Paper CT-22

Schedule Impossible: Using ODS and PROC REPORT to Create a Schedule Visualization

Jeffrey Reiss, University of Central Florida, Orlando, FL

ABSTRACT

When preparing reports for mass user consumption, those with unique requirements beyond an organization's expected norm may surface periodically. While some of these reports may appear to be impossible to create at first glance, the current abilities of PROC REPORT and SAS® ODS are sufficiently developed to make these reporting anomalies possible. In this paper, the author will discuss how a report that was once deemed "impossible" to produce in the SAS environment can indeed be made possible.

INTRODUCTION

Most SAS coders can likely cite at least one instance of having to push SAS procedures to their limits in order to properly complete a task. The Registrar's Office of a major metropolitan research university needed to take that precise action in order to successfully migrate to SAS an automated report that uses a side-by-side display pairing a table-generated graphical object alongside with a standard, text-based table. By utilizing some clever DATA step techniques and PROC TRANSPOSE to manipulate the dataset, pushing the limits of PROC REPORT, and implementing some experimental ODS commands to generate the display, a seemingly impossible report conversion was made possible.

THE PROBLEM

The Registrar's Office of a major metropolitan research university is in the process of converting their report generation process from a Microsoft Access-based automated system to a SAS-based automated system. Although most of the reports produced are simple to port to SAS, one particular report conversion was notably more daunting. As shown in Figure 1, this particular report lists all of the classes held in a given room on campus in a given semester. The list is accompanied by a visualization of the class start and end times against a set of fixed blocks for scheduling finals. While the list of classes is no different than the one found in any of the other reports, the accompanying visualization, originally constructed in Microsoft Excel with a Visual Basic macro, served as the component of concern. The other major issue is that the two components are side-by-side in nature, which is not generally handled in the most straightforward of fashions with most SAS procedures.

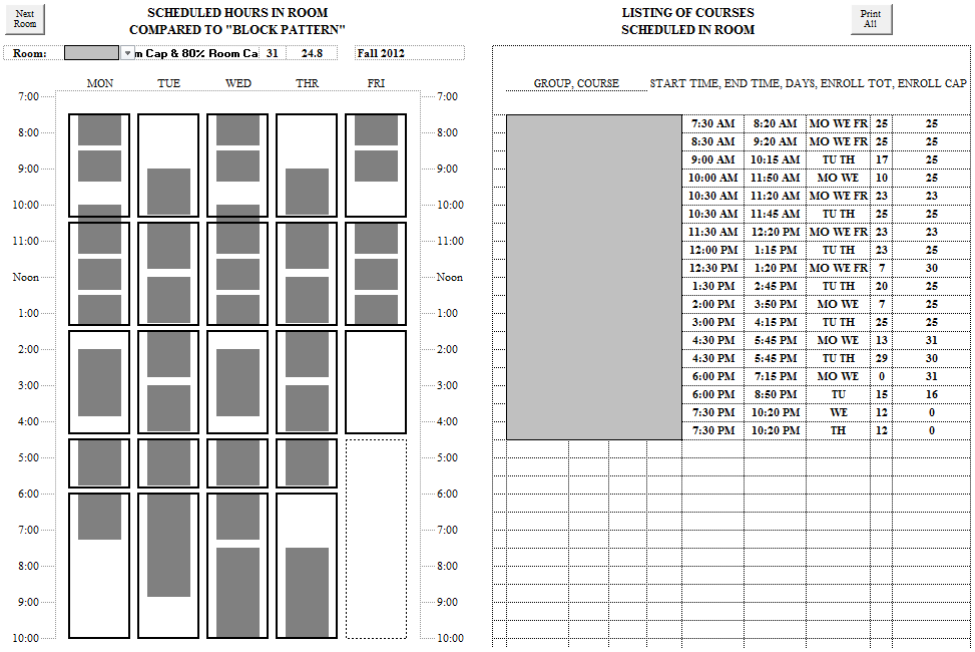


Figure 1. The Original Report

THE PLAN

Fortunately, the SAS ODS tools provide some useful features to overcome these hurdles. The experimental feature ODS REGION will correct the side-by-side issue. This feature creates absolute regions in which to place multiple result items. As of SAS 9.4, the status of this feature will turn from experimental to production. However, depending on the type of problem at hand, ODS COLUMNS could also work. The ODS COLUMNS approach does not work in this situation because of the risk of exceeding one page for the table portion of the report. While the obvious hurdle can be fixed by ODS, the other problem lies in the actual creation of the visualization. Fortunately, since the visualization is tabular in nature, the use of PROC REPORT with some extensive modifications can make this task possible.

