

Using HTML as a SAS® Documentation Tool

Paul Marsh, Electricities of NC, Inc., Raleigh, NC

Abstract

Classic documentation presents static information on a linear media - paper. Such presentation is less effective when complex interwoven relationships must be shown. This paper will show how HTML (HyperText Markup Language) is used as a method around the interwoven relationship problem.

Introduction

Electricities of North Carolina, Inc. is a non-profit organization that provides administrative, legal, legislative, and technical services to more than 75 municipally owned electric distribution systems in North Carolina, South Carolina, and Virginia. The full participant members own interests in the Duke Power Co. Catawba and Carolina Power & Light Brunswick nuclear power plants. The combined budgets to operate those power plants are approximately one billion dollars per annum. A collection of over 40 SAS/CALC® spreadsheets detail the figures.

Problem Statement

The budget equations are the line items related to the operation of a nuclear energy power plant. The equations are numerous (in the hundreds) and complicated (composed of many operands which often are the result of other equations). Many of the operands are inputs to several different equations.

The current documentation is a book containing the equations. An example illustrates how cumbersome this book can be:

$alp = a1 + b1 + c1 + d1 + e1$ (found on pg. 7)
 $a1 = a2 + a3 + a4 + a5$ (found on pg. 8)
 $d1 = p1 + p2 + p3$ (found on pg. 2)
 $b1 = beta1 + beta2 + beta3$ (found on pg. 20)

.... and so on for a complicated equation

Your first reaction is to reorder the equations so that all parameters immediately follow. That would work except that a particular parameter may be contained in three or four equations. You then think - just duplicate the parameter for all associated equations! Well, the book of equations is already very thick and to duplicate parameters as often as they appear, you would end up with a document as thick as a dictionary. Very intimidating.

But the fundamental problem with the documentation is that it fails to show the computations themselves, which is ultimately what the target audience (budget personnel) really want to see. A budgeter only wants to see the numbers (which change every budget year) to determine if the equation contains the correct result. If the end result appears erroneous, he will look at each component in the equation for any value that appears incorrect. He might trace each component several steps backward to determine its validity. The book of equations does not really aid such endeavors.

A Web Solution

As you back away, you see the equations can be illustrated by a spider web. A particular parameter is linked to several equations. Each of those equations can have many links to other parameters which are intertwined with yet other equations. You have a web of information you wish to fathom through. Exploring that type of framework is what World Wide Web information browsers (such as Netscape and Mosaic) are specifically designed for and do well. Fortunately, we do not need to be on the Internet (or even an intranet) to use these programs.

As the world of information browsers matures, we are offered several avenues to presenting the equations in a useful manner. Indeed, that world is evolving so quickly, I will not attempt to present a definitive answer to the problem.

	Col1	Jul95	Aug95
ROW11	Total Deposit to Bond Fund	24,535.82	24,535.82
ROW6	Decommissioning Trust	324.68	327.49

Figure 1. Sample spreadsheet segment

Instead, I will outline the steps I have taken, including some stumbles. I also offer the following web URL (I assume that the reader is familiar with the World Wide Web and information browsers) that I will periodically update as I find better ways of documenting the equations:

http://www2.ncsu.edu/eos/info/sasclass.info/www/sas_html_doc.html

I feel the first two steps to document the equations are stable; that is, they are the best avenue to travel at this time and are not likely to change in the near future. However, subsequent steps are dependent both on the information delivery tools available at a given point in time and the needs of a particular target audience.

Figure 1 represents part of a sample budget spreadsheet and is presented to help illuminate the documentation technique discussed below.

Step 1 - Create the database

Since all my data are in SAS/CALC spreadsheet catalogs, I had to convert those catalogs into SAS datasets. Fortunately, SAS/CALC allows one to convert a spreadsheet into a SAS dataset fairly easily. The rows of the spreadsheet (the various equations) became the rows in the new dataset and the spreadsheets columns (spanning the time period of the budget) became the new columns. Including the row label was important since we want subsequent documentation

listings to be as descriptive as possible. The resultant SAS dataset is similar to Figure 2.

