Paper 22-28

Next Generation Data _NULL_ Report Writing Using ODS OO Features

Daniel O'Connor, SAS Institute Inc., Cary NC

ABSTRACT

This paper is targeted for advanced data *null* report-writing programmers that are eager to take advantage of the new ODS Object Oriented (OO) features built directly into the data step environment, which will be experimentally available in Version 9.1. Features will allow you to build tables cell by cell with all of the ODS features that you have become accustom to such as proportional fonts, traffic lighting in cells, embedded images, but will also allow you to create non-tabular output, insert sub-totals, and manage your layout.

INTRODUCTION

The Output Delivery System (ODS) was introduced in Version 7 as a means for SAS programmers to quickly and easily produces output in a variety of industry standard forms. Until now the primary use of the ODS System has been to format procedure output in a tabular fashion while taking advantage of all the presentation quality characteristics that are expected in any reporting technology. SAS provides a variety of report writing procedures each designed with a particular technique of report writing in mind. However, not all requirements fit so neatly into one of these prepackaged techniques. For over 20 years now data *null* report writing has been an intricate part of the SAS report writing solution. It provides the greatest flexibility and control of all the SAS report writing techniques, so that even the most rigid reporting requirements can be met with ease. This paper will briefly review the existing ODS data step integration, and then will discuss the new OO features in great detail.

OUTPUT DELIVERY SYSTEM

The Output Delivery System is an objected oriented technology specifically designed to be able to take data and an abstract description of a report, and render that report in a variety of forms simultaneously. An Output Object is the fundamental component of ODS, and its sole responsibility is to package the data, and the description of how that data should be presented. The data portion of the output object is commonly referred to as the Data Object, and can be thought of as a tabular collection of data (similar to a data set). While the abstract report description is referred to as the Template Object (or just the Template). Templates contain tabular formatting information such as the number of columns, the order in which they are presented, formatting characteristics (Datetime16.), alignment, column headings, and may contain stylistic features such as colors, and fonts. A Template is a pre-packaged description that will only contain enough information necessary to produce 1 table. So, when you look at SAS output and see multiple tables being produced from a procedure each table may in fact have a unique template definition. There are two important pieces of information to keep in mind when thinking of a template Object. First, it is always tabular in nature, and secondly it is pre-packaged. Neither of these characteristics is flexible, nor allows the level of control often necessary. Which leads us to the purpose of this ODS OO data step interface that we will explore in this paper.

DATA STEP ODS INTERFACE

As part of the initial implementation of ODS in Version 7, a data step interface was also supported in a limited fashion. Consider this, If a data set is similar to a Data Object (tabular data) then in theory it should be relatively straightforward to package the data step with a pre-defined Template Object (abstract report definition) to create an Output Object (ODS Table) in any ODS output destination (HTML, RTF, PDF, etc.). The data step FILE statement supports an ODS option to enable the creation of an Output Object, and the data step PUT statement supports an `ODS` option to allow the data step to communicate with the Data Object component of the Output Object. Note that the ODS option may only be used on the FILE statement when the PRINT fileref (ie. FILE PRINT ODS = …) is being used.

EXAMPLE 1:

```
proc template;
define table Sugi28.ex1;
column name ssn bday sex;
  define name;
     width=14;
     header="Customer";
     just=R;
     style={font_weight=Bold};
  end;
  define bday;
     width=14;
     format=MMDDYY8.;
     header="Birth Day";
     cellstyle _val_< "01JAN67"D as
                        {foreground=red},
                _val_ < "01JAN70"D as
                        {foreground=blue},
                _val_ < "01JAN73"D as
                        {foreground=green},
                1 as    {foreground=black};
   end;
  define ssn;
     width=14;
     format=ssn.;
     header="SSN";
     just=C;
     justify=on;
  end;
  define sex;
     width=14;
      just = C;
     header="Sex";
     translate _val_ = 0 into "Male",
                _val_ = 1 into "Female";
                justify=on;
  end;
end;

ods rtf file="Sugi_ex1.rtf";

data test;
input name $6. bday mmddyy9. @18 ssn sex;
file print ods =(template="Sugi28.ex1");
put _ods_;
cards;
Dan    11/25/66  340987544 0
Chris  07/28/69  340980983 0
Matt   04/05/72  090985049 0
Patty  11/20/67  093488347 1
;
run;

ods rtf close;
```

