DMX zone

Bring your website to live with AJAX and DHTML

Bring your website to live with AJAX and DHTML

by Tom Dell'Aringa

Published by DMXzone.com
Dynamic Zones International
Hengelosestraat 705
7521 PA Enschede
The Netherlands

Trademark Acknowledgements

DMXzone has endeavoured to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals. However, DMXzone cannot guarantee the accuracy of this information.

Flash, Dreamweaver, Director, Ultradev, ColdFusion and Fireworks are trademarks of Macromedia. Photoshop is a trademark of Adobe.

8

55

132

Getting started with DHTML

- DHTML: Introduction To Scripting The Dom
- DHTML: Dynamic Client Side Table Sorting
- DHTML: Dynamic Class Switching
- Introduction To Regular Expressions
- Dynamic "Hide and Show" Content Using The DOM

REAL-WORLD DHTML

- Popup Windows: Doing Them Right
- A Simple DHTML Flyout Menu
- Formatting User Form Data
- More Controlling Fields
- "Alertless" Client Side Error Messages
- The New And Improved Date Picker
- Spicing Up Data Tables With Highlighting Rows
- An Abstract "Add Row" Class For Dynamic Table Rows (Like YAHOO!)

TAKE CONTROL WITH AJAX

- Introduction To AJAX
- The AJAX Filmstrip
- AJAX: Ready States, PHP and Arrays
- Google Maps And AJAX: Web 2.0 On Your Web

Introduction

This e-book will teach you how to use DHTML and AJAX to create amazing new applications and interactive features for your website. Tom Dell'Aringa shows you how to create Popup Windows, Fly out menus, Filmstrips, advanced form features and much more!

Ajax enables you to write interactive applications while reducing the amount of data interchanged between the web browser and web server. This results in shorter processing times so you can give your user the ultimate web experience.DHTML allows you to create amazing effects and allows you to enhance the interactivity of your webpage without using any Plug-ins. DHTML files are also smaller then HTML files thus speeding up your website even further.

What this book does, and who it's for.

This book is for anyone with an interest in developing their DHTML and AJAX skills. The writer uses very clear examples that will enable you to master the programming languages. It's also a useful reference for developers.

Please bear in mind that as websites are by nature populated with ever changing content, and are also often transient and quickly redesigned, many screenshots may not exactly match those seen if the link to the site is followed.

About Tom Dell'Aringa

Tom is our programming guru; he wants to teach you how to correctly use DHTML and AJAX by teaching the basics first, then covering some simple, commonly used scripting next. After that he will move on to building real world applications while making sure that the code is efficient and that the user experience is good.

During the early 1990's, Tom (http://www.pixelmech.com/) spent his days as a graphic designer for a small publishing house. His boss threw a World Wide Web book at him one day, "encouraging" him to research it. Tom fell in love and quickly made the transition from print to web technologies, teaching himself along the way. During the wild ride of the dot.com era, Tom worked for a start up, web integrator Scient and a failed small design firm that bounced his paychecks. Forever fascinated by Peanuts comic strips, animation and history, he recently completed his first mini 3d film. He holds a B.A. in Fine Art from Columbia College, Chicago.

Bring your website to live with AJAX and DHTML

Tom shows you how to use DHTML by starting with the basics first. After that he will move on with some real world applications. In the final part of this e-book tom shows you how to take control with AJAX to create amazing applications and features on your website.

Getting started with DHTML

This chapter explains the meaning and the power of DHTML and the DOM. Furthermore it demonstrates some of the features in step by step tutorials.

DHTML: Introduction To Scripting The DOM

Tom starts off by showing the meaning of the DOM and what you can do with it. Simply put, the DOM allows you to dynamically change the page all on the client side. We're not talking simple text color changes here, either. Want to add a whole table to your page? Go ahead and create one. Want to re-style every DIV on the page? Go ahead and do it. It will all happen at the click of a button, and it will all happen without sending the page back to the server. One seamless and smooth action that puts the punch in dHTML.

DHTML: Dynamic Client Side Table Sorting

In this chapter Tom shows you a powerful feature of the DOM; how to manipulate the data the client (user) has already received without making a round trip to the server.

DHTML: Dynamic Class Switching

Being able to change the look and feel of a page not only allows you to enhance the look of a page, but can produce benefits in usability as well.