After all the data values are in SAS datasets, PROC FORMAT was used to create look-up functions that will map a variable name to an observation in the respective SAS dataset. For the above dataset, the PROC FORMAT code could appear as:

```
proc format;
value $eq
    'row11page1' = 1
    'row6page2' = 2;
run;
```

Step 2 - Parsing the Code

Probably the hardest step is to convert the code behind the spreadsheets (the budget equations) into datasets so I could do table look-ups. This step will be similar for all SAS developers; it does not matter whether we are using the SAS language, Screen Control Language (SCL) code, or the equivalent in SAS/CALC. Save the source code to an ASCII file so that it can be parsed and stored as a SAS dataset. The equation coding for our example is as follows:

```
month1.row11.page1:month21.row11.page1 =
    month1.row8.page1:month21.row8.page1 +
    month1.row9.page1:month21.row9.page1;

row6.page2(i) =
    (begbal05.page2(i) +
    (row15.page1(i) * 1000 * 1/6)) *
    irdecom.page2(i) * .001;
```

variable name	variable label	Jul95	Aug95
ROW11	Total Deposit to Bond Fund	24,535.82	24,535.82
ROW6	Decommissioning Trust	324.68	327.49

Figure 2. Sample spreadsheet converted into a SAS dataset

A little explanation is in order here for readers not familiar with SAS/CALC:

a) Spreadsheets can be three-dimensional; in those cases, each cell of the spreadsheet is uniquely identified as a column:row:page triplet. In our example, we have labeled our columns as months.

b) The colons in equation 1 basically indicate that the same equation is used for each month of the budget cycle. That is, we will add the row 8 value on page 1 for July 95 to the row 9, page 1, July 95 value to get the row 11, page 1, July 95 answer. Similar additions are done for August 95 and the remaining 19 months of the cycle. The important information is that row 8 is added to row 9 to get the row 11 value.

c) Equation 2 is really contained in a DO-loop; the (i) indices refer to the months spanned by the DO. I will not cover DO-loop or conditional statement (e.g., IF-THEN) documentation since I feel that I have not really found a good method for those statements. For this paper, I will focus on the equation in a more static sense and its nesting (parentheses).

Each equation must be broken into two parts: the result (left-hand side of the equals sign) and the operators/operands (right-hand side). There will always be just one observation per equation in the former (after all, we can't have two results from a single equation!!) and possibly many in the latter. No problem here - as long as the equations are numbered, we can do a 1-to-many merge to recombine the data.

For our two equations, the left-hand and right-hand SAS datasets that hold the parsed equations are seen at the bottom of this page.

We have one observation in the left-hand dataset for each row-page combination (i.e., equation). The variable EQ_NUM contains the equation number and ROWLABEL is included so that subsequent HTML pages can be more descriptive. LROW and LPAGE do more than just identify the equation; their combined value provides the mapping for the user-written look-up scheme (remember the PROC FORMAT above?) that allows us to bring the monthly values themselves into the dataset. The resultant dataset contains both the description and value of the equation result.

Sample Left-Hand Dataset

EQ_NUM	LROW	LPAGE	ROWLABEL	JUL95	AUG95	...	remaining months
1	row11	page1	Total Deposit to Bond Fund	14234.68	14234.68		
2	row6	page2	Decommissioning Trust	471.93	476.00		

