

Paper PN05

ODS HTML Meets Real World Requirements

Lisa Eckler, Lisa Eckler Consulting Inc., Toronto, ON
Robert W. Simmonds, TD Bank Financial Group, Toronto, ON

ABSTRACT

This paper describes a customized information delivery system using ODS HTML. The challenge that spawned this was to automate delivery of a number of static SAS®-generated reports via a corporate intranet. ODS HTML appeared to be the obvious solution, yet -- as is often the case with obvious solutions -- it failed to address many of our business requirements. These requirements included compliance with existing standards for the look and feel of reports, control over who had access to various levels of reports, and print and export functionality. We took advantage of PROC TEMPLATE, ODS HTML, PROC REPORT and the creation of an Excel spreadsheet containing an external Table of Contents to accommodate these requirements. In doing so, we developed a highly customized but fully automated report delivery application.

BACKGROUND

We set out to generate HTML code from a SAS reporting application. The pre-existing mainframe application generated printed reports that were distributed across the country. This involved printing at a central facility, then manually bursting, sorting and mailing to branches. The branches repeated some of that effort to distribute a portion of some reports to the appropriate Credit Teams. Reprints and redistribution were a common nuisance. The reports were static (monthly) but there were a lot of them and some were very large, both in width and length. ODS HTML is a very valuable tool for information delivery which provides an elegant means of accomplishing much of what was needed, but certainly not all of it. We encountered many obstacles/challenges/problems in trying to make the standard SAS ODS solution fit the organization, because the organization was not willing to fit the standard solution. This is a large organization with comprehensively enforced intranet standards for colours, fonts, menus, etc. The results had to look and feel exactly like other Web reports—both those from other applications and those that have been semi-manually created. We are not aware of any one tool that would satisfy all of the requirements, and so decided make use of the strengths of SAS and integrate the results with an existing intranet application.

OUR APPROACH

The obvious first step was to take the existing PROC REPORT code ported from another platform and wrap ODS HTML statements around it. A simplified version of that code is shown in Figure 1. Executing that code produces the results shown in Figure 2.

Figure 1

```
ods html
  path = MYDATA
  body = 'First_Example.html';

proc report data = MYDATA.PORTFOLIO nowindows;
  by branch;
  column branch team Customer loan_auth loan_os deposit_vol;
  title1 "Example 1";
run;

ods html close;
```

Figure 2

Example 1					
Branch=1093					
Branch	Credit Team	Customer Name	Loans Authorized	Loans Outstanding	Deposit Volume
1093	7	Test Customer B	2,000,000	200,000	300,000
1093	8	Test Customer C	3,000,000	300,000	300,000
1093	9	Test Customer D	500,000	250,000	250,000
1093	10	Test Customer E	500,000	250,000	250,000
1093	11	Test Customer F	1,000,000	400,000	300,000
1093	12	Test Customer G	1,000,000	200,000	300,000
1093	13	Test Customer H	1,000,000	350,000	400,000
1093	13	Test Customer I	1,000,000	200,000	300,000
1093	14	Test Customer J	500,000	300,000	300,000

...

The very simple code in Figure 1 represented a big step in the right direction, but the default style didn't meet our standards, so we used PROC TEMPLATE to define a custom style sheet. Our customized template is shown in Figure 3. After invoking the PROC TEMPLATE to define our custom style, which we called CBD_STANDARD, the same PROC REPORT code with only the title and STYLE option changed (shown in Figure 4) produced the output shown in Figure 5. This may not look like a huge difference, but conforming to colour and font standards was a non-negotiable requirement for publishing in this environment.

