Paper 185-25

Style Sheets, JavaScript, and SAS® ??? Oh, My !!

Kerril Bauerly, Best Solutions Inc., Kansas City, MO

ABSTRACT

There are so many features available to make your web pages look terrific and perform dynamically. However, its hard to determine how to use them when creating pages with the SAS®/Intrnet Software. This paper explains a few basics on Cascading Style Sheets or CSS. CSS allows a user to set up a common Style definition so that all pages on your web site will look the same. It also covers very basic JavaScript code that will let the browser do the work instead of repeated calls to the SAS broker. It then shows how to put it all together and create great looking pages with a standard look that do some of the up front work for you.

INTRODUCTION

Cascading Style Sheets are used in conjunction with HTML to provide web page formatting capabilities. The great thing about them is they can be stored in a separate file and linked to whichever pages they apply to. This allows all web pages in a particular site to look the same. I'm going to cover how to specify formatting basics such as backgrounds, fonts, colors, and margins. I am going to be covering only Level 1 specifications (CSS1). These specifications have been defined by the World Wide Web Consortium (W3C) and most of what I will cover is supported by the 2 most popular browsers: Internet Explorer and Netscape.

JavaScript is a simple and easy to use scripting language which lends dynamic elements to web pages. I'm going to cover using JavaScript for basic data validation. This allows the browser to do the work rather than the broker.

How does this apply to SAS? Both CSS and JavaScript can be utilized in conjunction with SAS generated pages. This paper will show you how!

Cascading Style Sheets

You can specify font and color attributes in HTML tags. Why do you want to use Style sheets? Because you only have to specify a style once at the top of the page, but you can use it throughout your HTML code. You aren't defining the same attributes over and over and over again. If your color or font scheme needs to be updated, the change only has to be made in one place. The best part is that style definitions can be stored in a separate file and used across multiple pages. This allows all pages on your site to look the same.

There are 4 ways to specify styles in your HTML pages:

1. Use the <LINK> tag to link to an external style sheet.
2. Define styles in a <STYLE>...</STYLE> block in your HTML code.
3. Use the @import statement in a <STYLE>...</STYLE> block. This is not widely supported so I won't cover it.
4. Use a STYLE= parameter on an HTML tag. This is a nice feature for overriding one particular style definition but when used throughout the program defeats the purpose of utilizing global styles.

The basics of Style: Selectors

The easiest way to experiment with style is to use the HTML <STYLE> tag in your <HEAD> block. Style definitions have the following syntax:

```
<STYLE TYPE="text/css">
        selector { property: value; }
        selector { property: value; }
</STYLE>
```

For example, to assign all level 1 heading tags an attribute of extra large font size and color of red, you would use the following code:

```
<STYLE TYPE="text/css">
     H1 { font-size: x-large;
          color: red; }
</STYLE>
```

(Just like SAS code, don't forget your semi-colons!) Because we specified an HTML tag element value as a selector, the browser will assign every <H1> tag the x-large font attribute and make all the text within the tag red. Any HTML element is eligible to be used as a selector. Therefore, definitions can be created for headings, the body, paragraphs, etc.

The basics of Style: Class Selectors

While assigning attributes to a specific HTML element is definitely handy, there are times when display attributes need to be a little more specific. In this case, you would define class selectors. Class selectors can be used by any HTML element by specifying CLASS=class selector in the HTML tag. Define the style like this:

```
<STYLE TYPE="text/css">
     .hr {color: red; }
</STYLE>
```

This allows you to use the **hr** class on any HTML tag that will support it. I use something like this most often for a horizontal line.

```
<HR CLASS="hr">
```

This will turn this instance of the horizontal line red. Notice how the class definition in the style block contains a period, but the actual class reference does not. I can change the color value in the style block as often as I like without changing HTML code. If I don't want the horizontal line to have the defined attribute, I don't use CLASS= on the tag:

```
<HR>
```

A good rule of thumb is to name class selectors by their function rather than their appearance. While using `<HR CLASS="red">` is definitely easy to understand, changing the style definition to make the rule green could cause confusion.

The basics of Style: ID Selectors

ID selectors are similar to class selectors but are designed to be used on a per-element basis. They are defined with a # in front of the selector instead of a period. ID selectors are not widely supported and for that reason, I don't use them. I mention them here so you will recognize them if you see them.

The basics of Style: Contextual Selectors

Contextual selectors are stacked within a definition. For instance, I want all my strong text in level 1 headings to be blue but strong text in level 2 headings to be green.

```
<STYLE TYPE="text/css">
        H1 STRONG { color: blue;  }
        H2 STRONG { color: green; }
</STYLE>
```

HTML tags that looked like this `<H1>This part of the heading is the default color, <STRONG> this part is blue.</STRONG></H1>` would produce a portion of blue text. All other references to the <STRONG> tag outside <H1> and <H2> tags would be treated normally.

The basics of Style: Grouped Selectors

HTML elements that need to have the same attributes assigned can be stacked also, but will need to be comma delimited:

```
<STYLE TYPE="text/css">
       H1, H2, H3 { color: blue;  }
</STYLE>
```

All level 1, level 2, and level 3 headings will be blue.

