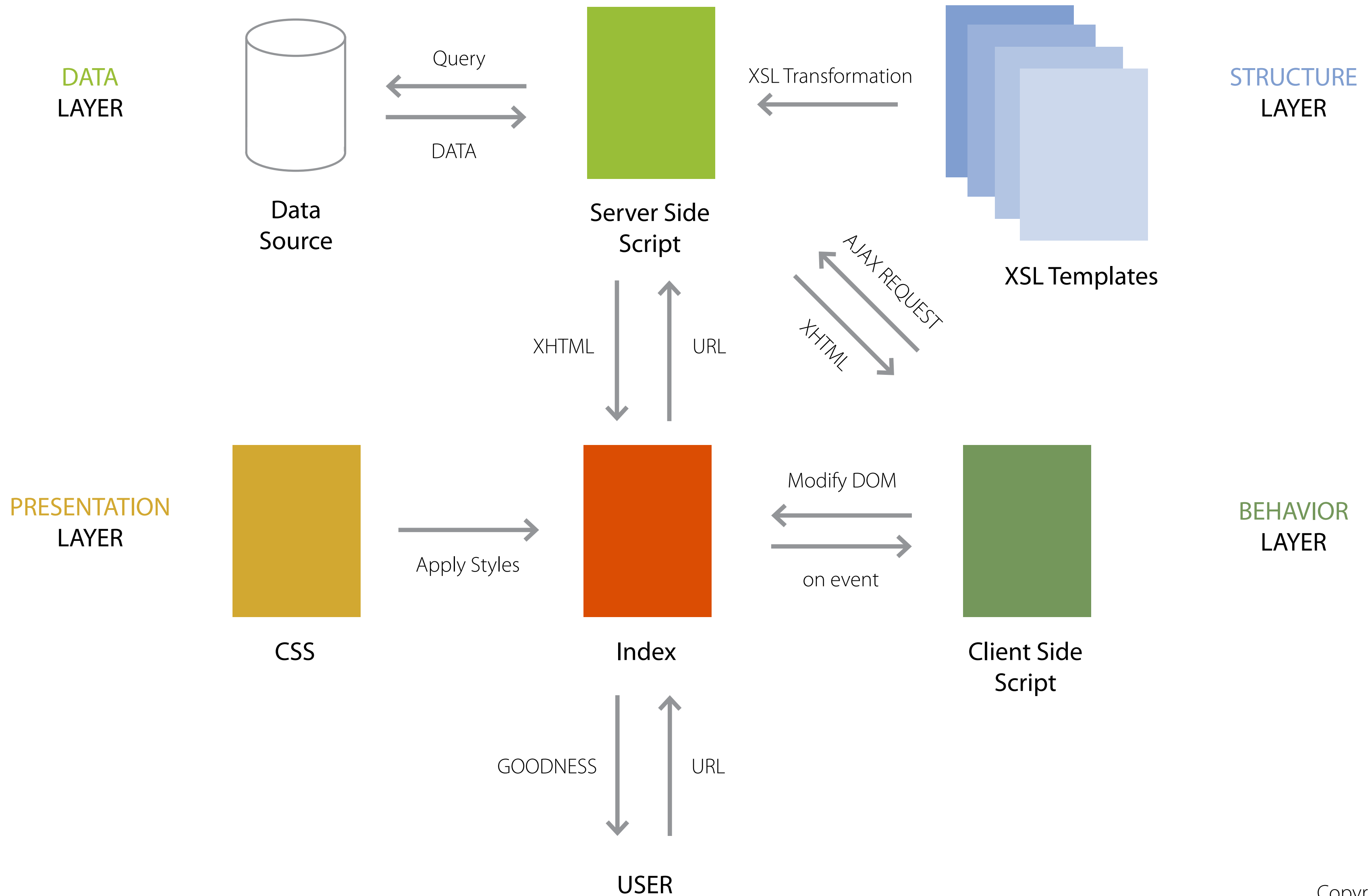
Introducing AJAX



Copyright © 2005
Particletree Inc.

4 Layers of Separation

Particletree Degradable Ajax Setup



Copyright © 2005
Particletree Inc.