Figure 3

```
proc template;
  define style STYLES.CBD_STANDARD;
    parent = STYLES.DEFAULT;
    options nodate nonumber nocenter;
    ods noprint;
    replace fonts /
      'TitleFont2' = ("verdana, helvetica, sans serif",8pt,Bold)
      'TitleFont' = ("verdana, helvetica, sans serif",8pt,Bold)
      'StrongFont' = ("verdana, helvetica, sans serif",8pt,Bold)
      'EmphasisFont' = ("verdana, helvetica, sans serif",8pt,Bold)
      'FixedEmphasisFont' = ("verdana, helvetica, sans serif",8pt)
      'FixedStrongFont' = ("verdana, helvetica, sans serif",8pt,Bold)
      'FixedHeadingFont' = ("verdana, helvetica, sans serif",8pt)
      'BatchFixedFont' = ("verdana, helvetica, sans serif",8pt)
      'FixedFont' = ("verdana, helvetica, sans serif",8pt)
      'HeadingEmphasisFont' = ("verdana, helvetica, sans
serif",8pt,Bold)
      'HeadingFont' = ("verdana, helvetica, sans serif",8pt,Bold)
      'DocFont' = ("verdana, helvetica, sans serif",8pt);
    replace table from output /
      borderwidth=0
      cellpadding=3
      cellspacing=1;
    replace frame from document /
      contentposition = L;
    replace color_list /
      'fgB1' = #2D5C3D
      'bgA4' = cxFFFFF
      'bgA3' = cxDDEBDE
      'bgA2' = cxC8DDCA
      'bgA1' = cxFFFFF
      'bgA' = cxFFFFF;
    end;
run;
```

While customization of the style sheet was tedious, the same TEMPLATE will be used for all SAS reports in this application, so it was a good investment. There are many excellent books and papers available to assist in developing a customized style using PROC TEMPLATE, especially Lauren Haworth's book on ODS.

Figure 4

```

ods html
  path = MYDATA
  body = 'Second_Example.html'
  style = CBD_STANDARD;

proc report data = MYDATA.PORTFOLIO nowindows;
  by branch;
  column branch team customer loan_auth loan_os deposit_vol;
  title1 "Example 2";
run;

ods html close;

```

Figure 5

Example 2**Branch=1093**

Branch	Credit Team	Customer Name	Loans Authorized	Loans Outstanding	Deposit Volume
1093	7	Test Customer B	2,000,000	200,000	300,000
1093	8	Test Customer C	3,000,000	300,000	300,000
1093	9	Test Customer D	500,000	250,000	250,000
1093	10	Test Customer E	500,000	250,000	250,000
1093	11	Test Customer F	1,000,000	400,000	300,000
1093	12	Test Customer G	1,000,000	200,000	300,000
1093	13	Test Customer H	1,000,000	350,000	400,000
1093	13	Test Customer I	1,000,000	200,000	300,000
1093	14	Test Customer J	500,000	300,000	300,000

....

Next, we needed to separate the data at the branch and/or Credit Team level. This was a practical requirement to allow for individual printing of what was relevant, and also to accommodate branches with lower bandwidth, pending a network upgrade. The lowest drill-down level is Credit Team within branch, so the highest-volume reports are selectable at that level. The code shown in Figure 6, using the 'newfile = bygroup' option will create a new HTML file each time the 'by variable' changes within the data set. The output essentially looks the same as the previous example, except that each branch – since BRANCH is the 'by variable' in the PROC REPORT – would be a physically separate HTML file.

Figure 6

```
ods html
  path = MYDATA
  body = 'Third_Example.html'
  style = CBD_STANDARD
  newfile = bygroup;

proc report data = MYDATA.PORTFOLIO nowindows;
  by branch;
  column branch team customer loan_auth loan_os deposit_vol;
  title1 "Example 3";
run;

ods html close;
```

Once we have physically separate files, of course we need a way to access the individual files directly. Figure 7 shows code to provide the simple solution to this problem, using the 'contents' option in the ODS HTML statement.

Figure 7

```
ods html
  path = MYDATA
  body = 'Fourth_Example.html'
  style = CBD_STANDARD
  newfile = bygroup
  contents = 'Contents.html';

proc report data = MYDATA.PORTFOLIO nowindows;
  by branch;
  column branch team customer loan_auth loan_os deposit_vol;
  title1 "Example 4";
run;

ods html close;
```

While the resulting automatic table of contents, shown in Figure 8 isn't particularly pleasing to look at, it might suffice for a small number of files and one or few 'by variables'. Authors of various SUGI and NESUG papers have addressed ways to improve the appearance of the standard table of contents and the naming of files. In our situation, however, it was essential to make access as simple and intuitive as possible to accommodate the varying skill levels of our clients in the field.

Figure 8

Table of Contents	
1. The Report Procedure	
·Branch=1093	Detailed and/or summarized report
·Branch=1287	Detailed and/or summarized report
·Branch=1485	Detailed and/or summarized report
·Branch=1559	Detailed and/or summarized report
·Branch=1893	
...	