The user doesn't always need to be aware of the change either. For example, I wrote about a script I wrote that took a search results table and shaded the rows with alternating colors. This produced the dual benefit of both looking nicer and making the table easier to read. This particular script ran using the onload event, and of course happens so fast the user is never aware the table rows were unstyled.

Introduction To Regular Expressions

It's time to get into regular expressions! Regular expressions can be very difficult to build, use and understand. Like anything else regular expressions are a tool (in this case, a very powerful one) that can be used to your advantage. Even if you never become adept at building the expressions themselves, you should at least be able to look at one, understand what is basically going on and make use of it in your scripts.

Dynamic "Hide and Show" Content Using The DOM

Tom demonstrates the flexibility that the DOM and CSS have brought to developers. Websites no longer have to be static, page by page linear trudges through information. And we no longer have to rely solely on the server to dynamically change page content.
One way we can use this behavior to our advantage is to tie more functionality to a page than is initially visible.

That hidden content can then react to the user based on their actions. Coupled with other DOM methods and methods like innerHTML you can develop a richer interactive interface for the user.

Real-World DHTML

This chapter shows you how to use DHTML for real world examples such as popup windows and menu's, forms and various applications.

Popup Windows: Doing Them Right

Tom addresses the right way to opening new windows. JavaScript can be somewhat picky about how you bring a new window into being, and not doing things exactly right can leave you pulling out your hair. But have no fear, Tom will show you how things work.

A Simple DHTML FlyOut Menu

If there is one thing that computer users are familiar with it is the "drop down" or "fly out" menu system. This is also sometimes referred to as a "cascading menu" system. We see this particularly in operating system interfaces.

Formatting User Form Data

Often when you have a form, the user inputs data that either doesn't match the format you desire, or it isn't as nice looking as it could be. A good example is the phone number. The typical 7 digit phone number looks like 555-1212. Quite often, people entering phone numbers on web forms omit the dashes and enter 5551212 instead.

While there isn't anything wrong with this, there is a chance that an error is harder to spot by the user.

What you could do is check the value of that phone number after it has been entered. If the dashes have been left out, you could simply add them. You could do the same thing for a 10 digit phone number or even an international phone number. Of course the more data you handle the more difficult it's going to be.

In this chapter we'll take a look at 7 and 10 digit phone numbers and how we can format (or mask) them to look nicer should the user decide not to enter dashes. While we're doing this we can introduce some handy string methods and talk about a very basic regular expression.

"Alertless" Client Side Error Messages

Tom shows you how to replace those old gray alert boxes in your forms with more colorful alert messages. Tom uses innerHTML to help you style your forms.

More Controlling Form Fields

Once you begin to use JavaScript you'll find that a lot of what you end up scripting involves dealing with forms. Form submissions, validations, dynamically displaying fields and even disabling and enabling fields. The disabling and enabling part can be confusing so we're going to cover how to deal with this subject with each type of form field: the textbox, radio button, checkbox, button and dropdown list.

The New And Improved Date Picker!
In January of 2004 Tom wrote an article called The JavaScript Date Picker. In this chapter Tom wrote a complete remake of the old Date Picker article with a new and improved code.

Spicing Up Data Tables With Highlighting Rows
This time Tom builds a script that will allow your table rows to "highlight" as the user mouses over the row. He does this using the DOM, and we'll even load the script at page load to keep things nice and clean.

An Abstract "Add Row" Class For Dynamic Table Rows (Like Yahoo!)
Tom shows you how to dynamically add rows to your page. A good example is the attach document page for Yahoo! Mail. On that page I have the choice of adding up to five attachments – but what if I want to add more?

The smart thing to do would be to give the user the option of adding more fields if they so desire. A great way would be to have a local script that instantaneously adds theform fields for the user. Tom shows you how to do this.

Take Control With AJAX

In the last six months there has been a lot of talk about AJAX, an acronym for "Asynchronous JavaScript + XML. Tom shows you how to take control of this new technology to create small but amazing applications.

Introduction To Ajax
Tom explains how to use a very popular technology in web 2.0; AJAX. The key element is the term "asynchronous" - in other words Tom shows you how to make a round trip to the server, without actually going there!