In order to make this report work, both components require their own dataset. The table portion is based on the original finalized dataset used by the report. The visualization portion uses a dataset that places the time data into a tabular map. Within this map, each time segment is given a numeric code to indicate whether there is no class scheduled, there is a class scheduled without issue, or if there is a class scheduled that violates the blocking rules. With these components, this seemingly impossible report will become functional.

PREPARING THE VISUALIZATION DATASET

The dataset for the table portion only includes start and end times as data points. In order to produce the visualization dataset as planned, however, the time between the start and end also needs to be recorded as data points. Therefore, the first step is to code the time. All scheduled times may run between 7:00 AM and 11:00 PM and need to be tracked in five-minute blocks to accommodate all possible times. The method of coding used to meet this need involved creating a scale between 1 and 193 where 1 represents 7:00 AM, 193 represents 11:00 PM, and each increment represents five minutes. Every start and end time was coded into this scheme. Furthermore, the end time is decreased by one on the scale to avoid the visualization from otherwise displaying the class as exceeding its time by an additional five-minute block of time. Each of the time blocks in which all the classes must reside feature ten-minute gaps between each other; these boundaries have static time codes. If a class's start and end time cross any of the time block boundaries, the class entry would be assigned a value indicating a violation. Otherwise, the entry would be assigned a standard value.

The next step is to translate the coded start and end times into the individual five-minute entries. This component is completed by making an array to fit the data for each day of the work week. By using a DO UNTIL(last.var) style loop known as the DOW loop, each classroom is assigned its own entry with arrays of the entire schedule.

```
do until(last.facility_id);
  set work.report_prep;
  by facility_id;
  do i = convstart to convend by 1;
    if mon = 'Y' then monday___[i] = eCode;
    if tues = 'Y' then tuesday__[i] = eCode;
    if wed = 'Y' then wednesday[i] = eCode;
    if thurs = 'Y' then thursday_[i] = eCode;
    if fri = 'Y' then friday___[i] = eCode;
  end;
end;
```

While the data are now presented in the necessary format, PROC REPORT cannot read horizontally. Thus, the data needs to be transposed. Although PROC TRANSPOSE can transpose all the fields at once without issue, all of the arrays will end up in one column. Fortunately, the original array names are saved in a column for easy splitting. After sorting by the time code so that all five days can be simultaneously processed, another DOW loop is used to split the one variable of transposed data into the individual days. Within this step, other preparations are made to get the data ready for the report, such as converting the time codes back to their respective hours for the axis and creating blank "buffer variables." After this last portion is completed, the visualization dataset is ready for processing. The following code takes care of these issues.

```
do until(last.obs);
  set work.bartrans2;
  by facility_id obs;
  if day = "monday" then Monday = tick;
  else if day = "tuesday" then Tuesday = tick;
  else if day = "wednesday" then Wednesday = tick;
  else if day = "thursday" then Thursday = tick;
  else if day = "friday" then Friday = tick;
```

```
    hour = floor(obs/12) + 7;
    _emptym = '';
    _emptyt = '';
    _emptyw = '';
    _emptyr = '';
    _emptyf = '';
    _emptys = '';
  end;
```

ASSEMBLING THE VISUALIZATION

While everything is in proper order for PROC REPORT to make the visualization component, the difficulty comes in the styling and formatting. The main areas for styling this visualization is traffic lighting for the schedule display, spacing the axis and schedule columns, and drawing the block borders.

TRAFFIC LIGHTING AND SPACING THE COLUMNS

Traffic lighting the observations is one of the simpler tasks in this project. The first step is to create a format that reflects the desired color scheme. The visualization uses a missing value to indicate there is no class scheduled, a '1' to indicate there is a class scheduled, and a '2' to indicate that there is a class scheduled that violates the blocking rules. The format associates each of these values with a color (in this case, white, gray, and red, respectively) while another format is applied to the variable to keep the values from printing in the report. Within each day's style statement, the background is set equal to the color format.

Due to the span of time used for the visualization, it cannot fit comfortably on a single page with default settings. Furthermore, the default settings of PROC REPORT generate gridlines around all cells. While the gridlines are fine for the other half of the report, they make the visualization difficult to read—especially with the time block borders. Thus, these extra gridlines need to be removed for both readability purposes and to make the visualization consistent with the Excel version. To remove these and ensure that all the cells are touching each other, using the following code will correct this issue:

```
style(report)=[rules=none frame=void cellspacing=0 cellpadding=0]
```

The RULES=none and FRAME=void options will remove the lines, while CELLSPACING=0 will ensure that there is no whitespace between the cells. CELLPADDING=0 was used to minimize the space occupied by the cell in order to make sure it all fits on the page. Even with these changes, the visualization is still larger. Fortunately, SAS will adjust the cell heights to the largest font size. Likewise, shrinking the font size will also shrink the cells. For the visualization, the font size was reduced to 3-point size, allowing the whole report to fit comfortably within the page.

