Paper 1689-2014

Simple ODS Tips to Get RWI (Really Wonderful Information) from your RWI (Report Writing Interface)

Gina Huff, Western Kentucky University, Bowling Green, KY

ABSTRACT

SAS ® continues to expand and improve its reporting capability. With new 9.4 enhancements in ODS (Output Delivery System), the opportunity to create stunning reports has expanded even further. If you are charged with creating relevant, informative, easy-to-read reports for clients and/or administrators, then the **ODS Report Writing Interface, ODS LAYOUT** enhancements, and the new **ODS TEXT** procedure are important tools to use. These tools allow you to create reports in a smart, eye-catching format that can be turned around quite quickly and programmed to provide optimum flexibility.

How many times have you worked hours to tweak and fine-tune a report directly in Excel, Word, Power Point or some other similar software only to be asked for a "quick update", which would then take hours to recreate because you are manually transferring data? Do you ever dread receiving the compliment, "This is really wonderful information!!!!" because you know it will be followed by "Can you run this for EVERY region?" Well, dread no more because when you harness the power of SAS® ODS, you can create first-rate, flexible, fabulous reports!

Join me as I share with you a real-world example of ODS capabilities using a marketing piece I designed to help the president of our university spotlight county and region specific data as he recruited across the state (figure 1).

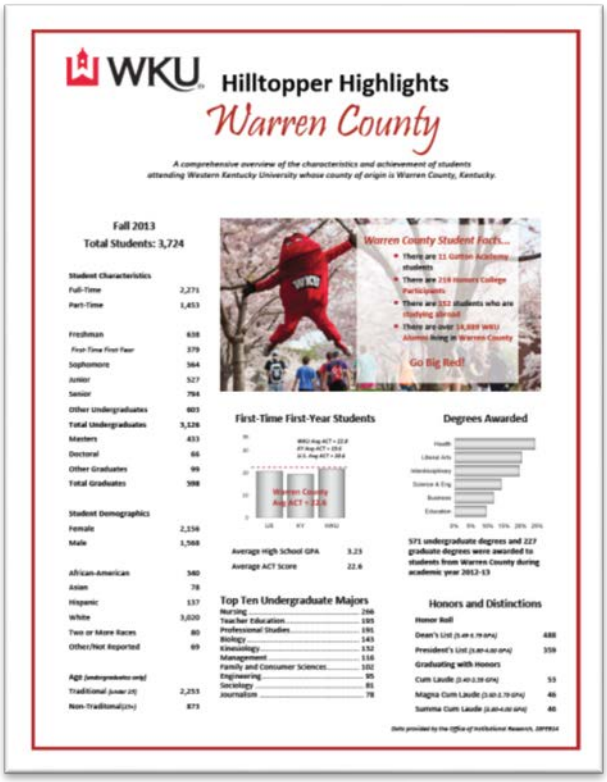
INTRODUCTION

The Office of Institutional Research (IR) at Western Kentucky University is responsible for the production and distribution of reports that provide information on university statistics. The president is one of our key clients because he values data—data for decision support and data to support our outreach and recruiting efforts. Because of his reliance on IR information, I created what we internally call a "trip report". (It is officially called "Hilltopper Highlights.) This report provides key student information segmented by county of origin that he, as well as enrollment management and admissions staff, can use on visits to area chamber of commerce meetings, recruitment trips to local high schools, and appointments with state legislature and government officials. Because this report was initially only used by the president for periodic visits, I created the report in Microsoft Publisher using PROC TABULATE Output that I manually transposed into the document.

I will never forget the day when IR received the urgent request to create these reports for every county in the state (all 120 of them) as well as by congressional districts and regional campus service areas. They were also going to need those updated every term. Had I not known about the power of the Output Delivery System, I may have panicked. Thank goodness SAS had the capability to power through this project and create output that was just as attractive as it was in Publisher, but without the additional, error-prone step of manually transferring data into another format.

This project illustrates how SAS® has refined their product in way that you can literally move from raw data to final, polished report in one program. My hope is that as you read this paper, you may find a tip or method that you can use to craft your own *Really Wonderful Information.*

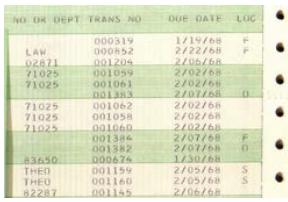
Figure 1.
Hilltopper Highlights

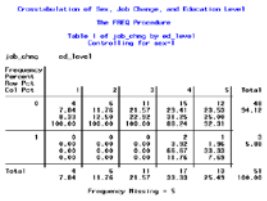


THE EVOLUTION OF INTERNAL BUSINESS REPORTING

Very apparent in this project is the fact that a "standard" report rarely meets the needs of our users any longer. In a 2013 Global Forum Paper, Daniel O'Connor commented, "Enterprise Business Reporting is the evolution of our traditional reporting framework to address advanced reporting requirements for the 21st century." Mr. O'Connor is absolutely correct to point out that our reporting needs have evolved over time. A "great" report created years ago wouldn't seem so great to us now. For example, many of us can look back to the days of green bar paper reports and recognize how antiquated that now appears. Imagine if you were presented with a green bar report today. What would your impression be? Would you even trust the data presented? I would argue that providing reports in regular SAS output is also well past its prime as is PDF output in the traditional SAS "Printer" style. SAS recognized this in the 9.4 release and changed the default style to "Pearl", a more refined and understated style. Clearly, clients want a more **refined** report.

Figure 2. The Evolution of Business Reporting.
From left: old school green bar report, SAS Output, STYLES.PRINTER output, STYLES.PEARL output (with additional formatting), and ODS LAYOUT output





	Level Code		
	GR	UG	All
	N	N	N
Gender			
F	2063	10424	12487
M	946	7691	8637
All	3009	18115	21124

	Level		
	GR	UG	Total
Female	2,063	10,424	12,487
Male	946	7,691	8,637
Total	3,009	18,115	21,124

The standard report also comes up wanting because our clients have grown to expect a complete or **whole** report. I remember the days—not too long ago, in fact—that the three-ring binder (often the 4 inch variety) was a staple in our office because many of the reports we ran had hundreds of pages. It wasn't that we were presenting more information in the past, it was just that each table of data was printed on a separate page. With the power of SAS, we can be much more efficient with our output and provide decision-makers with *information* rather than just pages of data tables.

