Paper 229-29

Using New Features in ODS to Create Master/Detail Reports

Jack Hamilton, First Health, West Sacramento, California

ABSTRACT

This paper discusses the concept of master/detail reports and discusses several methods of creating them in SAS, with an emphasis on using the Output Delivery System.

INTRODUCTION

Master/detail reports are reports are generated from two or more data sets containing related information. One of the data sets, the master, contains information pertinent to all of the related records. The other data sets contain information about specific items related to the master record. Master/detail relationships are widely used in business. Here is an example of master/detail records:

Invoices

Invoice_Num	Customer	Invoice_Date
101	Hugo Furst	01Jan2004
102	Freida Peeples	15Jan2004
103	Al E. Loohah	30Jan2004

LineItems

Invoice_Num	Line_Num	Item	Cost
101	1	Widget	12.00
101	2	Gadget	10.00
101	3	Frammet	36.00

Invoices is the master table. There is one record for each invoice, containing information which applies to all items in the invoice. This is a greatly simplified record; in real life, there would probably be address information, and potentially a host of other items

LineItems is the detail table. There are 0 or more records for each invoice, containing information about the items ordered on the invoice. Again, in real life there would probably be additional columns such as quantity and stock number.

A version of these data sets with more rows will be used for the remainder of the paper; the code need to create them is in Appendix I.

It's quite common to want to take two data sets like these and produce a single report. Invoices and bills are an example; class rosters are another. Many types of hierarchical and relational data can be thought of in terms of master/detail reports. Unfortunately, there's no reporting procedure in base SAS designed to produce such reports.

WHAT WE DID IN VERSION 5 (AND BEFORE)

In version 5 (and previous versions), so called "data _null_" reports[1] were typically used to create master detail reports.

DATA _NULL_ REPORTS

A data _null_ report uses data step code to produce a print file (listing output). A simple example is shown on the next page, with the code in the left column and the list output in the right column. Page breaks are indicated with the text "<page>".

Although this example is quite simple, data _null_ reports can become quite complex, producing multiple output files, page and report totals, and tables of contents. For some examples, see the SAS Institute publication *SAS Guide to Report Writing: Examples* (Pubcode 55045) . Sample code which you can download and run can be found at

<http://support.sas.com/samples/A55045>

Example 1: Simple DATA _NULL_ Reporting

The first step in master/detail reporting with data _null_ reporting in the data step is to join the two data sets together. It could be done in the report-writing step, but in this case I'll do it separately so I can use the resulting data set in later examples.

In the sample output, '<page>' is used to indicate a page break.

```
title 'Simple data _null_ report';

data invoices_and_items;
   merge sugi29.invoices
         sugi29.lineitems;
   by invoice_num;
run;

data _null_;

   set invoices_and_items;
   by invoice_num;

   file print;

   if first.invoice_num then
      do;
      if _n_ ne 1 then
         put _page_;
      put 'Invoice: ' invoice_num
          'for ' customer
          'on ' invoice_date;
      totalcost = 0;
      end;

   put @5 line_num  3.
       @10 item     $10.
       @22 cost     comma10.2;

   totalcost + cost;

   if last.invoice_num then
      put /
          @10  'Total:'
          @22  totalcost dollar10.2;

run;
```

```
Simple data _null_ report
Invoice: 101 for Hugo Furst on 01JAN2004
      1  Widget           12.00
      2  Gadget           10.00
      3  Frammet          36.00
      4  Thingies         37.56

         Total:          $95.56
<page>
Simple data _null_ report
Invoice: 102 for Freida Peeples on 15JAN2004
      1  Gadget           10.00
      2  Gizmo            50.37
      3  ItsIts            1.00
      4  Geehaws           5.76
      5  HooHaas          22.22

         Total:          $89.35
<page>
Simple data _null_ report
Invoice: 103 for Al E. Loohah on 30JAN2004
      1  Gadget           10.00
      2  Gizmo            50.37
      3  Whatsit         100.00
      4  ItsIts            1.00

         Total:         $161.37
```

There are two problems with data _null_ reporting: it can be very complicated to write (and to understand later), and it's designed for monospace, lineprinter output only. That was OK back in the days when programmers did all the programming and output went to lineprinters of lineprinters but nowadays many recipients want PDF or HTML output.

