Paper 085-30

ODS MARKUP: The SAS® Reports You've Always Dreamed Of

Eric A. Gebhart, SAS Institute Inc, Cary, NC

ABSTRACT

Find out why Output Delivery System (ODS) tagsets are going to take over the world! ODS tagsets are incredibly powerful tools that can simplify and conquer the worst of problems. New ODS markup destinations are being created all the time. One of the most powerful aspects of ODS markup is its ability to reuse and combine previous work to create more powerful and flexible ODS markup destinations.

INTRODUCTION

Many of the beautiful entities of our dreams are faceless and ephemeral. Their vague and abstract feeling of beauty leaves us with a desire for more. However, many escape as we run after them as if we are running in quicksand. Their lack of description makes them even harder to recreate.

SAS reporting is very much like those dreams—vague, abstract, impossible to recreate, and nothing compared to those ephemeral beauties of our night dreams. This is because we are pulled down by the legacy of what we know…or really…what we don't know. We tend to limit ourselves based upon what we think is possible. However, creating better SAS reports isn't a dream world. This paper casts a trampoline across the quicksand to enable you to capture your dreams and make them reality. You will learn new possibilities that will open your eyes to new ephemeral beauties in the world of SAS reporting that are waiting to be defined.

There are many new and exciting ways to improve your SAS reports. The few examples that are presented in this paper are chosen to show a range of output types and new ways to customize them. For example, Hypertext Markup Language (HTML) and Extensible Markup Language (XML) are the most common output types that the ODS MARKUP destination is used to create. This paper shows several HTML examples, one XML example with Microsoft Excel's SpreadsheetML, and a way to create PDF reports with LaTeX (a typesetting language) that would be impossible any other way. As a group, these examples will help you understand the power of the ODS MARKUP destination. These examples should spur your imagination to new solutions and ideas that you would have dismissed as impossible before now.

It is assumed that the reader has a basic familiarity with ODS, ODS styles, and the ODS MARKUP destination.

OUT OF THE QUICKSAND AND INTO THE AIR

The first step to creating the SAS reports of your dreams is to learn that it is possible to fly. With ODS MARKUP you can do things that you never thought you could do—like fly. ODS MARKUP is a framework, unlike any other, that allows the definition of new destinations through tagsets. A tagset is a type of template that defines how to generate output from the ODS MARKUP destination. Over 50 tagsets are shipped with SAS. All these tagsets can be used as the basis for new tagsets of your own design. A tagset can be general purpose, written specifically for a particular shape of data or even for a specific SAS program. Without a tagset, ODS MARKUP will do absolutely nothing.

The name ODS MARKUP implies that the output will be a markup language of some sort. That may seem limiting but not as much as you might think. Markup languages cover most, if not all, of the output types that anyone may desire, such as HTML and XML (which in itself covers a lot of output types). The number of applications that can load XML is nearly endless and growing everyday. Excel, Word, and OpenOffice all read XML. Other markup languages include the very powerful publishing languages of LaTeX and Troff which are both capable of creating very nice PDF, DVI, and postscript documents. PostScript is another markup language; I'll leave the tagset for you to create. Rich text format (RTF) is another very popular markup language. ODS MARKUP isn't restricted to markup languages. ODS MARKUP is equally capable of creating CSV and SQL output. It is possible to write PROC TEMPLATE code, even another tagset as the output from an ODS MARKUP tagset.

So, out of the quicksand and into the air. Let's fly with ODS MARKUP and create the SAS reports of our dreams.

DAYDREAMING

Have you ever wanted to put a SAS data set into an SQL database, but the network wasn't cooperating, or you lacked the best tools for the job? This is easy to do if you have SAS/CONNECT to connect to the database. However, a “low-tech” solution is sometimes easier to do than more superior methods and can be made to handle more complex data types, or to do updates instead of inserts.

This first example is rather simple, but it shows a different point of view that may expand the reporting possibilities in your dreams. It's not really a report either. The output of this SQL tagset is an SQL program that can be run against any SQL database to create a set of tables from the output of PROC PRINT. Usually, this sort of thing is written in DATA step for a specific data set. However, the nice thing about using this tagset is that it will work with any data set. Just run a series of PROC PRINTs on all your data. The output is an SQL program that creates a database for you.

```
ods tagsets.sql file="class.sql";
    proc print data=sashelp.class ;
 run;
 ods _all_ close;
```

The resulting output file looks like this:

```
 Create table CLASS
        (Name varchar(7), Sex varchar(1), Age float, Height float, Weight float);

 Insert into CLASS(Name, Sex, Age, Height, Weight)
        Values ('Alfred', 'M', 14, 69.0, 112.5);
 Insert into CLASS(Name, Sex, Age, Height, Weight)
        Values ('Alice', 'F', 13, 56.5, 84.0);
 .
 .
 .
```

The most interesting part of this tagset is the collection of column names and types for the CREATE TABLE statement. The Colspec_Entry catches all the columns, their types, and their widths as needed. The Output Event catches the Data Set name and uses DATA step functions to simplify it. By the time the observations start coming through, everything is ready for the CREATE TABLE statement

The insert statements are easily created by using the collected names and the data in each observation. The first step is to set everything up, including name and type translations, widths, and value quoting.

```
/*-----------------------------------------------------------------eric-*/
/*-- Set Everything up at the beginning.                             --*/
/*---------------------------------------------------------------11Feb04-*/
define event initialize;
    trigger type_translations;
    trigger name_translations;

    /* types that need widths */
    set $types_with_widths['string'] "True";

    /* types that need quotes */
    set $types_with_quotes['string'] "True";
end;

/*-----------------------------------------------------------------eric-*/
/*-- Set up some look-up tables for convenience.                     --*/
/*---------------------------------------------------------------11Feb04-*/
/* type translations */
define event type_translations;
    set $types['string'] 'varchar';
    set $types['double'] 'float';
    set $types['int']    'integer';
end;

/* column name translation */
define event name_translations;
```

```
    set $name_trans['desc'] 'description';
end;

define event colspec_entry;
    /*-----------------------------------------------------------eric-*/
    /*-- Ignore the obs column.  The value will get ignored because   --*/
    /*-- it will be in a header cell and we don't define a header     --*/
    /*-- event to catch it.                                            --*/
    /*-----------------------------------------------------------12Feb04-*/
    break /if cmp(name, 'obs');

    /*-----------------------------------------------------------------eric-*/
    /*-- Create a list of column names.  Translate the names          --*/
    /*-- if they are in the translate list.                           --*/
    /*-----------------------------------------------------------11Feb04-*/
    set $lowname lowcase(name);
    do /if $name_trans[$lowname];
        set $names[] $name_trans[$lowname];
    else;
        set $names[] $lowname;
    done;

    /* keep a list of types */
    set $col_types[] type;

    /* make a list of column type definitions */
    set $col_def $types[type];

    /* append width if needed */
    set $col_def $col_def "(" width ")" /if $types_with_widths[type];

    set $columns[] $col_def;
end;

/*-----------------------------------------------------------------eric-*/
/*-- Catch the data label and get the data set name from it.        --*/
/*-----------------------------------------------------------11Feb04-*/
define event output;
    start:
        set $table_name reverse(label);
        set $table_name scan($table_name, 1, '.');
        set $table_name reverse($table_name);
        set $table_name lowcase($table_name);
end;
```

This tagset and all its features are fully documented on the Base SAS Communities Web page (http://support.sas.com/rnd/base/topics/odsmarkup/sql.html).

SHARPENING PERCEPTIONS

You are dreaming of numbers. There are numbers everywhere; they are all different but somehow they all seem the same. Everything is fuzzy, out of focus, and the numbers aren't making sense. You want to read them, but can't. You think “Gee, this is a dream, but what I really want is for all the numbers to show their value relatively as boxes of variable size.” Suddenly all the numbers are a myriad of boxes of all different sizes (Figure 1). They are now easy to see and interpret.