The basics of Style: Inheritance
Most people don't realize that browsers have default style sheets and that users may set up default style sheets according to individual needs. Style sheets coming in from the HTML page will override browser and reader defaults. When the browser starts assigning styles to HTML elements, each style is assigned a weight. The style with the highest weight is used. You can increase the weight of your style by using the " !important" declaration in your style definition. However, doing this could override a hearing or vision impaired default style sheet so using this is not recommended. A general rule to follow is: Styles are assigned in reverse of the order received by the browser. Selectors nested within other selectors will inherit the attributes for the outer selector unless specifically defined. Thus the term "Cascading".

```
 <STYLE TYPE="text/css">
        H1, H2, H3 { color: blue;  }
        BODY {background-color: black;
              Color: white; }
        .ghost {background-color: white;
                Color: black; }
 </STYLE>
 <body>
 <h1>level one heading</h1>
 <h2>level two heading</h2>
 <h3>level three heading</h3>
 <p>this text is white on a black background</p>
 <p class=ghost>this text is black on a white
background</p>
 </body>
```

The above code will produce a page with a black background. The Body is the parent element in this case. Text in the level 1, 2, and 3 headings will be blue but the background remains black. The headings inherit the black background from their parent, the body. However, any text within a tag that has the class definition of "ghost" will be presented in black text on a white background. Because the class of ghost is named specifically in the <P> tag, its attributes override those same attributes in the parent. Thus this text will default to white text on a black background. Specific rules for cascading are on the W3C web page.

The basics of Style: Assigning properties - Fonts
The most common use of style sheets is assigning fonts. CSS font definitions allow the browser to use fonts already installed on the reader's machine. The rules for determining which font to use are on the W3C web page. The best guide here is to use font families rather than specific fonts. However, fonts are tried in the order listed, so list specific fonts first (such as Comic Sans) and then list one or more font families (such as Arial or Helvetica) and lastly list a generic family (like Sans-Serif). If the first font isn't installed on the user's machine, the browser will search for the second one and so on.

Fonts can be assigned by their specific attributes or can be assigned using a shorthand method.

Defining by specific attributes:

```
.P1 {font-family: arial, helvetica, sans-
serif;
   font-style: italic;
   font-variant: small-caps;
   font-weight: bolder;
   font-size: x-large; }
```

Defining by the shorthand method:

```
.P2 {font:  <font-style> || <font-variant> ||
    <font-weight> <font-size> <line-height>
              <font-family> }
```

or

```
.P2 {font:   italic; small-caps; bolder;
   X-large; 120%; arial, helvetica, sans-
serif;}
```

Available font properties and descriptions are given in detail on the Web Design Group web page.

The basics of Style: Assigning properties - Colors
Another advantage of using Style Sheets is the ability to set up color schemes. Colors can be specified in several different ways.

```
    .P1 {color: white; }   /* using a keyword */
    .P2 {color: #FFFFFF; } /* using hex rgb
                           definition */
    .P3 {color: rgb(255,255,255); } /* rgb decimal
                                  values */
    .P4 {color: rgb(100%,100%,100%; } /* rgb
                           percentages */
```

All of the above styles identify the same color. There are several good color tables available on the web. I prefer the one at htmlgoodies.com.

The basics of Style: Assigning properties - Backgrounds
As with fonts, backgrounds can be specified using individual specific attributes or using a shorthand method.

Defining by specific attributes:

```
Body {background-color: green;
      background-image: url(kitties.gif);
      background-attachment: fixed; }
```

Defining by shorthand method:

```
Body {background: url(kitties.gif) green
fixed; }
```

When specifying a background in a Style Sheet, it is important to remember 2 things. The URL of the background image may be a fully qualified reference or a reference relative to the Style Sheet source. Netscape browsers often look for the URL in relation to the HTML source as opposed to the Style Sheet source. Because I generally have to support all types of browsers, my personal preference is to store HTML code in the same directory as my Style Sheet to avoid browser confusion. The second item to remember is to specify a color for background in case there is a problem loading the background image.

The basics of Style: Assigning properties - Other Stuff
Margin allowances are specified in pixels or as percentages of the page width or length.

```
    Body { margin-left: 5%; }
```

Height and Width parameters are available but don't work for every element in every browser. The following style will produce a fat red horizontal rule line in Internet Explorer but a regular sized uncolored rule in Netscape.

```
    hr {color: red;
        height: 5px; }
```

The basics of Style: Notes for Users
Note 1: There are many available properties and corresponding values. I have only touched on a few here. Not all properties are supported by every browser. The Web Review web site has a terrific Master Grid of which browsers support which features. I test my HTML code with CSS in both a Netscape and an Internet Explorer browser before publishing. It is frustrating sometimes, when your pages appear differently in different browsers, and it is tempting to upgrade to the latest browser version so all your styles look great. However, you must keep in mind the tools your intended audience will be using and code your styles accordingly.

Note 2: Try not to make your web page dependent on the style sheet definitions. Display the page without any styles. Does your information still come across? Sure, it doesn't look as cool, but is the page still usable?

Note 3: Check your CSS code. There are CSS checkers on both the W3C page and Web Design Group page. Use them!

Note 4: Use comments to document!!! Comments are the same (/* ..*/) for Style Sheets as in SAS:

```
      /* I like to comment my hex color values */
```

So there is no reason not to comment your styles the same way you comment your code!

Note 5: Be careful using Styles in conjunction with HTML tables. Be sure to test in multiple browsers. Tables sometimes don't follow inheritance rules so you might need to make adjustments. I'll explain this more in the example.

Note 6: HTML font, size, and color specifications on the HTML tags themselves will override style sheet definitions.

The basics of Style: Building a Style Sheet

As I stated above, there are several ways to define styles for an HTML page. I prefer to put mine into an external file because I'm usually creating more than one page. If you prefer to keep your styles in your HTML code, just enclose them in <STYLE>...</STYLE> tags.

