Paper 093-30

Creating complicated Word documents from an intranet application

Frank Poppe, PW Consulting, the Netherlands

ABSTRACT

For the sales organization of a large pharmaceuticals company we developed an intranet application with SAS/IntrNet® giving an overview of several performance indicators. Some cover sales of key products; others address the activities of the sales reps who call at doctors' offices and pharmacies.

The application users come from all levels: sales reps, managers, and vice presidents. The indicators shown on screen partly depend on their role and will sometimes reflect individual scores or alternatively suitable team aggregates.

Within the application users can choose between products, between teams, and between geographical levels, limited by their role.

All users can activate the "Word export" function. The information from all screens that can be reached within the choices in effect will be sent to output in RTF format, sent to the browser, and displayed in Word. The created document contains "boxes" where the users can enter comments and evaluations. The documents should then be passed on to the users' managers.

This presentation focuses on the Word export. It describes how the application was organized so that much code creating the tables and graphs in HTML could also be used to create basically the same for RTF. Some aspects of the layout requested by the users were difficult to realize, such as side-by-side graphs and tables. Also several limitations or idiosyncrasies in relation to the RTF destination were encountered. Solutions and ways to circumvent problems are described.

INTRODUCTION

This application is used by the Dutch sales and marketing department of a large international pharmaceutical company. The representatives (‘reps’) visit doctors and pharmacists to inform them on the company’s main products. To monitor the efforts of the reps (‘input’) and the effects of their activities (‘output’) an application has been developed that shows ‘key performance indicators’ (KPI’s) on input and on output.

This has been implemented as an internet application.

The application has a range of different users. The largest group is the reps themselves, who through this application can follow their own actions, and the sales of the target products in their area.

Also their team managers use the application, and can have an overview on the total results in the areas for which they are responsible, and can see the indicators for the members of their team.

Finally managers of different levels of control can get information on the teams and areas they are responsible for.

Figure 1 (on the next page) shows the main screen.

The drop down boxes in the navigation area at the top of the screen give each user the possibility to select the product or the team and area he is interested in. The position and the role of the person using the application determine the options available

On the left side you see the main information on the sales of the product selected, and on the right those on the efforts of the person or team selected. For both input and output more detailed information is available on subsequent screens through buttons in the navigation area at the top.

The button labeled “Word-export” will generate a Word document with the same information as on the screen (see Appendix 2 for an impression), with the information of the detail screen, and for most of the selections that can be made from the selection boxes. The generation of this file is the main subject of this presentation.

The difference in general appearance will not be discussed in detail and is mainly done by selecting different ODS styles when opening the output destinations (although in the case of HTML an external CSS file is being used as well).