OUTPUT 1:

Customer	SSN	Birth Day	Sex
Dan	340-98-7544	11/25/66	Male
Chris	340-98-0983	07/28/69	Male
Matt	090-98-5049	04/05/72	Male
Patty	093-48-8347	11/20/67	Female

This example accomplished traffic highlighting, the use of customized stylistic attributes, a numeric translation, justification of data in a column, and uses proportional fonts all though an existing data null report writing techniques. It also conveys a valuable point that an enormous amount of customization can be accomplished without having the use the ODS OO features that we are about to introduce. So, when developing a report you must evaluate which report writing technology is best suited for the problem at hand. If the desired output is tabular in nature, the existing data NULL or any of the report writing procs is almost always the simplest approach. Traditionally data null has been the only technique that allows multiple input data sources, the use of multiple observations on a given line, multiple reports to be generated simultaneously, summarization (subtotal) records need to be inserted, or when data dependant headers are required. But, if your goal is to produce a tabular report the existing interface will continue to be your best choice. Data null report writing is also used to solve fare more complex reporting problems where the level of control that is provided by existing ODS interfaces proves to be inadequate. It is often used when output simply does not conform to a tabular format, and this will become your primary justification for using the ODS OO interface.

ODS INTEGRATION

The ODS Output Object experimental interface is integrated into all output destinations except for the Listing, Output, and Document destinations. The Output and Document destinations are more data oriented than report oriented and therefore do not conform directly, while the Listing destination already has a historical data *null* interface to address the needs of character cell based reporting. Which realistically leave the HTML, RTF, and Printer (PDF, PS, PCL) destinations.

OBJECT ORIENTED TECHNIQUE

Object oriented programming revolutionized the software development industry, and is a sophisticated technique that allows the developer to perform a specific task with less programming than a traditional Application Programming Interface (API) approach. As the name indicated the fundamental component of an object oriented is the “Object” itself. The object contains data as well as actions that can commonly be performed to accomplish a specific task. Our task today is to create a report one item at a time.

ODS OUTPUT OBJECT CREATION

The DECLARE (Dcl) and *NEW* statements allow the declaration and instantiation of ODS Output Object, and the data steps object dot syntax allow access to the resulting object’s attributes and methods.

Example 2.1:

```
declare odsout obj;
obj = _new_ obj();

or

dcl odsout obj();
```

This tells the data step compiler that a variable ***obj*** is of class type ***odsout***, and all necessary ODS initialization is performed. When using this technique for data *null* report writing we are essentially dynamically binding a data value with specific formatting information and therefore our previous concepts concerning Data Object, and pre-define Template Objects have no bearing on the output that will be generated, and therefore our use of the following will no longer be necessary.

```
file print ods =(template="Sugi28.ex1");
put _ods_;
```

This ***obj*** variable will become the means to communicate with ODS System for your reporting needs, and will remain available until the completion of the data step or until the explicitly deleted.

PERFORMING ACTIONS (METHOD CALLS)

Methods calls are nothing more than a functional API that is specific to the object class that has been declared. The general syntax of a method call is

```
obj.method( < method arguments > );
```

Where “*method*” is the API (action) to be performed, and “*method arguments* “ are additional information that the method understands how to interpret. It is also possible to call the method in a function-like manner to obtain a numeric return code, which would indicate the successfulness of the requested action.

```
Rc=Obj.method( < method arguments > );
```

Example 2.2:

```
obj.delete();
```

The *delete* method removes memory allocations, and performs any necessary termination processing required with the declared object. The ***obj*** variable will no longer be available in the context data step following this method call.

DATA DEPENDANT HEADINGS

Data dependant Title/Footer information has historically been a difficult for many of the report writing procedures.

Example 2.3:

```
obj.title(text: "SUGI Power & Light Account
Statement for " ||  name);
```

Notice that in addition to a simple piece of text, operators as well traditional data step formatting characteristics are supported (i.e. Put(date(), worddate18.)). The title will be produced using the characteristics defined by the ODS Style that is in effect for the active output destinations. This seamless integrations of the data step and ODS will ensure that all of your titles have the same style characteristics (fonts, colors) throughout any given report.

