# Mastering Ajax, Part 6: Build DOM-based Web applications

## Mix the DOM and JavaScript -- those perfect Ajax companions -- to change a Web page's user interface without page reloads

Skill Level: Intermediate

Brett McLaughlin
Author and Editor
O'Reilly Media Inc.

12 Sep 2006

Combine the Document Object Model (DOM) with JavaScript code to create interactive Ajax applications. In previous articles in this series, you examined the concepts involved in DOM programming -- how the Web browser views a Web page as a tree -- and you should now understand the programming structures used in the DOM. In this article, you put all of this knowledge into practice and build a simple Web page that has some nice effects, all created using JavaScript to manipulate the DOM, without ever reloading or refreshing the page.

You've had two full articles of introduction to the Document Object Model, or DOM; you should be pretty comfortable with how the DOM works by now. (See Resources for links to the first two DOM articles, as well as to earlier articles in the Ajax series.) In this article, you'll put that understanding into practice. You'll develop a basic Web application with a user interface that changes based on user actions -- of course, you'll use the DOM to handle changing that interface. By the time you're finished with this article, you'll have put most of the techniques and concepts you've learned about the DOM into practice.

I assume that you've followed along for the last two articles; if you haven't, review them to get a solid understanding of the DOM and of how Web browsers turn the HTML and CSS you supply them into a single tree structure that represents a Web page. All of the DOM principles that I've talked about so far will be used in this article to build a working -- albeit somewhat simple -- DOM-based dynamic Web page. If at any point in this article you get stuck, you can simply stop and review the earlier two articles, and come back.

## Getting started with the sample application

**A note on the code**

To keep the focus specifically on the DOM and JavaScript code, I've been somewhat sloppy and have written HTML with inline style (like the `align` attribute on the `h1` and `p` elements, for example). While this is acceptable for trying things out, I recommend that you take the time to put all your styles into external CSS stylesheets for any production applications you develop.

Let's start out by putting together a very basic application, and then adding a little DOM magic. In keeping with the idea that the DOM can move things around on a Web page without submitting a form -- thus making it a perfect companion for Ajax -- let's build a simple page that shows nothing but a plain old top hat, and a button labeled **Hocus Pocus!** (Can you guess where this will go?)

### The initial HTML

Listing 1 shows the HTML for this page; it's just a body with a heading and a form, along with a simple image and a button that you can push.