Lastly, our consumers don't want to wait on information; they want **immediacy**. If you have to invest time transferring information over into Publisher, Excel, or Power Point to create a professional report, you are sacrificing precious time and increasing your opportunity for error. Let me point out, however, that I am realistic enough to know that ODS LAYOUT isn't a solution for every little adhoc request that comes down the line. You have to weigh the time it takes to develop production ready output with the number of times/ways it will need to be accessed. Nevertheless, for ongoing reporting needs such as these, I am convinced that ODS LAYOUT is the best option. Furthermore, code like this can transition seamlessly into a stored process for complete flexibility and real-time reporting for users.

Real World Influence

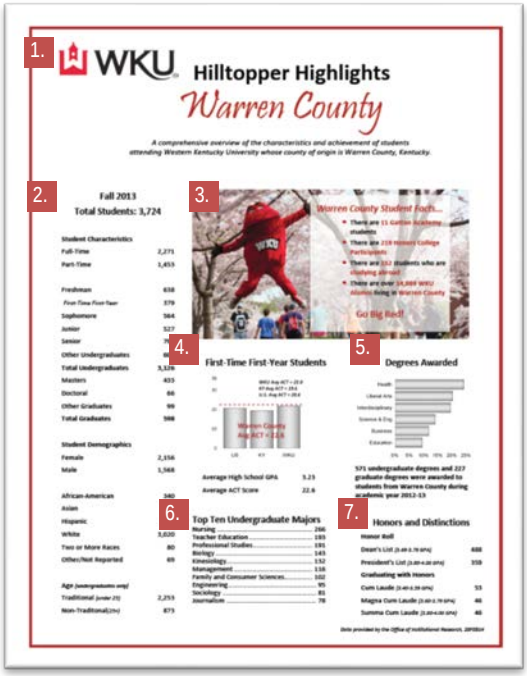
PROJECT: CREATING A PRINT-READY MARKETING PIECE

The "trip report" needed to be polished and ready for distribution in the "real world"; in other words, the report needed to be in a format that could be distributed externally to school counselors, chamber of commerce leaders, etc. This was the key challenge in successfully implementing this project. The data itself was fairly straightforward. Student demographics, popular majors, and degrees awarded are common statistics that most IR shops would have at-the-ready. Using tables from our existing data warehouse, I simply compiled summary datasets with the information needed by county, senate district, and regional campus service area. I then harnessed the power of SAS, specifically it's Output Delivery System (ODS) to create my **Really Wonderful Information.**

PLANNING THE PAGE

I elected to use a three column format with an understated feel. I used a style I created for IR reports using the TEMPLATE Procedure named calibri_pdf to ensure a consistent look with our other publications. I incorporated "basic statistics" about each county within the publication using a combination of the Report Writing Interface, ODS Text, PROC ODSTEXT, and the SGPLOT procedure.

Figure 3.
Hilltopper Highlights



1. **Page Set Up:** I selected a bold heading with the university logo and prominent county of focus displayed. I added a red border around the page as a simple visual anchor and a footnote that credits our office as well as the date the report was run.

2. **Students:** This section provides basic characteristics and demographics of students from the county selected.

3. **Layered Pix:** This section provides some novel visual appeal to the report. I have included a seasonal picture and a translucent box with county quick facts.

4. **FTFY Students:** This section provides information about our incoming freshman class, including average ACT score and high school GPA. The bar chart provides a quick comparison of WKU, state, and national ACT averages compared to the county average (in a red reference line).

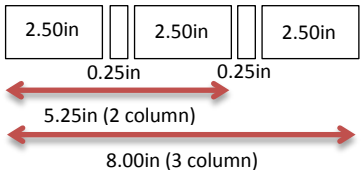
5. **Degrees:** The degrees awarded section provides bold text providing the number of degrees awarded. The horizontal bar chart provides the percentage of degrees awarded by academic college.

6. **Majors:** The top ten undergraduate majors are listed in this section along with the number of students enrolled in that major for the given term.

7. **Honors:** This is a very simple table that provides the number of students making the honor roll and those who graduated with distinction during the given academic year.

TIPS AND TIDBITS: Don't skip the planning process!

A little bit of preparation goes a long way when creating your report using ODS LAYOUT. I typically sketch out a sample page and determine column widths to help in determining regions. In this project, I elected to have a three column design. Columns were 2.50 inches with a quarter inch gutter in between. The total width was 8.00 inches, which equals standard letter size less a quarter inch margin on both sides.



CREATING THE REPORT

I always begin any ODS LAYOUT project by setting the appropriate OPTIONS for my page:

```
options  orientation=portrait papersize=letter
         nodate nonumber
         topmargin=0.25in
         bottommargin=0.25in
         leftmargin=0.25in
         rightmargin=0.25in;
ods escapechar="^";
title; footnote;
```

Defines an 8½ x 11 page.

Suppresses date and page numbers from appearing on pdf.

Because I don't have to print to the edge of the margin, I am setting a quarter inch margin. Please note that margins must be greater than 0.

Defines the character that will call inline formatting. Select a rarely used character like the caret (^) chosen at left or you may use (*ESC*) directly in the code if you prefer.

Clear all titles and footnotes.

Next, I define the pdf output delivery system and style and initiate ODS LAYOUT. Both use the "wrapper" method in that you will initiate at the beginning and then close at the end. Note that the border around the page was created here as well. I simply defined the style as I initiated the ODS LAYOUT code.

```
ods pdf  file="C:\TripReport.pdf"
         style=calibri_pdf;

    ods layout start    width=8.00in
                        height=10.00in
                        style={bordercolor=firebrick
                               borderwidth=3pt};

         ods region     x=___in
                        y=___in

         …insert code here…

    ods layout end;

ods pdf close;
```

Initiates the creation of PDF output. Note that the style is not a SAS style, but one I customized for use in IR reports.