The AJAX Filmstrip
For this chapter Tom is going to build a basic version of the Windows "Filmstrip" functionality. Filmstrip is a folder view on the Windows platform that allows you to browse to multiple images.

AJAX: Ready States, PHP and Arrays
In the previous chapter we built a basic filmstrip application using Ajax. In this chapter Tom shows you how to add new images and how to improve your code with less hard coded JavaScript. He'll use PHP to figure out how many images (and what they are) on your server.

Furthermore Tom uses arrays, both in JavaScript and PHP to do some of the data handling. Lastly, he'll make use of the xmlHTTPObject's ReadyState property to keep the user informed of the progress when loading new elements into their view.

Google Maps and AJAX: Web 2.0 On Your Web Site
Tom shows you how to implement a Google Map on your web site using AJAX.

Getting started with DHTML

Getting started with DHTML

DHTML: Introduction to scripting the DOM

The DOM – What is it?

Much has been made of the DOM (an acronym for Document Object Model) in recent years, especially since the term "dHTML" (for Dynamic HTML) was introduced to the web world. If you Google "DOM" you are bound to come up thousands upon thousands of references. Web lore has it that it's an off-putting, complicated and esoteric subject. However, the heart of the DOM is pretty straightforward.

Simply put, the DOM allows you to dynamically change the page all on the client side. We're not talking simple text color changes here, either. Want to add a whole table to your page? Go ahead and create one. Want to re-style every DIV on the page? Go ahead and do it. It will all happen at the click of a button, and it will all happen without sending the page back to the server. One seamless and smooth action that puts the punch in dHTML.

The W3C defines the DOM as "The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents." What the W3C lacks in clear speech they make up for in confusing sentences. But I digress; what all this means is that the DOM gives scripts (and for our purposes of course, JavaScript) direct access (or access at all) to all the content on a given HTML page.

I really like the way Peter-Paul Koch puts it, so let's quote that:

> "The Document Object Model (DOM) is the model that describes how all elements in an HTML page, like input fields, images, paragraphs etc., are related to the topmost structure: the document itself. By calling the element by its proper DOM name, we can influence it."

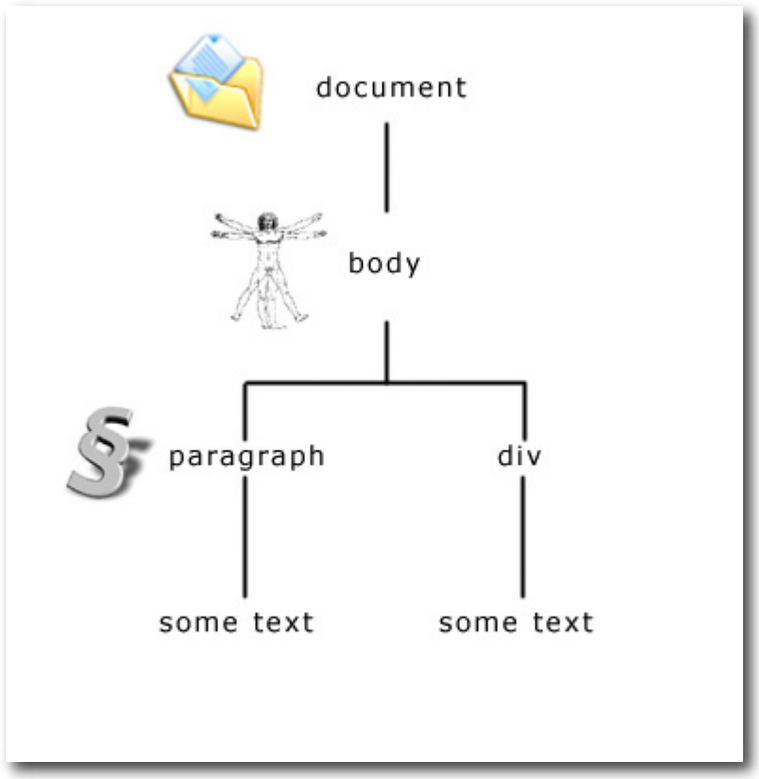
If you have been reading my articles already, then you might remember we already talked about accessing forms and form elements with the DOM level 0. I showed you how easy it is to get direct access to those elements. This same principle works for any object on your page. (Although not all access is created equal, as you will find out.) You may have guessed that there are other DOM levels (really "recommendations"). The current specification the W3C is working on is actually level 3.