### Listing 1. The HTML for the sample application

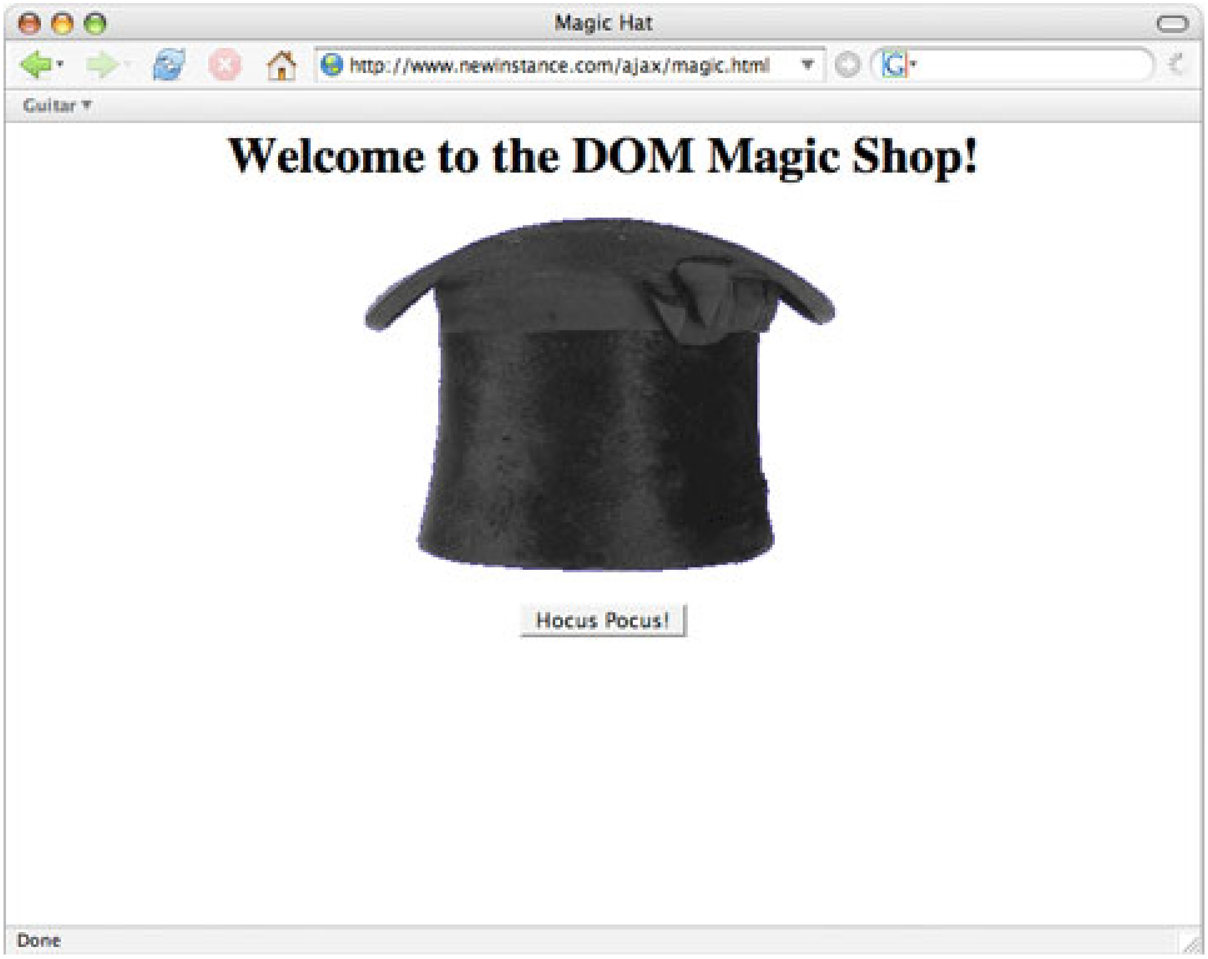
```
<html>
 <head>
  <title>Magic Hat</title>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" />
   </p>
  </form>
 </body>
</html>
```

You can find this HTML, along with all the images used in this article, available in Downloads at the end of this article. However, I highly recommend that you download just the images and then, as I build up the application throughout the article, type in the HTML by hand as you walk through the sample. You'll get a much better understanding of the DOM code in this way than you might by just reading the article and then simply opening up the completed application.

### Viewing the sample Web page

Nothing particularly tricky here; open up the page and you should see something like Figure 1.

**Figure 1. A rather boring-looking top hat**


Magic Hat
http://www.newinstance.com/ajax/magic.html
Guitar
Welcome to the DOM Magic Shop!
Hocus Pocus!
Done

### A final point about the HTML

One important point that you *should* note, though, is that the button on the form in Listing 1 and Figure 1 is of type `button`, and is not a submit button. If you do use a submit button, pressing the button will cause the browser to try and submit the form; of course, the form has no `action` attribute (which is entirely intentional), so this would just create an infinite loop of no activity. (You should try this on your own to see what happens.) By using a normal input button -- and avoiding the submit button -- you can connect a JavaScript function to the button and interact with the browser *without* submitting the form.

## Adding more to the sample application

Now, spruce up the Web page with some JavaScript, DOM manipulatin, and a little image wizardry.

## Using the getElementById() function

Obviously, a magic hat isn't worth much without a rabbit. In this case, begin by replacing the image in the existing page (refer back to Figure 1) with an image of a rabbit, like that shown in Figure 2.

**Figure 2. The same top hat, this time with a rabbit**

The first step in making this bit of DOM trickery occur involves looking up the DOM node that represents the `img` element in the Web page. Generally, the easiest way to do this is to use the `getElementById()` method, available on the `document` object that represents the Web page. You've seen this method before; it functions like this:

```
var elementNode = document.getElementById("id-of-element");
```

### Adding an id attribute to the HTML

This is pretty basic JavaScript, but it requires a bit of work in your HTML: the addition of an `id` attribute to the element that you want to access. That's the `img` element you want to replace (with a new image containing the rabbit); so you need to change our HTML to look like Listing 2.

