

Layout the Grid and you control the document: ODS meets OOP and you reap the benefits.

Daniel Ralyea, Rock Hill Schools; Karen Price, Rock Hill Schools

ABSTRACT

ODS is a journey not just a destination. You can exhibit a fine degree of control over your output once you understand the basic structures involved. Most of us are familiar with opening and closing an ODS destination (and it is good!) Exploring the construction of a destination allows a greater understanding of the power at your fingertips. ODS Layout provides a guiding structure for the information canvas. ODS Region provides control of a subset of the canvas and Object Oriented Programming allows cell by cell control of a custom built table. These tools, combined with the flexibility inherent in the output destination, allow a wide variety of production possibilities.

INTRODUCTION

Our ODS journey started with the realization that different destinations had inherently different properties and those properties could be exploited if we understood the control structures that distinguished the capabilities. For example the excel tagsets allow pagination by worksheet while ODS PDF defaults to a page per procedure, report or dataset. The manipulation of those options or tagsets describes the desired behavior to the output destination's rendering program rather than manipulating the information being submitted to the program. ODS layout and region control the structure of the output before it is sent to the output destination. Within reason, the output will look similar across different destinations. Extending that control to a more granular level, we can use Object Oriented Programming within the ODS to control cell by cell the position and format of the text in a table. The range of control allows SAS to produce documents with the fluidity of a website including graphics to the rigid structure of an invoice including a logo.

THE DESTINATION (INFORMATION) CANVAS

Every ODS destination has a distinct set of properties and constraints. Some of these are obvious, the size of the paper on which you are printing; some of these are not, xls tagset produces an xml document that excel imports and converts. ODS Layout has to function within the parameters of the destination. SAS uses the terms layout container and region container to describe the abstract structures being created. These concepts roughly parallel the layout region structures in Graph Template Language (GTL). Providing a quick hierarchy: 1) Destinations house the layout container, 2) Layout containers house region containers, and 3) Region containers house the results of procedures, reports and graphics.

The concept of housing the container becomes clearer as titles and footnotes are placed sequentially in the destination. The titles and footnotes are placed before the layout container is placed, bearing in mind that the layout container dictates the relationships between regions. Regions are then placed across the row, one column at a time by default (Row Major) this can be modified using the ORDER_TYPE=<ROW_MAJOR | COLUMN_MAJOR> option.

LAYOUT

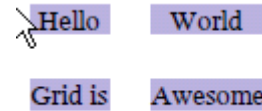
ODS Layout has two base variations, Absolute Layout and Gridded Layout. These variations offer distinct advantages depending on your goal. Absolute Layout places the programmer in "absolute" control of the positioning and size of each region, but "with great power comes great responsibility". The programmer must ensure that each defined dimension of the region will contain whatever output is placed in that region. By assuming that level of control you sacrifice SAS's capability to dynamically resize the container to accommodate the output. Gridded Layout sizes the height of the row to the height of the tallest region in the row and the width of the column to the width of the widest region in the column.

To illustrate these ideas, we have defined a style using proc template to set the background color to purple. This will cause the containers to be visible. The column_major option doesn't apply in a HTML destination, so open a pdf destination to make the printer styles available.

```
proc template;
  define style Styles.PBlock;
    parent =Styles.Printer;
    style LayoutRegion/
      background=cxbbb2e0;
  end;
```

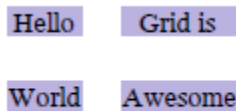
The following block of code will define a 2X2 grid with purple regions being filled row_major (the default setting)

```
ods pdf;
ods layout gridded columns=2 rows=2 order_type= row_major;
ods pdf style=Styles.pBlock;
ods region;
ods text="Hello";
ods region;
ods text="World";
ods region;
ods text="Grid is";
ods region;
ods text="Awesome";
ods layout end;
ods pdf close;
```



Error! Reference source not found..Error! Reference source not found. **Row Major default cell padding purple background**

Changing to column_major will produce the following.