The dimensions are calculated using the paper size less the margins defined in the options statement above.

I create a 3pt border around the outside of the layout container using the STYLE option.

Once you have defined the layout area, you will define the region where you would like the subsequent output to display. The X Option specifies horizontal placement, while the Y Option specifies the vertical placement. In some instances, you will also need to use the width and/or height option.

GETTING FAMILIAR WITH THE DATA

The data structure makes this project easier to execute. The data set is a simple summary table with a listing of all counties, senate districts, and regional campus sites. My first step in the program is to create a work dataset named COUNTY_FACTS that pulls the county desired. This allows me to use the single observation to create many of the tables shown.

County	Total	FT	PT	UG	GR	White	AA	Hisp
Warren	3,724	2,271	1,453	3,126	598	3,021	340	137

Etc.

The MAJORS table, on the other hand, has multiple rows per county, representing the top ten undergraduate majors. Again, I subset the specific county into a work dataset named TOP_MAJORS. Here is a sample of that dataset:

County	Department	Number
Warren	Nursing	266
Warren	Teacher Educaton	193
Warren	Professional Studies	191
Warren	Biology	143
Warren	Kinesiology	132

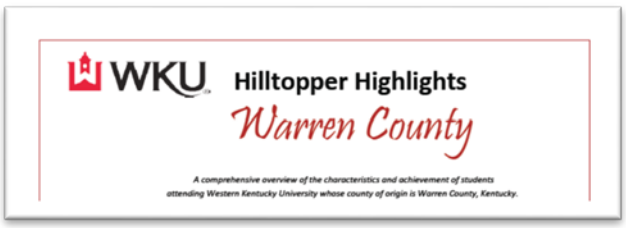
Etc.

USING PROC ODSTEXT AND ODS TEXT

Pre SAS 9.4, I would have created the title section shown at right using ODS TEXT. However, the new ODSTEXT Procedure offers more functionality and a clear, straightforward way to layout paragraphs and lists.

I am compelled to point out that PROC ODSTEXT is particularly useful in creating PowerPoint files using the new 9.4 ODS POWERPOINT destination. Take a moment to review Tim Hunter's excellent paper (provided in the reference section) for more information on this excellent new feature (2013).

Figure 4.
Header/Title



In this example, I have added a graphic, our university logo, to the first line of text. As you can see, the ODSTEXT Procedure supports inline formatting the same as its grandfather, ODS TEXT.

```
ods region  x=0.50in
            y=0.25in;

proc odstext data=county_facts;
```

Remember, county_facts has only one observation. In this instance the variable county = "Warren"

The P Statement signifies the start of a new paragraph in the ODSTEXT Procedure

```
   P "^{style[preimage='c:\WKUlogo.png']}
      Hilltopper Highlights"
      / style={font_face='calibri' fontsize=28pt just=l fontweight=bold};
```

I use inline formatting to insert our university logo

```
   P COUNTY||" County"
      / style={font_face='pristina' foreground=firebrick fontsize=48pt just=c};
```

Here, I concatenate the variable county ("Warren") with the word "County"

```
   P "A comprehensive overview of the characteristics and achievement of students"
      / style={font_face='calibri' fontsize=10pt just=c font_style=italic};

   P "attending Western Kentucky University whose county of origin is "||COUNTY||"
      County, Kentucky."
      / style={font_face='calibri' fontsize=10pt just=c font_style=italic};

run;
```

Again, I concatenate the appropriate county into the text.

One of the more powerful features of PROC ODSTEXT is that you can access a dataset to pull data from variables. In the past, I would have had to first create a macro variable to get the county name into the title, either by using a let statement or by using a DATA _NULL_ statement with CALL SYMPUT. Now, I simply have to access the summary table I am using (data=) and call the variable name (COUNTY) in the paragraph (P) statement.

Because the footnote was just a single line of text, I opted to use ODS TEXT to create it. ODS TEXT is still a great option to use for simple, single lines of text like the one in this example.

Figure 5. Footnote

```
ods region  x=5.25in
            y=10.15in;

ods text="^{style[font_face='calibri'
            fontsize=7pt just=l
            font_style=italic]Data
            provided by the Office of
            Institutional Research,
            &sysdate.}";
```

Inline formatting is used to define the style of the text that follows. I also used the macro variable &SYSDATE to display the date the report was prepared.

Just a Few Common Style Attributes:

sub/super	subscript/ superscript
newline	*inserts a blank line*
thispage	current page #
lastpage	# of pages
foreground	*color of text* (ex. red)
font_face	Arial, Calibri, etc.
fontweight	bold or medium
font_style	italic or roman
fontsize	10 pt, 12pt, etc.
just	left, center, right, dec
textdecoration	underline

TIPS AND TIDBITS: Get "In the Know" with Inline Formatting!

Inline formatting, also referred to as ODS ESCAPECHAR, is a powerful feature of the Output Delivery System. Make it a habit to set your ESCAPECHAR Character at the beginning of the program so that you can utilize it at any time.

Also, read Cynthia Zender's 2007 Global Forum paper titled "Funny Stuff in My Code: Using ODS ESCAPCHAR" to learn more. Honestly, reading anything by Ms. Zender will help improve your visualization and reporting skills!