**Listing 2. Adding an id attribute**

```
<html>
 <head>
  <title>Magic Hat</title>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" id="topHat" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" />
   </p>
  </form>
 </body>
</html>
```

If you reload (or reopen) the page, you'll see nothing different at all; the addition of an `id` attribute has no visual effect on a Web page. It does, however, make it possible to work with an element more easily using JavaScript and the DOM.

### Grabbing the img element

Now you can use `getElementById()` easily. You have the ID of the element you want -- `topHat` -- and can store that in a new JavaScript variable. Add in the code shown in Listing 3 to your HTML page.

**Listing 3. Getting access to the img element**

```
<html>
 <head>
  <title>Magic Hat</title>
  <script language="JavaScript">
    function showRabbit() {
      var hatImage = document.getElementById("topHat");
    }
  </script>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" id="topHat" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" />
   </p>
  </form>
 </body>
</html>
```

Again, loading or reloading the Web page at this point won't show anything exciting. Even though you now have access to the image, you haven't done anything with it.

## Changing the image, the hard way

You can make the change you want in two ways: the hard way and the easy way. Like all good programmers, I typically prefer the easy way; however, walking through the longer path is a great DOM exercise, and well worth your time. Look at how to change out the image the hard way first; later, you'll re-examine things to see how to make the same change in an easier way.

Here's what you need to do to replace the existing image with the newer, rabbit-inclusive image:

1. Create a new `img` element.
2. Get access to the element that is the parent -- that is, the container -- of the current `img` element.
3. Insert the new `img` element as a child of the container just *before* the existing `img` element.
4. Remove the old `img` element.
5. Set things up so that the JavaScript function you've created is called when a user clicks the **Hocus Pocus!** button.

### Creating a new img element

You should remember from my last two articles that the key to almost everything in the DOM is the `document` object. It represents an entire Web page, allows you access to powerful methods like `getElementById()`, and also allows you to create new nodes. It's this last property you want to use now.

Specifically, you need to create a new `img` element. Remember, in the DOM, everything is a node, but nodes are broken up further into three basic groups:

- Elements
- Attributes
- Text nodes

There are other groups, but these three will serve you for about 99 percent of your programming needs. In this case,you want a new element, of type `img`. So you need this code in your JavaScript:

```
var newImage = document.createElement("img");
```

This will create a new node, of type `element`, with the element name `img`. In HTML, that's essentially this:

```
<img />
```

Remember, the DOM will create well-formed HTML, meaning that the element is currently empty, with both a starting and ending tag. All that's left is to add content or attributes to this element, and then insert it into the Web page.

As for content, the `img` element is an empty element. However, you do need to add an attribute: the `src` attribute, which specifies the image to load. You might think that you need to use a method like `addAttribute()` here, but you would be incorrect. The DOM specification creators figured that as programmers, we might like a little shortcut (and we do!), so they created a single method to both add new attributes and change the value of existing ones: `setAttribute()`.

If you call `setAttribute()` and supply an existing attribute, its value is changed to the value you supply. However, if you call `setAttribute()` and supply an attribute that does *not* exist, the DOM quietly adds the attribute, using the value you provide. One method, two purposes! So you need to add the following to your JavaScript:

```
var newImage = document.createElement("img");
newImage.setAttribute("src", "rabbit-hat.gif");
```

This creates the image, and sets its source up appropriately. At this point, your HTML should look like Listing 4.

**Listing 4. Creating a new image using the DOM**

```
<html>
 <head>
```

```
  <title>Magic Hat</title>
  <script language="JavaScript">
    function showRabbit() {
      var hatImage = document.getElementById("topHat");
      var newImage = document.createElement("img");
      newImage.setAttribute("src", "rabbit-hat.gif");
    }
  </script>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" id="topHat" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" />
   </p>
  </form>
 </body>
</html>
```

You can load this page, but don't expect any action; you haven't really done anything that affects the actual Web page yet. Plus, if you look back up at step 5 in the list of things to do, you might notice that your JavaScript function isn't even getting called yet!

### Getting the parent of the original image

Now that you have an image ready to insert, you need somewhere to insert it. However, you're not inserting it into the existing image; instead, you want to put it before the existing image, and then remove the existing image. To do that, you need the parent of the existing image, which is really the key to all this inserting and removing.