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0

Figure 1. HTML Table Showing Data and Relative Values

This HTML example shows how adding a simple tagset can make all the difference in data presentation. Our dream of desire is to have a bar graph for each value in a given table column. There are complicated ways to do this with PROC REPORT and PROC GRAPH. However, this method only uses a tagset and will work with any PROC. Creating a bar in an HTML table cell is easy. The HTML code looks something like this.

```
<td class="data">
   <table width="100%">
     <tr>
         <td class="Header" width="45%"> 45% </td>
         <td></td>
     </tr>
   </table>
</td>
```

The table goes inside what would normally be the data cell. Its width is 100% which means that the data cell spans the entire width that is available to it. Inside that table are two cells. The first cell uses the header style so it will be in contrast to the data cells around it. Its width is a percentage of the total value of the cell. Following that is an empty cell which is added to take up the rest of the space. This solution works best if the values are percentages. The new tagset uses the HTML4 tagset as its parent. All the usual behavior of ODS HTML will be inherited. This tagset will add the slidebar functionality.

When it comes to tagsets, special behavior needs a cue for when it should act. This tagset is a simple example. It is not desirable to put bars on all columns so we need to tell the tagset where the bars should go. If we are going to tell the tagset where to place the bars, we might as well also tell it the columns' maximum value. We could use the header style for the slidebars, but the blue header style wouldn't match the rest of the data. Creating a new slider style will make everything match nicely.

Something like the following ODS style and PROC PRINT works nicely. The TAGATTR style attribute is an unused element that begs to be used and abused. It is useful for giving information to the tagset.

```
proc template;
    define style styles.slider;
        parent=styles.default;

        style table from output /
            rules=cols
            cellpadding=3
            cellspacing=1
        ;
        style slider from data /
            background=colors('headerbg')
        ;
    end;
run;

proc print data=sashelp.class;
    var name;
    var sex;
    var age;
    var height / style(data) = slider[just=center tagattr="slider-80"];
    var weight / style(data) = slider[just=center tagattr="slider-150"];
run;
```

Next, we need to tell the tagset how to calculate a percentage if it's given a maximum number. The following event parses the TAGATTR value and does the necessary math. If the value of the cell is a percentage, $width is set to that percentage. Otherwise some math is necessary to create a percentage. The resulting value of $width is all the tagset needs to create a bar:

```
   define event calculate_width;
    set $width value /breakif index(value, '%') > 0;

    eval $dash_pos index(Tagattr, '-');
    do /if $dash_pos;
        eval $dash_pos $dash_pos+1;
    done;
    eval $value inputn(value, '12.');
    eval $max inputn(substr(Tagattr, $dash_pos), '12.');
    eval $width ($value / $max) * 100;
    set $width $width '%';
    unset $dash_pos;
end;
```

The only other part of the tagset is the Data Event. Most of it was cut and pasted from the original data event in the htmlcss tagset. The new parts are added at the beginning of the start section and at the very end of the finish section. Aside from these two parts, the only other addition is printing the $width variable.

```
start:

    do /if index(tagattr, 'slider') > 0;
        put '<td width="150" class="data">' nl;
        put '<table width="100%" class="data" cellpadding=0; cellspacing=0;>' nl;
        trigger calculate_width;
    done;

    ....

finish:

    ...

    do /if $width;
        put '<td> </td>' CR;
        put "</table></td>" CR;
        unset $width;
    done;
```

The program that is used to create the example output (Figure 1) looks like this.

```
options obs=3;
ods tagsets.slider file='test.html' style=slider;

proc print data=sashelp.class;
    var name;
    var sex;
    var age;
    var height / style(data) = slider[just=center tagattr="slider-80"];
    var weight / style(data) = slider[just=center tagattr="slider-150"];
run;

ods tagsets.slider close;
```

To enable reuse, the style and tagset should be separated and placed in an itemstore that everyone can access. No one needs to know how the tagset works; they just need to know how to use it. This PROC PRINT statement is all the introduction they need.

TRAPPED IN A NIGHTMARE

Have you ever had a nightmare about Scrolling Tables (Figure 2)? Scrolling tables seem to be something everyone dreams of when their HTML tables start getting very long. It becomes hard to tell which column of data you are reading. As desirable as that is, most of us have dismissed scrolling tables as an impossible dream. This tagset can free you from this nightmare.

Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0
7	Jane	F	12	59.8	84.5
8	Janet	F	15	62.5	112.5

1-8 of 19

Figure 2. Scrolling HTML Tables

It's easy to wake up from this nightmare. Kevin Smith, another SAS ODS developer, wrote a JavaScript program that applies the appropriate scrollbars to any tables it encounters. The only thing that was needed was to load the JavaScript from the HTML page. Creating the tagset is just a matter of copying the JavaScript into the tagset at the proper place. The result is an HTML destination that writes extra JavaScript to a code file or to the body file as desired. This tagset uses the HTML4 tagset as its parent; the only added behavior is support for the new JavaScript code that creates the scrolling tables.

The trickiest part of loading the JavaScript from the HTML page is knowing where to print the JavaScript. An event called scroll_code defines how the JavaScript should print. Triggering the scroll code event will cause the entire JavaScript to be written to the output. Scroll_code is an event of convenience. Once the proper print locations are found, triggering scroll_code prints all the JavaScript that we need. Those print locations are these two events, which are internally generated by ODS.

```
define event code_body;
    trigger scroll_code;
    trigger contents_code;
end;

define event JavaScript;
    start:
        trigger javascript_tag start;
        trigger scroll_code /if !any(code_name, code_url);
    finish:
        trigger javascript_tag finish;    end;
```

It is also important to ensure that a link is made if the JavaScript is written to a JavaScript file. Code_link is another ODS event that was created for this purpose.

```
define event code_link;
    put "<script";
    putq " src=" code_name /if !exists(code_url);
    putq " src=" code_url /if exists(code_url);
    put ' language="JavaScript" type="text/JavaScript">';
    put "</script>" CR;
    put "<noscript></noscript>" CR;
end;
```

Using the tagset is as simple as this.

```
ods tagsets.htmlscroll file="test.html";

proc print data=sashelp.class;
run;

ods _all_ close;
```

Putting all the JavaScript into a separate file only requires adding the code file to the ODS statement.

```
ods tagsets.htmlscroll file="test.html" code="test.js";

proc print data=sashelp.class;
run;

ods _all_ close;
```

This tagset and all of its features are fully documented on the Base SAS Communities Web page (http://support.sas.com/rnd/base/topics/odsmarkup/htmlscroll.html)

MAKING DREAMS COME TRUE

Everyone who needs to publish output to a Web site knows what they have to do in order to get their reports accepted by the Web site administrator: They must meet the Web site administrator's well-defined and lengthy list of requirements. In our dream, our goal is well-defined. We just wish we could trade in our wading boots for a magic carpet so that we could publish our Web report easier. Web site integration can be a difficult and painful process. But it doesn't have to be. This example shows you how easy it is to make this dream come true.

Integrating ODS HTML output into an existing Web site (Figure 3) is not as seamless as it should be. Many Web site administrators have specific requirements for the HTML pages they publish. Most of those requirements are easily met by embedding the body of our HTML output between an HTML header and footer that is designed by those Web site administrators. There are ways to do this using the NO_TOP_MATTER and NO_BOTTOM_MATTER HTML options. However, these methods require you to juggle several files before a Web page is ready for publishing. The options also require the same painful techniques for each SAS program that is targeting its output to the Web.

The following simple tagset addresses all our needs and can be used by anyone, without needing to know or care about all the requirements for their Web sites.