Lets start by building a front page for Mom's Marbles. The background I have chosen has a border running down the side that is approximately 150 pixels wide. This is important to remember because I want to create a sidebar down the left side later. The background will be defined in the BODY definition:

```
    Body { background: url(paisley1.gif) black;}
```

I also specify the color black after the URL so that if for some reason the browser can't find the paisley1 graphic, it will still use a black background.

I mentioned above that tables often react unexpectedly with Style Sheets. To solve some of the inheritance problems we specify all the body attributes that we want to cascade into the table on one statement in the Style Sheet.

```
    Body, Table, TD   {
        font-family:"Comic Sans MS",   arial,
        helvetica, sans-serif;
     /* use comic sans font if its there,
        use arial or helvetica if its not,
      use a sans-serif font if all else fails */
        color: white;         /* set text color */
        font-size: 14pt;        /* set font to an
                                absolute size  */
        line-height: 120%;      /* line height is
                                120% of font
                                height, so not
                                quite double
                                spaced */
    }
```

Table attributes unique to a particular table should be specified in their own selector statement.

Next I want to define my heading attributes. I expect to use Level 1 and Level 2 headings:

```
    H1    { color: blue;        /* make all fonts
                                blue */
        font-family: "Brush455 BT",arial,
        helvetica, sans-serif; /* use a brush
                                font if its there
                                */
        margin-left: 155px;     /* Place heading
                                to the left of the
                                border */
        font-size: xx-large;    /* uses whatever
                                the browser
                                decides is xx-
                                large */
        line-height: 120%;      /* line height is
                                120% of font
                                height */
    }
H2    { color: red;     /* make all text in this
                        heading red */
        font-size: large;       /* uses whatever
                                the browser
                                decides is large
                                */
        margin-left: 155px;     /* Place heading
                                to the left of the
                                border */
    }
```

The information to be placed on the sidebar will be handled in the HTML table statements, but here I have to move the headings over 155 pixels so they will miss the sidebar.

Because I'm using a black background, I need to modify the anchor links to show up:

```
    A:link { color: yellow;  }        /* set link
                                      color */
    A:visited { color: orange; }      /* set
                                      visited
                                      link color
                                      */
    A:active { color: red; }  /* set active link
                              color */
```

Now I want all of my horizontal rules to be fat and red. Unfortunately, this will only work in an Internet Explorer browser. The line will still show up in Netscape, but Netscape ignores the line attributes:

```
    hr {color: red;
        height: 5px; }
```

There are going to be a few lines of text on my page that I want to call special attention to. I will create classes to define these:

```
    .p1 {color:Fuchsia; }
    .construct {color: yellow;
             font-size: smaller; }
```

To create a sidebar, I need to create a table with 2 columns and separate tables embedded in each column. I need to make the one containing the sidebar 155 pixels wide. While it would be easy to specify this in the style sheet, it is specific to this page, so I will define it on the table tag instead.

That's all there is to this Style Sheet. I'm going to save this to a file called Marblestyle.CSS (CSS is the reserved extension for Style Sheets.) To use it in my HTML, I will link to it:

```
<html>
<head>
<title>Lost your marbles?</title>
<link    rel=stylesheet    href="Marblestyle.css"
type="text/css">
<h1>Mom's Marbles </h1>
</head>
```

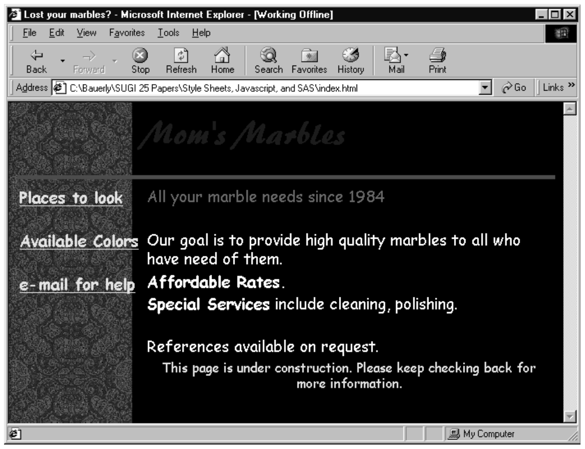
The rel and type parameters are very important! HREF can either be a relative or absolute reference (http://......).

I want to emphasize a couple of lines in my table, so I will use my class selectors:

```
<td class="p1">All your marble needs since
1984<br><br></td>
...
<td class="construct"><DIV align=center>This
page is under construction.  Please keep
checking back for more information.</DIV></td>
```

I can use these class selectors on any tag, I just happen to be using them on a Table Data tag here.

Once I get everything working, my page is going to look something like this:


Lost your marbles? - Microsoft Internet Explorer - [Working Offline]
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address C:\Bauerly\SUGI 25 Papers\Style Sheets, Javascript, and SAS\index.html
Go Links
My Computer

Mom's Marbles

Places to look

Available Colors

e-mail for help

All your marble needs since 1984

Our goal is to provide high quality marbles to all who have need of them.

Affordable Rates.

Special Services include cleaning, polishing.

References available on request.

This page is under construction. Please keep checking back for more information.

This page is from Internet Explorer but looks the same in Netscape except the horizontal rule has no color and is the default thickness.