You should recall from the earlier articles that the DOM really sees a Web page as a tree, a hierarchy of nodes. Every node has a parent (a node higher up the tree that it is a child of), and possibly some children of its own. In the case of this image, there are no children -- remember, images are empty elements -- but there is certainly a parent. You don't even care what the parent is; you just need to access it.

To do that, you can just use the `parentNode` property that every DOM node has, like this:

```
var imgParent = hatImage.parentNode;
```

It's really that simple! You know that the parent can have children, because it already has one: the old image. Beyond that, you really don't need to know if the parent is a `div`, or a p, or even the `body` of the page; it just doesn't matter!

### Inserting the new image

Now that you have the parent of the old image, you can insert the new image. That's fairly easy, as you can use several methods to add a child:

- `insertBefore(newNode, oldNode)`
- `appendChild(newNode)`

Since you want the new image to appear exactly where the old image is, you need `insertBefore()` (andyou'll also need to use the `removeChild()` method as well). Here's the line of JavaScript you'll use to insert the new image element just before the existing image:

```
var imgParent = hatImage.parentNode;
imgParent.insertBefore(newImage, hatImage);
```

At this point, the parent of the old image has *two* child images: the new image, immediately followed by the old image. It's important to note here that the content *around* these images is unchanged, and that the order of that content is exactly the same as it was before the insertion. It's just that the parent now has one additional child -- the new image -- directly before the old image.

### Removing the old image

Now all that you need to do is remove the old image; you only want the new image in the Web page. This is simple, as you already have the old image element's parent. You can just call `removeChild()` and pass in the node you want to remove, like this:

```
var imgParent = hatImage.parentNode;
imgParent.insertBefore(newImage, hatImage);
imgParent.removeChild(hatImage);
```

At this point, you essentially replaced the old image with the new one. Your HTML should look like Listing 5.

### Listing 5. Replacing the old image with the new

```
<html>
 <head>
  <title>Magic Hat</title>
  <script language="JavaScript">
    function showRabbit() {
      var hatImage = document.getElementById("topHat");
      var newImage = document.createElement("img");
      newImage.setAttribute("src", "rabbit-hat.gif");
      var imgParent = hatImage.parentNode;
      imgParent.insertBefore(newImage, hatImage);
      imgParent.removeChild(hatImage);
    }
```

```
   </script>
  </head>

  <body>
   <h1 align="center">Welcome to the DOM Magic Shop!</h1>
   <form name="magic-hat">
    <p align="center">
     <img src="topHat.gif" id="topHat" />
     <br /><br />
     <input type="button" value="Hocus Pocus!" />
    </p>
   </form>
  </body>
</html>
```