NEW FEATURES IN VERSIONS 6 AND 8

When version 8 came out[2], there were some substantial improvements. The Output Delivery System (ODS) gave the ability to send output to PDF, RTF, CSV, HTML, and other TLAs and ETLAs[3]. #BYVAL and #BYVAR allowed by values to be displayed in titles. Keyed access to data sets provided another way to join tables. And PROC REPORT added powerful and general support for custom reporting in a base SAS procedure.

USING #BYVAL TO CUSTOMIZE HEADERS

A common use of data _null_ was to add page titles whose text depends on data values. Sometime after version 5, the special variables #BYVAR and #BYVALUE became available to print BY data values in the header. This eliminated the need for some simple data _null_ reports.

EXAMPLE 2: USING #BYVAL

To save space, only the first page of output is shown.

```
options nobyline;
title "Invoice #BYVAL1 for #BYVAL2";

proc print data=invoices_and_items
        noobs;
   by invoice_num customer
      invoice_date;
   id line_num;
   var item cost;
   pageby invoice_num;sum cost;
   sumby customer;
run;
```

```
Invoice 101 for Hugo Furst on 01JAN2004

   Line_Num      Item            Cost

          1    Widget           12.00
          2    Gadget           10.00
          3    Frammet          36.00
          4    Thingies         37.56
-----------                  ----------
   Customer                    95.56
Invoice_Num                    95.56
```

The output isn't quite as pretty as that produced by the data step, but it is much easier to produce. And if you want to spiff it up, or send output to PDF or HTML, you can use ODS.

PROC REPORT

PROC REPORT is a very powerful reporting procedure, but is not the topic of this paper. I will give an example, but for more information see the SAS Institute documentation, or look at one of the many SUGI papers. I especially recommend papers by Lauren Haworth, Ray Pass, or Dan Bruns. PROC TABULATE is often an alternative; see papers by the same list of suspects, or Lauren Haworth's book.

EXAMPLE 3: PROC REPORT

```
title "Invoices from PROC REPORT";

proc report data=invoices_and_items
nowindows;

   column invoice_num customer
     invoice_date line_num item cost;
   define invoice_num / order noprint;
   define customer / order noprint;
   define invoice_date / display noprint;
   define line_num / order;
   define cost / sum;

   compute before _page_;
      line @1 'Invoice: ' invoice_num 3.
              ' for ' customer $10.
             ' on ' invoice_date
                    worddatx12.;
   endcomp;

   break after customer / summarize page
dol;

run;
```

```
Invoices from PROC REPORT

Invoice: 101 for Hugo Furst on 01 Jan 2004
   Line_Num  Item             Cost
          1  Widget          12.00
          2  Gadget          10.00
          3  Frammet         36.00
          4  Thingies        37.56
                          ==========
                             95.56
```

Pages 2 and 3 are omitted to save space.

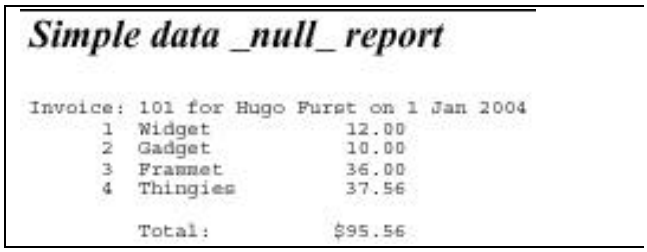
You may look at this and think "This isn't any simpler than writing a data _null_ report", but that's not quite the case. Data _null_ reporting gets more complicated more quickly than PROC REPORT, and PROC REPORT can more easily take advantage of ODS. In addition, PROC REPORT does any needed sorting and summarization for you; you don't have to explicitly code PROC SORT or summary statements.

SENDING DATA STEP OUTPUT TO ODS

You can use ODS in data _null_ reporting; just open an ODS destination, and the print output will go there. For example, if you take the program code from Example 1 and surround it with ODS statements, you can create a PDF file.