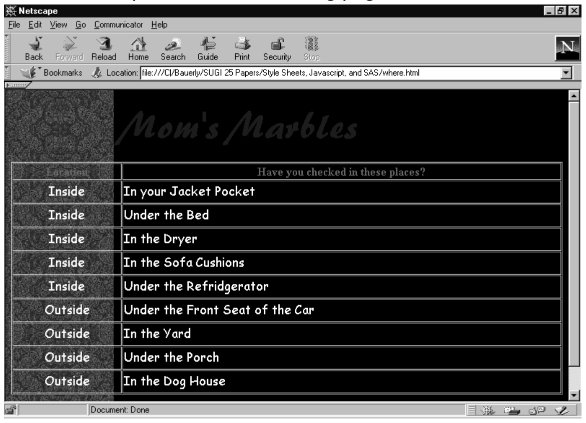
What does this have to with SAS?
Did you know you can use your defined Style Sheets with your SAS output? Now I can generate the pages corresponding to the links on my front page. The first example uses the DS2HTM macro to list a dataset of places to look for your marbles.

```
title "Mom's Marbles";
%ds2htm(data=one,  runmode=b,
        htmlfref=whereout,
        labels=y,  clclass=p1,
        id=loc,    ihalign=center,
        vhalign=left,
        ttag=HEADER 1,
        sshref1=Marblestyle.css,
        sstype1=text/css,
        ssrel1=stylesheet);
```

The parameters specific to Style Sheets are **clclass**, **sshref1, sstype1,** and **ssrel1**. The SS parameters correspond to the <LINK....> tag parameters that invoke the Style Sheet in HTML code. In fact, if you look at the HTML source for the SAS generated page, you will see the macro generated the <LINK...> tag. SSHREF1 is the URL of the Style Sheet. SSTYPE1 corresponds to the TYPE= parameter and SSREL1 corresponds to REL=. The CLCLASS parameter tells the macro to assign the P1 class to column labels. I reset the TTAG parameter. I forced it to HEADER 1 from the default of PREFORMATTED HEADER 3. I did this to force it to use the Style already defined for the <H1> tags.

Styles can be defined for most of the elements in all the HTML formatting macros. A complete listing of parameters including Style Sheet parameters is available on the SAS web site.

The above code produces the following page:


Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: file:///C|/Bauerly/SUGI 25 Papers/Style Sheets, Javascript, and SAS/where.html
Document: Done

Mom's Marbles

Location	Have you checked in these places?
Inside	In your Jacket Pocket
Inside	Under the Bed
Inside	In the Dryer
Inside	In the Sofa Cushions
Inside	Under the Refridgerator
Outside	Under the Front Seat of the Car
Outside	In the Yard
Outside	Under the Porch
Outside	In the Dog House

Notice how the pages have the same look and feel. They both have the same fonts, title, and backgrounds. This SAS generated page is being displayed in Netscape and there are few differences from the Internet Explorer version of the same page.

JavaScript
JavaScript is script code that resides in or is linked to by HTML code. Therefore, once your HTML page is loaded, your JavaScript is ready to run. JavaScript runs on the client side which is very convenient for basic form checks. It is more widely supported than Style Sheets and can lend your pages a more dynamic aspect. JavaScript is a complete language all its own. Since syntax details are beyond the scope of this paper, I am going to cover a few shortcuts for form validation.

Basic data checks on the client side save repeated calls to the SAS broker for form field validation. I am going to show you how to verify that a field is not blank, that if field A is checked field B has a value, and how to make sure a field is numeric.

The basic JavaScript functions are supported by most browsers although I have learned that Netscape gives more informative error messages when debugging than does Internet Explorer. JavaScript is also easier to learn and doesn't require a separate compiler like Java.

A few warnings are in order. There is lots of documentation related to JavaScript. Most of it is terrible. I picked it up by studying examples and lots of trial and error. JavaScript is not widely supported in older browsers so you will have to hide the script from those that can't handle it.

At this point, I am going to recommend you get a good HTML editor. While its definitely easier to code Style Sheets in a good HTML package, its practically a must for JavaScript. I prefer Homesite by Allaire. It comes with a nice little Style Sheet editor and is able to pick out most of my JavaScript errors.

Some Basics of JavaScript: A few rules
1. JavaScript code is enclosed in script tags within HTML code:

```
<SCRIPT LANGUAGE="JavaScript">
  <!-- Hide code from non-js browsers
....
// end hiding -->
 </SCRIPT>
```

The comment hides the enclosed script from browsers that do not support JavaScript.

2. JavaScript code consists of a series of functions called from within HTML code. Functions can have zero or more parameters.

```
function validateForm()
   {
...
 }
```

All functions and "do" related code must be enclosed in curly brackets.

3. The function I am going to use is called by an event handler. Some examples of events are onLoad (execute this function when the page is loaded) and onSubmit (execute this function when the user presses the submit button). The functions I am going to write are used with the onSubmit event.

4. Another important reminder is that arrays of any kind are indexed starting at zero. Form fields with the same name are automatically treated as arrays by JavaScript. This can come in extremely handy.

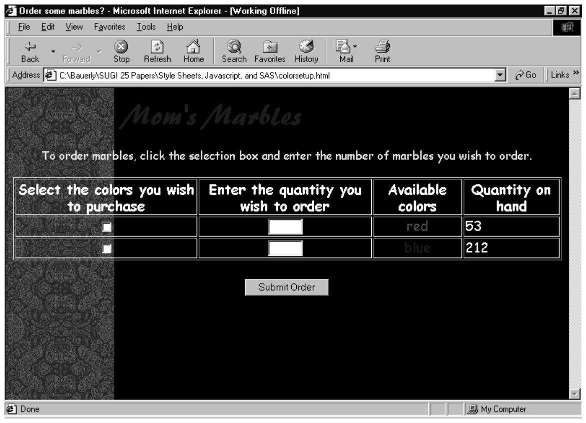
5. Form and Field names are case sensitive. Take extra care when using combinations of upper and lower case.