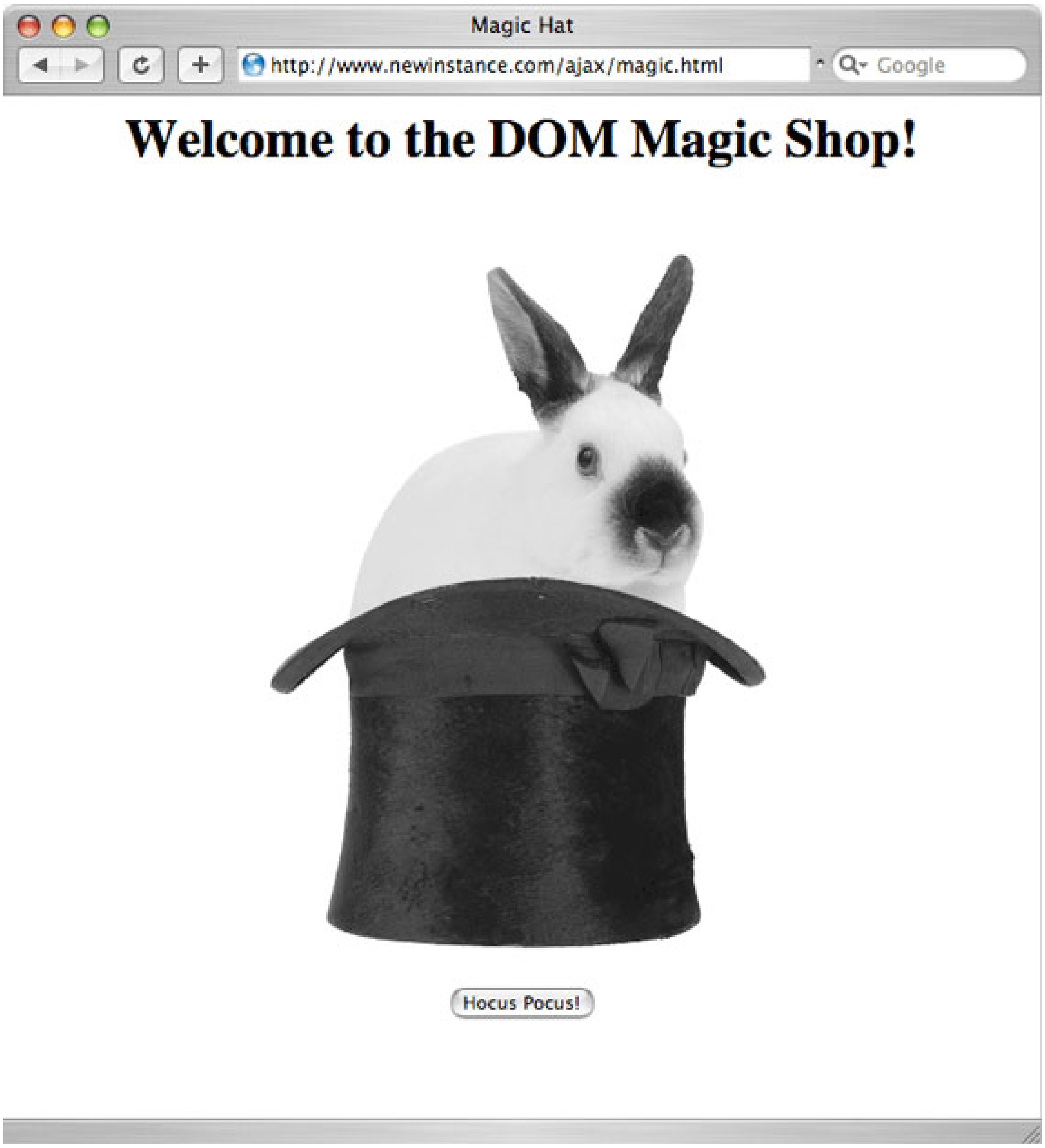
### Connecting the JavaScript

The last step -- and perhaps the easiest -- is to connect your HTML form to the JavaScript function you just wrote. You want the `showRabbit()` function to run every time a user clicks the **Hocus Pocus!** button. To accomplish this, just add a simple `onClick` event handler to your HTML:

```
<input type="button" value="Hocus Pocus!" onClick="showRabbit();" />
```

At this point in your JavaScript programming, this should be pretty routine. Add this into your HTML page, save the page, and then load the page into your Web browser. The page should initially look like Figure 1; click **Hocus Pocus!**, though, and you should get a result that looks like Figure 3.

**Figure 3. The rabbit has come out to play**


Magic Hat
http://www.newinstance.com/ajax/magic.html
Google
Welcome to the DOM Magic Shop!
Hocus Pocus!

## Changing the image, the slightly easier way

If you look back over the steps you took to change out the image, and then review the various methods available on nodes, you might notice a method called `replaceNode()`. This allows you to replace one node with another. Consider again the steps you took:

1. Create a new `img` element.
2. Get access to the element that is the parent -- that is, the container -- of the current `img` element.
3. Insert the new `img` element as a child of the container just *before* the existing `img` element.
4. Remove the old `img` element.
5. Set things up so that the JavaScript function you've created is called when a user clicks **Hocus Pocus!**.

With `replaceNode()`, you can now reduce the number of stepsyou need to take. You can combine steps 3 and 4 so the process looks like this:

1. Create a new `img` element.
2. Get access to the element that is the parent -- that is, the container -- of the current `img` element.
3. Replace the old `img` element with the new one you just created.
4. Set things up so that the JavaScript function you created is called when a user clicks **Hocus Pocus!**.

This may not seem like a huge deal, but it certainly simplifies your code. Listing 6 shows how you can make this change: by removing the `insertBefore()` and `removeChild()` method calls.

