PhUSE EU Connect 2019

Paper DV03

Using ODS LAYOUT to Create ABSOLUTE-ly Stunning Patient Profiles

Michael Allie, Oncology Biometrics, Early Oncology Programming, R&D, AstraZeneca, Cambridge, United Kingdom

ABSTRACT

Summarising key patient information usually involves incorporating a variety of data from multiple sources. For each topic included, there is an optimal way to display this information in an output and occasionally these differ from each other; varying numbers of columns, bespoke style elements and completely different output types (listing vs plot etc.) are not generally utilised within the same SAS procedure. This can make it difficult to create patient profiles which often involve a breadth of information which has no single best way to be displayed.

Enter ODS LAYOUT ABSOLUTE, a tool for arranging multiple SAS procedure outputs into a single combined output.

This paper describes the basics of using ODS LAYOUT ABSOLUTE to build custom arrangements of SAS outputs and explores methods to add variety and efficiency to the creation of unique composite outputs which effectively convey diverse data into a single patient profile.

INTRODUCTION

Efficiently portraying information increasingly involves making the most out of the available space in an output. This is especially true when monitoring individual patients on an ongoing basis; physicians greatly benefit from a comprehensive overview containing the most critical pieces of information at a glance. Programmatic creation of these outputs saves significant time and improves quality compared to compiling this information manually. This facilitates more responsive decision making over the course of a clinical trial. Accomplishing this goal within SAS is not always straightforward since procedures are designed to primarily create a single output per page.

There are methods for incorporating tabular information into a graphic such as axis tables, but this typically requires that the data in the table be directly linked to the data plotted. Combining data that is more indirectly linked (e.g. qualitative data and graphical data about a particular patient) is outside of the scope of most SAS procedures.

ODS LAYOUT allows you to combine the outputs from multiple SAS procedures together, creating a single output which displays multiple summaries of indirectly linked data. There are two variations available: GRIDDED and ABSOLUTE, each of which has unique advantages and disadvantages depending on the requirements of the output.

ODS LAYOUT: GRIDDED

The output area is divided up into a two-dimensional grid, where a procedure output occupies a cell of this grid. GRIDDED layouts are supported by HTML and printer destinations (PDF, PS and PCL) as well as the PowerPoint destination.

You begin by defining a layout with an ODS LAYOUT GRIDDED statement and finish the layout definition with an ODS LAYOUT END statement.

Syntax:
```
ODS LAYOUT GRIDDED <options>;
   [Output generating procedure code]
ODS LAYOUT END;
```

The required syntax is relatively simple and one of the largest advantages of the GRIDDED layout is that most of the sizing and alignment of items within the layout is done automatically.

Basic details of the layout are controlled by the options on the ODS LAYOUT GRIDDED statement:

- HEIGHT and WIDTH options control the overall dimensions of the layout
- COLUMNS and ROWS options control the number of columns and rows in the grid
- COLUMN_WIDTHS and ROW_HEIGHTS control the dimensions of the cells within the grid.
- COLUMN_GUTTER and ROW_GUTTER control the amount of empty space between grid cells
- The ORDER_TYPE option controls the order in which the layout is filled by default
    - COLUMN_MAJOR specifies that all regions in the first column are filled before moving on to the next column
    - ROW_MAJOR specifies that all regions in the first row are filled before moving on to the next row

There are other options available, but these control most of the characteristics of the layout.

While the dimensions of the cells can be assigned using these options when they are known in advance, it is not required to do so; the height and width of the columns will be calculated automatically based on the procedure output.

Controlling the specific details of each individual cell within the layout is accomplished through use of ODS REGION statements within the layout block, preceding the output generating procedures.

BUILDING REGIONS: GRIDDED

Multiple regions are created within a gridded layout using ODS REGION statements.

Syntax:
```
ODS LAYOUT GRIDDED <options>;
   ODS REGION <options>;
     [Output generating procedure code]
ODS LAYOUT END;
```

Note that there is no ODS REGION END statement; each region may only contain the output from a single SAS procedure.

Options within the ODS REGION statement specify which cell(s) of the total layout the region contains and some aesthetic details of the region.