The Four Layers

Presentation Layer - The Presentation Layer is responsible for styling the way your elements look. Primarily focused on the aesthetics of your application, it is important to work closely with the Structure Layer for compatibility. This layer, however, does not involve manipulating markup. If you're messing with XHTML, you're messing with structure.

The most dominant form of this layer is CSS, but you can also substitute/integrate the visual effects of Flash into this layer as well if the ActionScript is separated out into external files.

Behavior Layer - The Behavior Layer is responsible for allowing dynamic interactions and real-time effects. Popularized by the recent Ajax fervor, this layer is receiving a lot more attention by developers. It is important to note that this layer must be integrated unobtrusively to be considered sufficiently separated from the other layers (like structure). This layer is traditionally created using JavaScript, but feel free to also use ActionScript, JAVA and if you're an IE fan, VBscript.

Structure Layer - The Structure Layer determines the format your data will be presented to the user. Most of the time this is XHTML, but you can also transform data to look like anything else (PDF, Word Document, vCard, iCal, RSS, TXT and even unformatted XML).While the structural determination can be accomplished with server-side functions, this layer is best handled by XSL. By using XSLT, developers can cleanly delegate data processing to the Data Layer.

Data Layer - The Data Layer handles data coming from the user or from an outside data source. This layer is responsible for validating data, processing data and formatting data into a transportable format like XML (so it can be turned into markup based on templates provided by the Structure Layer).

Any server-side solution can be used to process this data (PHP, .NET, Python, Ruby, Perl, etc). And while the Data Source is usually a database (mySQL, SQL server, Oracle), it can also be an XML file, a text file, an RSS feed or even a Web 2.0 Service.

How it Works

The approach to a 4 layers project is probably the most important part of the implementation. When you separate CSS and JavaScript out of the structure, you do so because you want a reusable file that is easy to edit and maintain. The same mindset applies to using XSLT to separate the structural layer from the data layer.

The easiest way to think of XSLT is as an include file. We're all comfortable with creating a header and footer file that gets included on every page. You do the same thing with XSLT except you evaluate every include file in your document. Basically, if a section of our site will be dynamically added or removed via JavaScript, or if that section is bound to data, then we should convert it into an XSLT file.

Once we have decided which files the rule applies to, we can begin the conversion process. Remember, XSLT can display plain markup. Basically you're going

MORE DIAGRAMS :
To see full visual representation of the four layers in action, check out our 9 page PDF packet, 4 Layers of Separation. The PDF provides an high level overview of all of the layers, and how they operate.

to open a file, add the XSL header, write plain XHTML inside, and then save it with an extension of .xsl. When you are first starting out, don't think of an XSLT file as anything more than an HTML template document.

Binding Data

Just as you can put XHTML into any XSLT file, you can also mix in data with it. The only catch is that the data must be formatted as XML. So, you can pull data from a database, a web service, an ini file, or from a user as long as that data can be turned into an XML format. For a detailed look on how to convert information from a database into XML, see my tutorial on Database Simplicity - XML.

When you have an XSLT file, and a set of data you would like to associate with it, the next step is to perform a transformation. A transformation outputs XHTML which creates the structure of the page. Also, if you don't have any data to associate to one of your XSLT files, you can bind it to a blank XML file. A blank XML file only needs to consist of the XML header and one empty node. The following function will perform the transformation.

```
function transformData($xslFile, $rs, $parentNode, $childNode) {
  $xsl = new DomDocument();
  $xsl->load($xslFile);
  $input = new DomDocument();
  $input->loadXML($cDB->ConvertRStoXML($rs, $parentNode, $childNode));
  $proc = new XsltProcessor();
  $proc->importStylesheet($xsl);
  echo $proc->transformToXML($input);
}
```

Quick note about loadXML(): This function expects a string of XML to load, but if you actually have a physical XML file then just use load() instead of loadXML(). Other than that one line, the function is straightforward. It creates both an XML document and XSLT document, then it loads the stylesheet using importStylesheet() which then transforms the data using transformToXML(). After this function executes, you will see it display on your page.