Since our application exists in the very real world, of course we didn't set out to publish one report, but 10 different monthly reports across various levels of summarization (Branch, Credit Team within Branch, Region and Total Bank). With over 300 distinct Credit Teams, that added up to over 800 separate HTML files per month, with a need to keep 3 months accessible at any time. We began development with the anticipation of expanding to include additional monthly and quarterly reports in the near future. Access to over 2,400 distinct html files makes the standard scrollable table of contents list unworkable, so some sort of hierarchical access was required.

We also faced data privacy constraints. Although 9 of our 10 reports were different summarizations of the same source dataset, access to the various results had to be restricted to people at different levels in the organization.

At this point, we diverged from a full SAS solution and decided to deliver our SAS output through an existing report delivery application on the intranet. Our menu page was cloned from an existing one in the same organization. That guaranteed that the look and feel of the menus met internal standards. It also established hierarchical choices for accessing any of the more than 2,400 files, and allowed data security to be handled through Lotus Notes authentication, so that access to various levels of detail is automatically determined based on the user's job function. The menu is an Active Server Page (ASP), illustrated in Figure 9. The reports are displayed using VB Script and dynamic HTML code. Users with the highest level of authorization see the menu shown in Figure 9, while those whose job function dictates access to less information would only see the upper half of the menu, allowing access only to the Branch and Team Level reports.

Figure 9

Our ultimate SAS program is made up of a series of REPORT procedures individually wrapped in ODS HTML statements (each similar to the code shown previously in Figure 6, but with variations in the level of summarization in the PROC REPORT and the HTML file name) and a few simple PROC SUMMARY and DATA steps. Those steps summarize and manipulate the source dataset to create a single SAS dataset which contains one row representing each HTML file and includes an assigned report identifier that relates to the menu options and the key reporting variables (reporting month, Branch, or Branch and Credit Team etc.). This effectively provides a table of contents for our HTML reports. The final step within the SAS program uses PROC EXPORT to convert that SAS dataset to an Excel spreadsheet. Figure 10 shows a simplified version of this SAS code. The first processing step of the menu interface is to use the Excel spreadsheet created within SAS to populate a database that drives the drop-down lists on the menu. The table of contents database is in SQL Server 2000 (although it could be any relational database).

Figure 10

```

proc summary data = MYDATA.PORTFOLIO nway missing n;
    class branch team;
    output out = TEAMS(drop=_type_ _freq_);
run;

proc summary data = MYDATA.PORTFOLIO nway missing n;
    class branch;
    output out = BRANCHES(drop=_type_ _freq_);
run;

data TEAMS(keep = report_id branch report_month cteam file_name);
    length file_name $ 22;
    set TEAMS;
    cteam = left(put(team,3.));
    report_id = 501;
    if _n_ = 1 then file_name = 'team_list.html';
    else fname = 'team_list' !! trim(left(_n-1)) !! '.html';
    output;
    report_id = 504;
    if _n_ = 1 then fname = 'team_vol_seg.html';
    else fname = 'team_vol_seg' !! trim(left(_n-1)) !! '.html';
    output;
run;

data BRANCHES(keep = report_id slsbran report_month team regname
                fname);
    length fname $ 22;
    retain regname '          ' team '    ';
    set work.BRANCHES;
    report_id = 502;
    if _n_ = 1 then fname = 'branch_list.html';
    else fname = 'branch_list' !! trim(left(_n-1)) !! '.html';
    output;
    report_id = 503;
    if _n_ = 1 then fname = 'branch_load.html';
    else fname = 'branch_load' !! trim(left(_n-1)) !! '.html';
    output;
    report_id = 505;
    if _n_ = 1 then fname = 'branch_vol_seg.html';
    else fname = 'branch_vol_seg' !! trim(left(_n-1)) !!
'.html';
    output;
run;

proc append base = BRANCHES
    data = TEAMS
    force;
run;

proc export data = BRANCHES
    dbms=excel97
    replace
    outfile = "&FOLDERM\Monthly SAS reports TOC.xls";
run;

```

One of the biggest challenges was to allow clients to print reports using Excel in keeping with the Bank standard on other downloadable reports. To accomplish this, our technology group helped us to imbed a button in each HTML file to export to Excel with the same look and feel as other applications currently on our website.