- HEIGHT and WIDTH options determine the dimensions of the region within the cell. The overall dimensions are restricted by the COLUMN_WIDTH and ROW_WIDTH specified for the layout. Regions cannot be larger than the cells that they occupy
- COLUMN and ROW options specify the exact cell the region is contained within
- COLUMN_SPAN and ROW_SPAN options allow regions to span across multiple cells of the output
- The STYLE option specifies style elements such as border and background colours

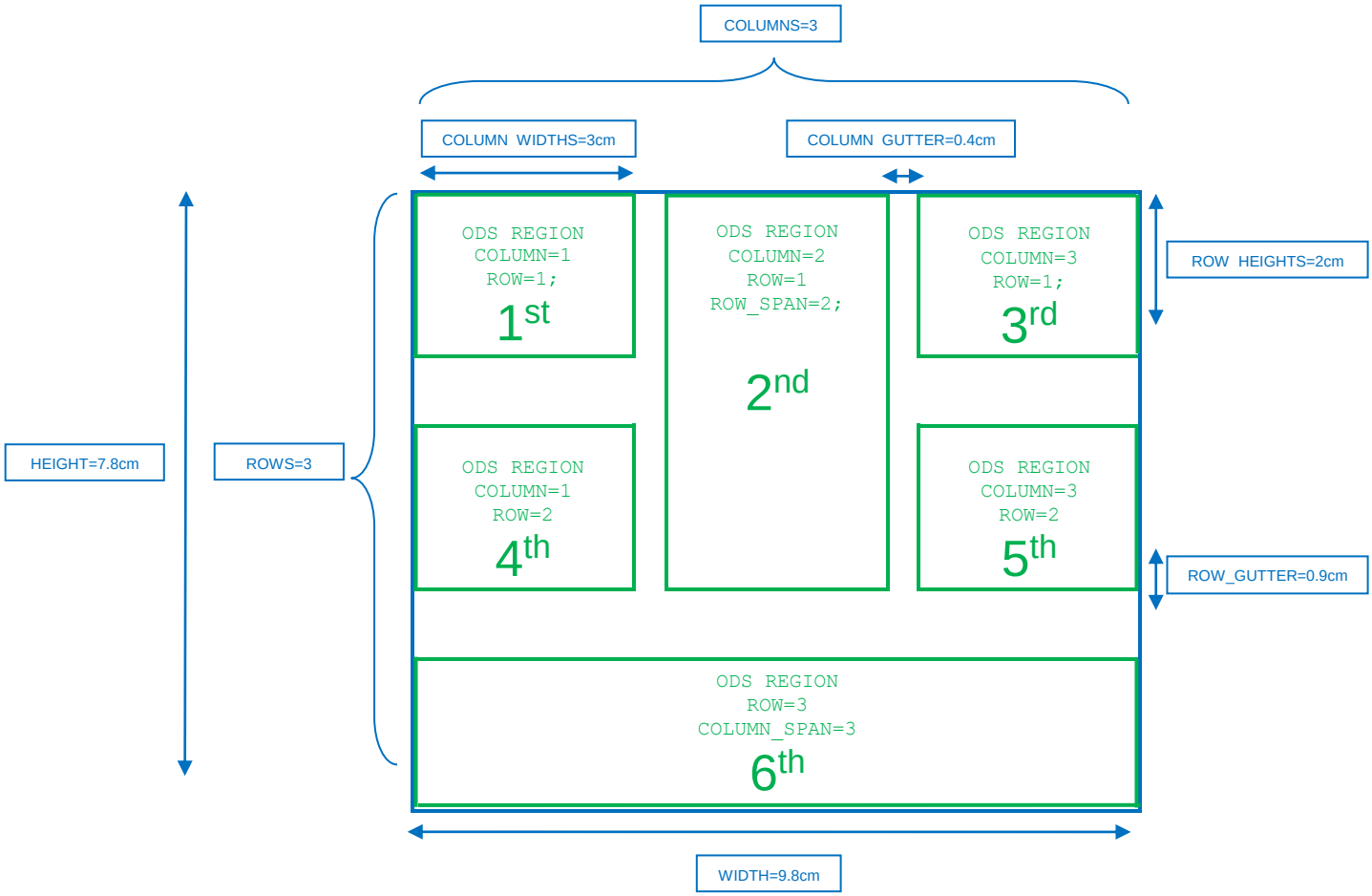
Regions in a gridded layout must be declared in order; e.g. For a 2X2 grid, you could not declare the region in row 2 column 2 before declaring the region in row 1 column 1.

The next page shows a diagram of what a GRIDDED layout would look like using the code below.