Shrinking the font, however, has one unintended consequence. The time column that represents the axis of the visualization is also forced to 3-point font. Even though the SPANROWS option is used for the report, PROC REPORT will still treat the one cell occupied by text as a single cell rather than a larger merged cell. To circumvent this issue, SAS provides a workaround that will assist in obtaining the desired result. By adding into the Time column's STYLE(column) definition the options for CELLHEIGHT, CELLWIDTH, and VJUST, the text can be broken out of its cell.

```
style(column) = {cellheight=.03in vjust=m cellwidth=.5in borderleftcolor=black
borderleftwidth=3 bordertopwidth=2 borderbottomwidth=0 fontsize=8pt}
```

This method shown above, however, has some significant caveats. First, the CELLHEIGHT option does affect the first row in each SPANROWS block. Therefore, for the purpose of this application, the height was adjusted to 0.03 inches to keep it in line with the height of the other cells. Because of the increase in font size for the Time column's text, the text ends up shifted upwards, crossing upper borders. This clearly is unacceptable and can be mitigated by using VJUST=m to center the text in each SPANROW box. As of the time of writing of this paper, it is currently not possible to align the text to the top of the SPANROW box without the text crossing the top border. It has been classified by SAS as a defect in the procedure and is currently being looked into further by programmers at SAS.

ADDING THE BORDERS

Correctly adding the borders for the time block boundaries is the part of the visualization that requires the most creativity. Fortunately, each variable can have its own definition and COMPUTE block. Hence, the variable's initial definition lays out the base rules, such as a side border, while the COMPUTE block styles a specific row for the designated areas. The order of the styling in the definition statements and COMPUTE blocks is imperative because the style definitions will override each other if they share a common option, with precedence put on the last definition.

```
define monday / 'Mon' style(column) = {background=$ticklight.
  foreground=$ticklight. borderleftcolor=black borderleftwidth=2 borderrightwidth=2
  fontsize=3pt cellwidth=50} style(header) = {borderbottomcolor=black
  borderbottomwidth=2 bordertopcolor=black bordertopwidth=2 };
```

As mentioned in the example above, the first style command in the definition statement styles the columns the way most of the rows will need to be styled, while the style statements in the compute block handle styles that are used for fewer cases. This methodology of styling from most used to least used also applies to other programming languages such as CSS for HTML.

For the visualization, each of the days will have outlines to display the time as was shown in Figure 1. Each of the days is initially styled with side borders, as most of the observations will be located within a block. If the report were to be run at this point, the day columns would just look like columns with the appropriate areas colored in, since the borders for the blocks are at specific points on the report. In order to tell SAS where these points are located, a hidden column is added to the report that contains all the observation numbers. The observation numbers are ordered in the same 1-193 scheme as described in the data preparation section. Therefore, the same values that were used to identify the gaps between the blocks can be used to identify the block border lines and blank regions. To ensure that the style goes across all the days and not the time column, the observation column is placed immediately after the time column so that only the columns to the right of the observation column are affected.

```
compute obs;
  if obs < 6 or obs > 181 then do;
    call define(_ROW_, 'style', 'style=[borderleftwidth=0 borderrightwidth=0]');
  end;
  else if obs = 6 or obs = 42 or obs = 78 or obs = 114 or obs = 132 then do;
    call define(_ROW_, 'style', 'style=[borderbottomcolor = black
    borderbottomwidth=2 borderleftwidth=0 borderrightwidth=0]');
  end;
  else if obs = 41 or obs = 77 or obs = 113  or obs = 131 or obs = 181 then do;
    call define(_ROW_, 'style', 'style=[bordertopcolor = black bordertopwidth=2
    borderleftwidth=0 borderrightwidth=0]');
  end;
endcomp;
```

In the COMPUTE block for the observation column above, a set of three IF statements determine the correct case (blank, bottom border, top border) and run the appropriate style definition. Finally, because Friday's evening blocks are different than the rest of the week, another compute block is written strictly for the Friday column using similar logic to the observation column COMPUTE block. While it is possible to add the initial style definition to the observation COMPUTE block, the method used here reduces the number of style calls. Each style call in the define statements are only called once, where the compute block runs for every line.