The most important piece of this solution is reading an external file. A simple example of this is documented on the Base SAS Communities Web page (http://support.sas.com/rnd/base/topics/odsmarkup/tagsets.html#exp14).

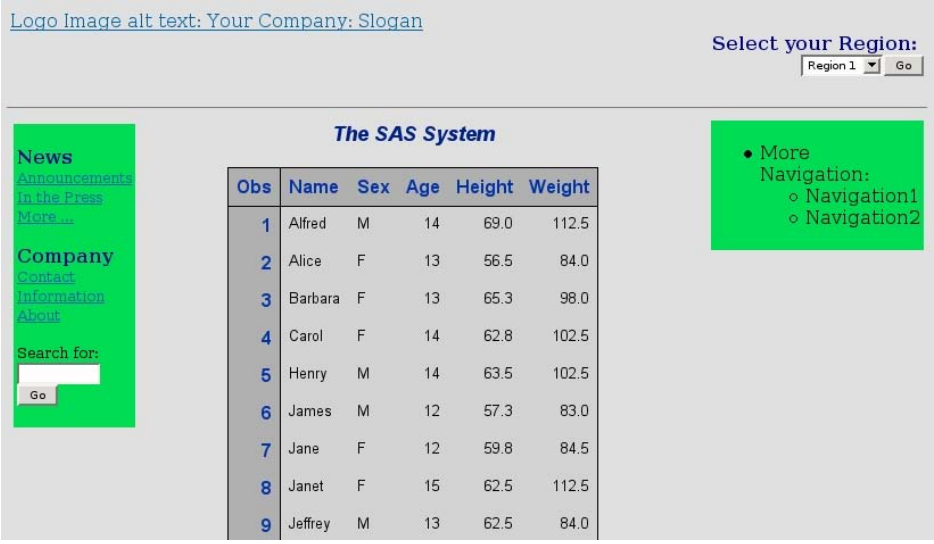


Figure 3. ODS HTML Output

Reading a file is easy, but there are other more minute details that need attention first. For example, most Web sites don't like embedded stylesheets. The Web administrator wants everything to look consistent, which means that you need to use their stylesheet. A link to that stylesheet is often embedded in the file header that this tagset is going to read. However, SAS output still needs some style information that is not defined in the Web site stylesheet. There are two ways to solve this problem. The first, and best, way is to ask your Web administrator to include the necessary style definitions in their stylesheet. The second way is to modify their header to load an ODS-generated stylesheet after it loads the Web site's stylesheet.

The first step is to disable embedded stylesheets for the new tagset. Adding this EMBEDDED_STYLESHEET attribute to the top of the tagset will do just that.

```
embedded_stylesheet=no;
```

The next step is to look for the closing </head> tag in the header file. If a stylesheet was specified on the ODS statement, insert a link to the stylesheet there. These two events handle that, and at the same time allow for easy redefinition through inheritance if modification is ever needed. The Integrated link event is mostly stolen code from the usual stylesheet_link event.

```
define event head_search;
    trigger integrated_link /if contains($file_record, '</head>');
end;

define event integrated_link;
    put '<style type="text/css">' CR '<!--' CR;
    trigger alignstyle;
    put '-->' CR '</style>' CR ;

    set $urlList stylesheet_url;
    set $urlList stylesheet_name /if !$urlList;
    trigger urlLoop ;
    unset $urlList;
end;
```

The only other missing piece is naming the header and footer files that are read. The following tagset has several paths to that end. First there is an event that can be easily redefined through inheritance. The Set_default_files event creates an $infiles variable with the names default_head.html and default_foot.html. There is also a macro variable called infiles that can be set with two file names. The last way is to use the Tagset ALIAS option. The alias is set on the ODS statement. Setting it to 'default' will cause it to use the default files. Anything else will cause the tagset to attempt to read the filenames that are given. This tagset will behave just like ODS HTML if neither alias nor

the infiles macro variable are set. One exception is that ODS HTML allows an embedded stylesheet while this tagset does not.

```
mvar infiles;

define event set_default_files;
   set $infiles 'default_head.html default_foot.html';
end;

/*-------------------------------------------------------------eric-*/
/*-- The files should be given as a list with spaces between them.--*/
/*-----------------------------------------------------------8Nov 03-*/
define event initialize;
    /*trigger set_just_lookup;*/

    do /if cmp(tagset_alias, 'default');
        trigger set_default_files;
    else;
        set $infiles tagset_alias;
        set $infiles infiles /if !$infiles;
    done;
    putlog "Infiles are" " :" $infiles;

    set $filename scan($infiles, 1, ' ');

    /*-------------------------------------------------------eric-*/
    /*-- Set a flag so the ending document tags can be suppressed.--*/
    /*-----------------------------------------------------8Nov 03-*/
    do /if $infiles);
        set $read_files 'true';
        set $filrf "headfile";
        set $read_head "true";
    done;
end;
```

The last step is to suppress the top and bottom matter in the HTML output. Specifying NoTop works great, but, without a bottom, the tagset never gets a cue for when it should read in the Web site footer. The tagset gets around this by keeping track of when it's reading Web files. When it is reading files, the bottom matter is suppressed and replaced with the Web site's footer file. Both the doc event and the body event break early if $readfiles is set.

```
define event doc;
    start:
       set $doctype '<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">';
       set $framedoctype '<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">';
        put $doctype CR;
        put "<html>" CR;

    finish:
        do /if $read_files;
            set $filename scan($infiles, 2, ' ');
            set $filrf "footfile";
            trigger readfile;
            break;
        done;

        put "</html>" CR;
end;

define event doc_body;
    put '<body onload="startup()"';
    put ' onunload="shutdown()"';
    put  ' bgproperties="fixed"' / WATERMARK;
    putq " background=" BACKGROUNDIMAGE;
       trigger style_inline;
    put ">" CR;
    trigger pre_post;
    put          CR;
    trigger ie_check;
```

```
  finish:
    trigger pre_post;

    break /if $read_files;
    put "</body>" CR;
end;
```

The following program shows how the tagset is used and is the same program that created the output shown in Figure 3.

```
ods tagsets.website stylesheet="website.css"
                         file="website.html"(notop)
                        alias='default';

proc print data=sashelp.class; run;

ods tagsets.website close;
ods _all_ close;
```

THE BEST DREAM YOU EVER HAD JUST GOT BETTER

In reality, you load ODS output into Excel and it works. In your dreams, not only does it work, it looks great, with proper column widths, fonts, and colors. Numbers are always numbers. Worksheets bend to your will, with names of your own choosing. Subtotals actually recalculate, and negative numbers are formatted differently for easy viewing. Well, your dream just became reality.

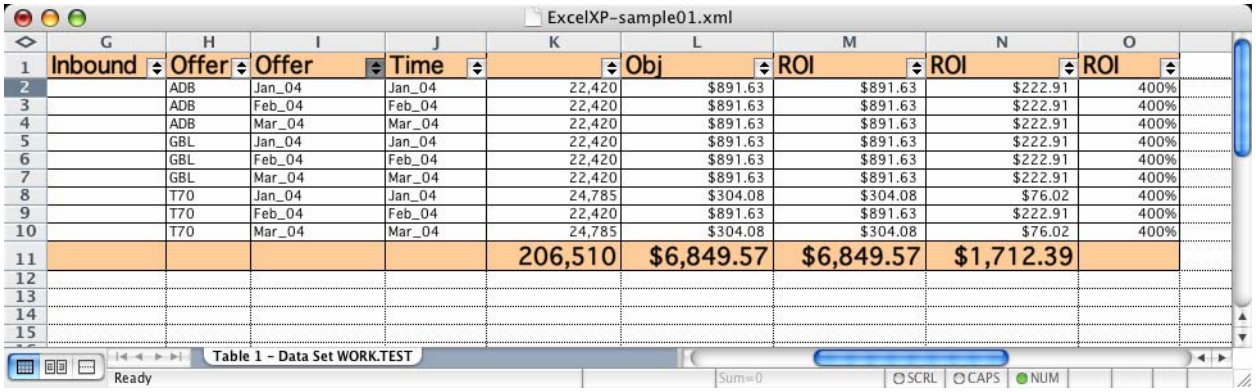
There are many ways to get SAS output into Excel, and every one of them can or do use ODS MARKUP. The simplest way is using CSV files. Next is HTML; last and most painfully, is Dynamic Data Exchange (DDE). While it is possible to use a tagset with DDE, most people use DATA Step. XML is by far the best way to load SAS output into Excel. XML has many advantages over all of the other methods, such as style, formats, numbers, formulas, better column widths, worksheet control, worksheet labels, landscape or portrait, auto filters, frozen header panes, and printable titles. XML falls short with graphs—but graphs are the least of your worries if you are loading data into Excel.