Layout code:

```
ODS LAYOUT GRIDDED height=7.8cm width=9.8cm
                   columns=3 rows=3
                   column_widths=3cm column_gutter=0.4cm
                   row_heights=2cm row_gutter=0.9cm
```

DIAGRAM OF GRIDDED LAYOUT



The diagram above details various elements that make up a gridded output. The options in blue are used in the ODS LAYOUT GRIDDED statement to control the structural elements of the overall layout, and the options in green are used in individual ODS REGION statements to build each region out of the all the cells of the layout. The ordering here is ROW_MAJOR because the layout is filled one row at a time.

LIMITATIONS OF THE GRIDDED LAYOUT:

A major limitation of the gridded layout is that regions may not overlap each other. Due to the amount of space that SAS allocates around each procedure output (to account for options such as thicker borders), there is a limit to how close adjacent regions can be to each other; even if a gutter width of 0 is specified, there will be noticeable space between regions which may be problematic for creating a composite output with the individual items closely packed together. This is not the case when using ODS LAYOUT ABSOLUTE, as that removes the requirement to conform to a non-overlapping grid structure, so the outputs can be packed so closely together that SAS believes they are overlapping. Absolute layouts have full flexibility over the location of regions so for outputs where minimising whitespace between regions is important, the additional control is essential.

ODS LAYOUT: ABSOLUTE

The output area is a single page where regions are specified by exact coordinate points; each region contains a single procedure output but regions are allowed to overlap with each other. Regions are rendered in the order they are declared within the code, so latter regions will appear over the top of previous ones in the instance they occupy common space within the layout.

Syntax:
```
ODS LAYOUT ABSOLUTE <options>;
  [Output generating procedure code]
ODS LAYOUT end;
```

The minimum syntax is almost identical to that of the GRIDDED layout but the specification of regions is more intricate with the ABSOLUTE layout; dimensions and location of each region are explicitly defined rather than simply referencing a cell based on row and column from a grid.

BUILDING REGIONS: ABSOLUTE

As with the GRIDDED layout, regions are created in an ABSOLUTE layout by issuing ODS REGION statements before code that generates a procedure output.

Syntax:
```
ODS LAYOUT ABSOLUTE <options>;
   ODS REGION <options>;
     [Output generating procedure code]
ODS LAYOUT END;
```

There are fewer options in the ODS REGION statement in an ABSOLUTE layout but in practice, defining these regions is more complicated than those of the GRIDDED layout because relying on the default values is often ineffective when creating composite outputs made up from multiple individual procedure outputs. For example, if you create multiple regions using the default X and Y values (0,0), they will be placed directly on top of each other.

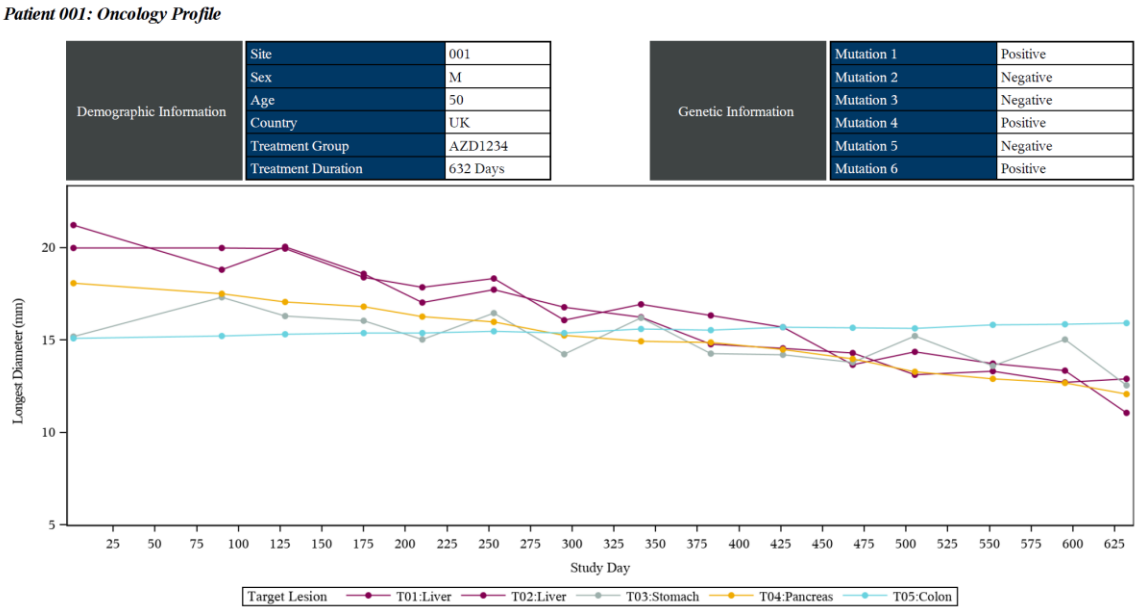
- HEIGHT and WIDTH options determine the dimensions of the region. The height and width of all regions may not exceed the total space allocated for the layout
- X and Y options specify the horizontal and vertical starting position of the region. Regions begin from their top left corner
- The STYLE option specifies style elements such as border and background colours