### Listing 6. Replacing the old image with the new (in one step)

```
<html>
 <head>
  <title>Magic Hat</title>
  <script language="JavaScript">
    function showRabbit() {
      var hatImage = document.getElementById("topHat");
      var newImage = document.createElement("img");
      newImage.setAttribute("src", "rabbit-hat.gif");
      var imgParent = hatImage.parentNode;
      imgParent.replaceChild(newImage, hatImage);
    }
  </script>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" id="topHat" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" onClick="showRabbit();"  />
   </p>
  </form>
 </body>
</html>
```

Again, this isn't a big change, but it does illustrate a rather important point in DOM coding: you can usually find a few ways to perform any given task. Many times, you can cut what takes four or five steps down to two or three if you carefully review the DOM methods available to you, and see if there are perhaps shorter ways to accomplish a task.

## Changing the image, the (really) easy way

Since I've pointed out that there is almost always an easier way to perform a task, I'll now show you that there's a *much* easier way to replace the top hat image with the rabbit image. Did you figure out what that approach is as you worked through this article? Here's a hint: it has to do with attributes.

Remember, the image element is largely controlled by its `src` attribute, which refers to a file somewhere (either a local URI or an external URL). So far, you've replaced the image node with a new image; however, it's much simpler to just *change the `src` attribute of the existing image!* This consolidates all the work of creating a new node, finding the parent, and replacing the old node into a single step:

```
hatImage.setAttribute("src", "rabbit-hat.gif");
```

That's all it takes! Look at Listing 7, which shows this solution in the context of an entire Web page.

**Listing 7. Changing the src attribute**

```
<html>
 <head>
  <title>Magic Hat</title>
  <script language="JavaScript">
    function showRabbit() {
      var hatImage = document.getElementById("topHat");
      hatImage.setAttribute("src", "rabbit-hat.gif");
    }
  </script>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" id="topHat" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" onClick="showRabbit();"  />
   </p>
  </form>
 </body>
</html>
```

This is one of the coolest thing about the DOM: when you update an attribute, the Web page immediately changes. As soon as the image points to a new file, the browser loads that file, and the page is updated. No reloading is necessary, and you don't even need to create a new image element! The result is still identical to Figure 3, shown above -- it's just that the code is far simpler.

## Hiding that rabbit

Currently, the Web page is pretty nifty, but still a bit primitive. Even though the rabbit pops out of the hat, the button on the bottom of the screen still reads **Hocus Pocus!** once he's out, and still calls `showRabbit()`. This means that if you click on the button after the rabbit is revealed, you waste processing time. More importantly, it's just plain useless, and useless buttons are never a good thing. Let's see if you can use the DOM to make a few more changes, and make that button useful whether the rabbit is in the hat or out of it.

### Changing the button's label

The easiest thing to do is to change the label of the button after a user clicks it. That way, it doesn't seem to indicate that anything else magical will occur; the worst thing a Web page can do is imply something to the user that's not correct. Before you can change the button's label, though, you need to get access to the node; and before you can do that, you need an ID to reference the button by. This should be old hat by now, right? Listing 8 adds an `id` attribute to the button.

**Listing 8. Adding an id attribute**

```
<html>
 <head>
  <title>Magic Hat</title>
  <script language="JavaScript">
    function showRabbit() {
      var hatImage = document.getElementById("topHat");
      hatImage.setAttribute("src", "rabbit-hat.gif");
    }
  </script>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" id="topHat" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" id="hocusPocus"
           onClick="showRabbit();" />
   </p>
```

```
    </form>
  </body>
</html>
```

Now it's trivial to access the button in your JavaScript:

```
function showRabbit() {
  var hatImage = document.getElementById("topHat");
  hatImage.setAttribute("src", "rabbit-hat.gif");
  var button = document.getElementById("hocusPocus");
}
```

Of course, you probably already typed in the next line of JavaScript, to change the value of the button's label. Again, `setAttribute()` comes into play:

```
function showRabbit() {
  var hatImage = document.getElementById("topHat");
  hatImage.setAttribute("src", "rabbit-hat.gif");
  var button = document.getElementById("hocusPocus");
  button.setAttribute("value", "Get back in that hat!");
}
```

With this simple bit of DOM manipulation, the button's label will change as soon as the rabbit pops out. At this point, your HTML and completed `showRabbit()` function should look like Listing 9.