Let's do a quick recap.

1. Create XSLT files based on sections of your site.

2. Build queries to associate with each XSLT file - use blank.xml if no data is associated.

3. Call transformData() in the order you wish sections to display on the page.

Benefits of 4 Layers Approach

• **Easier to Develop in Parallel** - The largest benefit that comes from the 4 Layers is that it separates most of the server side code out of the markup. This means that designers can edit the display of a page without ever having to look at what's going on behind the scenes. Additionally, files stay just as reusable as an include would.

• **Ajax is Easier to Implement** - In most Ajax applications markup is created in the JavaScript. This means that if the designer wishes to redo a section of a page, the associated JavaScript has to be completely redone as well. Using XSLT, we can dynamically change the display of a page based off of an external file.

• **Improves Database Queries** - One thing that I have learned is that almost every SQL SELECT can be executed in one query. However, when I was first learning SQL I did exactly the opposite, which led

***XML FILE :**
Actually, Particletree reader, Hilarie J., pointed out that XSL can handle more than one XML document by using the document function.

"The document() function is used to access nodes in an external XML document. The external XML document must be valid and parsable."

—W3Schools

Now that the option to link multiple XML files into one XSL template is available, that does not mean it should be encouraged. Writing efficient, single SQL queries to build your XML files is more appealing than multiple queries creating multiple XML files. Use this power with caution and understanding.

to many recordsets looping within one another. Since XSLT files can only accept one dynamic XML file*, you are forced to make sure you can retrieve all of your data in one query. This generally leads to more efficient queries.

- **Easier to Build Degradable Pages** - With or without JavaScript, the page will be reading the same external XSLT file. This means your site will contain the same exact same markup with or without JavaScript.

- **Choose Any Data Source** - You could easily connect to a different database with different table and field names and you would not have to worry about changing the code that displays the structure.

Limitations of 4 Layers Approach

- **XSLT** - It's a tough language. On top of that, complex equations become code heavy, and not intuitive at all. While tricks like working equations into your queries provide some benefit, XSLT overall is a problem in itself. This alone will discourage many people. Fortunately, this concept will work with any language that can transform XML, so substituting an alternative in the future won't be hard.

And while the fact that everyone on a team needs to learn XSLT is a limitation, but it is a limitation that I feel is acceptable considering the benefits. The ideal programmer should understand CSS and markup so that he can compliment the designers on a team (UI, visual, etc). And likewise, the ideal designer should create pages that are structured to help a programmer.

- **Speed** - Using XSLT makes both Ajax sites and a non-Ajax sites a bit slower because the transformations have to be processed before being presented to the user. However, I have found performance degradations to be negligible so far. The ideal situation for dividing your development into four layers occurs while using Ajax. While it is a tad slower than pure JavaScript Ajax, this technique is still much faster than a normal refreshing web site and has all of the added benefits of the segregation mentioned above.

- **Server Load** - Right now, dynamic transformations can't be reliably performed on the client because different browsers support XSLT differ-

XSL SUPPORT :
Safari 1.3 and above supports XSL transformations of XML pages at load time. XSLT support is limited to XSL page processing instructions embedded at the top of an XML page:

XSLT not available via JavaScript for application to arbitrary XML in an HTML page.

Safari Developer FAQ

ently. No surprise there. Firefox and IE have completely different methods and Safari isn't convinced yet.

Since there are no reliable dynamic client side solutions, I recommend doing all of the transformations on the server to ensure things work for the most number of users. This is fine for developers who do without the JavaScript fanciness, but on an Ajax site the potential server savings are sadly being missed. Google, however, is approaching the XSLT browser support problem with brute force by recreating all of the XSL transformation functions in JavaScript. This is why they created the open source project, AJAXSLT. We think their applications (like gmail) work like this:

The AJAXSLT JavaScript file will add about 90kb (fully commented and uncompressed) to your application. And while their implementation is faster than our degradable version (because it offloads the processing to the client), to make their projects degradable, they have to duplicate all of their functions both on the client and server side (which may be what they did to create the plain HTML version of gmail). We believe, however, that this is just a temporary solution for Google and that they (along with us) is betting on XSL support to just get better across the browsers.