To further confuse things, the DOM that we'll script, which currently has the most support among browsers right now, is the DOM level 1. We'll continue to use the parts that worked well for us in level 0 as well (such as the forms array.)

The reason we stick to the DOM level 1 is because it is the best supported among today's browsers (Netscape 6+, Mozilla 1+, Explorer 5+, Opera 5+). Currently the other DOMs available only support proprietary methods (such as document.layers from Netscape and document.all from Microsoft) or are not yet implemented.

How The DOM Works

So how does the DOM allow us to "influence" elements on a page? The DOM sees the whole document as a tree. If you think of the document as the "root" of the tree, you realize by following the root to the trunk and up to each branch, you could travel to every leaf on the tree, each leaf being an element on the page:



This is how the DOM gives you access to each element. You travel up the right branch to grab the leaf (or leaves) you want. Each leaf is technically called a "node." It was this kind of capability that really facilitated a lot of neat dHTML scripts.

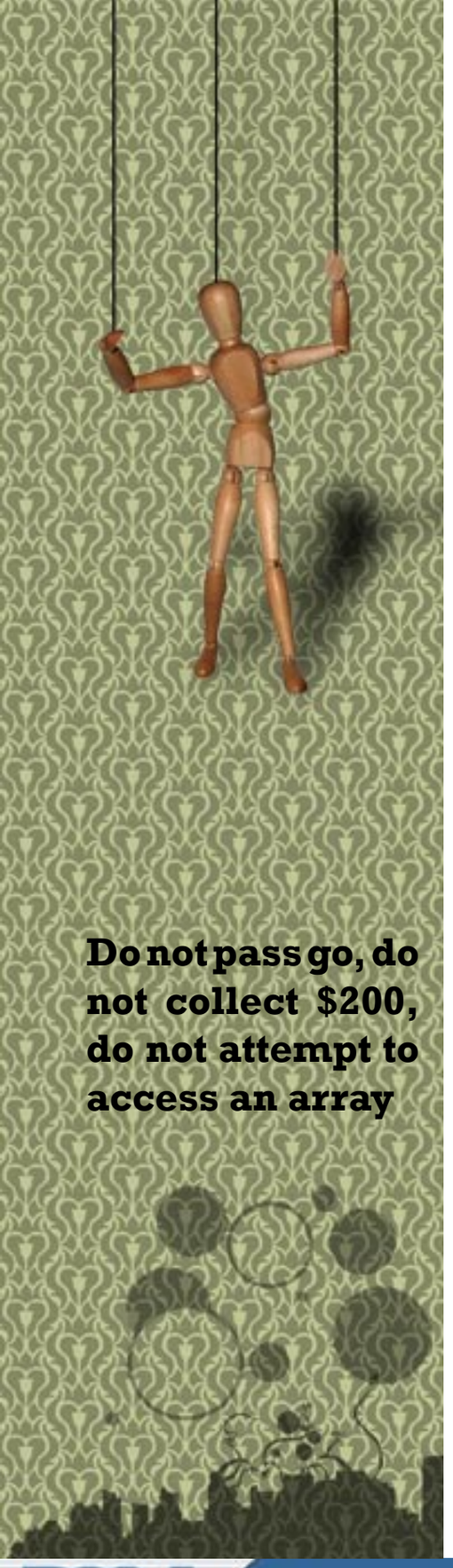
However, there are some simpler terms and methods we can use as an introduction to the DOM before we get to the tree and its more advanced features. Instead, we'll start with a couple of well known and well used methods to dynamically change our page.

document.getElementById()

This method gives you direct access to the element you desire. Do not pass go, do not collect $200, do not attempt to access an array. All you need is a unique ID attribute on your element and you are golden. Just like you assign NAME attributes to form elements, any element can have an ID attribute. The only difference is that the ID value must be unique. This is of course so you can uniquely identify any element on a page. In the case of form elements where you already have a NAME attribute attached, you can still add an ID:

```
<input type="text" name="firstName"
id="firstName" value="tom" />
```

You might remember that ID is what is used for the LABEL tag to tie the label to the form element.