TEXT & ODS STYLE CHARACTERISTICS

Data *null* report writing has often been touted as the solution for reporting when complete control of the exact placement of information is required. This could easily be attained by the use of a monospace font, and fixed character cell based addressing. With the introduction of proportional fonts, and style attributes in Version 7 this solution was no longer a viable alternative.

EXAMPLE 2.4:

```
obj.format_text(text: "Account Number: " ||
    account,
 overrides: "just=Left
             background=colors('headerbg')
             foreground=black
             font_weight=bold" );
```

Notice we are overriding the default style characteristics defined in the ODS Style of the active output destinations. The background=colors('headerbg") override looks distinctively different than the rest of the overrides. This is a ODS Style element reference which is selecting the 'headerbg' item from the color palette style element called 'Colors'. This may seem overly complex in comparison to foreground=black, but is a very powerful feature. A wide variety of styles can be applied to each individual output destination, and each has its own set of presentation quality characteristics. By referencing an element within the active ODS style your data *null* report will dynamically inherit those output characteristics. The expectation of this example is to make this text appear similar to a SAS title while being left justified, and always using black foreground color. One caveat must always be considered when assigning fixed style characteristics like foreground=black, if the ODS Style itself uses a very dark background color your black text may be very difficult to read. Consult ODS documentation for a list of all Style Attributes & Values.

NON-TABULAR OUTPUT

For tabular output each row of output traditionally corresponds directly to one row of data. Often one observation is expected to produce multiple lines of output, or even an entire page of output. SAS Procedures are simply not architected to allow such flexibility.

EXAMPLE 2.5:

```
obj.table_start(name: State,
 Label: "Customer Report for " || State,
 overrides: "just=L
             frame=void
             background=_undef_
             cellspacing=0
             cellpadding=0
             rules=none" );
obj.row_start();
obj.format_cell(text: name,
 overrides: "just=Left
             background=_undef_ " );
obj.row_end();
obj.row_start();
obj.format_cell(text: street,
 overrides: "just=Left
             background=_undef_ "  );
obj.row_end();
obj.row_start();
obj.format_cell(text: trim(City) || ", " ||
    State || " " || zip,
    overrides: "just=Left
                background=_undef_ "  );
obj.row_end();
obj.table_end();
```

OUTPUT 2.5:

SUGI Power & Light Account Statement for Jay

Account Number: 4983009
Jay
95 Wild Ranch Road
Taos, NM 89875

The ***table_start*** method marks the beginning of the table creation, and must be terminated with a corresponding ***table_end***. A table is very simply a collection of rows, which are comprised of a collection of cells. So it will not surprise you to see that we have modeled the interface to reflect those simple primitives. Notice that the obj.table_start() method has "name" and "Label" optional attributes. These are simply the Name/Label pair that is shown in the results window. A row is only valid as part of a table, and will be flagged as a run-time error if used outside that context. In the above example I chose not to provide any specific style attribute overrides on the rows, but it would be perfectly acceptable to do so if desired. In addition to style attributes rows do have some unique characteristics themselves.

EXAMPLE 2.6:

```
obj.table_start(name: put(account, 8.),
        overrides: "frame=box
                    background=_undef_
                    cellpadding=2
                    rules=none" );

obj.row_start(type: "Header");
obj.format_cell(text: "Account Status",
             column_span: 2 );
obj.row_end();
```

```
obj.row_start(type: "Header");
obj.format_cell(text: "Customer" );
obj.format_cell(text: "Balance Due" );
obj.row_end();

obj.row_start();
obj.format_cell(text: name,
             overrides: "just=Left
             background=_undef_");
curamt  = 0.088 * kwhusage;
obj.format_cell(text:
                trim(put(curamt, dollar10.2)),
             row_span: 3,
             override: "vjust=Center");
obj.row_end();

obj.row_start();
obj.format_cell(text: street,
             overrides: "just=Left
             background=_undef_ ");
obj.row_end();

obj.row_start();
obj.format_cell(text: trim(City) || ", " ||
                State || " " || Zip,
             overrides: "just=Left
             background=_undef_ ");
obj.row_end();
obj.table_end();
```

Output 2.6:

Account Status	
Customer	**Balance Due**
Jay **95 Wild Ranch Road** **Taos, NM 89875**	**$104.60**

Heading rows are often used to provide descriptive information about the data that is being presented, and often are presented using a different set of visual characteristics. In the event that a table encounters a page boundary heading rows also are repeated at the top of every page before continuing to present the rest of the data. This example also takes account some cell specific features. Notice that the "Account Status" cell actually spans the entire table, and the "$104.60" spans 3 rows.