Some Basics of JavaScript: Preparing the form
The first thing I need is a form definition tag in my HTML. The standard form tag contains the action parameter and sometimes a method specification. I need to add some parameters to the standard form tag in order to invoke my JavaScript:

```
<form   name="Bob"   action="/cgi-bin/broker.exe"
onSubmit="return validateForm()">
```

The first thing I have done is give my form a name, in this case "Bob". My action parameter is the standard call to the SAS broker. Next I have included my event handler for this form. When the user clicks on the submit button on the form, the validateForm JavaScript function will be invoked. It is expected to return a boolean value (true or false) and I am passing no parameters to the function. If a value of true is returned, the action in the action parameter will be performed, otherwise nothing happens and the user must make changes and try again. I will generate error messages in the JavaScript code to support the false condition.

If I'm using a form, I am going to need some kind of input fields to pass information into the broker. I want to allow my page visitors to order marbles from my inventory so I've mocked up a form to look something like this:


Order some marbles? - Microsoft Internet Explorer - [Working Offline]

Address C:\Bauerly\SUGI 25 Papers\Style Sheets, Javascript, and SAS\colorsetup.html

Mom's Marbles

To order marbles, click the selection box and enter the number of marbles you wish to order.

Select the colors you wish to purchase	Enter the quantity you wish to order	Available colors	Quantity on hand
☐		red	53
☐		blue	212

Submit Order

Done

My Computer

File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Go Links

I have created a table of my available inventory. To order marbles, the user will need to click in the checkbox and enter a numeric quantity. The checkbox tag will look like this:

```
<input type="checkbox" name="Shipit" value="red">
```

Notice that I have given the checkbox field the name "Shipit".

The field for quantity will look like this:

```
<input   type="text"   name="quantity"   size="5"
maxlength="3">
```

Again, I have given the text field a name, this time "quantity".

I can go ahead and leave out the JavaScript and let the SAS broker verify that the user entered a quantity after selecting an item and that the quantity was numeric and greater than zero. But each time the broker finds an error, it has to write a message back to the screen and let the user try again. This can be very time consuming especially if the user made multiple mistakes.

Some Basics of JavaScript: Creating a Validation Function

I'm going to go ahead and do the upfront verification in a JavaScript function. I've already shown you the script tags, now lets look at what goes in between them.

The first thing I'm going to do is name my function:

```
function validateForm()
    {
```

Notice that this corresponds to the name on my onSubmit parm in my form tag. The () indicates that I am passing no parameters to this function. The open curly bracket must precede the rest of my code and a close curly bracket will close the function after the last line of script code.

The first thing I want to verify is that at least one item in my form was selected. I do this by verifying that at least one checkbox is checked.

JavaScript treats every part of the form as an object. The entire page is a document object so is referenced by the term "document". The form itself is an object and in this case is named "Bob". The field or checkbox is another object and is named "Shipit". Each object has variables that describe it called properties. References to fields in a form in a document require a 3 level name and usually a reference to one of the field's properties:

```
 document.<form name>.<field name>.<property>
```

In this case, the above reference refers to the checked property of the Shipit field in the Bob form in my document. Because I gave all of my checkboxes the same name, JavaScript is going to treat them as an array. In JavaScript, array references start at zero and appear in square brackets. So this reference refers to the red marble checkbox:

```
(document.Bob.Shipit[0].checked == false) &&
```

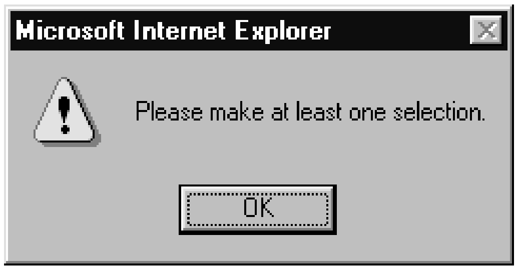
The checked property has values of true or false. The equal to comparison operator is a double equal sign and the AND logical operator is a double ampersand.

Now I can build the following code to make sure that at least one checkbox was checked:

```
if  ((document.Bob.Shipit[0].checked == false)  &&
     (document.Bob.Shipit[1].checked == false)) {
```

If neither boxes are checked, I must tell the user to do something about it. The alert method displays a message box with an "OK" button. The user must click on the button to continue.

```
    alert("Please make at least one selection.");
    return false;
    }
```


Microsoft Internet Explorer

Please make at least one selection.

OK

Once I have alerted the user to the problem, I want to return control to the HTML form and not call the broker. Therefore, I am returning a value of false to the onSubmit event. Finally, I close my block of code with a curly close bracket.

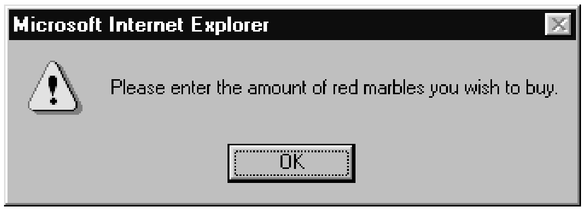
Once the user has checked a box, the first validation point will pass. Now I need to make sure that a valid order quantity has been entered. I only want to check the quantity fields where the user has checked the box, so I first verify that the checkbox has been clicked:

```
if  (document.Bob.Shipit[0].checked == true)   {
     /* check to make sure quantity field has a
     value */
```

Next, I want to make sure the user entered something and didn't leave the box blank. To do this, I am going to check the length of the value of the quantity field:

```
 if  (document.Bob.quantity[0].value.length == 0)
    {
        alert("Please enter the amount of red
marbles you wish to buy.");
        return false;
     }
```

If the user didn't fill in the box, I'm going to generate an alert:


Microsoft Internet Explorer

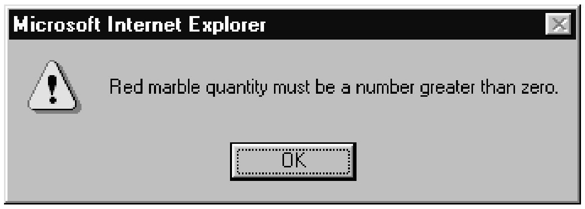
Please enter the amount of red marbles you wish to buy.