Do not pass go, do not collect $200, do not attempt to access an array

Now that you have named your unique element, you can use the getElementById() method to access its properties. So in the case of the above text input:

```
var name = document.
getElementById("firstName").value;
```

The variable "name" now holds the value "tom." Note that we did not have to access the forms[]. elements[] array to get the value. For forms and their elements we actually should stick to the forms array, but I wanted to show you an example you would be familiar with.

Another example would be a DIV tag that you had styled with CSS. Let's say you have the following simple CSS rule for a box on your page:

```
#box {
   border: 1px solid #cc0000;
   background: #ffffff; }
```

the "#" signifies that "box" is the ID of the element, and is unique to the page. If I wanted to change the background of the box to black instead of white, I could do the following:

```
var myBox = document.
getElementById("box");
box.style.background = "#000000";
```

Accessing the style properties of the box (since we want to change the style) we attach a new value for the background property, and viola, new background color.

TIP:
You should know that when accessing CSS properties this way with JavaScript, which you cannot use spaces or dashes, which would seem to create a problem. How will you access the border-bottom of an element? The answer is that you simply change what is two word separated by a dash into an inter-capped word. Therefore, border-bottom becomes borderBottom, and margin-left becomes marginLeft.

getElementsByTagName()

Direct access is nice, but if you need more than one element, it gets cumbersome very quickly. What if we want to change the properties of all the rows in a table? document.getElementsByTagName(tagName) returns an array of all elements of the specified tag name. We could actually get an array of every single DIV on a page like so:

```
myDivs =  document.
getElementsByTagName("DIV");
```

If our page contained the following DIVs:

```
<div id="header">content</div>
<div id="body">content</div>
<div id="footer">content</div>
```

Then we would end up with myDivs as an array filled with those three IDs. I could then loop through that array and perform some kind of operation on each one. Let's say we wanted to give each div a top border with the properties of 1 pixel in width, and a color of black:

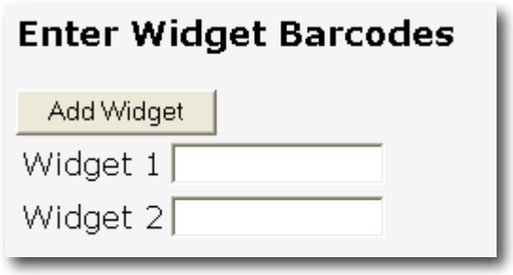
```
myDivs =
document.getElementsByTagName("DIV");

var divTotal = myDivs.length;

for(i=0; i<divTotal; i++)
{
 myDivs[i].style.borderTop = "1px solid
#000000";
}
```

This would run through ALL the DIVs on your page and give each one the border style you indicated. With these points in mind, let's move on and write a real world script.

Adding Dynamic Content

So here is our situation. We sell widgets to store owners. We have a form where the store owner has two lines to enter a barcode for the widget he wants to purchase. Then he would submit the form for processing and have his order delivered. There are two lines in which he can enter barcodes. However, what if he wants to enter more than two? He would have to reload the page again. We could offer more barcode lines in the form, but this takes up valuable space.

What we can do is write a function that dynamically adds rows to our table which has our form, and of course adds the form elements so he can add more widgets! And yes, we'll use the DOM to do this. Here is the simple version of what you'd see, without the submit button for the form or any other information:



Clicking the "Add Widget" button should give us another line to enter another widget. We aren't simply going to reveal a hidden row here, we are actually going to create a brand new one. This is the power of the DOM. We can literally rewrite the page on-the-fly, without the time needed to send the information down a wire to the server, and without the need to refresh the page to display the new fields.

Here is the HTML code for our little page:

```
<h3>Enter Widget Barcodes</h3>
<input type="button"
onclick="AddWidget();" value="Add
Widget" />
<table id="widgetTable">
<tbody id="widgetBody">
<tr>
    <td>Widget 1</td>
    <td><input type="text" size="15"
name="widget1" /></td>
</tr>
<tr>
    <td>Widget 2</td>
    <td><input type="text" size="15"
name="widget1" /></td>
</tr>
</tbody>
</table>
```

You'll notice a possibly unfamiliar tag <TBODY>. This tag is part of something called the complex table model, which is in contrast to the simple table model. All you really need to realize about this is that it gives you a finer level of control over the table, and we want that. Other than that, everything is fairly straightforward.