Controlling pagination is yet another feature commonly addressed with data *null* report writing. A page request is interpreted in accordance with the output destinations expectations. For example, HTML would produce a horizontal reference line while and output destination such as PDF would physically begin a new page.

EXAMPLE 2.7:

```
obj.page();
```

Reporting needs today encompass both electronic and printed documents, and each some very unique requirements such as hyper linking, static images (logo's), reference lines. Here are a few additional methods that will allow you to provide even more customization to your reporting needs.

```
Obj.line(size: "6in");

Obj.href(text: "http://www.sas.com/");

Obj.image(text: "SugiLogo.gif");
```

COMPLETE PAGE CONTROL

Having complete control of the content of a page is probably the most common argument for using data *null* reporting. Thus far we have seen examples that empower the report writer to create some very dynamic reports that to not conform to traditional proc output, but have yet to introduce the ability to control the exact placement of the content within the context of a printed page. ODS Layout is an experimental feature that has been developed in Version 9.1 to accommodate this type of reporting. Whether it is combining graphical output and existing procedural output, or taking advantage of the ODS Layout capabilities built into the Data *NULL* you will see that these features are one of the primary development enhancements for the ODS System. There are two different layout techniques introduced in version 9.1, Absolute Layout, and Gridded Layout. Absolute Layout can be used for reporting needs where pre-printed forms are involved, or precise placement is a requirement. However, absolute layout is restricted to a single page of output. Gridded Layout on the other hand provides a little less control, but is much more accommodating when you have several items that stretch multiple columns, rows, or even pages.

ABSOLUTE LAYOUT

Due to the fact that the ODS printer (PS, PCL, PDF) output destinations are the only destinations that currently have the concept of a physical page absolute layout is only supported by these output destinations.

Lets assume that we have a pre-printed form letter that needs to be filled in with basic customer account information. Historical data null reporting has been able to accomplish this when using dot matrix printers (remember those things) and monospace output, but technology has simply antiquated those techniques. The defacto standard today is presentation quality reporting, and the historical data null reporting falls short of adequately addressing those expectations. The absolute layout area assumes the height and width of the printable page. Additional options that affect the printable area of a page are topmargin, bottommargin, leftmargin, rightmargin, nodate, and nonumber.

```
obj.layout_absolute();
…
obj.layout_end();
```

All absolute reporting directives must be encapsulated by the layout methods. The fundamental component of any layout technique is defining a region in which some data should be placed.

```
obj.region();
```

REGION OPTIONS:

X

X location relative to the upper-left hand corner of the Layout area.

Y

Y location relative to the upper-left hand corner of the Layout area.

Width

The width of the region.

Height

The height of the region.

All region parameters use the style attribute dimension units of measure that are used by ODS template language (See ODS documentation for additional details).

cm	centimeters
mm	millimeters
in	inches
pt	printers pt
px	pixels(target device pixel units)

EXAMPLE 3.1:

```
obj.layout_absolute();
obj.region(y: "1in",
           x: "2in",
           width: "1in");
obj.format_text(text: put(account,8.),
    overrides: "foreground=blue
                font_weight=bold");

obj.region(y: "1in",
           x: "5in",
           width: "2in");
obj.format_text(text:
        trim(put(date(), monname.)) ||
        " 15, 2002",
        overrides: "font_weight=bold");

obj.region(x:"1.25in",
           y: "6in",
           width: "4in",
           height: "2in");
obj.table_start(name: name,
          overrides: "frame=void
                      cellspacing=0
                      cellpadding=0
                      rules=none" );
obj.row_start();
obj.format_cell(text: name,
             overrides: "just=Left" );
obj.row_end();
obj.row_start();
obj.format_cell(text: street,
              overrides: "just=Left");
obj.row_end();
obj.row_start();
obj.format_cell(text: trim(City) ||
        ", " || State || " " || zip,
        overrides: "just=Left"  );
obj.row_end();
obj.table_end();
```

```
obj.region(x: "5in",
           y: "6in",
           width: "1.5in");
obj.format_text(text:
   put(totalamtdue, dollar10.2),
   overrides: "font_weight=bold" );

obj.layout_end();
```

Absolute layout provides report writers the ultimate control of the page in a simple yet powerful manner.