EXAMPLE 4: ODS FROM THE DATA STEP

```
ods listing close;
ods pdf
    file='example4.pdf';

%include 'example1.sas';

ods pdf close;
ods listing;
```



Simple data _null_ report

```
Invoice: 101 for Hugo Furst on 1 Jan 2004
      1  Widget           12.00
      2  Gadget           10.00
      3  Frammet          36.00
      4  Thingies         37.56

         Total:          $95.56
```

USING INDEXES

Using the data step allows you to use the KEY= option on the SET statement, which means that you don't have to merge your two input data sets (but the detail data set does have to be indexed).

EXAMPLE 5: USING INDEXES

```
title 'Data Step Example With Index';

data _null_;

   set sugi29.invoices;

   file print;

   if _n_ ne 1 then
      put _page_;

   put 'Invoice: ' invoice_num
       'for ' customer
       'on' invoice_date;
   totalcost = 0;

   _iorc_ = 0;
   do while (_iorc_ = 0);
      set sugi29.lineitems
          key=invoice_num;
      if _iorc_ = 0 then
         do;
         put @5 line_num  3.
             @10 item     $10.
             @22 cost     comma10.2;
         totalcost = totalcost + cost;
         end;
   end;
   put @5 'Total'
       @22 totalcost comma10.2;
   _error_ = 0;

run;
```

```
Data Step Example With Index
Invoice: 101 for Hugo Furst on 1 Jan 2004
      1  Widget           12.00
      2  Gadget           10.00
      3  Frammet          36.00
      4  Thingies         37.56
    Total                 95.56
<page>
Data Step Example With Index
Invoice: 102 for Freida Peeples on 15 Jan 2004
      1  Gadget           10.00
      2  Gizmo            50.37
      3  ItsIts            1.00
      4  Geehaws           5.76
      5  HooHaas          22.22
    Total                 89.35
<page>
Data Step Example With Index
Invoice: 103 for Al E. Loohah on 30 Jan 2004
      1  Gadget           10.00
      2  Gizmo            50.37
      3  Whatsit         100.00
      4  ItsIts            1.00
     Total                 161.37
```

Note: In real life, you would need additional code for exception detection - for example, if an invoice has no lines, you might want to print an explanatory note to that effect. All possible values for _IORC_ under various error conditions are shown in the document-ation for the %SYSRC system autocall macro. I don't know of anything listing the values for KEY= in particular. For this example, checking for a value of 0 (got a record) versus non-0 (didn't get a record) is sufficient.

You might look at this and think "This is much more complicated than the other code! Why bother!?", and you'd be right, but only because this is a simple example. If you have more levels of detail - cities within counties within states within regions, for example - it quickly becomes even more difficult to keep track of all the IN= and FIRST. and LAST. variables. This method allows you to put all the code dealing with one data set in one place.

IN-LINE FORMATTING IN VERSION 8.2

Version 8.2 of SAS added new formatting capabilities in titles and footnotes. Using an ODS escape character and special formatting commands, you can control the font and (to some extent) the location of individual pieces of your titles and footnotes. Although this does not add any additional data capabilities, it does increase the control your have over your output. The example below uses a style to change the appearance of some text, and the m and n commands to align text. The m tells ODS to save the current location; -2n tells ODS to go to that location on a new line.

EXAMPLE 6: IN-LINE FORMATTING

```
options nobyline;
title "^S={font=(helvetica)
font_size=3}Invoice ^m#BYVAL1 ^-2nfor
#BYVAL2 ^-2non #BYVAL3^S={}";

proc report data=invoices_and_items
nowindows
      style(summary)={font_weight=bold};
   by invoice_num customer invoice_date;

   column line_num item cost;

   define line_num / order;
   define cost / sum;

   rbreak after / summarize page dol;

run;
```

Invoice 101
for Hugo Furst
on 1 Jan 2004

Line_Num	Item	Cost
1	Widget	12.00
2	Gadget	10.00
3	Frammet	36.00
4	Thingies	37.56
		95.56

Note: This particular example isn't very interesting. A real-life example would probably eliminate some of the internal divider lines and add a description for the total line. Those things are easy enough to do, but are outside the scope of this paper.