The ExcelXP tagset was complex when it started. Everyone I talk to seems to have another idea about how it should work and what it should do. The features have become so numerous that I added Help to explain all the options. You can obtain the Help with the following ODS Statement.

```
ods tagsets.excelxp file="test.xml" options(doc='help');
```

The Help text reveals many options including, orientation, frozen_headers, frozen_rowheaders, autofilter, autofilter_table, default_column_widths, sheet_interval, sheet_name, sheet_label, and auto_subtotals.

Pictures don't do it justice—actually using Excel is much more telling. Figure 4 verifies that this is the SAS and Excel integration of your dreams. Do you see the autofilter buttons, subtotals, percentages, and the complex worksheet name?



ExcelXP-sample01.xml

	G	H	I	J	K	L	M	N	O
1	Inbound	Offer	Offer	Time		Obj	ROI	ROI	ROI
2		ADB	Jan_04	Jan_04	22,420	$891.63	$891.63	$222.91	400%
3		ADB	Feb_04	Feb_04	22,420	$891.63	$891.63	$222.91	400%
4		ADB	Mar_04	Mar_04	22,420	$891.63	$891.63	$222.91	400%
5		GBL	Jan_04	Jan_04	22,420	$891.63	$891.63	$222.91	400%
6		GBL	Feb_04	Feb_04	22,420	$891.63	$891.63	$222.91	400%
7		GBL	Mar_04	Mar_04	22,420	$891.63	$891.63	$222.91	400%
8		T70	Jan_04	Jan_04	24,785	$304.08	$304.08	$76.02	400%
9		T70	Feb_04	Feb_04	22,420	$891.63	$891.63	$222.91	400%
10		T70	Mar_04	Mar_04	24,785	$304.08	$304.08	$76.02	400%
11					206,510	$6,849.57	$6,849.57	$1,712.39	

Table 1 - Data Set WORK.TEST

Ready

Figure 4. SAS and Excel Integration

This first example uses many things—auto filter, frozen headers, automatic subtotal formulas, and specific formats and formulas for some columns, and parenthesized negative values that are highlighted in red.

The very first thing it does is to create a new style to better define the table borders for Excel. Excel is limited when it comes to borders, so this style accommodates Excel's needs.

```
proc template;
  define style styles.XLStatistical;
    parent = styles.Statistical;

    style Header from Header /
      borderwidth=2;

    style RowHeader from RowHeader /
      borderwidth=2;

    style Data from Data /
      borderwidth=2;
  end;
run; quit;
```

The ODS statement is where most of the options are turned on. Frozen headers, autofilters, and automatic subtotals are all specified.

```
ods tagsets.excelxp file='ExcelXP-sample02.xml' style='XLStatistical'

    options(autofilter='all' frozen_headers='yes' frozen_rowheaders='yes'
auto_subtotals='yes');
```

The actual PROC PRINT code is where the format, formula, and style overrides are specified for particular columns in the PROC PRINT output.

```
proc print data=prdsale noobs label;
  id country region division;
  var prodtype product quarter month year;
  var predict actual / style={tagattr='format:$#,##0_);[Red]\($#,##0\)'};
  var difference / style={tagattr='format:$#,##0_);[Red]\($#,##0\) formula:RC[-1]-RC[-2]'};
  sum predict actual difference / style={tagattr='format:$#,##0_);[Red]\($#,##0\)'};;
run; quit;
```

All of these fancy features are great, but most of us were dreaming of multiple worksheets and having the ability to control their names. Excel is only capable of displaying one table per worksheet. The XML schema dictates that there is one table per worksheet. That is what the ExcelXP tagset does by default. Making it appear as if there is more than one table per worksheet is a bit more work. At first, it seemed that everyone dreamed of a single table per worksheet. That is one of the things that is hard to do with CSV and HTML. It soon became apparent that there

are just as many people who like multiple tables as those who do not. It took a fair amount of work, but you can use the Excel XP tagset to display multiple tables per worksheet. At least, it appears that way. Appearances are everything, especially in our dreams.

The ExcelXP tagset handles multiple tables well. It even allows for auto filters to be applied to the table of your choice. Here is an example that has a PROC PRINT and a PROC TABULATE displayed across three worksheets, with two tables per region (Figure 5). Autofilters are placed on the second table.

```
data prdsale;
  set sashelp.prdsale;
  Difference = actual-predict;
run;

proc sort data=prdsale; by country region division year; run; quit;

title; footnote;

/*
  Create a workbook with multiple tables per worksheet, and specify
  the name for the worksheets.  Autofilters will be applied to the
  second table in each worksheet.  The SUM statement of PROC PRINT
  will result in a subtotal row, but note that the SUBTOTAL function
  is not used for the DIFFERENCE column because a formula was
  specified on the SUM statement.  In this case, the user-specified
  formula takes precedence over the auto subtotal.
*/

ods listing close;
ods tagsets.excelxp file='ExcelXP-sample09.xml' style='XLStatistical'
    options(auto_subtotals='yes'
            default_column_width='7, 10, 10, 7, 7'
            frozen_rowheaders='yes'
            sheet_interval='none'
            sheet_name='Canada'
            autofilter='all'
            autofilter_table='2');

*;
*  The output from the following two procs will be in a single worksheet
*  with a user-specified name of 'Canada'.
*;

  proc tabulate data=prdsale;
    where country eq 'CANADA' and year eq 1993;
    var predict actual;
    class region division prodtype;
    table
     region*(division*prodtype all={label='Division Total'}) all={label='Grand Total'},
      predict={label='Total Predicted Sales'}*f=dollar10.*sum={label=''}
      actual={label='Total Actual Sales'}*f=dollar10.*sum={label=''};
  run; quit;

  proc print data=prdsale noobs label split='*';
    where country eq 'CANADA' and year eq 1993;
    id country region division;
    var prodtype product quarter month year;
    sum predict / style={tagattr='format:Currency'};
    sum actual / style={tagattr='format:Currency'};
    sum difference / style={tagattr='format:Currency formula:RC[-1]-RC[-2]'};
    label prodtype = 'Product*Type'
          predict  = 'Predicted*Sales'
          actual   = 'Actual*Sales';
  run; quit;

  ods tagsets.excelxp options(sheet_interval='none' sheet_name='Germany');

*;
*  The output from the following two procs will be in a single worksheet
*  with a user-specified name of 'Germany'.
*;
```

```
  proc tabulate data=prdsale;
    where country eq 'GERMANY' and year eq 1993;
    var predict actual;
    class region division prodtype;
    table
     region*(division*prodtype all={label='Division Total'}) all={label='Grand Total'},
      predict={label='Total Predicted Sales'}*f=dollar10.*sum={label=''}
      actual={label='Total Actual Sales'}*f=dollar10.*sum={label=''};
  run; quit;

  proc print data=prdsale noobs label split='*';
    where country eq 'GERMANY' and year eq 1993;
    id country region division;
    var prodtype product quarter month year;
    sum predict / style={tagattr='format:Currency'};
    sum actual / style={tagattr='format:Currency'};
    sum difference / style={tagattr='format:Currency formula:RC[-1]-RC[-2]'};
    label prodtype = 'Product*Type'
          predict  = 'Predicted*Sales'
          actual   = 'Actual*Sales';
  run; quit;

  ods tagsets.excelxp options(sheet_interval='none' sheet_name='United States');

*;
*  The output from the following two procs will be in a single worksheet
*  with a user-specified name of 'United States'.
*;

  proc tabulate data=prdsale;
    where country eq 'U.S.A.' and year eq 1993;
    var predict actual;
    class region division prodtype;
    table
     region*(division*prodtype all={label='Division Total'}) all={label='Grand Total'},
      predict={label='Total Predicted Sales'}*f=dollar10.*sum={label=''}
      actual={label='Total Actual Sales'}*f=dollar10.*sum={label=''};
  run; quit;

  proc print data=prdsale noobs label split='*';
    where country eq 'U.S.A.' and year eq 1993;
    id country region division;
    var prodtype product quarter month year;
    sum predict / style={tagattr='format:Currency'};
    sum actual / style={tagattr='format:Currency'};
    sum difference / style={tagattr='format:Currency formula:RC[-1]-RC[-2]'};
    label prodtype = 'Product*Type'
          predict  = 'Predicted*Sales'
          actual   = 'Actual*Sales';
  run; quit;
ods tagsets.excelxp close;
```