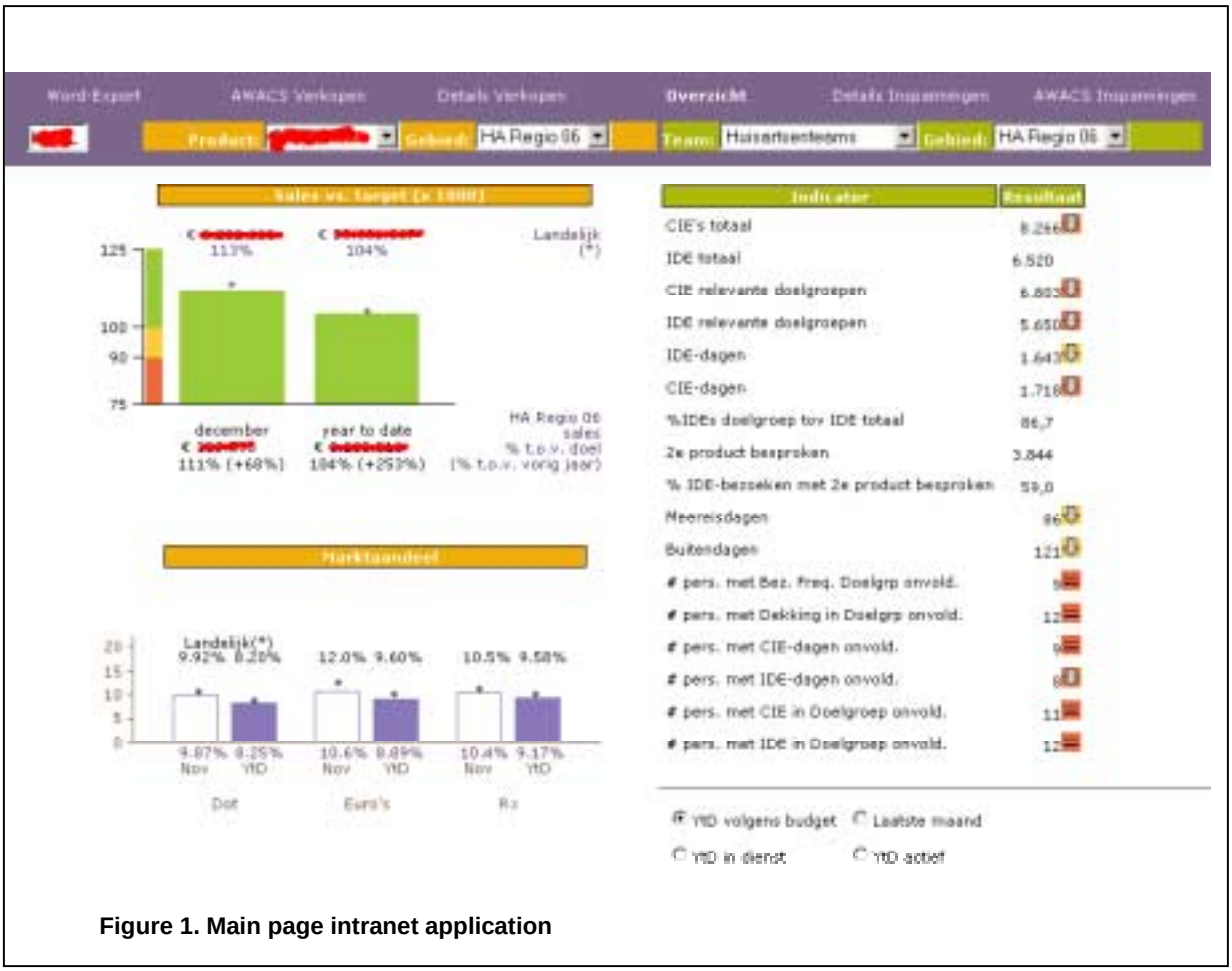

Figure 1. Main page intranet application

The information for the tables and graphs comes from the same sources and will be generated by essentially the same code. The next section describes the general organization of the source data sets. Then the subsequent sections discuss the code used to generate the Word document, focusing on the differences between the code used for the HTML output shown on the screen, and the code for the RTF output that makes up the Word document.

THE UNDERLYING DATA SETS

The data comes from a variety of internal and external sources, and is stored in different forms in the data warehouse. For this application a set of intermediate data sets has been developed that is updated on a monthly basis. There is a data set for each indicator that can be shown in any of the screens, and the data sets have an identical organization.

THE VARIABLES

One set of variables defines the aggregate level of the observation. Typical variables within this group are *userID*, team code, area code, etc.

Another set of variables contains the measurements for that level: realization year-to-date and last-month for the current year, and the comparable figures for last year. Finally there are variables that list the targets for the year-to-date and last-month values for the current year, if targets have been set for the indicator.

Not all indicators are available for all the aggregation levels, so the number of aggregation variables in the indicator data sets may differ. The meta data for the application specifies which aggregation variables are present in a specific indicator data set.

SELECTING THE INFORMATION

The different parts of the report are filled by different programs, but the data needed from the indicator data sets is always collected in the same way. For each indicator that is needed the meta data is consulted to determine which aggregate variables are present in the data set. Using the values from the drop down boxes a where clause is built for

those variables. Then the indicator data set is queried using that where clause, which should result in a single observation being returned.
This will result in a work data set containing observations for all the indicators, with one variable containing the internal number for the indicator, and the measurement and target variables described in the previous paragraph.
This data set is turned over to the code that will actually produce the graph or the table needed in that part of the report.

THE TABULAR INFORMATION

All the tables, within the intranet application and for the word document, are generated with the REPORT procedure. The tables generally show each observation (containing information for a single indicator) as a row in the table. But there are a lot more variables to show (realization year-to-date and last month, targets) than are actually visible on the screen. In fact only one figure per row is shown in the table.
However, more is available. When you look at the screen print you see that a mouse over effect has been used to show targets (and the percentage of the realization versus that target). At the bottom of the screen you see some radio buttons that allow the users to switch between year-to-date and last-month figures (and if they are team managers or higher they can also opt for figures that correct for vacancies in the team, but that is beyond the scope of this presentation).
The mouse over effect has been accomplished by assigning javascript functions to the 'onMouseOver' and 'onMouseOut' events for each table cell. The cells and the columns have also been assigned HTML class and id values, and the radio boxes activate javascript code that will turn on and off the visibility for the right cells and columns using those class and id values.
Incidentally, to get these attributes properly within the right HTML tags (through the TAGATRIB style element) the standard ODS HTML output destination was not good enough. We had to resort to a tagset defined output destination for that.
Now have a look at the analogous part of the Word document (see Appendix 2). Here both the information for year-to-date and last-month are presented, side-by-side in landscape format. The headings are repeated, maybe a bit superfluously, but that was a consequence of some requirements for the general layout.