PUTTING IT ALL TOGETHER

With the visualization completed, all that remains is to place the two reports together. Because an absolute layout is being used, each side of the report must be created in alternation. Therefore, each room needs to be processed individually. To accomplish this task, all the prior code is stored into a macro with the ODS layout and region options. Since each report will generate its own set of bookmarks, one of the two needs to be suppressed as both reports are on the same page. Using the BOOKMARKGEN=yes option before the visualization report and BOOKMARKGEN=no before the regular report will fix this issue. Adding in some CONTENTS options to the visualization report will also help clean up the bookmarks. For example, adding in an extra column that contains the same value through the report can remove the third-level bookmarks that do not serve a purpose in this particular example:

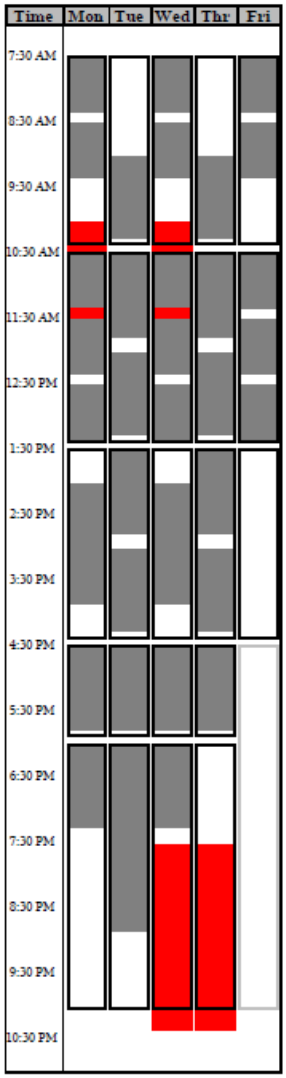
```
break before building / page contents='';
```

Setting CONTENTS equal to a blank value prevents the bookmark from appearing. Using CONTENTS in the report options will change the second-level bookmarks and ODS PROCLABEL will change the first level.

After the ODS LAYOUT and REGION have been set, the final step is to use the macro to loop through the dataset. Since this dataset should be sorted by term, building, and room for easy reading and navigation, two levels of looping need to be implemented. Furthermore, other variables such as room capacity and term need to show up in the report. Because of the by row constraint, two nested CALL EXECUTE statements were implemented. The first statement cycles through a dataset with unique term numbers, while the second one cycles through the unique rooms pre-sorted by building. Within the macro called by the term execute, the PDF output is initialized with the term number and other variables needed for the title lines are prepared. Within the macro called by the room execute is the rest of

the reporting code. Thus, two PROC REPORT statements are called for each room for each term. Figure 2 shows a page of the finished product.

Block Scheduling report for [redacted] *in Fall 2012* 1
Room capacity: 31 (80%: 25)



					Enrollment	
Group	Course	Start	End	Days	Tot	Cap
		7:30 AM	8:20 AM	M W F	25	25
		8:30 AM	9:20 AM	M W F	25	25
		9:00 AM	10:15 AM	T R	17	25
		10:00 AM	11:50 AM	M W	10	25
		10:30 AM	11:20 AM	M W F	23	23
		10:30 AM	11:45 AM	T R	25	25
		11:30 AM	12:20 PM	M W F	23	23
		12:00 PM	1:15 PM	T R	24	25
		12:30 PM	1:20 PM	M W F	7	30
		1:30 PM	2:45 PM	T R	20	25
		2:00 PM	3:50 PM	M W	8	25
		3:00 PM	4:15 PM	T R	25	25
		4:30 PM	5:45 PM	M W	12	31
		4:30 PM	5:45 PM	T R	29	30
		6:00 PM	8:50 PM	T	15	16
		6:00 PM	7:15 PM	M W	0	31
		7:30 PM	10:20 PM	W	12	0
		7:30 PM	10:20 PM	R	12	0

Figure 2. The SAS Version of the Report

CONCLUSION

While it may not look possible at first, creating a table-based graphic in PROC REPORT is definitely possible. Depending on how it is used, trial and error is definitely involved—especially if specific border lines are involved. Regardless, SAS does provide all the tools needed to get the job done.

REFERENCES

- A. M. Booth. 2011. “Beyond the Basics: Advanced REPORT Procedure Tips and Tricks Updated for SAS 9.2.” *PharmaSUG 2011*. Nashville, TN. Available at http://www.pharmasug.org/proceedings/2011/SAS/PharmaSUG-2011-SAS-AD02.pdf
- E. Tilanus. 2007. “Turning the Data Around: PROC TRANSPOSE and Alternative Approches.” *SAS Global Forum 2007*. Orlando, FL. Available at http://www2.sas.com/proceedings/forum2007/046-2007.pdf
- P. Dorfman, L. Shajenko. 2008. “The DoW—Loop Unrolled.” *SESUG 2008*. St. Pete Beach, FL. Available at

http://analytics.ncsu.edu/sesug/2008/CS-059.pdf

- R. Nelson. 2010. "ODS LAYOUT to Create Publication-Quality PDF Reports of STD Surveillance Data." *SAS Global Forum 2010*. Seattle, WA. Available at http://support.sas.com/resources/papers/proceedings10/216-2010.pdf

CONTACT INFORMATION

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

Jeffrey Reiss
University of Central Florida
Registrar's Office
PO Box 160114
Orlando, FL 32816-0114
(407) 823-0587
Jeffrey@ucf.edu