ExcelXP-sample09.xml

	A	B	C	D	E	F	G	I	J	K
1				Total Predicted Sales	Total Actual Sales					
2	Region	Division	Product type							
3			FURNITURE	$11,081.00	$12,483.00					
4		CONSUMER	OFFICE	$21,939.00	$16,991.00					
5			FURNITURE	$12,972.00	$14,467.00					
6		EDUCATION	OFFICE	$16,434.00	$20,189.00					
7	EAST	Division Total		$62,426.00	$64,130.00					
8		Division	Product type							
9			FURNITURE	$10,286.00	$10,380.00					
10		CONSUMER	OFFICE	$16,042.00	$16,371.00					
11			FURNITURE	$12,816.00	$11,234.00					
12		EDUCATION	OFFICE	$17,759.00	$18,905.00					
13	WEST	Division Total		$56,903.00	$56,890.00					
14	Grand Total			$119,329.00	$121,020.00					
15										
16	Country	Region	Division	Product	Product	Quarter	Month	Predicted	Actual	Difference
197	CANADA	WEST	EDUCATION	FURNITURE	SOFA	1	Jan	$421.00	$72.00	($349.00)
198	CANADA	WEST	EDUCATION	FURNITURE	SOFA	1	Feb	$433.00	$926.00	$493.00
199	CANADA	WEST	EDUCATION	FURNITURE	SOFA	1	Mar	$394.00	$850.00	$456.00
200	CANADA	WEST	EDUCATION	FURNITURE	SOFA	2	Apr	$338.00	$826.00	$488.00
201	CANADA	WEST	EDUCATION	FURNITURE	SOFA	2	May	$764.00	$651.00	($113.00)
202	CANADA	WEST	EDUCATION	FURNITURE	SOFA	2	Jun	$216.00	$854.00	$638.00
203	CANADA	WEST	EDUCATION	FURNITURE	SOFA	3	Jul	$96.00	$899.00	$803.00
204	CANADA	WEST	EDUCATION	FURNITURE	SOFA	3	Aug	$550.00	$309.00	($241.00)
205	CANADA	WEST	EDUCATION	FURNITURE	SOFA	3	Sep	$636.00	$943.00	$307.00
206	CANADA	WEST	EDUCATION	FURNITURE	SOFA	4	Oct	$427.00	$138.00	($289.00)
207	CANADA	WEST	EDUCATION	FURNITURE	SOFA	4	Nov	$652.00	$99.00	($553.00)
208	CANADA	WEST	EDUCATION	FURNITURE	SOFA	4	Dec	$478.00	$270.00	($208.00)
257								$5,405.00	$6,837.00	$1,432.00

Canada Germany United States

Figure 5. SAS and Excel Integration

The previous example manipulated the worksheets manually. It is also possible to tell the tagset to break up the pages according to Procedure, Page, Table, or None.

This next example uses the procedure interval. It also does some rather dream-like, nonsensical subtotals of height and weight. The tagset correctly creates subtotal functions for the height and weight columns in each table, despite that they are all really one table. The bylines are also printed above each table as shown in Figure 6. Just like in our dreams, not everything is how it appears.

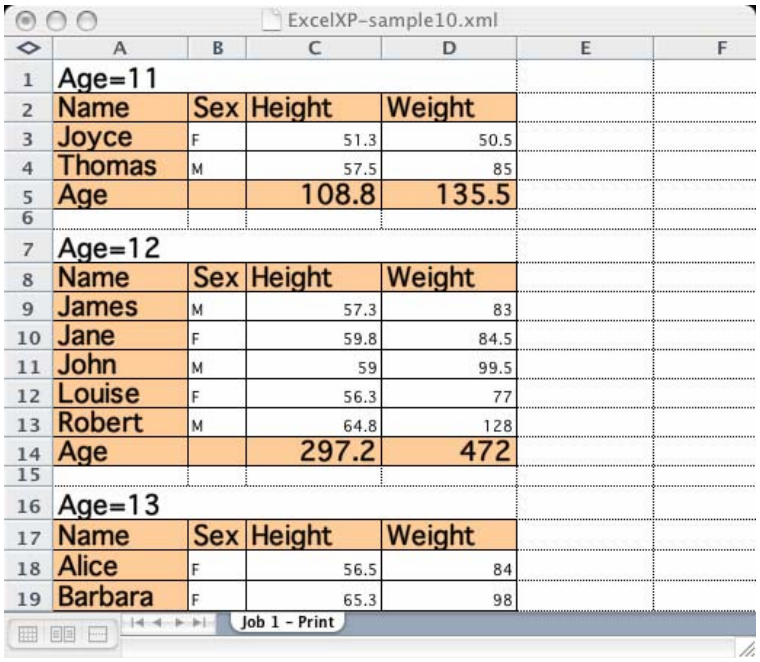
```
*;
*  Use the SHEET_INTERVAL option to force the output from each procedure into a single worksheet.
*  Subtotals will be added to the HEIGHT and WEIGHT fields as a result of the SUM statement.
*;

ods listing close;

ods tagsets.excelxp file='ExcelXP-sample10.xml' style=XLStatistical
    options(sheet_interval='proc' auto_subtotals='yes');

  proc print data=class;
    by age;
    id name;
    sum height weight;
  run; quit;

ods tagsets.excelxp close;
```



ExcelXP-sample10.xml

	A	B	C	D	E	F
1	Age=11					
2	Name	Sex	Height	Weight		
3	Joyce	F	51.3	50.5		
4	Thomas	M	57.5	85		
5	Age		108.8	135.5		
6						
7	Age=12					
8	Name	Sex	Height	Weight		
9	James	M	57.3	83		
10	Jane	F	59.8	84.5		
11	John	M	59	99.5		
12	Louise	F	56.3	77		
13	Robert	M	64.8	128		
14	Age		297.2	472		
15						
16	Age=13					
17	Name	Sex	Height	Weight		
18	Alice	F	56.5	84		
19	Barbara	F	65.3	98		

Job 1 - Print

Figure 6. Multiple Tables with Bylines per Worksheet

WAKE ME UP … I DON'T KNOW WHERE I AM

Have you ever been lost in your dreams? While exploring your dream world you suddenly realize that you have no idea where you are or where you've been. This happens when looking at really wide HTML tables too. Scrolling rows doesn't help in this case. What we really need is ID columns on both sides of our tables.