Overzicht inspanningen, lopend jaar	Resultaat	Overzicht inspanningen, laatste maand	Resultaat
CIE's totaal	8.266	CIE's totaal	506
IDE totaal	6.520	IDE totaal	451
CIE relevante doelgroepen	6.803	CIE relevante doelgroepen	446
IDE relevante doelgroepen	5.650	IDE relevante doelgroepen	398
IDE-dagen	1.643	IDE-dagen	114
CIE-dagen	1.718	CIE-dagen	114
%IDEs doelgroep tov IDE totaal	86,7	%IDEs doelgroep tov IDE totaal	88,2
2e product besproken	3.844	2e product besproken	315
% IDE-bezoeken met 2e product besproken	59,0	% IDE-bezoeken met 2e product besproken	69,8
Meereisdagen	86	Meereisdagen	3
Buitendagen	121	Buitendagen	4
# pers. met Bez. Freq. Doelgrp onvold.	9	# pers. met Bez. Freq. Doelgrp onvold.	10
# pers. met Dekking in Doelgrp onvold.	12	# pers. met Dekking in Doelgrp onvold.	9
# pers. met CIE-dagen onvold.	9	# pers. met CIE-dagen onvold.	8
# pers. met IDE-dagen onvold.	8	# pers. met IDE-dagen onvold.	6
# pers. met CIE in Doelgroep onvold.	11	# pers. met CIE in Doelgroep onvold.	10
# pers. met IDE in Doelgroep onvold.	12	# pers. met IDE in Doelgroep onvold.	10

Figure 2. Main table on input data in Word document

GENERATION OF THE CODE

The code for HTML and for RTF is generated by the same macro. The macro inspects the different variables that make up the columns. The variables are named according to a schema that reflects the content of the variables. Then, depending on the current context, it determines what to do with it: skip, display, perhaps add a color coded arrow, etc. Within PROC REPORT this generates a CALL DEFINE statement for each variable: there are a lot of variables (much more than are visible in the actual table). It appears some interaction between the execution of macro code and of these CALL DEFINE statements sometimes upsets SAS. The log is being repeated from some seemingly random point, but the output is as expected.

(The interaction between CALL DEFINE and macro execution is also evident from the fact that turning on the system option MPRINT will also generate more information in the log on the execution of CALL DEFINE.)

ON THE ARROW SYMBOLS

Let me first explain what the meaning of the symbols is. They convey two bits of information. The color signifies with green, orange or red if the indicator is above, on (within margins) or below target is. The symbol itself shows whether the current score with respect to the current target is an upward or downward trend compared to the previous month. The image is inserted by doing some comparing in a COMPUTE block, and then selecting the appropriate picture by setting the POSTIMAGE attribute in a CALL DEFINE statement. Using a GIF picture is preferable, because it usually gives a crisper result, but PROC REPORT wants JPEG pictures, especially for the RTF destination.

SOME CODE

Below I give an extract of the macro code that does the things described above. The following macro values are used:

&computevar: the name of the column being handled;

&realvar: the variable containing the current realization;

&targetvar: the variable containing the accompanying target;

&realvar_v and *&targetvar_v*: the pair of variables with realization and target for the previous month.

I have left out the code that checks for missing values and prevents divisions by zero; what is shown is only the essential.

Style is the variable that will contain the value that in the end can be used in the CALL DEFINE statement. *Plaatje* is a variable that just contains the part for the selection of the picture, and will be inserted into *style*.