LAYERING REGIONS FOR A POLISHED LOOK

Figure 6. Picture/At a Glance

Let me be a little biased for a moment and state emphatically that WKU is a beautiful campus! We have lovely buildings and grounds and elegant sculptures that add to the landscape. Additionally, Kentucky weather is famous for gorgeous and remarkable seasons, from lovely snow covered winters to beautiful blooming springs, hot summers to vibrant, colorful autumns. I decided to take advantage of our campus beauty by customizing the picture based on the season in which the report is being run. If the report is run in the winter (fourth quarter), the snowy scene at right would be generated. In the spring (first quarter), users would get our lovable mascot *Big Red* enjoying a sea of freshly blooming Dogwoods behind him. This is very easy to achieve in SAS with a pretty basic macro:

```
%macro m_picture;
   %global season;
   data _null_;
      if qtr(today()) = 1 then call symputx("season", "spring");
      else if qtr(today()) = 2 then call symputx("season", "summer");
      else if qtr(today()) = 3 then call symputx("season", "autumn");
      else if qtr(today()) = 4 then call symputx("season", "winter");
   run;
%mend m_picture;

ods region  x=2.75in
            y=2.75in;
ods text="^{style[preimage='c:\&season..jpg']}";
```

You must make the macro variable global so that the macro variable &SEASON can be accessed outside of this macro

The macro variable created (spring, summer, autumn, winter) is the name of a jpg file. This will be accessed later in the ODS TEXT statement.

The image selected will be determined based on the macro variable created above.

TIPS AND TIDBITS: Make your SYMPUT "X" rated!

CALL SYMPUT is an excellent way to create macro variables from within a data step. However, I have begun replacing CALL SYMPUT with CALL SYMPUTX because it adds these valuable features:

1. Trims leading and trailing blanks
2. Allows a field width of up to 32 chars
3. Converts numeric to character without writing a message to the log

Now that we have the picture in place, I need to add interesting facts about students/alumni from the county. I didn't want to lose the appeal of the picture, but just placing the text directly over the image was unreadable. Using a percent opacity solves the problem. Being able to define opacity value was new to SAS 9.3 and can really add a nice graphic component to your output.

```
ods region  x=4.75in
            y=2.85in
            width=2.35 in
            height=2.30in;
goptions reset=all cback=rgbafffffcc;
proc gslide;
     note;
run;
quit;
```

Notice that unlike some of the other sections, I have to add height and width here. If I failed to do so, the box would begin at the x and y designators but would not have the width and length desired.

PROC GSLIDE is a quick, simple way to create boxes. In this instance I am defining the fill color using the CBACK (Color of Background) option in the GOPTIONS Statement. If I wanted to define a border, I would use the CFRAME (Color of Frame) and/or the WFRAME (Width of Frame) option directly on the GSLIDE option: `proc gslide cframe=black wframe=2;`

Let's take a closer look at the GOPTIONS statement, particularly the CBACK= option.

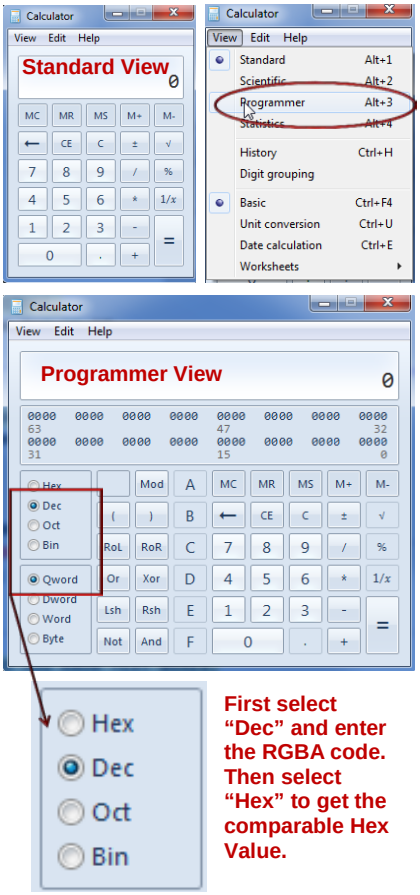
r g b a	f f	f f	f f	c c
color model	**red**	**green**	**blue**	**alpha (opacity)**

The first four characters indicate that we will be using the RGBA color model. The next two characters reflect the hexadecimal value for red and so forth, with the final two characters providing the percent opacity of the color. There are many resources on the web that will provide a sampling of colors with their hex values. I have

used http://cloford.com/resources/colours/ to find web-smart colors as well as about 150 named colors. (Take that Crayola!) And, yes, ghost white, floralwhite, oldlace, linen, whitesmoke, and ivory are all DIFFERENT shades of white.

If you already have an RGB color and just need to convert it to hex code, you need look no farther than your Windows7 calculator! Pull up the basic calculator from your list of programs. Chances are it is in standard view like the first image in the figure at right. Select **View >> Programmer** to get the Programmer View that will allow you to convert your values.

Figure 7.
Using a Windows7 Calculator to Convert RGBA to Hex Code



First select "Dec" and enter the RGBA code. Then select "Hex" to get the comparable Hex Value.

Let's use WKU Red as our example. As most companies do, we have a branding manual that provides our "official" colors. Our official red is RGB(198, 12, 48), or WKU Red. To determine the Hex code, I would simply type in the R code of 198 while "Dec" is selected at the left. Then, I would select "Hex" and the code would be changed to its two character equivalent (198 = C6). The G code of 12 translates to "C" on the calculator. Any time you get a one character response, remember to add a zero to the beginning (12 = 0C). The last color code, B, has a value of 48, which translates to 30, which is possibly the only time you will see the following equation (48 = 30). You combine the three values to get the hex value of "C60C30", or WKU Red.

Translating the hex value for the opacity percentage is a little more difficult. Because RGBA values can range between 0 to 255, you would need to determine the numeric value. If you wanted 100% opacity (or completely opaque), you would type in 255 and receive back a value of FF. If you wanted 60% opacity, you would calculate 0.60 of 255, or 153, and get back a hex value of 99.

In this project, I wanted a transluscent white box. The six character code "FFFFFF" denotes white and the last two characters "CC" calls for an 80% opacity. The box is clearly visible but allows a bit of the picture to show through.