We've used a table for layout only to have our form nicely lined up in rows. (And were we to want to style our form, we would use CSS, not presentational markup). Although not shown, (since we aren't using the form fu nctionality for this script), you would of course have your FORM tag in such a page. It's not necessary for our script to work so we'll leave it out for simplicity's sake.

We're going to trigger our script with a simple button using the onclick method. But we're going to need to send our function some parameters.

The Script

All our script is going to need to work properly are two parameters. First, the table ID, and second, the table body ID. You'll notice in the code above that we've given each one of those elements a unique ID, "widgetTable" and "widgetBody." This is going to allow us to reference our table directly just like we did with the DIVs earlier.
The start of the script looks like this:

```
var rowNumber = 3;
function AddWidget(bodyName, tableName)
{
    var docBody = document.getElementById(bodyName);
    var tableName = document.getElementById(tableName);
```

We've set a global variable called "rowNumber" which allows us to write the correct widget number as text, as well as set the right NAME attribute on each text field. Remember a variable that is outside your function is "global" in the sense that the whole page can see it. It will hold its value after we change it inside our function.

You can see we have specified the bodyName (which is the ID for the <tbody>) and the tableName (the ID for the table itself) parameters. Below that is our call to the getElementById() method. Passing our parameters in (since they are in fact the unique Ids) gives us that direct reference to each element, which is what we want.

A New Twist

And now for something completely different:

```
var newRow = document.
createElement("TR");
var newText = document.
createTextNode("Widget " +
rowNumber);
```

Now we'll begin to see the real power of the DOM in action. Here we have two new methods of the DOM. Notice that each new method is connected to the document object. Remember our goal here is to dynamically create new content on the page, no reloading needed. We need to be able to produce new content on-the-fly. These two methods will do that for us.

createElement()

The createElement() method does just that – creates an element! What we want is new rows in our table, so we pass this method the tag of the element we want to create. In our case, it's a TR. If we wanted to create a paragraph, we would have done document. createElement("P"). Nice and simple.

createTextNode()

There's that funky word "node" again. Remember that the document is a tree and we talked about leaves. A node is simply one leaf on the tree. In this case, that node is going to be text. So, this method allows us to create some text. Logically, you pass the method a string of what you want created.

In our case, we passed "Widget" and then we concatenated the rowNumber so we'll end up labels like "Widget 3, Widget 4" and so on. Here you see our global variable rowNumber in action. If that variable had been inside our function, we would have always gotten "Widget 3" no matter how many rows we added.

Okay, that's all fine and dandy, but this new "stuff" has only been assigned to variables. How do we actually get it into the page?

Stepping Through

Since we are going to want to write a whole row, we are going to need to loop through the row and write out what we want in each table cell. (In fact, we are going to have to write out the cell first!)

We'll create a FOR loop for that:

```
for (i = 0; i < 2; i++)
   {
```

This will allow us to step through our loop 2 times (0, 1) which is what we want since we have 2 table cells. We talked about the SWITCH control structure, and we'll make use of that next:

```
switch(i)
{
   case 0:
   newCell = document.
createElement("TD");
   newCell.appendChild(newText);
   newRow.appendChild(newCell);
   break;

   case 1:
   newCell = document.
createElement("TD");
   newField = document.
createElement("INPUT");
   newField.type = "text";
   newField.name = "widget" +
rowNumber;
   newField.size = "15";
   newCell.appendChild(newField);
   newRow.appendChild(newCell);
   break;
}
```

There's a lot going on here, because this is the meat of the function. First, notice we are using our i variable for the SWITCH. We could have used an IF...ELSE here, but that would have become cumbersome had we needed more table cells. (In fact, I wrote a similar script for a real contract where there were sometimes up to 12 cells in a row, so the SWITCH was really the only way to go!)

So case 0 is the label cell (Widget #) and case 1 is the text field input. Once again you see the createElement() method, this time creating a new TD. Remember we've already created our row (theoretically, it hasn't been added to the document quite yet) and now we are working on the row's content.

appendChild()

Now we'll actually do something to the document itself. The appendChild() method actually appends (adds) some element to another. It's called append child because you are adding a child node (leaf) to a parent node (leaf). A table cell has a parent-child relationship with the table row in which it resides. The same can be said of the text within a cell. The cell is the parent and the text is the child.