Error! Reference source not found.. Error!
Reference source not found. **Column Major default cell padding purple background**

It's the ability to change layout region style settings. The row major setting allows each region to be controlled individually but changing the style after the first region is placed using column major resulted in no output being produced in ods pdf. Using different background colors for the region will help visualization of the concepts being illustrated. The following code will define styles to produce those background colors.

```
proc template;
define style Styles.PBlock; /*Purple*/
parent =Styles.Printer;
style LayoutRegion/
background=cxbbb2e0;
end;
define style Styles.RBlock; /*Red*/
parent =Styles.Printer;
style LayoutRegion/
background=CXFF0000;
end;
define style Styles.BBlock; /*Light Blue*/
parent =Styles.Printer;
style LayoutRegion/
background=CXADD8E6;
end;
define style Styles.GBlock; /*Green*/
parent =Styles.Printer;
style LayoutRegion/
background=CX00FF00;
end;
define style Styles.WBlock; /*White*/
parent =Styles.Printer;
style LayoutRegion/
background=CXFFFFFF;
end;
define style Styles.LBlock; /*Black*/
parent =Styles.Printer;
style LayoutRegion/
background=CX000000;
end;
run;
```

Gridded Layout offers other options to control the dimensions, row gutters, column gutters even the x and y starting position for the layout on the page. Those options are self-evident. The region statement offers more granular control of the output.

REGION

ODS Region defines the style, size and position of a more localized container. The output SAS has sent to ODS is placed within regions. The region command interacts within the layout statement and can advance the position within the layout though the regions are processed in the order in which they are encountered inside the layout statement. Once the opportunity to write within a particular region has advanced beyond that position in the layout that region cannot be changed. The region will display the active style with no SAS output.

```
ods pdf;
ods layout gridded columns=2 rows=3 order_type= row_major;
ods pdf style=Styles.BBlock;
ods region row=1 column=2; /* Filled 1st */
ods text="Hello";
ods region row=2 column=1; /* Filled 2nd */
ods text="World" ;
ods region row=3 column=1; /* Filled 3rd */
ods text="Hello";
ods region ; /* Filled 4th */
ods text="World" ;
ods layout end;
ods pdf close;
```



Figure 3. Using row and column placement

Further interaction between the region statement and the layout statement is available with the `column_span` and `row_span` options. Recall that by default gridded layout automatically resizes to accommodate the largest region within a row or column. A large region can cause the balance of a row or column to shift which can result in an excess of whitespace. The `column_span` and `row_span` options can allow a multiple regions and the gutters between them to merge and form a single region without impacting the balance of the grid. This results in a very flexible design opportunity.

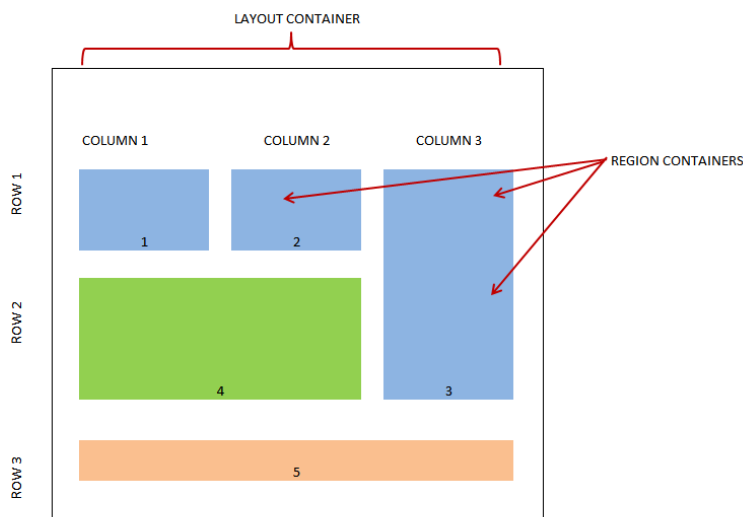


Figure 4. Layout design for `column_span` and `row_span`

The layout starts with a 3X3 grid indicating nine regions. The design plan encompasses 5 regions with distinct size requirements. Working row_major the 1st region occupies row 1 column 1, the 2nd region occupies row 1 column 2, the 3rd region occupies row 1 column 3 and row 2 column 3. The `row_span` option on the 3rd region will allow a merge of row 1 column 3 with row 2 column 3. The `row_span` is the count of the rows being merged within a column so the `row_span`=2. The 4th region occupies row 2 column 1 and row 2 column 2. The `column_span` option will allow a merge of row 2 column 1 and row 2 column 2 forming region 4. The `column_span` is a count of the columns being merged so the `column_span`=2. The 5th region occupies columns 1-3 in row 3 so the `column_span` is 3.

```
ods pdf;
ods layout gridded columns=3 rows=3 order_type= row_major;
ods pdf style=Styles.BBlock;
ods region row=1 column=2; /* Filled 1st */
ods text="Hello";
ods region row=1 column=1; /* Filled 2nd */
ods text="World" ;
ods region row_span=2; /* Filled 3rd */
ods text="Grid is";
ods pdf style=Styles.GBlock;
ods region column_span=2; /* Filled 4th */
ods text="Awesome";
ods pdf style=Styles.PBlock;
ods region column_span=3 ; /* Filled 5th */
ods text="and Flexible";
ods layout end;
ods pdf close;
```