TIPS AND TIDBITS: Save Time by Using Common Opacity Percentages

You can save a little time by using the common values when creating your own transparencies. Use these to fill in the last two characters of your RGBA code:

% Opaque	Hex Value	% Opaque	Hex Value
100%	FF	40%	66
90%	E6	30%	4D
80%	CC	25%	40
75%	BF	20%	33
70%	B3	15%	26
60%	99	10%	1A
50%	80	5%	0D

Now that we finally have the backgrounds defined, it is time to add the text. I decided to again use the powerful ODSTEXT Procedure. You will see that this time I have a title, "Warren County Student Facts" that I created using the Paragraph (P) statement as before. However, to create the bulleted list, I used the LIST and ITEM statements. Note that you must tuck the ITEM Statements in between the LIST and END Statement wrappers.

```
data _null_;
   set county_facts;
   if gatton > 1 then do;
      call symputx("m_verb", "are");
      call symputx("m_s", "s");
   end;
   else do;
      call symputx("m_verb", "is");
      call symputx("m_s", "");
   end;
run;
```

The Gatton Academy is a small, specialized residential high school designed to give gifted students the opportunity to earn college credit and conduct research in the sciences, all while completing their last two years of high school.

Because this is a selective program, there are many counties who have only one Gatton Student. To accommodate the difference needed in subject-verb agreement, I created the macro at left.

```
ods region  x=4.80in
            y=3.10in
            width=2.25in;

proc odstext data=county_facts;
   p county||" County Student Facts..."    / style={just=c foreground=firebrick
                                           fontsize=12pt fontweight=bold
                                           font_style=italic};
```

Create the title first using the P Statement

```
   list / style={liststyletype=square};
```

Define the bullet type (filled in square)

This ITEM Statement calls the macro vars created above.

```
      item "^{style[foreground=black fontweight=medium]There &m_verb. }"
           ||strip(put(gatton, best12.))
           ||" Gatton Academy ^{style[foreground=black fontweight=medium]student&m_s.}"
           / style={fontsize=9pt foreground=firebrick fontweight=bold};
```

Each ITEM Statement uses nested styles. In-line formatting allows you to customize within the statement itself.

```
      item "^{style[foreground=black fontweight=medium]There are }"
           ||strip(put(honors, best12.))
           ||" Honors College Participants"
           / style={fontsize=9pt foreground=firebrick fontweight=bold};

      item "^{style[foreground=black fontweight=medium]There are }"
           ||strip(put(study_abroad, best12.))
           ||"^{style[foreground=black fontweight=medium] students} studying abroad"
           / style={fontsize=9pt foreground=firebrick fontweight=bold};

      item "^{style[foreground=black fontweight=medium]There are over }"
           ||strip(put(alumni, comma9.))
           ||" WKU Alumni ^{style[foreground=black fontweight=medium]living in}"
           || county
           || "County"
           / style={fontsize=9pt foreground=firebrick fontweight=bold};

   end;
```

I could achieve the same result as the NEWLINE attribute presented below by adding a blank paragraph statement above.

```
   p "^{newline}Go Big Red!" / style={just=center fontsize=9pt};
run;
```

The result is a bulleted list providing fun facts about the university with important words in a bold red color. Again, this could have been achieved in ODS TEXT rather than the procedure, but hopefully you can see how PROC ODSTEXT has streamlined the text writing process.

One final reminder, when you layer elements the order in the program is important. The first output object will be in the back and each new output object will layer on top of the others. In this instance, the code for the picture is first (back), then the translucent box code (middle), and finally the fact list code (top).

USING THE REPORT WRITING INTERFACE (RWI) TO DESIGN TABLES

Let me just say it out loud; the Report Writing Interface is a control freak's dream! I absolutely love how much control you have to create the table that you desire. Additionally, because RWI is called within the data step, you can use all of the data step functionality to further refine the output. I used RWI to create all three of the tables in this project.

I must admit that the table at right (figure 8) could be achieved with a couple of text lines and PROC TABULATE as well. I have always been a fan of both the TABULATE and the REPORT Procedures. However, there have been times when my desired output simply did not fit within the confines of these two procedures. RWI really provides the programmer complete fluidity and flexibility of design. I am glad to add this important technique to my tool kit.

As mentioned, this table is a fairly basic example of the utility of RWI, but I feel that it clearly demonstrates how intuitive the code is and will allow you to quickly get started in object oriented programming. This example uses the TABLE Methods to output two column tabular output.

Figure 8. Student Table

To initiate the Report Writing Interface, you must first declare the ODSOUT Object. This simply alerts SAS to write the specified objects out to ODS. Procedures (like TABULATE and REPORT) automatically declare ODS Output. With DATA _NULL_ Report Writing, we must manually declare it. I always name the object OBJ( ) because it's easy to remember and clearly denotes what it is, but you can name it what you choose.

Although not necessary for my project since, as you will recall, I have only one observation, I conditionally run the DECLARE Statement when it is the first observation; you only need to declare it once. The `if _n_ = 1` conditional logic is used to run the DECLARE Statement on the first observation.

Similarly, I initiate the table (using the TABLE_START Method) on the first observation. This would be particularly important if I had more than one row as it would allow me to add rows to the table I am creating rather than making a new table at the start of each new observation.

After I have created a table, I can then define rows (OBJ.ROW_START) and next define and format the cells within those rows. In the OBJ.FORMAT_CELL Method, you will always want to define your data element (i.e. answer the question, *What goes in this cell?*) The answer could be a manual text entry denoted in quotation marks, a variable from the input dataset, a variable created within the new dataset, or some combination the three. You will denote this in the required argument, DATA. There are several optional arguments that are available to customize your table as well. I will point some of them out in the sample code below.