GRIDDED LAYOUT

Gridded layout allows you to arrange regions in a series of rows and columns with different widths and heights. This approach is similar to the table that was produced earlier that used the column_span, and row_span attributes. Gridded Layout can be much more complex than absolute layout and it is strongly recommended that you layout the entire report on a piece of paper prior to beginning development. Defining a Gridded Layout can be accomplished by using the following methods.

```
Obj.layout_gridded(<gridded layout options > );

…
obj. layout_end();
```

GRIDDED LAYOUT OPTIONS:

Width

The width of the overall layout (dimension units).

Height

The height of the over all layout (dimension units).

Columns

Specifies the number of columns to use in the gridded layout(Default 1 column).

Rows

Specifies the number of rows in the gridded layout. The option should seldom be used. The number of rows is automatically calculated for you based on the number of regions that are created prior to the layout_end() method.

Column_widths

Specify the width of each column (dimension units). The default is to simply divide the horizontal space evenly across all of the number of columns. You must specify the columns= option, and specify this option once for each column defined.

Row_hieghts

Specify the width of each row (dimension units). You must specify the rows= option, and specify this option once for each rows defined.

Column_gutter

Specify the width of the space between columns.

Row_gutter

Specify the space between rows.

Similar to Absolute layout, the fundamental component the Gridded layout is the region in which an item should be placed.

GRIDDED REGION OPTIONS:

X, Y
Invalid for Gridded Layout.

Width
Normally used for absolute layout.

Height
Normally used for absolute layout.

Column
The Gridded column. Only required when skipping columns.

Row
The Gridded row. Only required when skipping rows.

Column_span
The number of gridded layout columns that the region should occupy. The default is 1 column.

Row_span
The number of gridded layout rows the region should occupy. The default is 1 row.

Column_gutter
Overrides the column_gutter.

Row_gutter
Overrides the row_gutter.

Lets take a close look at a simple Gridded Layout. We are going to reproduce something similar to Example 2.7.

EXAMPLE 3.2:

```
obj.layout_gridded(columns: 2);

/* Row 1 */
obj.region(column_span: 2);
obj.format_text(text: "Account Status",
        overrides: "just=Center
        foreground =colors('headerfg')
        background =colors('headerbf')
        font_weight=bold" );

/* Row 2 */
obj.region();
obj.format_text(text: "Customer",
        overrides: "just=Left");
obj.region();
obj.format_text(text: "Balance Due",
        overrides: "just=Left");

curamt  = 0.088 * kwhusage;

/* Row e */
obj.region();
obj.layout_gridded(columns: 1,
        column_gutter: "0.1in",
        row_gutter: "0.1in");
obj.region();
obj.format_text(text: name,
        overrides: "just=Left
        foreground=red");
obj.region();
obj.format_text(text: street,
        overrides: "just=Left");
obj.region();
obj.format_text(text: trim(City) ||
        ", " || State || " " || Zip,
        overrides: "just=Left");
obj.layout_end();

obj.region();
obj.format_text(text:
         trim(put(curamt, dollar10.2)),
         overrides: "just=Left");

obj.layout_end();
```