Sample Right-Hand Dataset

EQ_NUM	DEPTH	RROW	RPAGE	RROWLAB	RJUL95	RAUG95
1	0	row8	page1	Interest Account	10859.680	10859.680
1	0	+				
1	0	row9	page1	Principal and Retirement Account	3375.000	3375.000
2	-1	bal			47124603.000	47532183.000
2	-1	+				
2	-2	row15	page1	Decommissioning Fund	407.580	407.580
2	-2	*				
2	-2	1000			1000.000	1000.000
2	-2	*				
2	-2	1			1.000	1.000
2	-2	/				
2	-2	6			6.000	6.000
2	0	*				
2	0	irdecom	page2	Interest Rate Decommissioning Fund	0.010	0.010
2	0	*				
2	0	.001			0.001	0.001

The right-hand dataset is really only a bit more complicated. We have one observation for each operand or operator in the equation. The RROW and RPAGE variables fully describe each operand so that user-written formats can be used to bring the monthly values of those variables into the dataset. The values of equation constants (e.g., 1000) can be considered monthly values also. Similar to the left-hand dataset, EQ_NUM allows the two datasets to be merged and RROWLAB is for descriptive purposes. Only one variable yet to discuss - DEPTH. Nesting (parentheses) controls the order of arithmetic operation in the SAS language, so we must keep track of that information as we parse the equations. Equation 1 contains no parentheses, so all operators/operands are on the "surface" - no nesting. The DEPTH is zero. Looking at equation 2, we see that the right-hand dataset observations referring to the equation part

row15.page1(i) * 1000 * 1/6

all have DEPTH=-2. The -2 refers to the second pair of parentheses that contains this part of the equation. Those arithmetic operations would be performed first (since they have the deepest nesting). In turn, that intermediate result would be added to BEGBAL05.PAGE2 next since the

intermediate result can be considered having DEPTH=-1. Finally, this secondary intermediate result (in effect having DEPTH=0) would be multiplied by IRDECOM and .001 to get the final answer. Negative values are used for DEPTH to reinforce the idea that you must perform the arithmetic operations from the greatest depth of nesting to the surface. It eases subsequent documentation work to explicitly perform those intermediate calculations and include them as additional observations in the right-hand dataset. The new right-hand dataset is seen at the bottom of this page.

Note: the last observation for each equation now contains its result. You can merge these observations with the left-hand dataset and compare the monthly values. If those values agree, you know you have successfully parsed the code.

Step 3 - Create HTML image of spreadsheet

Now that the left-hand and right-hand phase II datasets are set up, we can think about creating HTML files from that information and what alternative methods to pursue. Figure 3 is a screen snapshot illustrating how an information browser such as Netscape may present the original spreadsheet segment.

Sample Right-Hand Dataset - Phase II						
EQ_NUM	DEPTH	RROW	RPAGE	RROWLAB	RJUL95	RAUG95
1	0	row8	page1	Interest Account	10859.680	10859.680
1	0	+				
1	0	row9	page1	Principal and Retirement Account	3375.000	3375.000
1	0	result		Total Deposit to Bond Fund	14234.680	14234.680
2	-1	bal			47124603.000	47532183.000
2	-1	+				
2	-2	row15	page1	Decommissioning Fund	407.580	407.580
2	-2	*				
2	-2	1000			1000.000	1000.000
2	-2	*				
2	-2	1			1.000	1.000
2	-2	/				
2	-2	6			6.000	6.000
2	-1	int1		intermediate result #1	67930.000	67930.000
2	0	int2		intermediate result #2	47192533.000	47600113.000
2	0	*				
2	0	irdecom	page2	Interest Rate Decommissioning Fund	0.010	0.010
2	0	*				
2	0	.001			0.001	0.001
2	0	result		Decommissioning Trust	471.925	476.001

Category		jul96	aug96
ROW11	Total Deposit to Bond Fund	<u>14265.3</u>	<u>14265.3</u>
ROW6	Decommissioning Trust	<u>407.58</u>	<u>407.58</u>