This solution is a tagset that inherits from the HTML4 tagset. This tagset uses six streams to capture the desired header cells for each row. A special event is set up just to determine if a cell should be captured and which stream it should be captured in. The tagset uses the colstart variable, which is the current table column, to make its decision. It prints all the headers that it captures as soon as it encounters a cell that should not be captured. When the row ends, all the captured headers are printed again, but in reverse order.

There are a number of ways to do this. This solution could be more versatile. The only real drawback to this method is that the justification cannot be reversed for the headers on the right side of the table. That's a nice thing that most people won't even notice.

This event does all the decision making and prints the beginning of the row when all the cells that need repeating have been captured.

```
define event header_streams;
    eval $column inputn(colstart, "3.");
    do /if $column <= $column_count;
        unset $row_headers_printed ;

        do /if $column = 1;
            open column_1;
        else /if $column = 2;
            open column_2;
        else /if $column = 3;
            open column_3;
        else /if $column = 4;
            open column_4;
        else /if $column = 5;
            open column_5;
        else /if $column = 6;
            open column_6;
```

```
            done;

        else /if $column = $column_count + 1;

            trigger put_header_streams;

        done;

    end;
```

This event just prints the beginning of the row if it hasn't been printed already.

```
    define event put_header_streams;

                flush;

                break /if $row_headers_printed;

                /* just in case */
                close;

                /*-------------------------------------------------------eric-*/
                /*-- go back to the header row stream if need be.          --*/
                /*-----------------------------------------------------22Oct04-*/
                open header_row /if cmp(section, "head");

                /* the rowheaders for this row have been printed */
                set $row_headers_printed "True";

                /*-------------------------------------------------------eric-*/
                /*-- This chunk prints out the headers at the beginning of the row. --*/
                /*-----------------------------------------------------22Oct04-*/
                put $$column_1;
                put $$column_2;
                put $$column_3;
                put $$column_4;
                put $$column_5;
                put $$column_6;
                flush;

        end;
```

The header event is just like the header event as defined in the HTML4 tagset that is used by ODS HTML. It adds a section for the 'body' of the table that determines when to repeat the table header rows and, more importantly, triggers the header_streams event to catch the beginning of the row for later.

```
   define event header;
       start:

           do /if cmp(section, "body");
               do /if !$row_started;

                   do /if $data_row_count > $header_interval;
                       do /if cmp(colstart, '1');
                           put $$header_rows;
                           eval $data_row_count 1;
                       done;
                   done;

                   put "<tr>" CR;
                   set $row_started "True";
               done;

           done;

           /*-----------------------------------------------------------------eric-*/
           /*-- open up a column stream if needed.                      ----------*/
           /*-----------------------------------------------------------------22Oct04-*/
```

```
        trigger header_streams;

        put "<th";
        putq " title=" flyover;
        trigger classalign;
        trigger style_inline;
        trigger rowcol;
        put ">";
        trigger cell_value;
    finish:
        trigger cell_value;
        put "</th>" CR;
      end;
```

The data event only adds a trigger header_streams statement to catch any data cells that might need repeating.

```
 define event data;
     start:
         /* this would work but sometimes htmlclass is empty... */
         trigger header /breakif cmp(htmlclass, "RowHeader");
         trigger header /breakif cmp(htmlclass, "Header");

         /*-----------------------------------------------------------------eric-*/
         /*-- open up a column stream if needed.                        ---------*/
         /*-----------------------------------------------------------22Oct04-*/
         trigger header_streams;

         put "<td";
         putq " title=" flyover;
         do /if !cmp(htmlclass,'batch');
             trigger classalign;
             trigger style_inline;
         done;
         trigger rowcol;
         put " nowrap" /if no_wrap;
         put ">";
         trigger cell_value;
    finish:
         trigger header /breakif cmp(htmlclass, "RowHeader");
         trigger header /breakif cmp(htmlclass, "Header");

         trigger cell_value;
         put "</td>" CR;
  end;
```

The event to pull all this together is triggered from the row end. Reverse_Header_Streams prints out the six possibly captured cells in reverse order and then deletes them.

```
/*-----------------------------------------------------------------eric-*/
/*-- Print out the columns in reverse order and delete them.        --*/
/*-----------------------------------------------------------22Oct04-*/
define event reverse_header_streams;


    flush;

    /*-----------------------------------------------------------eric-*/
    /*-- For the case where are row has fewer cells than what we want-*/
    /*-- to repeat.                                                 --*/
    /*-----------------------------------------------------------22Oct04-*/
    trigger put_header_streams;

    put $$column_6;
    put $$column_5;
    put $$column_4;
    put $$column_3;
    put $$column_2;
    put $$column_1;
```

```
        unset $$column_6;
        unset $$column_5;
        unset $$column_4;
        unset $$column_3;
        unset $$column_2;
        unset $$column_1;
    end;
```

Obviously, there is more to this tagset, but those are the important pieces. Now all we need is the key to the map that shows our newly created coastline on the other side of the huge data sea of our dreams.

This tagset pre-dates the options option, so macro variables are used. Here is an example of how it's used. The output is shown in Figures 7 and 8.

```
%let column_count=2;
%let header_interval=5;

options obs=10;

ods tagsets.two_sided file="twosided3.html";

proc print data=sashelp.class;run;

%let header_interval=10;

proc tabulate missing data=sashelp.revhub2;
   class type;
   class hub source;
   var revenue;
   table hub*source, type*(revenue="Average Revenue"*(mean*f=10.0))
      / rts=40;
   keylabel mean=' ';
run;

ods tagsets.two_sided close;
```

The tabulate output in this example is quite narrow, but it fits on this page and on the presentation screen. It is important to test tagsets with a variety of procedures such as PROC TABULATE and PROC PRINT to ensure that they work appropriately.

Obs	Name	Sex	Age	Height	Weight	Name	Obs
1	Alfred	M	14	69.0	112.5	Alfred	1
2	Alice	F	13	56.5	84.0	Alice	2
3	Barbara	F	13	65.3	98.0	Barbara	3
4	Carol	F	14	62.8	102.5	Carol	4
5	Henry	M	14	63.5	102.5	Henry	5
Obs	Name	Sex	Age	Height	Weight	Name	Obs
6	James	M	12	57.3	83.0	James	6
7	Jane	F	12	59.8	84.5	Jane	7
8	Janet	F	15	62.5	112.5	Janet	8
9	Jeffrey	M	13	62.5	84.0	Jeffrey	9
10	John	M	12	59.0	99.5	John	10

Figure 7. PROC PRINT with Two-Sided Row Headers

		TYPE				
		Direct	Indirect	Other		
		Average Revenue	Average Revenue	Average Revenue		
HUB	REVENUE SOURCE				REVENUE SOURCE	HUB
Frankfurt	Freight	1464938	198942	144685	Freight	Frankfurt
	Other	111193	12057	10717	Other	
	Passenger	3460389	421999	337599	Passenger	
	Service	444773	.	.	Service	

Figure 8. PROC TABULATE with Two-Sided Row Headers

THE RECURRING SLIDEBAR DREAM

Remember the dream you were having in the “Sharpening Perceptions” section? You had fuzzy numbers everywhere. The numbers were related but they weren’t clear and their relationships weren't discernable. Well, you’re having that dream again. This time, you dream of adding colors and boxes to your data tables to help clarify things greatly but you are sure that it is too difficult or impossible to do.

This example started with a DATA step program that created a table with complex interlocking slidebars for survey results. The output from the DATA step was HTML. It looked great! But the DATA step was specifically tailored to the data set. The other limitation was HTML. The desire was to have both HTML and PDF. It would also be nice if other data sets could be used. The current DATA step could work only with one particular data set.