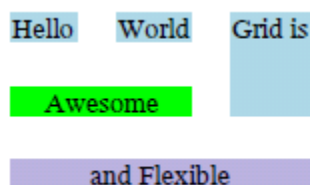


Figure 5. Actualized design plan using column_span and row_span

TRANSITIONING TO REPORT WRITING INTERFACE (RWI)

ODS is referred to in terms of a destination. Send the report to ODS, select an ODS destination when in fact a virtual object/file is being created. Most of the time that object only exists for the duration of the SAS instance in which the object was created. ODS handles this so smoothly it rarely requires any additional consideration. The examples above illustrate a more granular level of control than usually necessary. The Report Writing Interface (RWI) allows the ODS object to be crafted inside of a data step. This allows the granularity of control to be combined with the power of the data step, the ability to apply conditional logic to the crafting of the ODS object. The RWI is structured around Object Oriented Programming. The goal of the process is to create a report, an object, that object is going to have a set of characteristics. A Class provides a template, a predefined set of processes (methods) and characteristics (attributes) that will be used to define the object. The characteristics being defined in the object will parallel the options used in the ODS output created previously.

```
ods pdf;
data _null_;
declare odsout doc();
doc.layout_gridded(columns=3, rows=3, order_type="row_major");
doc.region(style_attr:'backgroundcolor=CXADD8E6');
doc.format_text(data:"Hello");
doc.region(style_attr:'backgroundcolor=CXADD8E6');
doc.format_text(data:"World");
doc.region(row_span=2, style_attr:'backgroundcolor=CXADD8E6');
doc.format_text(data:"gridded layout");
doc.region(column_span=2, style_attr:'backgroundcolor=CX00FF00');
doc.format_text(data:"is awesome");
doc.region(column_span=3, style_attr:'backgroundcolor=cxbbb2e0'); /*ERROR
reported to SAS*/
doc.format_text(data:"and flexible", style_attr:'backgroundcolor=cxbbb2e0');
doc.layout_end();
run;
ods pdf close;
```

The declare statement does two things in one step. It assigns a new variable (doc) based on the class odsout and initializes the variable. The object oriented notation now has a context. The doc.layout_gridded method is now applied creating three column markers and three row within the doc object. The doc.region statement identifies the characteristics of the region that has been defined. In this case, the starting position is next cell available, the size is dynamic and the background color is being set to Light Blue. Once the region is defined the output can be placed in

it. The `doc.format_text` method creates the data and the characteristics of the output to be placed in the region container within the layout grid. A convenient way to conceptualize the object is as a document with a series of generated tags.

DOCUMENT OBJECT

The class instantiation of the ods object lays out a series of sections on a page. It defines a header section which contains the intended destination and other information and a body section that defines the structure of the output. The `layout_gridded` method creates a set of tags which define the number of columns. The regions define a section break and then another document object is inserted within that section. The document object could be a table or procedure output or a graphic object. The `format_text` method creates a table with a single cell which is then placed within in the section.