While the dimensions of each region can be determined dynamically based on the procedure output, other regions in the layout will likely need their location specified. It is beneficial to know the expected dimensions of each region in advance so subsequent regions can be positioned in a way that avoids unintended overlap.

EXAMPLE 1: ONCOLOGY DASHBOARD

Below is an example output summarising some key oncology information* for a single patient.

*All data in this paper has been simulated



There are 6 different procedure outputs within this layout

- The output title is generated with proc ODSTEXT to allow close positioning to the rest of the outputs in the layout in order to save space
- The grey section headers “Demographic Information” and “Genetic Information” are created using proc ODSTEXT with the background colour being controlled with a style= option
- The tabular information in these sections is displayed using proc REPORT
- The graphic at the bottom of the layout is created by proc SGRENDER and GTL

The ABSOLUTE layout is required to achieve the tight spacing between the various layout regions. Technically, the section headers, tabular information and bottom graphic all overlap with each other according to the amount of space allocated by SAS for each output; in a GRIDDED layout, they would be forced to appear further apart.

Below is the full code used to create this output:

```
ODS LAYOUT ABSOLUTE;

ods graphics on / border=off;
ods region x=0.165in y=2in width=13in height=5in style=[borderBottomColor=CXFFFFFF];
%linePlot(data=diameters);
ods graphics off;

ods region x=0.165 y=0.15 width=3in height=0.4in;

proc odstext;
  p "Patient 001: Oncology Profile" / style=[font_weight=bold font_style=italic
                                             fontSize=15pt vjust=m just=center
                                             cellHeight=0.3in];
run;

ods region x=0.92in y = 0.47in width=2in height=1.58in
style=[backgroundcolor=CX3F4444];

proc odstext;
  p "Demographic Information" / style=[color=CXFFFFFF vjust=m just=center
                                       cellHeight=1.58in];
run;

ods region x=2.945in y=0.47in width=3.5in height=1.59in;
%tabular(data=demog);

ods region x=7.56in y=0.47in width=2in height=1.58in style=[backgroundColor=CX3F4444];

proc odstext;
  p "Genetic Information" / style=[color=CXFFFFFF vjust=m just=center
                                   cellHeight=1.58in];
run;

ods region x=9.585in y=0.47in width=3.5in height=1.59in;
%tabular(data=genetic);

ODS LAYOUT END;
```

There are a few interesting (and possibly unintuitive) details about this code: The graphic is actually added first, so that the other outputs appear above the additional white space allocated as part of the graphic. Additionally, the regions for the proc odstext headers are slightly smaller than the regions for the proc report outputs they align with. This is due to more white space placed around the report. The cellHeight specified in the title of the first proc odstext style= option is smaller than the actual height of the region, this allows the text to be placed nearer to the top of the block than the middle.

There was a lot of trial and error involved in selecting the dimensions and positions of each element in the layout; now that it’s completed, the input data can simply be changed to produce this layout for different patients but a structural change (e.g. a new category in either of the tables) will require more adjustments to almost every element. This can be quite a time-consuming process even for this relatively simple layout.

The next example is of a more complicated layout; Example 2A is my initial draft, which defines all the regions manually in open code as I have done above. Example 2B is an improved version which utilises some techniques to efficiently and dynamically build regions using macros.

EXAMPLE 2A: SAFETY SUMMARY (MANUAL REGIONS)

There are 12 procedure outputs in total here. 6 proc odstext blocks on the left to label each major section of the layout and an accompanying proc report to the right of each one which displays the relevant information from the clinical database.