A parent element/object is simply the container for the child object. If I had a DIV with some text, the DIV would be the parent and the text would be the child.

If I had a document with nothing but a paragraph, the document is the parent and the P is the child.

So we have this reference to a new cell we wanted to create. And we have a new text node we created above. All we have to do now is actually add that text node to the cell with this method:

```
newCell.appendChild(newText);
```

This places our new text into the cell we just created! Now you might have realized that we haven't actually appended this new cell to anything yet. That's why the next line is:

```
newRow.appendChild(newCell);
```

Now the cell is added to the row. You probably have further noticed that the row has not been added to anything yet! Patience, grasshopper.

The second CASE is our second cell and has a couple more items to deal with. First, we again create a new element, this time it's an INPUT. That's all fine and dandy, but what KIND of input! We'll tell the script by using the TYPE property, which you may remember we've used in previous scripts.

```
newField.type = "text";
```

We could have made it "checkbox" or any other type of input, even a button. Our text input needs a name, so we set the name property as well. Notice we added the rowNumber just like we did with the label in the previous cell. We also set the size to 15 to make sure it is the same size as our existing inputs.

And what is left to do but append them!

```
newCell.appendChild(newField);
newRow.appendChild(newCell);
```

The text input is now added to the cell, and the cell to the row. We're almost there.

Wrapping it Up

All we need to do now is attach our row to the table.

```
rowNumber += 1;
docBody.appendChild(newRow)
```

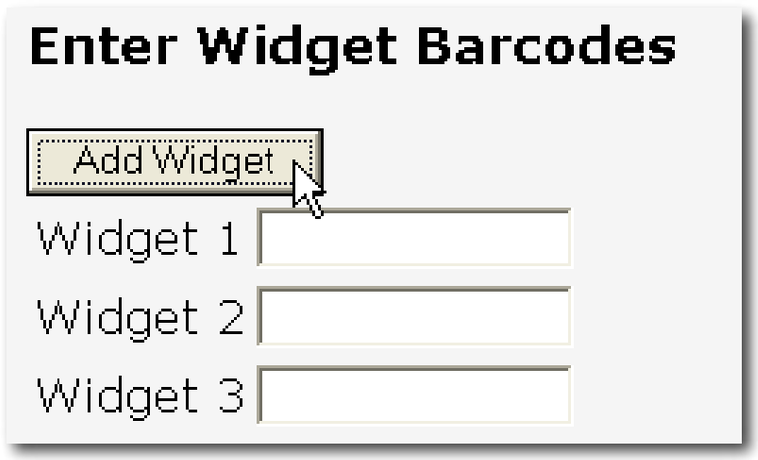
Notice we have to attach the row to the document body tag (<tbody>), not the table itself. This was the reason why we included it in the HTML and gave it an ID. I guarantee the reasons why this is the case would be a discussion to put you to sleep.

We've also incremented our rowNumber variable (since we just added a row) with a fancy-schmancy shortcut.

It means the same thing as:

```
rowNumber = rowNumber + 1;
```

Therefore, clicking the button should give you:



And you have just dynamically created a row and its content all on the client side!

Here is the full function:

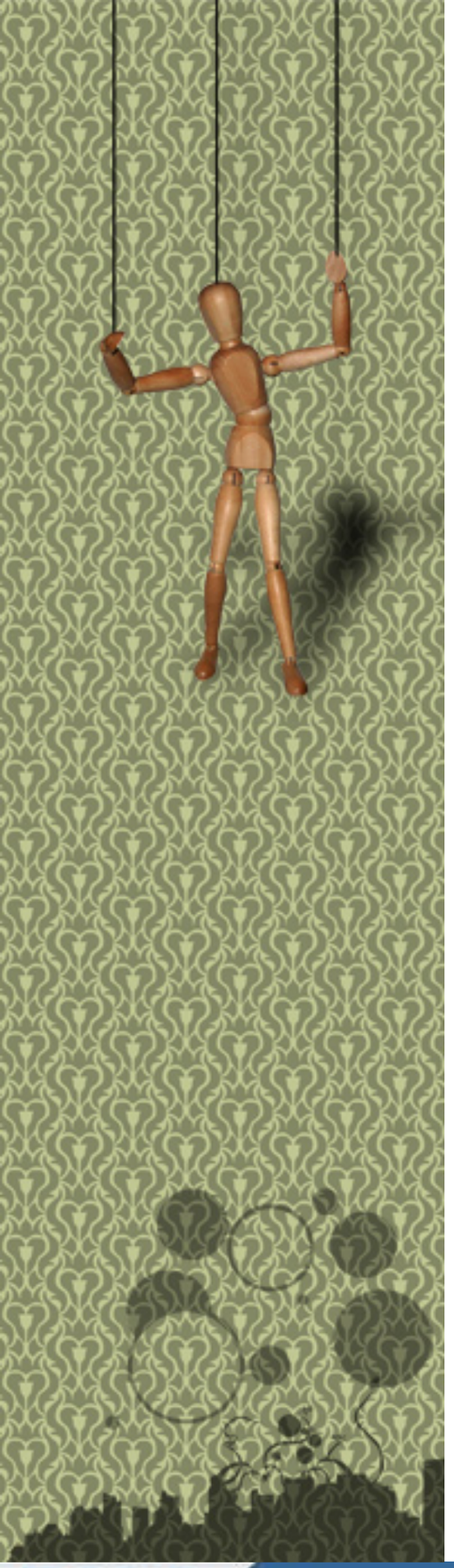
```
var rowNumber = 3;
function AddWidget(bodyName, tableName)
{
    var docBody = document.
getElementById(bodyName);
    var tableName = document.
getElementById(tableName);
    var newRow = document.
createElement("TR");
    var newText = document.
createTextNode("Widget " + rowNumber);
}
```

Conclusion

We've just merely scratched the surface of what is possible with the DOM. When you think about the fact that you have access to every element on a page, and that you can modify each element, create and remove elements at your whim, you realize there is great power to be had.

But like Spiderman's Uncle Ben said: "With great power comes great responsibility." Well, maybe not in this case. At any rate, you'll find that the DOM isn't quite the panacea it appears to be. There are certainly some browser issues to face as not all browsers implement the DOM level 1 quite exactly the same. Peter-Paul Koch's resource on the DOM and what is supported is invaluable in this case.

See if you can change the script and HTML to add new cells and content and see what you come up with!



About DMXzone

The History of DMXzone

DMXzone was founded in Feb 2001 by George Petrov. It was then called UDzone after the Macromedia product UltraDev that preceded Dreamweaver MX. By April 2001 we'd already been asked by Macromedia to speak at the Macromedia UCON 2001 conference in New York. Since then, we've grown to over 300,000 registered members of all levels and locations, who come together to share knowledge and learn from each other. We are an independent community and are in no way connected with Macromedia, the makers of Dreamweaver MX.

In May 2003, we launched our very successful Premium Tutorials track, publishing professionally written tutorials by a team of authors for an affordable price every day, as we ourselves were tired of shelling out lots of money for computer books full of redundancy and newbie's explanation. This premium track runs alongside the free content submitted by members.

Introduction

This e-book will teach you how to use DHTML and AJAX to create amazing new applications and interactive features for your website. Tom Dell 'Aringa shows you how to create Popup Windows, Fly out menus, Filmstrips, advanced form features and much more!

Ajax enables you to write interactive applications while reducing the amount of data interchanged between the web browser and web server. This results in shorter processing times so you can give your user the ultimate web experience.

DHTML allows you to create amazing effects and allows you to enhance the interactivity of your webpage without using any Plug-ins. DHTML files are also smaller then HTML files thus speeding up your website even further.

What this book does, and who it's for.

This book is for anyone with an interest in developing their DHTML and AJAX skills. The writer uses very clear examples that will enable you to master the programming languages. It's also a useful reference for developers.

About Tom Dell'Aringa

Tom is our programming guru; he wants to teach you how to correctly use DHTML and AJAX by teaching the basics first, then covering some simple, commonly used scripting next. After that he will move on to building real world applications while making sure that the code is efficient and that the user experience is good.
During the early 1990's, Tom (http://www.pixelmech.com/) spent his days as a graphic designer for a small publishing house. His boss threw a World Wide Web book at him one day, "encouraging" him to research it. Tom fell in love and quickly made the transition from print to web technologies, teaching himself along the way.

Tom Dell'Aringa

DMX zone

DMX zone e-book