Please be aware that the number of columns will be determined by the number of OBJ.FORMAT_CELL Methods placed in your **first** OBJ.ROW_START Method and/or the number of columns identified in the COLUMN_SPAN attribute. Finally, don't forget that when you start a table or row, you will also need the corresponding OBJ.TABLE_END() or OBJ.ROW_END() to denote the end of the table or row.

```
data _null_;
   set county_facts end=last;
   if _n_ = 1 then do;
      declare odsout obj();
      obj.table_start ();
   end;
```

LAST will be the final observation in the dataset

Initiate the ODSOUT object and start the table on the first observation

You must initiate the table (OBJ.TABLE_START above) before you can initiate rows. Similarly, you must declare the row (OBJ.ROW_START) before you can format cells within the rows.

Figure 9.
Top Portion of Student Table

```
   obj.row_start();

      obj.format_cell  ( data: "Fall 2013"
                       , column_span: 2
                       , style_attr:"fontweight=bold font_size=12pt" );
```

The COLUMN_SPAN Attribute allows you to merge cells across more than one column. Because this is the first row defined and COLUMN_SPAN is two, the table will be two columns wide.

You can assign style attributes such as font, type, size, and color in STYLE_ATTR.

```
   obj.row_end();
```

Once you have defined all cells in a row, you must end the row object.

```
obj.row_start();
   obj.format_cell  ( data: "Total Students: "||put(TOTAL, comma9.)
                    , column_span: 2
                    , style_attr: "fontweight=bold font_size=12pt" );
obj.row_end();
```

```
   obj.row_start();
      obj.format_cell ( data: " "
                      , column_span: 2 );
   obj.row_end();

   obj.row_start();
      obj.format_cell ( data: "Student Characteristics"
                      , column_span: 2
                      , style_attr: "fontweight=bold"" );
   obj.row_end();
```

The cell boxed in red above uses the PUT Function to format the variable TOTAL. Using the FORMAT attribute would not work in this case because it has been concatenated with text. If you put the two items in separate DATA Attributes, however, the FORMAT would work. Here is the alternate code (same output achieved):

```
( data: "Total Students: "
, data: TOTAL
, format: "comma9."
, column_span: 2
, style_attr: … )
```

(program continues on next page)

```
    obj.row_start();
       obj.format_cell ( data: "Full-Time"
                       , style_attr: "width=1.5in"
                       , just: "left" );
       obj.format_cell ( data: FT
                       , format: "comma9.")
                       , style_attr: "width=0.5in"
                       , just: "right" );
    obj.row_end();

    obj.row_start();
       obj.format_cell ( data: "Part-Time"
                       , just: "left" );
       obj.format_cell ( data: PT
                       , format: "comma9." )
                       , just: "right" );
    obj.row_end();
```

This is the first instance when both cells are defined, so I include the width of each cell in the style attribute.

The JUST Attribute determines alignment. Default alignment is Center.

The FORMAT Attribute allows you to assign a format to the variable listed in the DATA Attribute.

...code continues...

I haven't provided the entire block of code here because it is follows the same pattern as presented above. I would like to point out one additional attribute that I utilized toward the end of the table to format a portion of the text within a cell. In the first example below, I wanted the text "undergraduates only" to be in italics. The INLINE_ATTR made this very easy. Unlike the STYLE_ATTR which assigns a style to all of the text in the cell, INLINE_ATTR allows you to specify a font within the line, for a specific portion of the text. Specifically, the INLINE Attribute will format the DATA Attribute that immediately precedes it and trumps the STYLE_ATTR for that text should the same attributes be specified.

```
    obj.row_start();
       obj.format_cell (  data: "Age "
                       , data: "(undergraduates only)"
                       , inline_attr: "font_style=italic font_size=7pt"
                       , style_attr: "fontweight=bold font_size=9pt"
                       , just: "left"
                       , column_span: 2)
    obj.row_end();

    obj.row_start();
       obj.format_cell ( data: "Traditional "
                       , data: "(under 25)"
                       , inline_attr: "font_style=italic font_size=7pt"
                       , just: "left");
       obj.format_cell ( data: TRAD
                       , format: "comma9."

                       , just: "right");
    obj.row_end();

    obj.row_start();
       obj.format_cell ( data: "Non-Traditonal "
                       , data: "(25+)"
                       , inline_attr: "font_style=italic font_size=7pt"
                       , just: "left");
       obj.format_cell ( data: NONTRAD
                       , format: "comma9."
                       , just: "right");
    obj.row_end();

    if last then obj.table_end ();
run;
```

I put the text into two DATA Attributes because I want them to have different font styles.

The INLINE_ATTR will cause "(undergraduates only)" to be output in italics. The STYLE_ATTR of bold will be applied to both DATA Attributes, but the font_size of 7pt will trump the 9pt font in the STYLE_ATTR.

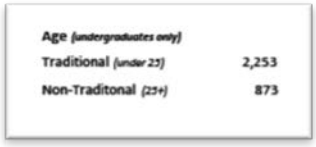


Figure 10.
Bottom Portion of Student Table

Again, because my code is based on one observation, this isn't technically necessary, but it is a good habit to end the table on the last observation of the table (unless otherwise desired).

Please note that the "Honors and Distinctions" Table (Section 7) was created using the same method. Because the approach is identical, I elected not include the code for this table in the paper.

It is important to note that had the data been structured differently (vertical rather than horizontal), then my approach would have been different. In the next example, I use a more vertical data structure to create text using the Report Writing Interface.

The second table in the publication provides a straightforward example of the vertical data structure and how it is handled in RWI. To create the table at right, I can create a new variable that has the major, the leaders (….) using inline formatting, and the count of majors. Because I am in the data step already, creating a new variable is quite simple. After the new variable is created, I can use it to create the Top Ten Undergraduate Majors you see at the right. Or, as I have done below, I can just concatenate the information directly within the OBJ.FORMAT_TEXT Statement.

Figure 11.
Top Ten Majors Table

```
ods region  x=2.75in
            y=8.25in
            width=2.25in;
```

You must define a width or the leaders (…) will fill in the remaining width of the page.

```
data _null_;
   set top_majors;
```

The DO Statement will allow you to declare the object and create a title on the first observation.

```
   if _n_ = 1 then do;

      declare odsout obj();
```

Declare object on the first observation.