Because this PROC REPORT uses a BY statement, you can't take advantage of PROC REPORT's ability to sort your data automatically; the input data set must already be sorted by or indexed on the BY variables.

NEW FEATURES IN VERSION 9

PROC DOCUMENT

PROC DOCUMENT is a base SAS procedure which lets you reorder and print output from SAS procedures and the data step. PROC DOCUMENT works with DOCUMENT objects, which are created by the new ODS DOCUMENT destination. This new facility (some of which is still under development) is discussed in several other sessions at this SUGI; please read the associated papers for more details. This paper presents a way to use ODS DOCUMENTs for master/detail reports, but does not explore the details.

There are two steps to the process. First, you create a document object for each section of output. For our data, there will be 3 invoice documents (one for each invoice in INVOICES) and 3 lineitem documents (again, one for each distinct invoice). Notice that it's not necessary to join the two tables together. PROC REPORT doesn't support ODS DOCUMENT as well as PROC TABULATE does, so I'm using PROC TABULATE.

EXAMPLE 7A: CREATING PROC DOCUMENT OBJECTS

```
ods document name=example7;

proc tabulate data=sugi29.invoices;
   by invoice_num;
   class invoice_num customer;
   var invoice_date;
   keylabel max=' ';
   table invoice_num * customer,

invoice_date*max*format=worddatx12.;
run;
```

```
proc tabulate data=sugi29.lineitems;
   by invoice_num;
   class line_num item;
   var cost;
   keylabel sum='';
   table (line_num * item) all,
         cost*sum;
run;

ods _all_ close;
```

You can use PROC DOCUMENT to get a list of the document objects the previous procedures created:

EXAMPLE 7B: LISTING DOCUMENT OBJECTS

```
ods document name=example7;

proc document name=example9;
   list / levels=all;
   run;
quit;
```

produces:

```
Listing of: \Work.Example9\
Order by: Insertion
Number of levels: All

  Obs    Path                                          Type
       1 \Tabulate#1                                   Dir
       2 \Tabulate#1\ByGroup1#1                        Dir
       3 \Tabulate#1\ByGroup1#1\Report#1               Dir
       4 \Tabulate#1\ByGroup1#1\Report#1\Table#1       Table
       5 \Tabulate#1\ByGroup2#1                        Dir
       6 \Tabulate#1\ByGroup2#1\Report#1               Dir
       7 \Tabulate#1\ByGroup2#1\Report#1\Table#1       Table
       8 \Tabulate#1\ByGroup3#1                        Dir
       9 \Tabulate#1\ByGroup3#1\Report#1               Dir
      10 \Tabulate#1\ByGroup3#1\Report#1\Table#1       Table
      11 \Tabulate#2                                   Dir
      12 \Tabulate#2\ByGroup1#1                        Dir
      13 \Tabulate#2\ByGroup1#1\Report#1               Dir
      14 \Tabulate#2\ByGroup1#1\Report#1\Table#1       Table
      15 \Tabulate#2\ByGroup2#1                        Dir
      16 \Tabulate#2\ByGroup2#1\Report#1               Dir
      17 \Tabulate#2\ByGroup2#1\Report#1\Table#1       Table
      18 \Tabulate#2\ByGroup3#1                        Dir
      19 \Tabulate#2\ByGroup3#1\Report#1               Dir
      20 \Tabulate#2\ByGroup3#1\Report#1\Table#1       Table
```

All we're interested in is the tables; the two in bold are the ones associated with the first BY-group.

