Working with cookies

By Cal Henderson

2009-05-11: This article was never completed, but numerous guides to handling cookies in JavaScript have been written since. Head to Google!

Cookies are usful for lots of things - tracking user logins, remembering where users have been, etc. Cookies work by the web server sending "Set-Cookie" HTTP headers to the user agent. Whenever the user agent requests a page from a server whose cookies it's carrying, it sends back "Cookie" HTTP headers to the server. This allows server-side languages to customsie the output of a page depending upon the cookie values.

But it's not just server-side languages that can manipulate cookies - you can work with them client-side aswell. It's a bit tricky to understand at first, because with client-side scripting, the reading a writing of cookies is done *after the page has loaded*. This means that when you set a cookie using javascript, the user agent will pass that cookie back to the server during subsequent requests (until the cookie expires).

Unfortunately, the javascript interface to cookies is almost non-existent. It consists solely of the *document.cookie* property, which behaves quite strangely. When you read it, it returns a list of key/value pairs: `key=value;key=value;key=value`

```
e.g.
visits=11;time=48324984;user_id=1284
```

But when you write to it, you write a single key/value pair with (optional) extra parameters:

```
key=value;expires=date;path=path

e.g.
visits=12;expires=Fri, 5 Jul 2002 15:26:35 UTC;path=/
```

Because Javascript provides no functions to get and set cookies, it's a good idea to add these routines to your own Javascript library. So let's start out by learning to read cookies. We'll make a simple function whoch we can call to display all the cookies associated with the page.

```
function show_all_cookies(){
 var cookie = document.cookie;

 // split it into key-value pairs
 var cookie_pieces = cookie.split(';');

 // for each of those pairs, split into key and value
 for(var i=0; i<cookie_pieces.length; i++){

  // get the cookie piece and trim it
  var piece = trim(cookie_pieces[i]);

  // find the location of the '=' and split the string
  var a = piece.indexOf('=');
  if (a == -1){
   // there was no '=' - so we have a key and no value
   var key = piece;
   var value = '';
```

```
   }else{
    // we found an '=' - split the string in two
    var key = piece.substr(0,a);
    var value = piece.substr(a+1);
   }

   // now display our cookies
   alert('Key: ' + key + "  Value : "+ value);
  }
 }
```

The function is quite straight forward - we take the cookie variable and split it into pieces using the *String.split()* method. We then loop for each of the cookie pieces, to parse and display them. First we trim any spaces from the beginning and end of the cookie because some browsers add spaces into the cookie data. Once the data has been trimmed, we use the *String.indexOf()* method to find the first equals sign ('=') in the string. If we don't find one, which shouldn't happen, we recover gracefully by taking the whole string to be the key. If we do find one, we use the *String.substr()* method to extract the key and the value. These are then displayed by concatenating them into a single string which is displayed using the *window.alert()* method.

The trim routine is fairly simple and should be a part of any serious Javascript programmer's library. It loops removing spaces from the beginning of the string, then from the end, and finally returns the trimmed string.

```
 function trim(str){

  // trim off leading spaces
  while (str.charAt(0) == ' '){
   str = str.substring(1);
  }

  //trim off trailing spaces
  while (str.charAt(str.length-1) == ' '){
   str = str.substring(0,str.length-1);
  }

  return str;
 }
```

Now that we've got a routine to read our cookies, we should set some to test it. Something like this will suffice:

```
 document.cookie = "bob=11";
document.cookie = "jack=12";
show_all_cookies();
```

Sure enough, this code will display two alert boxes, one for each cookie. But didn't I say before that there are several parameters for writing cookies? Well there are, but they are all optional. They follow the key/value pair and are delimited by semi colons (';'). Although these values are written to the special *document.cookie* property, they can never be read back. Thus, if you want to know when a cookie was set or when it will expire, set an accompanying cookie containing this information as

the value.

Perhaps the most important property is *expire*:

```
expire=date/time

e.g.
expire=Fri, 5 Jul 2002 15:26:35 UTC
```

The time format is dependant upon the browser, BUT it's always the same format as is returned by the *Date.toGMTString()* method. The *expire* property marks when the cookie should be thrown away. Cookies are typically stored in memory by the user agent until the agent is closed. At this point, if they haven't expired, they are saved to a file for later. When the user agent is restarted, it loads the saved cookie files and checks if they have expired. If you don't specify an *expire* value then the cookie will expire when the current session ends. A session typically ends when the user agent is restarted, but is sometimes when the user leaves the domain the cookie belongs to.

The next cookie property can also be quite important. *path* controls which documents on the server will be sent the cookie when they are requested (and which documents can access the cookie client-side).

```
path=/path/on/server

e.g.
path=/mysite/myfolder/
```

The default for *path* is a single forward slash which indicates the whole of the domain. If you are on a shared domain or you have more than one cookie enabled application on your domain, then it's important to set the *path* property for every application, otherwise they may overwrite each other's values.

Coupled with the *path* property is the *domain* property.

(end of article)