Below the code I'll give some further explanation on what is being done here.

```
compute &computevar ;
    LENGTH color arrow $ 6 plaatje style $300 ;
    score=&realvar..sum/&targetvar..sum;
    lastscore=&realvar_v..sum/&targetvar_v..sum;
    trend=score/lastscore;
    IF &realvar..sum > 1 THEN color='red';
          ELSE IF &realvar..sum < 1 THEN color='green';
          ELSE color='orange';
    IF trend < 1 THEN arrow ='down';
          ELSE IF trend > 1 THEN arrow ='down';
          ELSE pijl='equal';
    plaatje='postimage="' || "&graphDir/" ||
          TRIM(LEFT(color))||'_'||TRIM(LEFT(arrow)) || '.JPG"';
    style="style=[tagattr='onMouseOver=" ;
    style=trim(style) || '"' ;
    style=trim(style) || "return overlib('";
    style=trim(style) || trim(flyover);
    style=trim(style) || "');";
    style=trim(style) || '" onMouseout="return nd();"';
    style=trim(style) || "' ";
    style=trim(style) || ' htmlid="';
    style=trim(style) || compress(put(kpiid,32.));
    style=trim(style) || '_';
```

```
      style=trim(style) || "&computevar";
      style=trim(style) || '"';
      style=trim(style) || ' htmlclass="" ';
      style=trim(style) || trim(plaatje);
      style=trim(style) || "]" ;
      CALL DEFINE ( _COL_ , 'style' , style ) ;
  EndComp ;
```

First the scores are being computed: the realization versus the target, and then the trend.
From those values the color and the direction of the arrow is determined. These are concatenated into the name of a JPG picture, together with a path to the location of that picture, and made into a style attribute assignment.
Then the whole string for the CALL DEFINE statement is being built. First the value for the TAGATTR attribute is built. This contains javascript functions and their parameters, and requires some careful manipulation of the quotes. The HTMLID and HTMLCLASS attributes get a value, and the string for the picture is inserted.

The same technique is used for other screens as well. Team managers for instance get an overview for all the members of their team, filled in the same way with numbers and arrows. Below (Figure 3) an example of that is given, as it appears in the Word document (the initials of the members are hidden behind the red line).

Figure 3. Detail team members

GRAPHIC INFORMATION

The main screen for the intranet application already shows two graphs. The detail screen for the output information shows more graphs, and also a selection of other products of course gives other graphs. In the Word document all of these graphs should get their place. A strong wish of the users was to have graphs placed side by side on the page, like in this example in Figure 4 on the following page (some information has been air brushed away).

This was accomplished in the following way.
First I should explain how the work is organized for the intranet application. Some work for those graphs is already done before opening the ODS HTML destination. This is because some graphs require some manipulation through other SAS procedures that need the ODS Listing destination open (For those interested: the PLOT Procedure from Base SAS® Software is being used to determine the placement of labels in such a way that they do not overlap – at least not often). Opening the ODS Listing destination in turn requires the HTML destination being closed (or these intermediate results would show up there as well). Now it turned out to be a bad idea to close and open again the ODS HTML destination when that destination is _webout (i.e. ‘streaming’ towards the webserver). So that work has been moved to the beginning of the process.