We used the 'HEADTEXT' option on the ODS HTML statement. The button appears at top of every file when HEADTEXT is specified in the ODS HTML call with the appropriate instruction string, and the associated Javascript code is included in the file. There are other ways to accomplish this in SAS: using the PREHTML option in the defined style would place a button at the top of every file that uses that style sheet. Similarly, POSTHTML would place a button at the bottom of every file. Since our style definition syntax is quite complex, we preferred to use one common stored style for all of our reports and vary the ODS HTML call. Figure 11 is a simple example that would include the string "*** This is sample header text ***" at the top of each file. Our actual header text string is more complex, since it defines the appearance and location of a button and includes a Javascript function call which is triggered when the body is loaded. The header text string is limited to a maximum of 256 characters.

Figure 11

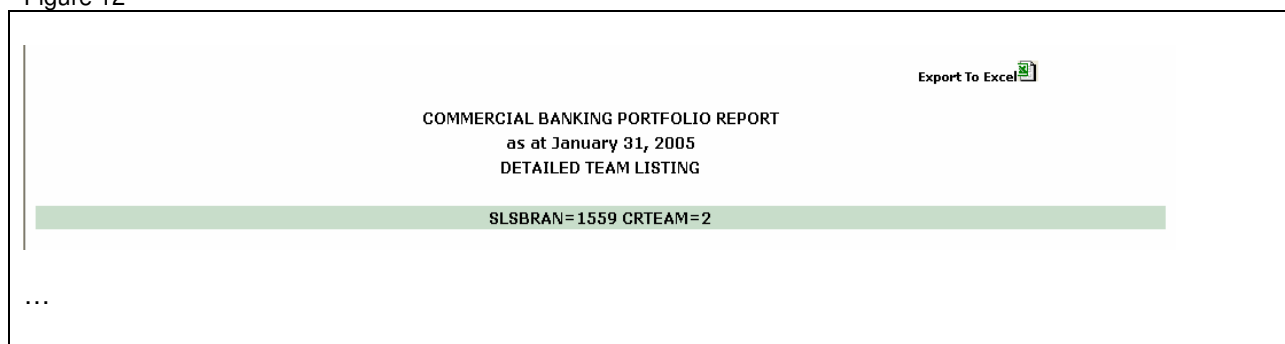
```
ods html
  path = MYDATA
  body = 'Fifth_Example.html'
  style = CBD_STANDARD
  newfile = bygroup
  headtext = '*** This is sample header text ***';

proc report data = MYDATA.PORTFOLIO nowindows;
  by branch;
  column branch team customer loan_auth loan_os deposit_vol;
  title1 "Example 5";
run;

ods html close;
```

Figure 12 shows the top section of a file created with our customized header text, with the "Export to Excel" button in the upper right corner.

Figure 12



Of course, the export button doesn't prevent any authorized user from saving or printing the files in other ways, but it is consistent with other internal applications, provides a sure way for non-technical users to export and print, and – since it is deliberately absent from the longer reports – serves to remind the users which listing reports were not intended to be printed. Use of the “@media” rule in header text further discourages printing of very long documents by generating only a single blank page if the user attempts to print the HTML file displayed on the screen using the normal menu option for printing. While the “@media” rule is recommended in SAS online documentation as a way to prevent printing of confidential information from an ODS HTML document, it should be noted that the user would still be able to select the screen contents and copy them to another format, such as a Word document for printing. See Figure 13 for an example using the “@media” rule to discourage printing.

Figure 13

```
ods html
  path = MYDATA
  body='Sixth_Example.html'
  style = CBD_STANDARD
  newfile = bygroup
  headtext=
  '<style type="text/css"> @media print {body {display:none}}</style>';

proc report data = MYDATA.PORTFOLIO nowindows;
  ...
run;

ods html close;
```

CONCLUSION

ODS HTML is a great tool. It can easily be leveraged with other elements of SAS to accommodate the need for customization and as with most SAS results, it dovetails neatly with other applications. We also learned, or at least had reinforced, the importance of collaborating with the web team in a project such as this to find the best possible solution.

REFERENCES

Haworth, Lauren E., *Output Delivery System: The Basics*, Cary, NC: SAS Institute Inc., 2001

ACKNOWLEDGEMENTS

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 registered trademarks or trademarks of their respective companies.

We would like to thank Vijaya Krishna and Michael Stoyke of TD's Commercial Banking Technology Group for their part in developing the front-end menu application, and for their collaboration.

CONTACT INFORMATION

Your comments and questions on related experiences are welcomed. You can reach the authors at:

Lisa.Eckler@sympatico.ca
Robert_Simmonds@TD.com