Possibilities

Non compiled .NET - The inspiration for this project came from .NET code behind pages. They beautifully separate server side code from structure. My problem with them is that they need to be compiled, and the overhead in many cases is huge. This technique may lead to a similar setup without the need for a framework.

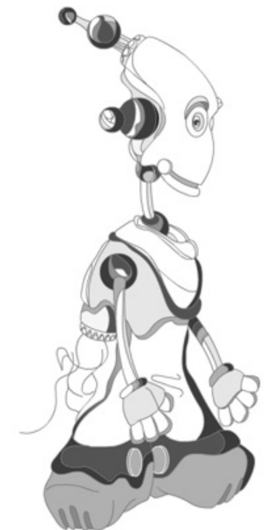
• **Eliminate Proprietary Templates** - When you use your content manager, I am sure there are tags used to retrieve and display data. It would be nice if you could instead refer to a dynamic XML file, and use XSLT to display your data.

• **Browser Support** - Client side processing would be a welcome addition. It would reduce the server load, and allow us to possibly switch to some form of REST or SOAP solution instead of a URL based solution.

In Summary

XSLT helps to provide another level of separation in web pages. Before the age of Ajax, a technique such as this hardly seemed worth the effort. It can result in a slower site with a higher learning curve. However, with more sites duplicating markup in both PHP and JavaScript, XSLT provides a nice compromise. It is a language that both JavaScript and PHP are able to understand, which removes redundancy.

T-SHIRTS BY BROKEN KODE



www.brokenkode.com
in association with StreetShirts.co.uk

TEN TIPS TO A BETTER FORM

By Chris Campbell

The most monotonous entities in the known universe, forms, are a staple of every web programmer's balanced diet. Whether we like them or not, forms are the gatekeepers to our site's goodies and often their design alone determines whether a user will try what you're selling or simply walk away. Without pomp or circumstance, here are ten tips to transform your plain vanilla into double chocolate chunk with marshmallows.

1. REMEMBER YOUR MARKUP

We've notice a lot of people forgetting to use the tools that are already made accessible to them by the very medium that they work in. And so we've highlighted a few HTML elements below that are made especially for forms. Just to refreshen the ol' web noggin.

Label

A label is used to attach information to a control. If you focus on a label, its associated control will gain focus. This is useful when a user clicks on a label name and the associated field gains focus. There are two ways to markup labels in your forms:

```
<label for="email">Email: </label>
<input type="text" id="email">
<label for="name">Name: </label>
<input type="text" id="name">
```

or

```
<label>
Email: <input type="text" id="email">
</label>
```

THE CODE :
The code examples used through out thise article will not work "as is." They are dependant on an external library, prototype.js with addEvent included. Additionally, the functions below need to be attached to events, such as onclick or onmouseover.

Fieldset
The FIELDSET element allows authors to group thematically related controls and labels. Grouping controls makes it easier for users to understand their purpose while simultaneously facilitating tabbing navigation for visual user agents and speech navigation for speech-oriented user agents. The proper use of this element makes documents more accessible.

W3C Recommendation

Legend
The LEGEND element allows authors to assign a caption to a FIELDSET. The legend improves accessibility when the FIELDSET is rendered non-visually.

W3C Recommendation

Tabindex
This attribute specifies the position of the current element in the tabbing order for the current document. This value must be a number between 0 and 32767. User agents should ignore leading zeros. The tabbing order defines the order in which elements will receive focus when navigated by the user via the keyboard. The tabbing order may include elements nested within other elements.

W3C Recommendation

Accesskey
This attribute assigns an access key to an element. An access key is a single character from the document character set. Note. Authors should consider the input method of the expected reader when specifying an accesskey.

W3C Recommendation

Password
By adding type="password" to your input field, characters entered will be transformed into a series of asterisks.

Application designers should note that this mechanism affords only light security protection. Although the password is masked by user agents from casual observers, it is transmitted to the server in clear text, and may be read by anyone with low-level access to the network.