OK

Next I want to make sure that the user did enter an order quantity and that is was numeric and non-zero.

```
    /* check to make sure its numeric */
else if ((document.Bob.quantity[0].value == 0)
|| (document.Bob.quantity[0].value /
    document.Bob.quantity[0].value != 1)) {
        alert("Red marble quantity must be a
        number greater than zero.");
       return false;
        }
     } /* close to if ... .checked */
```

There are many ways to validate numeric fields. My preference is dividing the value by itself to see if it equals 1. I check to see if a zero has been entered for two reasons. First, zero is not a valid quantity. Second, I don't want to divide by zero. The double bar is used here as a logical OR and not equal is indicated by != (notice the comments!). If this test fails, yet another error box is displayed to the user:


Microsoft Internet Explorer

Red marble quantity must be a number greater than zero.

While all the different error boxes may be viewed as overkill, I would rather display more informative messages. Its very frustrating to receive a generic error message and not have a clue what you are doing wrong.

This code repeats for each checkbox/quantity field on the page. My examples involve writing pretty much the same chunk of code for validation of each box. I can put this in a loop for ease of coding but that is beyond the scope of this paper. I encourage you to try it on your own.

The last thing I need to do is finish up my function. If all validation tests have been passed, I want to let the form know its ok to send the iall the form values onto the broker. To do this, I simply use the following as the last statement in my function:

```
  return true;
         }
```

The last curly bracket closes out the function.

JavaScript and SAS

Once I have my JavaScript code working correctly, I am ready to combine it with SAS. What I would like to do is dynamically generate a table based on what's currently in stock and allow users to make selections based on that information.

Since I know what my JavaScript code needs to look like, I am going to have SAS generate the validation function for me based on my current inventory.

The first step is to run a PROC FREQ on my inventory to get counts of marbles:

```
proc freq data=colors;
  tables color / out=inventory;
run;
```

Because I want to be lazy, I want SAS to build the actual table and then populate it. So once again I will want to use the %DS2HTM macro. I am going to need to add a few extra fields to my dataset first:

```
data inventory1;
  set inventory;
  chckbox = '<input type="checkbox"
name="Shipit" value="' || trim(color) || '">';
  quantbox = '<input type="text" name="quantity"
size="5" maxlength="3">';
  fmtcolor = '<font color="' || trim(color) ||
'">' || trim(color) || '</font>';
  label chckbox = 'Select the colors you wish to
purchase'
        quantbox = 'Enter the quantity you wish
to order'
        fmtcolor = 'Available colors'
        count = 'Quantity on hand'
               ;
run;
```

The new variables are CHCKBOX which is my HTML code checkbox, QUANTBOX which is the HTML code quantity box, and FMTCOLOR which I'm using to show the marble color in that color. I also assigned column labels.

I'm using a DATA _NULL_ to generate my JavaScript code:

```
data _null_;
 ... file and set and initial HTML and
JavaScript stuff ...

    webline = '  if (';
    do i = 0 to &nobs-1;
      if i ne &nobs-1 then
        webline = trim(webline) ||
               '(document.Bob.Shipit['
               || trim(left(put(i,3.))) ||
               '].checked == false) && ';
      else
        webline = trim(webline) ||
                '(document.Bob.Shipit[' ||
                trim(left(put(i,3.))) ||
                '].checked == false)) {';
      put webline;
      webline = '       ';
    end;
```

This code creates the first check in the function which checks to make sure all boxes are checked. &NOBS is the number of items in my inventory dataset.

This next piece of code in the DATA _NULL_ generates the box value checks for each item in the inventory:

```
 /* for each inventory item, create a separate
piece of JavaScript code */
  webline = '    if (document.Bob.Shipit[' ||
trim(left(put(_n_-1,3.))) || '].checked == true)
{';
  put webline;
  put '/* check to make sure quantity field has
a value */';
  webline = '    if (document.Bob.quantity[' ||
trim(left(put(_n_-1,3.))) || '].value.length ==
0) {';
  put webline;
  put '          alert("Please enter the amount
of ' color ' marbles you wish to buy.");';
  put '          return false;';
  put ' }';
  put '    /* check to make sure its numeric
*/';
  webline = '  else if ((document.Bob.quantity['
|| trim(left(put(_n_-1,3.))) ||
            '].value == 0) ||
(document.Bob.quantity[' || trim(left(put(_n_-
1,3.))) ||
            '].value / document.Bob.quantity['
|| trim(left(put(_n_-1,3.))) || '].value != 1))
{';
  put webline;
  put '    alert("' color ' marble quantity must
be a number greater than zero.");';
  put '    return false;';
  put '       }';
  put ' }';
```

I try to maintain spacing when writing out HTML and JavaScript code from SAS. It makes it much easier to debug when something goes wrong!

The last piece of DATA _NULL_ code closes out the script and head tags and more importantly defines the form:

```
    Put '<body>';
    put '<form name="Bob"'
         ' action="/cgi-bin/broker.exe"'
         ' onSubmit="return validateForm()">';
```

That's all for the first DATA _NULL_.