### Listing 9. The completed (for now) Web page

```
<html>
 <head>
  <title>Magic Hat</title>
  <script language="JavaScript">
    function showRabbit() {
      var hatImage = document.getElementById("topHat");
      hatImage.setAttribute("src", "rabbit-hat.gif");
      button.setAttribute("value", "Get back in that hat!");
    }
  </script>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" id="topHat" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" id="hocusPocus"
           onClick="showRabbit();" />
   </p>
  </form>
 </body>
</html>
```

### Putting the rabbit back

As you might guess from the new button label, it's time to get the rabbit back into the hat. This essentially just reverses the process by which you got him out: changing the image's `src` attribute back to the old image. Create a new JavaScript function to do this:

```
function hideRabbit() {
  var hatImage = document.getElementById("topHat");
  hatImage.setAttribute("src", "topHat.gif");
  var button = document.getElementById("hocusPocus");
  button.setAttribute("value", "Hocus Pocus!");
}
```

This is really just a matter of reversing everything that the `showRabbit()` function does. Set the image to the old, rabbit-free top hat, grab the button, and change its label back to **Hocus Pocus!**

## Dealing with event handlers

The sample application at this point has one big problem: even though the *label* of the button changes, the action that occurs when you click that button does *not.* Fortunately, you can change the event -- the action that occurs -- when a user clicks the button using the DOM. So if the button reads **Get back in that hat!,** you want it to run `hideRabbit()` when it's clicked. Conversely, once the rabbit is hidden, the button returns to running `showRabbit()`.

**Avoid addEventHandler()**
In addition to the `onclick` property, there's a method intended to be used to add an event handler, like `onClick` or `onBlur`; it's unsurprisingly called `addEventHandler()`. Unfortunately, Microsoft™ Internet Explorer™ doesn't support this method, so if you use it in your JavaScript, millions of Internet Explorer users are going to get nothing from your page other than an error (and probably some ideas about complaints). Avoid this method; you can get the same results with the approach in this article, which *does* work on Internet Explorer.

If you look at the HTML, you'll see that the event you deal with here is `onClick`. In JavaScript, you can reference this event using the `onclick` property of a button. (Note that in HTML, the property is generally referred to as `onClick`, with a capital *C,* and in JavaScript, it's `onclick`, all lowercase.) So you can change the event that's run on a button: you just assign a new function to the `onclick` property.

But there's a slight twist: the `onclick` property wants to be fed a function reference -- not the string name of a function, but a reference to the function itself. In JavaScript, you can reference a function by its name, without any parentheses. So you might change the function that runs when a button is clicked like this:

```
button.onclick = myFunction;
```

In your HTML, then, making this change is pretty simple. Check out Listing 10, which toggles the functions that the button runs.

**Listing 10. Changing the button's onClick function**

```
<html>
 <head>
  <title>Magic Hat</title>
  <script language="JavaScript">
    function showRabbit() {
      var hatImage = document.getElementById("topHat");
      hatImage.setAttribute("src", "rabbit-hat.gif");
      var button = document.getElementById("hocusPocus");
      button.setAttribute("value", "Get back in that hat!");
      button.onclick = hideRabbit;
    }

    function hideRabbit() {
      var hatImage = document.getElementById("topHat");
      hatImage.setAttribute("src", "topHat.gif");
      var button = document.getElementById("hocusPocus");
      button.setAttribute("value", "Hocus Pocus!");
      button.onclick = showRabbit;
    }
  </script>
 </head>

 <body>
  <h1 align="center">Welcome to the DOM Magic Shop!</h1>
  <form name="magic-hat">
   <p align="center">
    <img src="topHat.gif" id="topHat" />
    <br /><br />
    <input type="button" value="Hocus Pocus!" id="hocusPocus"
           onClick="showRabbit();" />
   </p>
  </form>
 </body>
</html>
```

This is a complete, ready-to-use DOM application. Try it out and see for yourself!

## In conclusion

At this point, you should be pretty comfortable with the DOM. In previous articles, you saw the basic concepts involved in working with the DOM, and got a detailed look at the API; now you've worked through a simple DOM-based application. Be sure to take your time with this article, and try it out online for yourself.