CREATING A ODS TABLE

The process of creating an ODS table using the RWI is a logical extension of the process for generating the document. A variable must be declared and instantiated. Methods and attributes further define the object. A table is still a member of the odsout class. This can be done as an extension of the current project or as a new object being placed in the ODS layout framework. Designing the table is dependent on your purpose. When planning on a set statement then establishing the table start and table end statements in terms of the data set is important, if the report is written in a single pass then the table start and table end statement can just follow the program flow. The basic structure is to declare the ODS variable using the odsout class. The `table_start` method is followed by a `row_start`, within the row the `format_cell` method places the text and formats the cells which establishes the table header and sets the row widths for the following rows. Subsequent rows contain the body data.

```
data _null_;
declare odsout dtable();
/* start the table */
dtable.table_start();
/* start the row */
dtable.row_start();
/* these cells establish the columns for the table */
dtable.format_cell(text: 'Name', overrides: 'background=green fontweight=bold
cellwidth=2.5in');
dtable.format_cell(text: 'Address', overrides: 'background=green fontweight=bold
cellwidth=2.5in');
dtable.format_cell(text: 'Phone', overrides: 'background=green fontweight=bold
cellwidth=3.5in');
/* end the row */
dtable.row_end();
/* these rows form the body of the table */
dtable.row_start();
dtable.format_cell(text: 'Bozo', overrides: 'cellwidth=2.5in');
dtable.format_cell(text: '15 Mocking Bird Rd.', overrides: 'cellwidth=2.5in');
dtable.format_cell(text: '555-0987', overrides: 'cellwidth=3.5in');
dtable.row_end();
dtable.row_start();
dtable.format_cell(text: 'Ronnie Regan', overrides: 'cellwidth=2.5in');
dtable.format_cell(text: '1600 Pennsylvania Ave.', overrides: 'cellwidth=2.5in');
dtable.format_cell(text: '555-1234', overrides: 'cellwidth=3.5in');
dtable.row_end();
dtable.row_start();
dtable.format_cell(text: 'Mr. Smith', overrides: 'cellwidth=2.5in');
dtable.format_cell(text: '51 Main St.', overrides: 'cellwidth=2.5in');
dtable.format_cell(text: '555-5643', overrides: 'cellwidth=3.5in');
dtable.row_end();
/*close the table*/
dtable.table_end();
run;
```

Name	Address	Phone
Bonzo	15 Mocking Bird Rd.	555-0987
Ronnie Regan	1600 Pennsylvania Ave.	555-1234
Mr. Smith	51 Main St.	555-5643

Figure 6. Static table produces with ODS

This approach will work to produce a static object but doesn't begin to harness the power of the data step. The following example loads the data into a table and uses conditional logic to initialize and close the output table (O'Connor, 2009).

```
data names;
infile datalines delimiter='$';
format name $15. address $25. phone $11.;
input name $ address $ phone $;
datalines;
Bonzo$15 Mocking Bird Rd.$555-0987
Ronnie Regan$1600 Pennsylvania Ave.$555-1234
Mr. Smith$51 Main St.$555-5643
;

data _null_;
set names end=fend;
if _n_ eq 1 then do;
  declare odsout dtable();
  /* start the table */
  dtable.table_start();
  /* start the row */
  dtable.row_start();
  /* these cells establish the columns for the table */
  dtable.format_cell(text: 'Name', overrides: 'background=green fontweight=bold
cellwidth=2.5in');
  dtable.format_cell(text: 'Address', overrides: 'background=green fontweight=bold
cellwidth=2.5in');
  dtable.format_cell(text: 'Phone', overrides: 'background=green fontweight=bold
cellwidth=3.5in');
  /* end the row */
  dtable.row_end();
end;
dtable.row_start();
dtable.format_cell(text: name);
dtable.format_cell(text: address);
dtable.format_cell(text: phone);
dtable.row_end();
if fend eq 1 then dtable.table_end();
run;
```

CONCLUSION

ODS Layout and ODS Region statements provide a structure for the information canvas that will adjust dynamically addressing the need to produce multiple outputs on a single page. The ODS Report Writing Interface extends SAS report writing capabilities to address custom report writing needs. It is fully integrated with ODS features such as changing fonts, using background colors, incorporating images, etc. and provides the user with the ability to programmatically apply this formatting for individual cells. The interface allows the user to combine the power and flexibility of the DATA step with the output capabilities of ODS, making even the most complex reporting requirements possible.

REFERENCES

- Kummer, Daniel. (2014). Toe to Toe: Comparing ODS LAYOUT and the ODS Report Writing Interface. *SAS Global Form*. Washington, DC. Available at: <http://support.sas.com/resources/papers/proceedings14/SAS330-2014.pdf>
- O'Connor, D. (2009). The Power to Show: Ad Hoc Reporting, Custom Invoices, and Form Letters. *Western Users of SAS Software*. San Jose.
- O'Connor, D., & Huntley, S. (2009). Breaking New Ground with SAS 9.2 ODS Layout Enhancements. *SAS Global Forum*. Washington, DC.

CONTACT INFORMATION

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

Name: Daniel Ralyea
Enterprise: Rock Hill Schools
Address: 660 North Anderson Rd.
City, State ZIP: Rock Hill, SC 29730
Work Phone: 803-981-1050
E-mail: dralyea@rhmail.org

Name: Karen Price
Enterprise: Rock Hill Schools
Address: 660 North Anderson Rd.
City, State ZIP: Rock Hill, SC 29730
Work Phone: 803-981-1082
E-mail: kprice@rhmail.org

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

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