Tagsets opened the door to all of these possibilities. The first step was to write an HTML tagset that could make interlocking slidebars—a proof of concept. This new HTML tagset inherits most of its behavior from the HTML4 tagset. The second step was to write a new LaTeX tagset using the same principles. The LaTeX tagset inherits most of its behavior from the LaTeX tagset that ships with SAS, alongside HTML4. LaTeX would be our path to PDF. The interface to the procedures would be the same for both tagsets. The primary goal for the tagset was to create interlocking slidebars from three consecutive percentage columns (Figure 9).

Figure 9. Part of an HTML Output Showing Interlocking Slidebars

The concept is very similar to the slidebar example explained in the Sharpening Perceptions section. The only real difference is the connection of consecutive data cells within a single table. The HTML for a single set of these looks like this.

```
<td width="30%" class="data"colspan="3">
<table border="1" cellpadding="0" cellspacing="0" width="100%"><tr>

<td class="c Unfav" start width="  11% ">  11% </td>
<td class="c Neutral" in width="  16% ">  16% </td>
<td class="c Fav" end width="  73% ">  73% </td>

</tr></table >
</td>
```

The cell that contains all this needs a width so that the overall width isn't determined by the browser. The cell also needs to span three columns because there still needs to be three column headers; therefore, three columns will be

merged into one. When the first cell is encountered, the surrounding cell and table need to be set up. When the last cell closes, the table and parent cell can be closed. The cells in between are just like every other data cell in the table, except that they have different styles and a width that is specified as a percentage, which is actually the cell value.

The heart of this solution is in three events. The header and data events and a new special event called chart_table that handles the surrounding cell and table.

After some experimentation, it was discovered that the column headers needed to be treated the same as the data. They were given an automatic width of 33%. That made everything look right for a three-column chart. While we are at it, why not make the whole thing more versatile? It could potentially combine any number of columns, as long as they were percentages that added up to 100.

```
define event chart_table;
    start:
        do /if $header;
            put '<th width="30%" class="header"';
        else;
            put '<td width="30%" class="data"' ;
        done;

        do /if !slidebar_column_count;
            putq 'colspan="3"';
        else;
            putq "colspan=" slidebar_column_count;
        done;
        put ">" nl;

        put '<table border="1" cellpadding="0" cellspacing="0" width="100%"><tr>' nl;

    finish:
        put '</tr></table >' nl;
        do /if $header;
            put '</th>' nl ;
        else;
            put '</td>' nl ;
        done;
end;
```

The header event is mostly stolen code. There are two new parts of this event. The first two lines open and close the chart table that the chosen columns will go in. The do_hwidth event sets the header width as appropriate for headers that are in the chart columns.

```
define event header;
    start:
        set $header "True";
        trigger chart_table start /if cmp(tagattr, "start");
        unset $header;
        set $slide_bar_column "TRUE" /if tagattr in ('start', 'in', 'end');
        set $end_column "TRUE" /if cmp(tagattr, 'end');

        put "<th";
        putq " title=" flyover;
        trigger classheadalign;
        trigger style_inline;
        trigger rowcol;

        trigger do_hwidth /if $slide_bar_column;

        put ">";
        trigger cell_value;
    finish:
        trigger cell_value;
        put "</th>" CR;
        set $header "True";
        trigger chart_table finish /if $end_column;
```

```
        unset $header;
        unset $end_column;
 end;
```

The data event turned out to be a little more complicated. If a value is missing, it shouldn't be in the chart. The whole data cell has to be saved in a stream that is called data_cell, just in case that happens. When it does, the data cell is thrown away instead of becoming part of the chart. The other complication came from the unruly behavior of PROC REPORT. PROC REPORT is different and less well-behaved than all the other procedures. Always be sure to test your dream tagset against PROC REPORT; you may be surprised to find a monster under the bed.

```
define event data;
   start:
      /*-------------------------------------------------------------eric-*/
      /*-- Put the entire cell in a stream so we can throw it away if--*/
      /*-- its value is missing and it's in a slidebar               --*/
      /*-------------------------------------------------------16Oct04-*/
       open data_cell;

      /* this would work but sometimes htmlclass is empty... */
      trigger header /breakif cmp(htmlclass, "RowHeader");
      trigger header /breakif cmp(htmlclass, "Header");

     /*-------------------------------------------------------------eric-*/
     /*-- this is the slidebar part.  start the table if need be.    --*/
     /*-- set slide_bar_column to true - if it is.  Makes it easier  --*/
     /*-- later.  set end_column if it is. - for report, which gives --*/
     /*-- a cell one excruciating piece at a time.  Set the width for--*/
     /*-- the same reason.                                           --*/
     /*-------------------------------------------------------10Oct04-*/
      trigger chart_table start /if cmp(tagattr, "start");
      set $slide_bar_column "TRUE" /if tagattr in ('start', 'in', 'end');
      set $end_column "TRUE" /if cmp(tagattr, 'end');
      set $width value /if $slide_bar_column;

      put "<td";
      putq " title=" flyover;
      do /if !cmp(htmlclass,'batch');
        trigger classalign;
        trigger style_inline;
      done;
      trigger rowcol;
      put " nowrap" /if no_wrap;

      /*-----------------------------------------------------eric-*/
      /*-- for report.  no reason to finish the cell.  we    --*/
      /*-- need to put width in when we finally get the      --*/
      /*-- value.  - if it's a slidebar column.              --*/
      /*-----------------------------------------------10Oct04-*/
      do /if !value;
          set $need_tag_end "TRUE" /breakif cmp(data_viewer, "report");
      done;

      trigger do_width /if $slide_bar_column;
      put ">";
      trigger cell_value;

   finish:
      trigger header /breakif cmp(htmlclass, "RowHeader");
      trigger header /breakif cmp(htmlclass, "Header");


      /*-----------------------------------------------------eric-*/
      /*-- If for some awful reason we never got a value and --*/
      /*-- the td tag was never closed - we need to close it now.--*/
      /*-----------------------------------------------10Oct04-*/
      put ">" /if $need_tag_end;
      unset $need_tag_end;
```

```
        trigger cell_value;
        put "</td>" CR;

        close;
        put $$data_cell /if !$miss;
        unset $$data_cell;
        unset $miss;

        /*-----------------------------------------------------eric-*/
        /*-- Close up the slidebar if we need to.  unset the   --*/
        /*-- variables, ready for the next go around.          --*/
        /*-----------------------------------------------10Oct04-*/
        trigger chart_table finish /if $end_column;
        unset $end_column;
        unset $width;
    end;
```

The put_value event is normally very simple. However, this is one of the monsters that is spawned from PROC REPORT. PROC REPORT gives each cell one piece at a time. It is the only PROC that gives the data separately from the data cell. Because of that, the put_value event has to catch the value and add the cell width that is created from it—but only if this is a slide bar column.

```
    define event put_value;
        do /if $need_tag_end;
            trigger do_width /if $slide_bar_column;
            put ">";
            unset $need_tag_end;
        done;

        trigger cell_value;
    end;
```

The tagset turned out to be fairly simple, but there is much more to this problem than just some tagset code. Some styles need to be created in order to improve the look of the chart. The tagset also needs an interface.

The primary styles that we care about are the percentage columns. Here's how we create those. The full style definition that is used for this example has other entries as well, but they have been left out to preserve space.

```
   proc template;
     define style styles.survey;
     parent = styles.minimal;

          style Unfav /
              background = red
              foreground = cxFFFFFF
              font = fonts("chartFont")
              borderColor = black
          ;
          style Neutral from Unfav/
              background = yellow
              foreground = cx000000
          ;
          style Fav from Neutral/
              background = cx7FFF00
          ;

          end;
   run;
```

Now for the interface. There has to be a method for the procedure to tell the tagset which columns are to go in the chart. Style overrides and the TAGATTRstyle attribute work well for this.