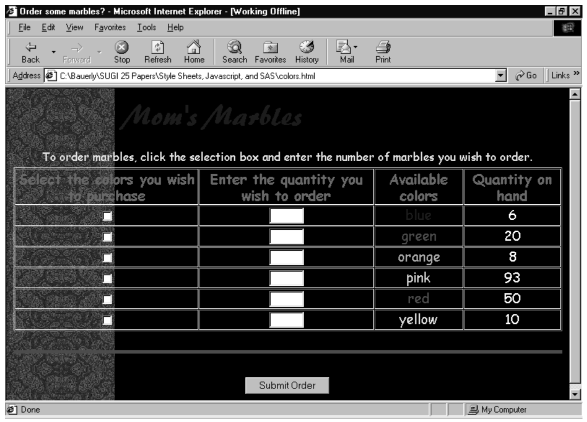
As I stated above, I am going to let SAS do all the work with the actual inventory table. I have set my OPENMODE parameter to APPEND. This will append the table HTML code to what was written out above from the DATA _NULL_. Because I have actual HTML tag values in some of my variables, I have to set ENCODE to N. The PAGEPART=BODY parameter tells SAS I only want the table generated. I don't need any of the other HTML flags. Lastly, I have used a %STR() around my caption to hide the comma in it from the SAS tokenizer.

```
%ds2htm(data=inventory1,
         runmode=b,
         htmlfref=colorout,
         openmode=append,
         encode=n,
         var=chckbox quantbox fmtcolor count,
         pagepart=body,
         labels=y,
         clclass=p1,
          caption=%str(To order marbles, click the
selection box and enter the number of marbles
you wish to order.),
          cclass=construct,
         vhalign=center,
          ttag=HEADER 1,
          sshref1=Marblestyle.css,
         sstype1=text/css,
         ssrel1=stylesheet);
```

The last thing I need is a cleanup DATA _NULL_ step to add the submit button and close out open tags:

```
data _null_;
  file colorout mod;
  put '<div align="center"><br><input
type="submit" name="Submit" value="Submit
Order"></div>';
  put '</form>';
  put '</body>';
  put '</html>';
run;
```

The resulting page looks like this:


Order some marbles? - Microsoft Internet Explorer - [Working Offline]
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address C:\Bauerly\SUGI 25 Papers\Style Sheets, Javascript, and SAS\colors.html
Go Links
Submit Order
Done
My Computer

Mom's Marbles

To order marbles, click the selection box and enter the number of marbles you wish to order.

Select the colors you wish to purchase	Enter the quantity you wish to order	Available colors	Quantity on hand
☐		blue	6
☐		green	20
☐		orange	8
☐		pink	93
☐		red	50
☐		yellow	10

This page looks just like the mock-up version but was completely generated by SAS.

CONCLUSION

Style Sheets can add a professional look to your web pages. As I have shown, you can make pages look the same whether they were generated manually or dynamically. This was not a comprehensive look at Style Sheets but I hope it was enough to get you started and interested in researching them further.

JavaScript is a challenge because all the documentation is so bad. However, it is a relatively easy language to pick up and once you get the hang of it, offers potential I have only touched on here. Programming by example is the easier way to learn it. I use JavaScript extensively to save browser calls although there are many more uses. I encourage you to look into it further as well.

REFERENCES

SAS® Institute Web Tools Documentation

World Wide Web Consortium (W3C) Style Sheet Page:
http://www.w3c.org/Style/CSS

Web Design Group Page:
http://www.htmlhelp.com/reference/css/properties.html

WebReview Page:
http://webreview.com/pub/guides/style/style.html

Dr. Joe Burns:
http://www.htmlgoodies.com/

Reaz Hoque, *Practical JavaScript Programming*, New York: Henry Holt & Company, Inc, 1997.

ACKNOWLEDGMENTS

SAS and SAS/Intrnet are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kerril Bauerly
Best Solutions, Inc.
4901 Main Street
Suite 402
Kansas City, MO 64112
Work Phone: (816) 931-9802
Fax: (816) 931-0682
Email: kerrilbauerly@teambsi.com
Web: http://www.teambsi.com

Marblestyle.CSS

```
Body { background: url(paisley1.gif) black;}
Body, Table, TD, Form {
        font-family: "Comic Sans MS",arial, helvetica, sans-serif;  /* use comic sans font if its there,
                                                                      use arial or helvetica if its not,
                                                               use a sans-serif font if all else fails */
       color: white;        /* set text color */
        font-size: 14pt;     /* set font to an absolute size  */
        line-height: 120%;   /* line height is 120% of font height, so not quite double spaced */
       }
H1    { color: blue;      /* make all fonts blue */
        font-family: "Brush455 BT",arial, helvetica, sans-serif;
       margin-left: 155px;
        font-size: xx-large;    /* uses whatever the browser decides is large */
        line-height: 120%;   /* line height is 150% of font height */
   }
H2    { color: red;
        font-size: large;
    }
A:link { color: yellow;  }   /* set link color */
A:visited { color: orange; }  /* set visited link color */
A:active { color: red; }  /* set active link color */

hr {color: red;
      height: 5px; }

.p1 {color:Fuchsia;    }
.construct {color: yellow;
            font-size: smaller; }
```