Figure 3. Sample Netscape display of spreadsheet segment

Let's examine the HTML (HyperText Markup Language) code for this display:

```
<HTML>
<HEAD>
<TITLE> Rate Setting Worksheet </TITLE>
</HEAD>
<BODY>
<TABLE BORDER>
<TR>
<TH colspan=2>Category</TH>
<TH>jul96</TH>
<TH>aug96</TH>
</TR>
<TR>
<TD>ROW11</TD>
<TD>Total Deposit to Bond Fund </TD>
<TD ALIGN=RIGHT><A HREF=f00101.dat>14265.3
</A></TD>
<TD ALIGN=RIGHT><A HREF=f00101.dat>14265.3
</A></TD>
</TR>
<TR>
<TD>ROW6</TD>
<TD>Decommissioning Trust </TD>
<TD ALIGN=RIGHT><A HREF=f00201.dat>407.58
</A></TD>
<TD ALIGN=RIGHT><A HREF=f00202.dat>407.58
</A></TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

HTML provides formatting instructions to the information browser via the <instruction> tags where instruction describes the tag type. HTML files always start with a <HTML> tag and end with a </HTML>. In general, tags always have complementary /tags that end the instruction. The <HEAD> tag that follows the <HTML> and describes the document header. The remainder of the document is enclosed within the

<BODY>, </BODY> tag pair. The body is the main display area and is what is shown in Figure 3. We use a <TABLE>, </TABLE> tag pair to present the spreadsheet facsimile. In HTML, a table is a rectangular arrangement of cells. Each row of a table is described within a <TR>, </TR> tag pair and the cells within a row are contained within <TD>, </TD> pairs. The <TH>, </TH> pairs describe the information for the table column headings.

Refer back to Figure 3. Note that the cells are either text or underlined numbers. The HTML code for the top-left corner of the figure is

```
<TD>ROW11</TD>
```

Pretty simple - just place the text between the table cell tags. Now let's look at the bottom-right corner of the table, the underlined number 407.58. The underline means that the information is linked this page to additional information. In our case, we want to display how that number was calculated. The code for this cell is:

```
<TD ALIGN=RIGHT><A HREF=f00202.htm>407.58
</A></TD>
```

We tell the information browser to right-align the number in the <TD> cell definition. Within that tag pair we have an additional <A>, anchor pair which allows us to define a link (the Hypertext REFerence). We have a number of ways we can define this link.

Step 4a - Brute Force

The brute force method is just to provide links to other HTML documents. For this method to

be successful, we need a naming convention for these files. The naming convention used above is

xxxxyy where xxx is the equation number

and

yy is the month (1, 2,)

The .htm file extension identifies that the file contains HTML code. Thus, the anchor tag

[HREF=f00202.htm](#)

is a hypertext reference to the file containing the computation for equation 2, month 2.

If you have a moderate number of equations, the brute force method is probably acceptable. It is easy to write a bit of SAS code that builds an HTML file for each equation/monthly value combination. Each of the HTML files is relatively small, so the delivery and display of that information will be suitably fast. If all those files reside on a local disk, you need not even be on the Internet or an intranet. The ability of the browser to be a client is not needed since only a formatting/presentation tool is required.

The brute force method begins to break down when you have a substantial number of equations. Even though each individual HTML file is small, the disk space to store hundreds or even thousands of them becomes substantial. This is the case with the ElectricCities budget - it covers 21 months of time and roughly 500 equations for each. Managing 10000-odd files occupying 20+ megabytes of disk space is a chore that I would like to avoid.

Step 4b - Writing HTML on the Fly

Earlier in the paper, I noted that HTML provides us with many techniques for certain situations. How can I avoid the 10,000 files problem? Well, let's look again at the HTML link:

```
<TD ALIGN=RIGHT><A HREF=f00202.htm>407.58
</A></TD>
```

The HREF link can also point to a Common Gateway Interface (CGI). A CGI is a reference to a script (e.g., in Perl or the native OS command language) or executable (e.g., written in C) that resides on a web server machine. We can now use the ability of the information browser to be a client and communicate its needs to another program (the server) by passing parameters via the CGI. So instead of placing 10,000 HTML files onto the server, all we need to do is convert the right-hand phase II SAS dataset into an ASCII flat file and store that flat file on the server. Only one file to maintain and that file will not be enormous even for thousands of equations. The HTML link to the equation computation will be similar to:

```
<TD ALIGN=RIGHT><A HREF=
http://web.serv/eqlook.pl?eqno=2+month=2>407.58</A><
/TD>
```

Http indicates a HyperText Transfer Protocol call to the machine web.serv (our web server). On that machine, we will execute the eqlook.pl Perl script. We pass two parameters to that script - the equation number and the month. The Perl script will take those parameters, look up the relevant information in the ASCII flat file, and return an HTML file to the client browser. Indeed, the HTML file will be identical to the one created in the brute force method so visually the user will see no difference.

I know what you are thinking - we have reduced our database management needs and the end result is the same to the user. Why not build the files on the fly? The answer is performance. Using CGI files forces the web server to do more work and whenever additional work must be done, performance will suffer. The links to the equation computations will not be as quick. However, since we build only small HTML files, the degradation is small.

Figure 4 shows the equation computation display and its associated HTML code.

Row	Page	Label	AUG 96	
begbal05	2		47532183.00	
row15	1	Decommissioning Fund		407.58
				* 1000.00
				* 1.00
				/ 6.00
intermediate result #1			+ 67930.00	
intermediate result #2			47600113.00	
irdecom	2	Interest Rate Decommissioning Fund	* 0.010	
			* 0.001	
final result:			476.001	

Figure 4. Sample equation computation display

Summary

At this time, it is pretty obvious building the HTML files on-the-fly is the best approach. The exact path along these lines is less obvious. The world of HTML and web servers is rapidly evolving. HTML 3.0 is the newest version of that specification but not all information browsers support it. HTML 3.2 is on the horizon as are new servers and clients. Web developers are

beginning to have little languages (such as Java) and powerful widgets (e.g., ActiveX) at their disposal. Advancements on these two fronts will allow the information provider to present more dynamic information tailored to specific users.

SAS and SAS/CALC are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ®Indicates US registration.

```

<HTML>
<HEAD>
<TITLE>Decommissioning Trust</TITLE>
</HEAD>
<BODY>
<TABLE BORDER>
<TR><TH>Row</TH><TH>Page</TH><TH>Label</TH><TH colspan=3>AUG 96</TH><TH></TH><TH></TH></TR>
<TR><TD>begbal05</TD><TD>2</TD><TD></TD><TD align=right>47532183.00</TD><TD></TD></TR>
<TR><TD>row15</TD><TD>1</TD><TD>Decommissioning Fund</TD><TD></TD><TD align=right>407.58</TD></TR>
<TR><TD></TD><TD></TD><TD></TD><TD align=right>* 1000.00</TD></TR>
<TR><TD></TD><TD></TD><TD></TD><TD align=right>* 1.00</TD></TR>
<TR><TD></TD><TD></TD><TD></TD><TD align=right>/ 6.00</TD></TR>
<TR><TD colspan=6><HR></TD></TR>
<TR><TD colspan=6>intermediate result #1</TD><TD align=right>+ 67930.00</TD></TR>
<TR><TD colspan=6><HR></TD></TR>
<TR><TD colspan=6>intermediate result #2</TD><TD align=right>47600113.00</TD></TR>
<TR><TD>irdecom</TD><TD>2</TD><TD>Interest Rate Decommissioning Fund</TD><TD align=right>* 0.010</TD><TD></TD></TR>
<TR><TD></TD><TD></TD><TD></TD><TD align=right>* 0.001</TD><TD></TD></TR>
<TR><TD colspan=6><HR></TD></TR>
<TR><TD colspan=6>final result: </TD><TD align=right> 476.001</TD><TD></TD></TR>
</TABLE>
</BODY>
</HTML>

```