The largest issue with this first draft using manually defined and fixed region dimensions is the amount of blank space in each region. This is caused by a varying amount of information to be displayed for each patient. For example, this mock patient only has 2 lines of information under the “Seizures on Study” section. Another patient may have more lines here, so the additional space must be present in order for the layout to have enough space to accommodate all the patients.

Additionally, any updates to the dimensions or position of a region requires manual updates to all subsequent regions. Considering changes in the data can necessitate increases to height of regions, there is a likely chance of being dragged into a series of very time-consuming manual updates.

Below is an extract of code to create one section of the layout:

```
* Horizontal title panel;
  ods region width=1in height=0.85in x=0.1in y=5.41in
             style=[backgroundcolor=CX830051];

    proc odstext;
      p "Missed Doses" / style=[vjust=m just=center cellheight=0.85in
                                color=CXFFFFFF];
    run;
```

```
ods region width=9in height=0.85in x=1.14in y=5.41in style=[backgroundcolor=CXD9D9D9];

proc report data=report7 style(report)=[rules=none outputwidth=9in frame=void
cellspacing=0] noheader;
  columns ord COL2;
  define ord / noprint order order=data;
  define COL2 / left width = 20 style(column)=[width=9in backgroundcolor=CXD9D9D9];
run
```

EXAMPLE 2B: SAFETY SUMMARY (DYNAMIC REGIONS)

GBM History	**E0000001: Male, 50, (White) Most Recent Visit (FOLLOW-UP 12SEP2019)**				
	Initial Diagnosis 01JAN2000			**GBM progression date** 02FEB2002	
	Treatment for initial GBM **Cancer Therapy:** Rituximab 02JAN2000-03MAR2001 Pembrolizumab 10APR2001-05MAY2001			**Treatment for relapsed GBM** **Radiotherapy:** 52Gy 06MAR2002-18MAR2002	
GBM-Seizure History / Treatment	GHM Seizure History: Freq [<1/MONTH] Features [Focal, Generalised] (Pre-Study) Levetiracetam 100mg QD 01JUN2005-28JUL2006				
Steroid Use	(Pre-Study) Hydrocortisone 10mg QD 01APR2005-08AUG2006 Prednisone 40mg BID 02OCT2018-16OCT2018 Dexamethasone 10mg BID 12NOV2018-22DEC2018 Dexamethasone 12mg BID 23DEV2018-12FEB2019 Dexamethasone 10mg QD 13FEB2019-Present				
Cycle Dates	**Cycle 0**	**Cycle 1**	**Cycle 2**	**DLT Period Complete**	**Follow-up #1**
	17SEP2018	24SEP2018	12OCT2018	01NOV2018	15JAN2019
Missed Doses?	AZD1234: Dose Interrupted on 15OCT2018 (Cycle 2) due to Adverse Event (Nausea) Radiotherapy: No missed doses				
Seizure(s) on study	2 Episodes of [Focal] seizure on 04DEC2018 1 Episode of [Generalized] seizure on 18DEC2018				

Above is an upgraded version of the output where each region is sized depending on the size of the report it contains. This mimics one of the main benefits of the GRIDDED layout while still maintaining ABSOLUTE control of the position of all elements of the layout.

While technically the height and width of a layout are optional in the ods region syntax, in practice the size of each report region in this layout must be known in order to use a matching height for the proc ods text regions on the left which label each major section of the layout. More importantly, the position of a report region is dependent on the height and position of all preceding report regions. Implementing proper spacing to avoid overlaps involves positioning each region further down on the page than the height of the previous region.

The dimensions of the page have also been adjusted to a landscape aspect ratio. Widening the layout also helps minimise the chance of any text wrapping occurring; this version of the code operates on the assumption that there will be no wrapping in the proc report.

This effect was achieved with two utility macros.

MACRO 1: COUNT_REP_ROWS

The biggest factor in determining the amount of space required for a proc report is the number of rows needed to present all the information in the final output. This idea led to the creation of the first utility macro %count_rep_rows. It counts the number of observations in the input dataset and the instances of a specified newline delimiter (e.g. ^{newline}).