Create the title on the first observation only. Otherwise, your output will look like Figure 12, with a title before every row.

```
      obj.format_text ( data: "Top Ten
                               Undergraduate Majors"
                      , just: "center"
                        , style_attr: "fontweight=bold"
);

   end;
```

End the DO Statement.

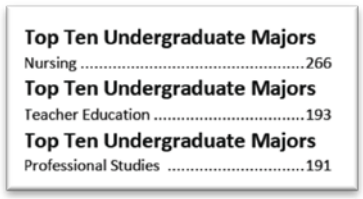


Figure 12.
Top Ten Majors Gone Wrong

Create a text line that houses the major followed by leaders (…)and then ending with the number of majors.

```
   obj.format_text ( data: strip(department)||"^{leaders .}"||strip(number);
                           , style_attr: "cellwidth=100pct font_size=9pt" );

run;
```

The technique I used above works well to list our top majors, but is equally useful to develop table of contents and similar lists. I hope that these examples help to illustrate how powerful the Report Writing Interface can be, although I have merely scratched the surface with these two examples.

TIPS AND TIDBITS: Let Your Commas Dangle

This is a little programming standardization tip that has paid dividends in saved minutes looking for syntax errors. In the past, I was probably the victim of missing comma errors even more than missing semicolons in my code. However, by putting the commas FIRST in a new line of code rather than the end of the previous line, I can now quickly check to make sure all my commas are in place. It provides the added benefit of allowing me to easily comment out code when needed. You can see this trick used in the above statements and is quite useful in PROC SQL as well.

SQL Code with "dangling commas":

```
proc sql;
   create table TIP1 as
   select    name
           , id
           , address
           , email
           , gender
           , classification
   from data1;
quit;
```

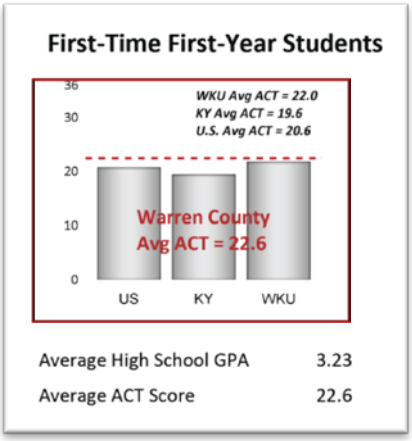
Quickly see that all commas are in place.

USING PROC SGPLOT FOR CHARTS AND GRAPHS

In previous projects, I created basic charts and graphs using the SAS/Graph Procedures GCHART and GPLOT. However, in the spirit of trying new things, I decided to explore the newer SGPLOT Procedure that SAS began offering with version 9.2. When working with SGPLOT, I suppose the main benefit I found was the simplicity and efficiency of the syntax. Rather than having to go to separate, global statements—GOPTIONS, AXIS, LEGEND, PATTERN, SYMBOL, NOTE, etc—these properties are controlled directly within the procedure, making the code a little more svelte. Another feature that is touted in several other papers is the benefit of creating a vector image; this creates a better graphic while decreasing the overall file size.

The first table that I decided to create was a basic bar chart that provided a comparison of average ACT scores on a national, state, and university level compared with the average ACT score of students entering WKU from the specified county. The section at right was achieved using three different outputs, ODS TEXT, PROC SGPLOT, and RWI, but only SGPLOT will be explored here since the others have been discussed in previous sections. I have outlined what I created directly in SGPLOT with the red box. I was pleasantly surprised that I could add all of additional text elements from within the procedure rather than having to create separate ODS TEXT code to overlay on top of the graph.

The code below provides some features I found useful when exploring the SGPLOT Procedure.



The area outlined in red was created in PROC SGPLOT

Figure 13.
First-Time First-Year Student Table

```
ods region    x=2.88in
              y=5.85in
              width=1.95in;

proc sgplot   data=act
              noborder
              aspect=.85;

   vbar       type  /     response=avg_act
                          fillattrs=(color=gainsboro)
                          dataskin=pressed;

   refline    &county_avg   /     lineattrs=(color=firebrick pattern=2 thickness=2);

   xaxis      discreteorder=data
              display=(nolabel noline noticks)
              valueattrs=(family='calibri' size=8pt) ;

   yaxis      values=(0 to 30 by 10 36)
              display=(nolabel noticks)
              valueattrs=(family='calibri' size=8pt) ;

   inset      "U.S. Avg ACT = &US_VALUE"
              "KY Avg ACT = &KY_VALUE"
              "WKU Avg ACT = &WKU_VALUE"
                   /      position=topright
                          textattrs=(family='calibri' weight=bold size=8pt);

   inset     "&mycountyname. County"
              "Avg ACT = 22.6"
              " "
                   /      position=bottom
                          textattrs=(family='calibri' color=firebrick weight=bold size=12pt);

run;
```

For simplicity of design, remove the plot border around the graph.

Aspect ratio defines the dimension of the graph in a range of 0 to 1 with 1 equal to a perfect square and .01 being very short and wide.

This option defines a variable to be the plot value for the bars.

This determines the color of the bars.

DATASKIN provides texture and dimensionality without diminishing readability like a 3-D effect can.

You can combine more than one graph type. Here, I have added a reference line to my vertical bar chart. Another common combination would be combining a bar and line chart.

Bars will be in the same order as they are in the dataset.

Simplify when you can. Remove unnecessary labels & guides.

I indicate the values I wish to display. Here I have 10, 20, 30, and the maximum ACT value of 36.

You can insert text using the INSET statement. Here I have added the averages in the top right corner of the table. To start a new line, simply set the text values in separate quotation marks as I did.

To make this print a little higher than the bottom, I "cheated" by adding a blank line to the INSET statement positioned at the bottom middle part of the page.

Again, this is a pretty basic example, but I feel that it demonstrates how you can set colorization, font specifications, axis information, etc. directly from within the graphing procedure. I would like to note that you can use the template procedure to set style rules for SGPLOT. This will be a fantastic way to standardize and streamline output from your office, ensuring reports are consistent and attractive. I plan to develop our own office template in the future, but that is clearly for another paper, another day.