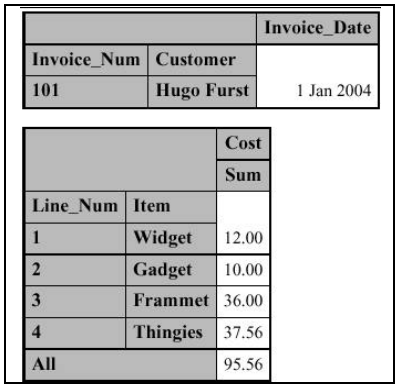
Next, you play the output back, in the correct order and with appropriate page breaks. Unfortunately, the process of setting this up is not as automated as it ought to be; I think that future releases of SAS will make this easier. We want to print out the header for the first invoice, which is the first BY-group in the first TABULATE, followed by the data for that invoice, which is the first BY-group in the second TABULATE:

EXAMPLE 7C: REPLAYING OUTPUT

```
ods pdf
file="%sysfunc(pathname(sugi29))\..\example
9.pdf" notoc;
ods pdf startpage=never;

proc document name=example9;
replay
\Tabulate#1\ByGroup1#1\Report#1\Table#1;
replay
\Tabulate#2\ByGroup1#1\Report#1\Table#1;
run; quit;
```

```
ods pdf startpage=now;

proc document name=example9;
   replay
\Tabulate#1\ByGroup2#1\Report#1\Table#1;
   replay
\Tabulate#2\ByGroup2#1\Report#1\Table#1;
run; quit;

ods pdf close;
```

This produces the following output:



		Invoice_Date
Invoice_Num	Customer	
101	Hugo Furst	1 Jan 2004

		Cost
		Sum
Line_Num	Item	
1	Widget	12.00
2	Gadget	10.00
3	Frammet	36.00
4	Thingies	37.56
All		95.56

This obviously needs more work to get the appearance right, but the concept is there.

This approach seems like it's the most work of any seen so far. So why use it? Because it lets you produce all the pieces you want separately (probably using BY-groups) and combine them in a different order later. If you want to produce a one page report with two tables and a graph, there's probably no reason to use DOCUMENTs. On the other hand, if you want to produce that page for 100 different invoices, it's going to be much more efficient to run two PROC TABULATEs and one PROC GPLOT than to run 200 PROC TABULATES and 100 PROC GPLOTS, each with a WHERE statement.

DATA STEP OBJECT-ORIENTED INTERFACE TO ODS

The object-oriented data step interface to ODS, new in release 9.0 and still experimental in release 9.1, provides a way to send information to ODS. It lets you combine the power of the data step with the output capabilities of ODS. A short way of describing it is to say that it lets you generate tables with varying numbers of rows and columns, with the ability to programmatically apply formatting to each individual cell.

The interface regards the output as a series of objects, each of which might contain one or more other objects. For our sample output, each page contains a table object containing the invoice information and another table object containing the lineitem information. Each table contains rows, each row contains cells, and each cell contains text. Each object can use the default formatting for the current style, or you can assign a different formatting.

The code for this is basically the same as that for Example 5, Using Indexes, but instead of writing lines to the output file you make function calls to the interface. Example 5 wrote text, but with the OO interface you can write to any ODS destination.

EXAMPLE 7: OBJECT ORIENTED ODS

```
data _null_;

   declare odsout Example10();

   set sugi29.invoices end=end;

   Example10.table_start();
   Example10.row_start();
   Example10.format_cell (text: 'Invoice: '
       || put(invoice_num, 3.)
       || ' for ' || trim(customer)
       || ' on ' || left(put(invoice_date, worddatx12.)));
   Example10.row_end();
   Example10.table_end();

   totalcost = 0;
   _iorc_ = 0;
   Example10.table_start();
   Example10.row_start();
   Example10.format_cell(text: 'Line Num');
   Example10.format_cell(text: 'Item');
   Example10.format_cell(text: 'Cost');
   Example10.row_end();
   do while (_iorc_ = 0);
      set sugi29.lineitems key=invoice_num;
      if _iorc_ = 0 then
         do;
         Example10.row_start();
         Example10.format_cell( text: put(line_num, 3.));
         Example10.format_cell( text: item );
         Example10.format_cell( text: put(cost, comma10.2));
         Example10.row_end();
         totalcost = totalcost + cost;
         end;
   end;
   Example10.row_start();
   Example10.format_cell(text: 'Total', column_span: 2,
                         Overrides: "font_weight=bold");
   Example10.format_cell(text: put(totalcost, comma10.2)
                         Overrides: "font_weight=bold"));
   Example10.row_end();
   _error_ = 0;
   Example10.table_end();

   if not end then
      Example10.page();

run;

ods _all_ close;
```

As you can see, table_start() starts a new table, and table_end() ends it. row_start() starts a new row, and row_end() ends it. format_cell adds a new column; the text parameter defines the contents, and the optional column_span() and overrides parameters specify additional formatting for a particular cell. Many of the style attributes that can be used elsewhere in SAS can be used in the format_cell function.