The newline delimiter is a keyword parameter of the macro (newline=). The instances of this string are counted to determine how many additional rows will be needed to present this data in the final output.

```
%* Counting number of newline delimiters present in dataset;
data _null_;
  set &data. end=EOF;
  array cols{*} _character_;
  retain __extraRowCount __newlinesInRow 0;
  length __word $ 10000;
```

Instances are counted within each variable.

```
  do __i = 1 to dim(cols);
    __ii = 0;
    __found = index(cols{__i},"&newline.");
    do while (__found > 0);
      __ii+1;
```

The substr() and index() functions are used to continually partition the string into the sections between each instance of the delimiter.

```
      if missing(__word)
        then __word = substr(cols{__i},__found + length("&newline.") );
      else __word = substr(__word,__found + length("&newline.") );
      __offset = length(__word);
      __found = index(__word,"&newline.");
      output;
    end;
```

The maximum number of instances across the variables is stored as the number of additional report rows needed for this observation.

```
    __newlinesInRow = max(__newlinesInRow,__ii);
  end;
```

Each maximum is then summed up to calculate the total number of extra rows needed. This value is then stored in a macro variable when the end of the input dataset is reached.

```
  __extraRowCount = sum(__extraRowCount,__newlinesInRow);

  if EOF then call symputx("extraRows",__extraRowCount);
run;
```

The number of observations, newlines and sum of the two are all stored as global macro variables to be used in further macros. These are separated because it takes less space in the final output to display a single observation split into two lines than two observations due to cell padding in proc report.

```
%global sasObs newlines totalRows;
%let sasObs = &obs;
%let newlines = &extraRows;
%let totalRows = %eval(&sasObs. + &newlines.);
```

MACRO 2: BUILD_REGION

This macro does most of the heavy lifting in creating the 2nd version of the safety summary output. It handles calculation of the height needed for each region and assigns the y coordinate based on the position and height of the previous region, as well as a specified row gutter distance.

There are quite a few parameters to build in flexibility for creating variations of this output:

```
%macro build_region(

isFirstRegion=N
```

The isFirstRegion parameter is a Y/N flag that tells the macro to set the Y coordinate to the &initialY value.

```
,initialY=0
```

initialY is just the starting Y coordinate position to use for the first region

```
,xPos=0
```

For this particular layout, the x coordinates are static so they are parameterised as a constant.

```
,reportData=
```

The data being used for the proc report of the row section is required as it is used by call_rep_rows to help calculated the required height.

```
,reportMacro=
```
For flexibility, the proc report code is contained in an external macro which can be passed to the build_region macro. This allows programmers to design their own proc report and simply use this macro for region generation.

```
,header=N
```
If the output contains a header, additional height is allocated to it.

```
,width=9
```
Width is a fixed constant for this layout

```
,headerHeight=0.082
,rowHeight=0.4
,newlineHeight=0.25
,rowGutter=0.04
,unit=in
```
The amount of space added for headers, rows, newlines and the row gutter are parameterised. This is to account for different font sizes.

```
,style=%str([backgroundcolor=CXD9D9D9])
```
Any style elements needed for the region can be specified with this parameter.

```
,incrementY=Y
```
This flag controls whether the Y coordinate for the next region is incremented based on the height of the current region. This is because the safety summary output has 2 regions per row section one for proc odstext and one for proc report. These pairs should occupy the same Y coordinate position so there must be an option to maintain the coordinate position.

```
);
```

Despite the large number of parameters, the logic is straightforward:

1. A call is made to count_rep_rows to determine the number of rows in the proc report of this row section.

```
%* Calculating height required for region;
%count_rep_rows(data=&reportData.);
```

2. Allow extra space for a proc report header if required.

```
  %if %upcase(&header.) = Y %then %let heightForHeader = &headerHeight.;
  %else %let heightForHeader = 0;
```

3. Multiply rows and newlines by their height parameters and add header height to calculate total height required.

```
%let requiredHeight = %sysevalf(&sasObs*&rowHeight + &newlines.*&newlineHeight. +
                                &heightForHeader.);
```

The height for vertical adjustment of the text displayed with proc ods text is stored in a global macro variable. In practice, I noticed that if there was exactly one row in the report then slightly less space was needed so I added a scale-down for this scenario.

```
  %if &totalRows. = 1 %then
    %let vJustHeight=%sysevalf(%sysfunc(compress(&requiredHeight.,&unit.))*0.9);
  %else %let vJustHeight=&requiredHeight.;
```

4. Calculate Y coordinate for region.

```
%*** Calculating Y coordinate for region ***;
```

The case for the first region is the easiest. It is set to the initialY parameter value.

```
* Use initial value for first region;
%if %upcase(&isFirstRegion.) = Y %then %do;
  %let currentY = &initialY.;
  %let previousY = 0;
%end;
```