In the second example, I create a horizontal bar chart that provides information on the percentage of degrees awarded by each of our six academic colleges. Because I want the user to be able to quick ascertain the county's most popular college, I first order the data in descending order of frequency so that when I define the DISCRETEORDER as "data", it will populate the table in descending order.

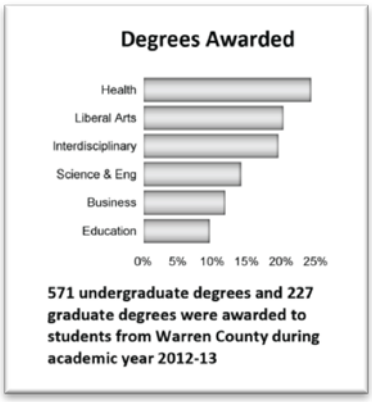


Figure 14.
Degree Awarded Table

Both the title and the narrative at the bottom was created as an ODS TEXT object and so will not be addressed here as to avoid repetition of previous topics.

```
ods region x=5.50in
           y=5.85in
           width=2.00in;
```

Sort first so that the data are ordered by the number of students per college, with the largest being first and the smallest being last.

```
proc sort data=degree_college; by descending coll_count ;
run;

proc sgplot data=degree_college noborder aspect=1.0;
     hbar    college  /  response=coll_count
                        stat=percent
                        fillattrs=(color=gainsboro)
                        dataskin=pressed;
```

I use the variable "coll_count" as my RESPONSE variable, the variable that defines the plot value. Because I want a percentage rather than a sum (which is the default statistic), I define the STAT option as percent. Please note that both RESPONSE and STAT are optional. If I didn't specify, the graph would reflect a basic frequency by category.

```
     xaxis   display=(nolabel noline noticks)
             valueattrs=(family='arial' size=8pt) ;

     yaxis   discreteorder=data
             display=(nolabel noline noticks)
             valueattrs=(family='arial' size=8pt);

     format college $x_coll.;
```

This is where you will define the display order for the graph so that the graph will present the data from largest to smallest.

You can add formats just as you would in other procedures.

```
run;
```

As with the ODSTEXT Procedure and the Report Writing Interface, I have merely scratched the surface of the capability of SGPLOT (not to mention SGPanel and SGSCATTER). Hopefully, however, I have demonstrated the ability to create an attractive, readable graph all from within the SGPLOT Procedure.

TAKE IT TO THE NEXT LEVEL

Using Macros to Maximize Flexibility

One of the greatest features of programming for publications in SAS is that you can use to the power of macro programming to create great, customized reports on the fly. In this instance, I was able to loop through each of our 120 counties, 38 senate districts, 100 house districts, and three regional campus, and create a specialized report for each.

Please take a moment to review my 2011 paper, "Absolutely Fabulous…" for further details on how to "macro-ize" a program (Huff, 2011).

Functionality as a Stored Process

This lends itself beautifully to the Stored Process. By using the Stored Process Wizard in Enterprise Guide, I can add Base SAS Code so that users can select parameters (Report Type, County/District/Campus Name, Term) to create their report. Please note that you should remove the ODS PDF statement from the code when you paste it into Enterprise Guide because the stored process macros created in the Wizard will initialize ODS.

CONCLUSION

If nothing more, I hope that I have accurately demonstrated SAS's capacity to create eye-catching reports that will serve the needs of your users. It can be easy to balk at the time and effort it takes to learn some of the data visualization techniques SAS provides; after all, it should be about the data and not how "pretty" the report is. Right? (Words that I have heard on occasion.) I would have to disagree and share this simple reminder…*a user has to actually look at the report to garner any useful information from it.* Hopefully, some of the tips and techniques I provided will spark your own creativity so that you can develop some RWI, some Really Wonderful Information, of your own!

REFERENCES

Huff, Gina (2011). "Absolutely Fabulous: Tips on Creating a Publication-Ready Report Using ODS Absolute Layout Functionality." *Proceedings of the 2011 SAS Global Forum Conference.* Available at http://support.sas.com/resources/papers/proceedings11/293-2011.pdf.

Hunter, Tim (2013). "A First Look at the ODS Destination for PowerPoint." *Proceedings of the 2013 SAS Global Forum Conference.* Available at http://support.sas.com/resources/papers/proceedings13/041-2013.pdf.

Huntley, Scott (2012). "A Different Point of View with ODS PDF in SAS 9.3" *Proceedings of the 2012 SAS Global Forum Conference.* Available at http://support.sas.com/resources/papers/proceedings12/260-2012.pdf.

Lund, Pete (2013). "Have It Your Way: Creating Reports with the Data Step Report Writing Interface." *Proceedings at the 2013 SAS Global Forum Conference.* Available at http://support.sas.com/resources/papers/proceedings13/040-2013.pdf.

O'Connor, Daniel (2013). "Take Home the ODS Crown Jewels: Master the New Production Features of ODS LAYOUT and Report Writing Interface Techniques." *Proceedings of the 2013 SAS Global Forum Conference.* Available at http://support.sas.com/resources/papers/proceedings11/293-2011.pdf.

O'Connor, Daniel (2009). "The Power to Show: Ad Hoc Reporting, Custom Invoices, and Form Letters." Available at https://support.sas.com/rnd/base/datastep/dsobject/Power_to_show_paper.pdf.

Zender, Cynthia L. (2007). "Funny ^Stuff~ in My Code: Using ODS ESCAPECHAR." *Proceedings of the 2007 SAS Global Forum Conference.* Available at http://www2.sas.com/proceedings/forum2007/099-2007.pdf.

CONTACT INFORMATION

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

Gina Huff
Office of Institutional Research
1906 College Heights Blvd. #11011
Bowling Green, KY 42101
(270) 745-3250
gina.huff@wku.edu