Index.HTML

```
<meta http-equiv="Description" content="Mom's Marbles page">
<meta http-equiv="Keywords" content="mom, advice, marbles">
<html>
<head>
<title>Lost your marbles?</title>
<link rel=stylesheet href="Marblestyle.css" type="text/css">
<h1>Mom's Marbles </h1>
</head>

<body>
<hr align="CENTER" color="Red">
<table width="100%" cellpadding="0" cellspacing="0">
<tr><td valign="top">
       <table width="155px">
          <tr><td><a href="where.html"><b>Places to look</b></a><br><br></td></tr>
          <tr><td><a href="colors.html"><b>Available Colors</b></a><br><br></td></tr>
          <tr><td><A href="mailto:ksbauerly@aol.com"><b>e-mail for help</b></A></td></tr>
       </table></td>
<td colspan="8">
       <table align="left">
         <tr><td class="p1">All your marble needs since 1984<br><br></td></tr>
         <tr><td>Our goal is to provide high quality marbles to all who have need of them.<br></td></tr>
         <tr><td><b>Affordable Rates.</b><br></td></tr>
         <tr><td><b>Special Services</b> include cleaning, polishing.<br><br></td></tr>
         <tr><td>References available on request.<br></td></tr>
         <tr><td class="construct"><DIV align=center>This page is under construction. Please keep
                    checking back for more information.</DIV></td></tr>
       </table>
</td>
</tr>
</table>
</body>
</html>
```

Marbles.SAS

```
 /* create data for Style Sheet example */
data one;
  input loc $char8.
        place $char40.        ;
  label loc = 'Location'
```

```
        place = 'Have you checked in these places?'      ;
  cards;
...;
run;

proc sort data=one;
  by loc;
run;

filename whereout "where.html";

title "Mom's Marbles";

 /* build HTML page */
%ds2htm(data=one,
        runmode=b,
       htmlfref=whereout,
       labels=y,
       clclass=p1,
       id=loc,
       ihalign=center,
       vhalign=left,
        ttag=HEADER 1,
       caption=Have you looked here?,
        sshref1=Marblestyle.css,
        sstype1=text/css,
       ssrel1=stylesheet);

 /* create data for JavaScript example */
data colors(keep=special color);
  length special $ 8;
  input color $char8.   ;
  randflag = int(ranuni(-1) * 100);
  do i = 1 to randflag;
    output;
  end;
  cards;
...;
run;

proc freq data=colors;
  tables color / out=inventory;
run;

data inventory1;
  set inventory;
  chckbox = '<input type="checkbox" name="Shipit" value="' || trim(color) || '">';
  quantbox = '<input type="text" name="quantity" size="5" maxlength="3">';
  fmtcolor = '<font color="' || trim(color) || '">' || trim(color) || '</font>';
  label chckbox = 'Select the colors you wish to purchase'
        quantbox = 'Enter the quantity you wish to order'
        fmtcolor = 'Available colors'
        count = 'Quantity on hand'           ;
run;

proc sql;
  reset noprint;
  select count(*) into: nobs
  from inventory1;
quit;
%put nobs=&nobs;

filename colorout "colors.html";

data _null_;
  length webline $ 400;
  file colorout;
  set inventory1 end=eof;
  if _n_ = 1 then do;
       put '<html>';
       put '<head>';
       put '<title>Order some marbles?</title>';
       put '<link rel="Stylesheet" href="Marblestyle.css" type="text/css">';
       put '<SCRIPT LANGUAGE="JavaScript">';
       put '  <!-- Hide code from non-js browsers';
       put 'function validateForm()';
       put '   {';
```

```
        webline = '  if (';
        do i = 0 to &nobs-1;
          if i ne &nobs-1 then
            webline = trim(webline) || '(document.Bob.Shipit[' || trim(left(put(i,3.))) ||
                     '].checked == false) && ';
          else
            webline = trim(webline) || '(document.Bob.Shipit[' || trim(left(put(i,3.))) ||
                     '].checked == false)) {';
          put webline;
         webline = '      ';
        end;

       put '    alert("Please make at least one selection.");';
        put '    return false;';
        put '    }';
  end;

   /* for each inventory item, create a separate piece of JavaScript code */
  webline = '    if (document.Bob.Shipit[' || trim(left(put(_n_-1,3.))) || '].checked == true)  {';
  put webline;
  put '/* check to make sure quantity field has a value */';
  webline = '    if (document.Bob.quantity[' || trim(left(put(_n_-1,3.))) || '].value.length == 0) {';
  put webline;
  put '           alert("Please enter the amount of ' color ' marbles you wish to buy.");';
  put '           return false;';
  put ' }';
  put '    /* check to make sure its numeric */';
  webline = '  else if ((document.Bob.quantity[' || trim(left(put(_n_-1,3.))) ||
            '].value == 0) || (document.Bob.quantity[' || trim(left(put(_n_-1,3.))) ||
            '].value / document.Bob.quantity[' || trim(left(put(_n_-1,3.))) || '].value != 1)) {';
  put webline;
  put '   alert("' color ' marble quantity must be a number greater than zero.");';
  put '    return false;';
  put '        }';
  put ' }';

  if eof then do;
    put 'return true;';
    put '          }';
    put '// end hiding -->';
    put ' </SCRIPT>';
    put '</head>';
    put '<body>';
    put '<form name="Bob" action="/cgi-bin/broker.exe" onSubmit="return validateForm()">';
  end;
run;

%ds2htm(data=inventory1,
        runmode=b,
       htmlfref=colorout,
       openmode=append,
       encode=n,
       var=chckbox quantbox fmtcolor count,
       pagepart=body,
       labels=y,
       clclass=p1,
      Caption=%str(To order marbles, click the selection box and enter the number of marbles you wish to
order.),
        cclass=construct,
       vhalign=center,
        ttag=HEADER 1,
        sshref1=Marblestyle.css,
       sstype1=text/css,
       ssrel1=stylesheet);

data _null_;
  file colorout mod;
  put '<div align="center"><br><input type="submit" name="Submit" value="Submit Order"></div>';
  put '</form>';
  put '</body>';
  put '</html>';
run;
```