If this is not the first region and an increment is requested then the previous Y value, row gutter and height of the previous output are added together to determine the current Y coordinate value.

```
%* Increment based on previous Y coordinate, output height and row gutter;
  %else %if %upcase(&incrementY.) = Y %then
    %let currentY = %sysevalf(&previousY. + &rowGutter. + &previousHeight.);
```

When no increment is requested, the previous Y value is used for the current region.

```
%* Use previous Y coordinate if no increment required;
%else %let currentY = &previousY;
```

5. Issue ODS REGION statement.

```
%* Building region;
ODS REGION width=&width.&unit. height=&requiredHeight&unit. x=&xPos.&unit.
           y=&currentY.&unit. style=&style.;
```

All the macro logic resolves to a single ODS REGION statement specifying all the details of the region.

6. Run proc report macro if required.

```
%* Executing proc report code if one was specified;
%if %length(&reportMacro.) %then %&reportMacro.;
```

As the proc report is abstracted out as a macro and issued as an input parameter, it can be easily called by the build_region macro.

7. Store height and Y coordinate values for use in calculation of the next region.

```
%if &countRepRowsRC < 0 %then %do;
  %let previousY = &currentY.;
  %let previousHeight = &requiredHeight.;
%end;
```

The return code for count_rep_rows is checked before storing this information in the global macro variables.

USING THE MACRO:

build_region creates regions one at a time. This meant that it was initially slightly cumbersome to use despite the relative simplicity of it. The combined output requires 12 calls to build_region in total. I decided to take advantage of the fact that the dataset and report macros were iteratively named, and created a wrapper macro to efficiently run build_region in a loop.

```
ODS LAYOUT ABSOLUTE;

do ii = 1 %to 6;
%* Selecting correct reporting macro and parameters based on section number;
```

< code omitted for brevity >

```
%build_region(isFirstRegion=&firstFlag.
              ,header=&headerFlag.
              ,reportData=&sumData..&currentPatient._sec&ii.
              ,reportMacro=ps_rep&macroNum.(data=&sumData..&currentPatient._sec&ii.)
              ,width=11.75
              ,unit=in
              ,xPos=1.04
              );
```

```
%build_region(isFirstRegion=N
              ,header=&headerFlag.
              ,reportData=&sumData..&currentPatient._sec&ii.
              ,width=1
              ,style=%str([backgroundcolor=CX830051])
              ,incrementY=N
              );
```

Each proc ods text section label is contained in a format. %sysfunc() is then used to apply the formatted value through use of the 2nd argument.

```
proc odstext;
  p "%sysfunc(left(&ii.),blockLabel)" / style=[color=CXFFFFFF background=CX830051
                                        vjust=m just=center cellheight=&vJustHeight.in];
run;

%end;

ODS LAYOUT END;
```

CONCLUSION

Both GRIDDED and ABSOLUTE layouts have their advantages. For combined outputs where regions are not required to be very close to each other, I would recommend the GRIDDED layout. Creating regions by simply assigning row and column position of a grid is far simpler than specifying exact coordinate position and dimensions of regions. There are features of the GRIDDED layout that I have not personally explored because the white space surrounding procedure outputs was detrimental for the specific output I was asked to create. Given more time, I would like to experiment with different outputs where this layout is more applicable. The fact that it supports the PowerPoint ODS destination (which ABSOLUTE does not) is an especially interesting feature of the GRIDDED layout which I hope to take advantage of at some point.

When more flexibility is required in the layout (for example close positioning or genuine overlapping such as graph inset which focusses on a section of interest), the ABSOLUTE layout is required. While this layout is generally more complicated to use than GRIDDED, this paper has shown how some of the difficulty and limitations of ABSOLUTE layouts can be mitigated through relatively simple programming techniques. If I were to continue to develop this output further, I would like to expand the scope of the build_region macro to allow dynamic calculation of the widths of outputs to further mimic the automation found the GRIDDED layout and continue to reduce the complexity of the ABSOLUTE layout from the macro user's perspective.

RECOMMENDED READING

SAS ODS LAYOUT tip sheet - http://support.sas.com/rnd/base/ods/Tipsheet_ods_layout.pdf

CONTACT INFORMATION

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

Michael Allie
AstraZeneca
Oncology Biometrics
Da Vinci Building
Melbourne Science Park
SG8 6HB
United Kingdom
Work Phone: +44 7785432591
Email: michael.allie1@astrazeneca.com