W3C Recommendation

2. CSS

This isn't new, but CSS can turn your eyesore of a form into something negative fugly. There is no need to reinvent the wheel here, so check out the following sources on enhancing your form with some CSS and a little JavaScript.

- **Style Those Buttons** - The Man In Blue shows us how to make those buttons not look so cheap.

- **Niceforms** - Niceforms does a great job of turning that ugly form into something much more tolerable using a little CSS action.

- **Hide Optional Fields** - In this example, a little CSS and Javascript is used to make a form better looking and more usable.

- **CSS Forms** - Jeff Howdens shows us how to create a well laid out and styled form without using tables.

3. AUTOTAB

When navigating through a form, the user traditionally presses the tab button in order to advance to the next form control. This AutoTab function automatically sets the focus to the next control after the control's maxlength is reached. This allows for the user to no longer manually tab through fields with a maxlength. This function is particularly useful for fields such as social security or phone numbers containing a character limit for each input field. For example, after a user enters the area code of a phone number, the form automatically tabs to the next input box allowing the user to continue entering their phone number without interruption.

To flag an input element to be autotabbed, you only need to include three things in your markup: tabindex, className of autoTab and maxlength.

```
<input type="text" name="areacode" class="autoTab"
tabindex="1" maxlength="3"  />
```

On page load, it'll attach the events to input fields for autotabbing. If a field has a maxlength and that maximum is reached, the focus is automatically set to the control with the next highest tabindex value. Here's a quick look at our function, autoTab():

```
function autoTab(e) {
  if(this.value.length == this.getAttribute("maxlength") &&
        e.KeyCode != 8 && e.keyCode != 16 && e.keyCode != 9)
   {
    new Field.activate(findNextElement(this.getAttribute("tabindex")));
   }
}

function findNextElement(index) {
    elements = new Form.getElements('shippingInfo');
    for(i = 0; i < elements.length; i++) {
        element = elements[i];
        if(parseInt(element.getAttribute("tabindex")) ==
          (parseInt(index) + 1)) {
            return element;
        }
    }
    return elements[0];
}
```

Two things worth noting:

1. You could also use `input.form[(getIndex(input)+1)].focus()`, but that causes a weird javascript error when using Firefox. An example of this can be seen at The Javascript Source.

2. When a user presses shift-tab, the previous element should still get focus, and autotab with automatically disable.

4. FIELD INFORMATION

It's always good policy to provide information describing a field's requirements and restrictions. How else is the user going to know a password must have 3 capital letters, an exclamation point and be somewhere between 6 and 17 characters long? Inspired by the forms used over at Tangent, we came up with some additional suggestions.

- **Store all related information in a fields label tag.** You can place a className of required, an accesskey, and a descriptive title in there, so that all information is in one place. This makes it easier to roganize, and easier for JavaScript to pull from.

- **We choose to use onfocus instead of onmouseover when displaying information to the user.** While mostly personal preference, it is still nice to know that the proper information will be displayed on the field with focus rather than where the mouse is.

- **It is easier to give labels an id similar to the field they are related to.** For example, if a field is named fname, call the label lfname. It makes things much easier on the JavaScript side to pull information. You can see the labels title being accessed below:

```
p = document.createElement("p");
p.innerHTML = $("l" + this.id).title;
span.appendChild(p);
```

5. ERROR DISPLAYS

When a user makes a mistake, it's your duty to show them errors quickly and efficiently. Here are some ideas to make your forms dis-

play errors better:

• **Don't just show the user one error.** If they left 3 required fields blank, make sure that you tell them they have three errors, this way they can correct them all in one fell swoop.

• **Provide as much information to the user beforehand as possible.** Examples of this would be marking a field as required, or explaning the minimum password length (See #2 above).

• **Be aware of the three validation options at your disposal**: 1) you can give responsive feedback straight from the JavaScript. The user benefits from instant feedback, but you will have to duplicate your validation functions on the client and server. 2) You can provide Degradable Ajax Validation that gets rid of the duplicated code, but increases the server load. 3) You may validate only on form submit which leaves you with no duplicated code, no additional server load, and unfortunately, no instant feedback.