Although this is the last of the articles in this series specifically focusing on the Document Object Model, it's not the last you'll see of the DOM. In fact, you'll be hard-pressed to do much in the Ajax and JavaScript worlds without using the DOM to at least some degree. Whether you create complex highlighting and movement effects, or just work with text blocks or images, the DOM gives you access to a Web page in a really easy-to-use manner.

If you still feel a little unsure of how to use the DOM on your own, take some time to revisit this set of three articles; the rest of this series will use the DOM without much explanation, and you don't want to get lost in those details and miss out on important information about other concepts, such as XML or JSON. Be sure you're comfortable using the DOM, write a few DOM-based applications on your own, and you'll be ready to tackle some of the data format issues I'll be discussing over the next several months.

## Downloads

| Description | Name | Size | Download method |
|---|---|---|---|
| Example graphics only | wa-ajaxintro6/ajax_6-images_download.zip | 91 KB | HTTP |
| Complete example, including HTML and graphics | wa-ajaxintro6/ajax_6-complete_examples.zip | 93 KB | HTTP |

Information about download methods

## Resources

### Learn

- Exploiting DOM for Web response and Advanced requests and responses in Ajax: Read the first and second artices in this series on DOM programming, or see the entire Mastering Ajax series.
- "Build dynamic Java applications" (Philip McCarthy, developerWorks, September 2005): Take a look at Ajax from the server side, using a Java™ perspective.
- "Java object serialization for Ajax" (Philip McCarthy, developerWorks, October 2005): Examine how to send objects over the network, and interact with Ajax, from a Java perspective.
- "Call SOAP Web services with Ajax" (James Snell, developerWorks, October 2005): Dig into this fairly advanced article on integrating Ajax with existing SOAP-based Web services; it shows you how to implement a Web browser-based SOAP Web services client using the Ajax design pattern.
- The DOM Home Page at the World Wide Web Consortium: Visit the starting place for all things DOM-related.
- The DOM Level 3 Core Specification: Define the core Document Object Model, from the available types and properties to the usage of the DOM from various languages.
- The ECMAScript language bindings for DOM: If you're a JavaScript programmer and want to use the DOM from your code, this appendix to the Level 3 Document Object Model Core definitions will interest you.
- "Ajax: A new approach to Web applications" (Jesse James Garrett , *Adaptive Path,* February 2005): Read the article that coined the Ajax moniker -- it's required reading for all Ajax developers.
- developerWorks Web architecture zone: Expand your Web-building skills with articles, tutorials, forums, and more.
- developerWorks technical events and Webcasts: Stay current with these software briefings for technical developers.

### Get products and technologies

- *Head Rush Ajax,* Brett McLaughlin (O'Reilly Media, 2006): Load the ideas in this article into your brain, Head First style.
- *Java and XML, Second Edition* (Brett McLaughlin, O'Reilly Media, Inc., 2001): Check out the author's discussion of XHTML and XML transformations.
- *JavaScript: The Definitive Guide* (David Flanagan, O'Reilly Media, Inc., 2001):

Dig into extensive instruction on working with JavaScript and dynamic Web pages. The upcoming edition adds two chapters on Ajax.

- *Head First HTML with CSS & XHTML* (Elizabeth and Eric Freeman, O'Reilly Media, Inc., 2005): Learn more about standardized HTML and XHTML, and how to apply CSS to HTML.
- IBM trial software: Build your next development project with software available for download directly from developerWorks.

**Discuss**

- developerWorks blogs: Get involved in the developerWorks community.
- Ajax forum on developerWorks: Learn, discuss, share in this forum of Web developers just learning or actively using AJAX.

## About the author

Brett McLaughlin

Brett McLaughlin has worked in computers since the Logo days. (Remember the little triangle?) In recent years, he's become one of the most well-known authors and programmers in the Java and XML communities. He's worked for Nextel Communications, implementing complex enterprise systems; at Lutris Technologies, actually writing application servers; and most recently at O'Reilly Media, Inc., where he continues to write and edit books that matter. Brett's upcoming book, *Head Rush Ajax*, brings the award-winning and innovative Head First approach to Ajax. His last book, *Java 1.5 Tiger: A Developer's Notebook*, was the first book available on the newest version of Java technology. And his classic *Java and XML* remains one of the definitive works on using XML technologies in the Java language.