And the first page of the output looks like this:

Invoice: 101 for Hugo Furst on 1 Jan 2004

Line Num	Item	Cost
1	Widget	12.00
2	Gadget	10.00
3	Frammet	36.00
4	Thingies	37.56
Total		**95.56**

REFERENCES

All references are available through the World Wide Web. I suggest using Lex Jansen's search site at <http://www.lexjansen.com/sugi/> to find individual papers whose location is not specified below, or use the author's name and the paper title in Google - that works for everything except "The DOCUMENT Procedure", and is likely to keep working even if sites are reorganized..

Andrew Karp, Sierra Information Systems, *A Peek at PROC DOCUMENT*, <http://www.sierrainformation.com/presentationPDF/presentation_11.pdf>.

Daniel O'Connor, SAS Institute, *Next Generation Data _NULL_ Report Writing Using ODS OO Features*, SUGI 28.

Brian T. Schellenberger, SAS Institute, *ODS Layout: Arranging ODS Output as You See Fit*, SUGI 28.

Jason Secosky, SAS Institute, *The Data step in Version 9: What's New*, <http://support.sas.com/rnd/papers/sugi27/dsv9-sugi.pdf>

SAS Institute, *The DOCUMENT Procedure*, <http://support.sas.com/91doc/getDoc/odsug.hlp/a002217000.htm>.

SAS Institute, *New Features for ODS Destinations in 8.2*, <http://support.sas.com/rnd/base/topics/odsprinter/new82.html>.

CONTACT INFORMATION

Jack Hamilton
First Health
750 Riverpoint Drive
West Sacramento, California 95605 USA
jackhamilton@firsthealth.com
www.excursive.com/sas

APPENDIX I – CREATE SAMPLE DATASETS

```
options compress=no nocenter
        nodate nonumber;

data sugi29.Invoices;
   infile cards;
   input @1  Invoice_Num   3.
         @5  Customer      $14.
         @20 Invoice_Date  date9.;
   format Invoice_Date worddatx12.;
cards;
101 Hugo Furst     01Jan2004
102 Freida Peeples 15Jan2004
103 Al E. Loohah   30Jan2004
;;;;

data items;
   infile cards;
   input @1  Item  $10.
         @12 Cost  10.2;
   format cost comma10.2;
cards;
Widget     12.00
Gadget     10.00
Frammet    36.00
Gizmo      50.37
Whatsit    100.00
ItsIts     1.00
Thingies   37.56
Sprockets  13.00
Geehaws    5.76
HooHaas    22.22
;;;;
```

```
data sugi29.LineItems;
   keep Invoice_Num Line_Num Item Cost;

   set sugi29.invoices (keep=invoice_num);

   Line_Num = 0;

   do i = 1 to 10;
      if ranuni(95819) > .5 then
         do;
         line_num + 1;
         set items point=i;
         output;
         end;
   end;

run;

proc datasets library=sugi29 nolist;
   modify LineItems;
      index create Invoice_Num;
run; quit;
```

creates these data in sugi29.LineItems:

Invoice_Num	Line_Num	Item	Cost
101	1	Widget	12.00
	2	Gadget	10.00
	3	Frammet	36.00
	4	Thingies	37.56
102	1	Gadget	10.00
	2	Gizmo	50.37
	3	ItsIts	1.00
	4	Geehaws	5.76
	5	HooHaas	22.22
103	1	Gadget	10.00
	2	Gizmo	50.37
	3	Whatsit	100.00
	4	ItsIts	1.00

[1] Data _null_ reports are data steps which produce only an output print file, but no output data sets. In practice, data _null_ reports often produce summary data sets along with print output, but the name "data _null_" is still used.
[2] I'm ignoring version 7, which had only a limited release.
[3] TLA = Three Letter Acronym; ETLA = Extended Three Letter Acronym.