1. **Put some effort into the display of your error messages.** Make them bold, noticable, and throw in a bit of creativity. It is also best to stick with colors that the user is comfortable with:red for errors, yellow for warnings, green for success. Obviously you can switch those up based on your evaluation, but going to far and making an error message pink could cause some confusion.

6. POSTBACKS

There is nothing worse than filling out a form, encountering an error, and having to retype all of your information all over again. In order to save your users from needless frustration, we need to ensure that all data is preserved. This means if there is an error, the fields should be repopulated. If we have a multistep form, back and forward navigation should also keep the form populated. A common approach is to set a form's action to it's current URL. That way, you can read in the form value and populate the fields immediately if there is an error. For example, just set the value to the post:

```
<input type="text" name="fname"
value="<?=$_POST["fname"] ?>" />
```

And the field will be populated. If there is no error, just redirect the page to the form completion page.

7. ONFOCUS

Visual cues, such as changing a field's border color, help show the user which field has focus. CSS provides an easy solution for adding borders with the border selector, but this feature isn't currently supported in IE and doesn't work at all with select elements. Our solution creates custom borders and backgrouds by adding a span to each form element using a little J-Juice.

The concept here is that by enclosing our form element within a span, we can produce a unique effect. Don't worry, you don't need to hand code the span—this unobtrusive JavaScript function will dynamically add and remove it automatically. The nice part is that the span can have background images, borders and any other desired combination of the two to create an effect that will work on text, textarea, and select elements. Here is how everything looks when it is done:

```
<span class="focus">
<input type="text" id="fname" name="fname" />
</span>

function showFocus() {
  this.parentNode.className = "focusHover";

function hideFocus() {
  this.parentNode.className = "focus";
}
```

The reason a state is needed for the span when the field does not have focus is to prevent shifting of the page. For example, if our custom span has 3 pixels of padding onfocus, we need 3 pixels of transparent padding when there is no focus, so that we do not see form elements "jumping" as the user navigates them.

8. LABEL CLICK

When a label receives focus, whether it be

through onclick or through accessKeys, the associated element specified in the labels for attribute should receive focus also.

> *When a LABEL element receives focus, it passes the focus on to its associated control.*
>
> *W3C Recommendation*

Unfortunately, this is not guaranteed in all browsers. To remedy the situation, a simple JavaScript function will do the trick:

```
function initLabels() {
    labels = document.getElementsByTagName("label");
    for(i = 0; i < labels.length; i++) {
        addEvent(labels[i], "click", labelFocus);
    }
}

function labelFocus() {
    new Field.focus(this.getAttribute('for'));
}
```

9. DOUBLE SUBMIT

Nooo! Our user has submitted the form twice, probably because the site did not respond fast enough. Not only will the data be processed twice, but the last submission will be the active submission which often will provide errors that will confuse the user even more. Three things to keep in mind for this situation:

- If JavaScript is enabled, use it to submit the form. After one submission, disable the button.

- If JavaScript is not enabled, provide your users with a clear message asking them not to submit critical data more than once.

- Do what you can on the server to prevent the processing duplicate information. Track payments, accounts, etc and flag them when an action has occured. This way, the second round of processing will just be ignored.

10. LEFTOVERS

And we're going to close this off by highlighting some awesome links to some features people have already implemented. You may not need them in your particular situation, but they're definitely lifesavers when you do.

- **Mini Calendar** - Whenever there is a date field, it is always best to provide a popup calendar. Ideally, this calendar will not open in a new window, and will be responsive. To get started, Dynarch has the best solution.

- **Combo boxes** - The element that missed the HTML cut is the half select, half text input breed known as the combo box. This concept allows the user to type in their own option if they can't find what they're looking for in the drop down. Take a look at Upgrade Your Select Element to a Combo Box to see how we approached a solution.

- **Visual Maps** - Just like the popup calendar, popup maps can also be effective. Take a look at Orbitz, and click on airport code. That takes you to a text listing of all airports. This could easily be a dynamic in page popup that displays a clickable map, which in turn populaltes the field when clicked.

YOU GOT A MESSAGE?
WE GOT A MEDIUM.

ADVERTISE IN TREEHOUSE.