Here's what we can do in PROC PRINT. PROC PRINT sets the style for each column and uses TAGATTR to indicate the start, middle, and ending columns. A macro variable can be used to override the default of three column charts. The only reason the tagset needs to know this is so that it can span the columns and set the header widths appropriately.

```
*%let slidebar_column_count=3;

proc print data=a label noobs;
    /*-----------------------------------------------------------eric-*/
    /*-- set up the variables and assign the styles and             --*/
    /*-- justification.  We could have put the justification in     --*/
    /*-- the style definition but that takes away versatility.      --*/
    /*-----------------------------------------------------------10Oct04-*/
    var code / style(head) = header
                  style(data) = header;
    var diff validN / style=[just=center];
    var percentt1 / style(head) = Unfav[just=center tagattr="start"]
                 style(data) = Unfav[just=center tagattr="start"];
    var percent2 / style(head) = Neutral[just=center tagattr="in"]
                 style(data) = Neutral[just=center tagattr="in"];
    var percent3 / style(head) = Fav[just=center tagattr="end"]
                 style(data) = Fav[just=center tagattr="end"];

    /* set up the headings */
    label code = 'Items';
    label diff = 'Diff % Fav';
    label validN = 'Valid N';
    label percent1 = 'Unfavorable';
    label percent2 = 'Neutral';
    label percent3 = 'Favorable';
run;
```

The interface looks good. Most of the control is given to the procedure. Finally all the parts are in place. The final PROC PRINT output is shown in Figure 10.

Department	Diff % Fav	Valid N	Unfavorable	Neutral	Favorable	Year
Design	10	30	13%	50%	37%	2002
Marketing	20	30	10%	40%	50%	2002
Devel	10	50	19%	60%	21%	2002
Pubs	40	20	10%	67%	23%	2002
Design	10	30	5%	48%	47%	2002
Marketing	40	40	10%	50%	40%	2002
Devel	10	30	23%	74%	3%	2002
Pubs	50	60	17%	35%	47%	2002
Design	10	30	50%	10%	40%	2002
Marketing	20	20	10%	40%	50%	2002
Devel	10	30	50%	10%	40%	2002
Pubs	10	10	61%	12%	27%	2002
Design	70	30	27%	36%	37%	2002
Marketing	10	80	10%	40%	50%	2002
Devel	40	30	58%	20%	22%	2002
Pubs	10	40	80%	15%	5%	2002

Figure 10. HTML Output with Interlocking Slidebars Generated by PROC PRINT

However, this is a survey! Where are the questions? Typically, every time I get close to my dream, something goes strangely awry. It's time for PROC REPORT. The entire program using the survey style and both the HTML and LaTeX tagsets looks like this.

```
data test;
format pct1 pct2 pct3 percent.;
INPUT @1 question $50. @52 dept $10. @63 diff 2. @66 valid 2. @69 (pct1 pct2 pct3) (percent.)
@86 year;

cards;
The computer I have is adequate for my needs       Design     10 30 13%   50%   37%    2002
The computer I have is adequate for my needs       Marketing  20 30 10%   40%   50%    2002
The computer I have is adequate for my needs       Devel      10 50 19%   60%   21%    2002
The computer I have is adequate for my needs       Pubs       40 20 10%   67%   23%    2002
The Software on my computer is always up to date   Design     10 30 5%    48%   47%    2002
The Software on my computer is always up to date   Marketing  40 40 10%   50%   40%    2002
The Software on my computer is always up to date   Devel      10 30 23%   74%   3%     2002
The Software on my computer is always up to date   Pubs       50 60 17%   35%   47%    2002
I feel adequately protected from computer viruses  Design     10 30 50%   10%   40%    2002
I feel adequately protected from computer viruses  Marketing  20 20 10%   40%   50%    2002
I feel adequately protected from computer viruses  Devel      10 30 50%   10%   40%    2002
I feel adequately protected from computer viruses  Pubs       10 10 61%   12%   27%    2002
The OS and tools are the best available for my job Design     70 30 27%   36%   37%    2002
The OS and tools are the best available for my job Marketing  10 80 10%   40%   50%    2002
The OS and tools are the best available for my job Devel      40 30 58%   20%   22%    2002
The OS and tools are the best available for my job Pubs       10 40 80%   15%   05%    2002
;

run;
/*-----------------------------------------------------------------eric-*/
/*-- This variable tells the tagset how many columns will be in     --*/
/*-- the slidebar.  For HTML that tells it to colspan this many     --*/
/*-- columns.  It is also used to calculate the header widths.      --*/
/*-- The default is 3 so there is no need to set it in that case.   --*/
/*-----------------------------------------------------------------13Oct04-*/
*%let slidebar_column_count=3;

title1 height=3 justify=l 'Survey Responses for Computers';
title2 height=6 justify=l 'Items within them:  Computers';

ods tagsets.html_slidebar style=survey file="survey.html";
ods tagsets.latex_slidebar style=survey
                           file="survey.tex"
                           stylesheet="survey.sty" (url="survey");

PROC REPORT DATA=test LS=138 PS=55  SPLIT="/" HEADLINE HEADSKIP CENTER nowd;
by year;

COLUMN question ( dept ( diff valid pct1 pct2 pct3 ) );
define question / order noprint;
DEFINE  dept / ORDER FORMAT= $16.     WIDTH=16 SPACING=2 LEFT   "ITEMS" style=header;
DEFINE  diff / DISPLAY FORMAT= BEST9.     WIDTH=9 SPACING=2 center "Diff % Fav";
DEFINE  valid / SUM   FORMAT= BEST9.   WIDTH=7  SPACING=2 center "Valid N" ;
DEFINE  pct1 / SUM   FORMAT= PERCENT6. WIDTH=6  SPACING=2 center "Unfavorable"
style=UnFav[tagattr="start"];
DEFINE  pct2 / SUM   FORMAT= PERCENT6. WIDTH=6  SPACING=2 center "Neutral"
style=Neutral[tagattr="in"];
DEFINE  pct3 / SUM   FORMAT= PERCENT6. WIDTH=6  SPACING=2 center "Favorable"
style=Fav[tagattr="end"];

break before question /;
compute before question / style=Question[just=left]  ;
 line question $80.;
endcomp;

RUN;


ods tagsets.html_slidebar close;
ods tagsets.latex_slidebar close;
```

Figure 10. HTML Output with Interlocking Slidebars Generated by PROC Report

Applying the same techniques that we learned with the HTML tagset, we created a new LaTeX tagset. There are several LaTeX tagsets; their use is documented on the Basa SAS ODS MARKUP Resources Web page (http://support.sas.com/rnd/base/topics/odsmarkup/).

The final PDF generated from LaTeX looks great (Figure 11). The ODS LaTeX tagset does have some limitations, but none of them are show here.

Figure 11. PDF Output Via LaTeX with Interlocking Slidebars Generated by PROC Report

CONCLUSION

With the techniques described in this paper, now you should be able to capture some of the reports of your dreams. You may even dream of new things that your subconscious never allowed for.

It's time to jump out of the quicksand and into the air. Realize the possibilities that ODS MARKUP and tagsets provide for you.

RECOMMENDED READING

Gebhart, Eric A. 2005. "Tagset Spelunking and Cartography: Debugging and Exploring Tagsets with Battery-Powered Headlamps", *Proceedings of the Thirtieth Annual SAS Users Group International Conference.* Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2004. *SAS® 9.1 Output Delivery System: User's Guide.* Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The author would like to thank Kathleen Walch and Mike Boyd of SAS Institute for their contributions to this paper.

CONTACT INFORMATION

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

Eric Gebhart
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8000
eric.gebhart@SAS.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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