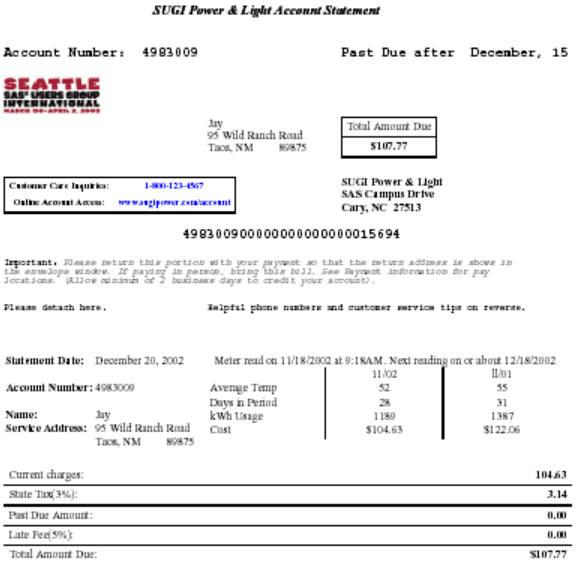
OUTPUT 3.2:

This example establishes a gridded layout that contains 2 gridded columns that we will place regions into. Notice that the first region (“Account Status”) spans the entire gridded layout (row 1). The next two separate regions (“Customer”, “Balance Due”) comprise the information that makes up row 2. Row 3 is very unique in the sense that it contains a nested gridded layout that controls the presentation of the customer name and address within the context of the first column. When you take a look at the output that was produce your first expectation would be that the layout would probably contain 5 rows when in fact it only contains 3 rows due to the nesting of the gridded layout. You can begin to see why gridded layout can be far more complex and powerful than absolute layout.

MORE ELABORATE GRIDDED SCENARIO

The final example that I will leave you with uses a variety of topics covered in this paper, and is intended to inspire you to take advantage of this new technology to solve all of your elaborate reporting needs. The source code will be available at http://www.sas.com/rnd/base/index-early-access.html
The goal of this example is to produce an invoicing report for each customer of the “SUGI Power & Light” company. By utilizing the data step by statement I can produce one report per customer using a complex gridded layout technique.

Complex Gridded output:



SUGI Power & Light Account Statement

Account Number: 4983009 Past Due after December, 15

SEATTLE SAS® USERS GROUP INTERNATIONAL MARCH 30-APRIL 2, 2003

Jay
95 Wild Ranch Road
Taos, NM 89875

Total Amount Due
$107.77

Customer Care Inquiries: 1-800-123-4567
Online Account Access: www.sugipower.com/account

SUGI Power & Light
SAS Campus Drive
Cary, NC 27513

49830090000000000000015694

Important. Please return this portion with your payment so that the return address is shown in the envelope window. If paying in person, bring this bill. See Payment information for pay locations. (Allow minimum of 2 business days to credit your account).

Please detach here. Helpful phone numbers and customer service tips on reverse.

Statement Date: December 20, 2002
Meter read on 11/18/2002 at 9:18AM. Next reading on or about 12/18/2002

Account Number: 4983009
Name: Jay
Service Address: 95 Wild Ranch Road
Taos, NM 89875

	11/02	11/01
Average Temp	52	55
Days in Period	28	31
kWh Usage	1180	1387
Cost	$104.63	$122.06

Current charges:	104.63
State Tax(3%):	3.14
Past Due Amount:	0.00
Late Fee(5%):	0.00
Total Amount Due:	$107.77

CONCLUSION

SAS provides a variety of report writing solutions that is intended to address everything from the most basic reporting needs to the most esoteric requests. The ODS Output Object and Layout features that's being integrated into the data step is intended to round out our SAS/Base report writing technology product offering. This technology will empower the SAS programmer with the tools to conquer any request with the greatest of ease.

REFERENCES

Reporting from the Field: SAS Software Experts Present Real-World Report-Writing Applications, Cary, NC, USA. SAS Institute Inc.

SAS Guide to report Writing Examples: Examples, Version 6, First Edition, Cary, NC, USA. SAS Institute Inc.

ODS: The Data Step Knows, by William F. Heffner, SAS Institute Inc.

ACKNOWLEDGMENTS

The author greatly appreciates input on content from Janice Bloom, Chevell Parker, as well as the rest of the Technical Support staff that made contributions. An immense amount of gratitude also goes to Brian Schellenberger for much of the development associated with the layout technology, and as well as the rest of my colleagues on the ODS Development Team.

Contact Information

Your comments and questions are valued and encouraged.

Contact the author at:

Daniel O'Connor
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8000
Fax: (919) 677-4444
Email: Dan.Oconnor@sas.com