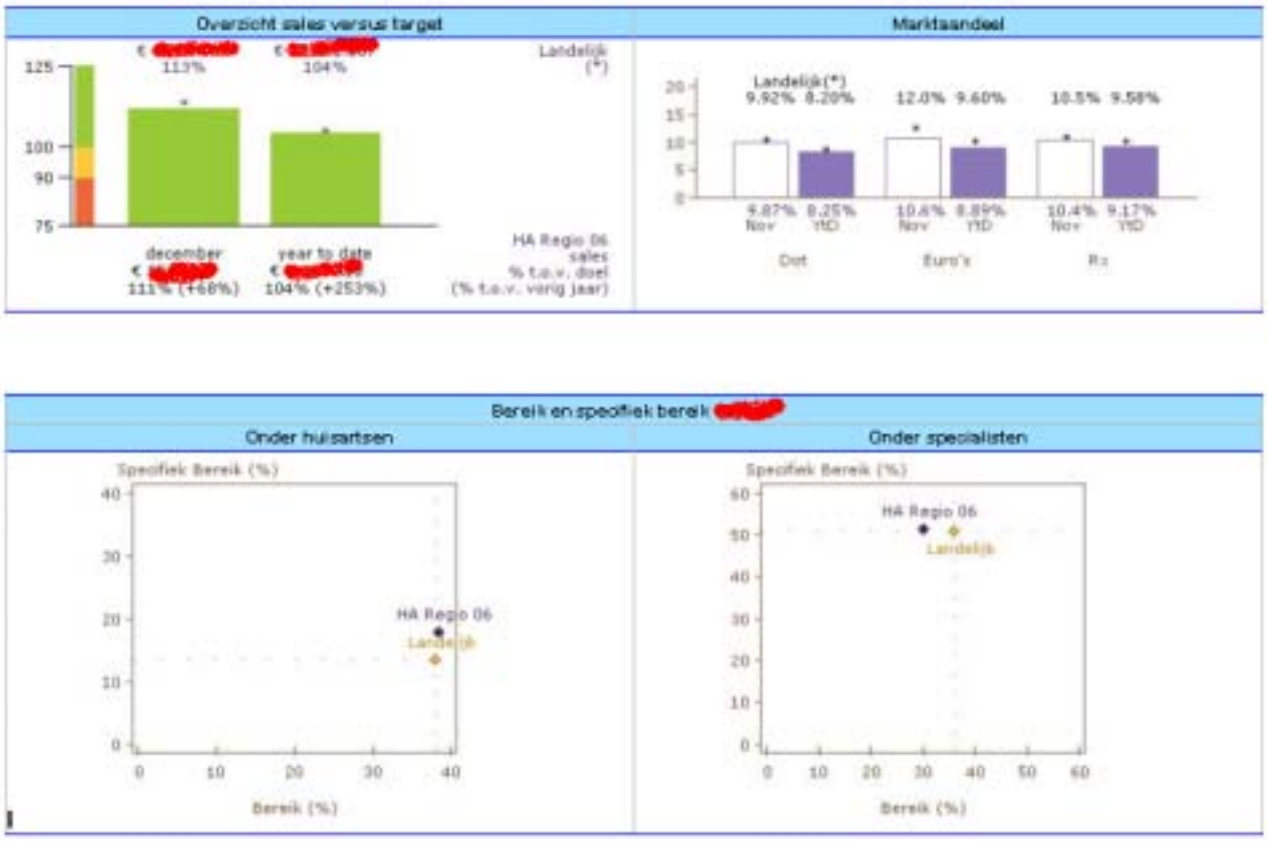

Figure 4. Example layout graphics in Word document

GETTING GRAPHS SIDE BY SIDE – WHAT DOES NOT WORK

Now there are lots of ways to place two graphs side by side. At first we attempted to solve that within the 'graphic department', using the GREPLAY Procedure. This is what that procedure is for. You create your graphs 'internally', as GRSEG entries in a graphic catalog, and using PROC GREPLAY you combine those entries into one graph. You do not need to have an ODS destination open when creating those catalog entries, so that work could be moved conveniently to the start of the whole process as well.

But it did not work out very well. It proved to be very difficult to prevent distortions of the individual graphs when they were being combined. As you may have guessed looking at the graphs, we have made heavy use of ANNOTATE. And more over, system fonts have been used instead of the SAS graphic fonts (mostly Verdana, and occasionally Wingdings). And it seems that texts created with ANNOTATE, particularly when system fonts are being used, are difficult to reproduce correctly through PROC GREPLAY.

Now PROC REPORT can use those GRSEG entries as well. This is a not very well known (nor documented) possibility:

```
call define ( _COL_ , "GRSEG" , left ) ;
```

should put the designated entry as a picture in the current column. 'Should', because, although it works fine for the HTML destination, it does not for the RTF destination. Sometimes the images are placed in the header above the cell, sometimes even above the table. This is a bug that will not be solved before SAS 9.2 arrives, and even that is not yet sure. So for the time being it is very wise that this has hardly been documented!

But there are others possibilities.

GETTING GRAPHS SIDE BY SIDE – WORK AROUND

In the previous paragraphs we have already seen the POSTIMAGE style attribute being used. Using this an image can be inserted after (or, using PREIMAGE, before) the actual content of the cell. When the content does not exist, or is made 'empty' in the COMPUTE block that is needed anyway to execute the CALL DEFINE statement, this is equivalent to transforming a cell to an image. This proved to work fine, but there are two disadvantages.

Firstly, now the picture has to be available on file. All the other solutions tried so far kept the image internally to SAS (as an entry in a SAS catalog). Now a physical location has to be determined where the picture can be written to, and

can be read from later on. Usually SAS/IntrNet applications do not need such locations. For this, the physical location of the WORK libref has been used (using the PATHNAME function).
Secondly, as noted above, it is only possible to use JPG files this way, while the GIF format generally provides crisper images. The JPG format has a quality factor that cannot be influenced when using SAS/GRAPH procedures, and seems to be set relatively low. Only when using the IMGOP function in SCL one seems to be able to influence that quality factor - but I haven't gone as far as that (and I don't know of anybody who has). Reportedly it is possible to use PNG files in SAS 9.1.3 (which would probably be a minor improvement) but this application has to run in version 8.2.

INSERTING (ALMOST) STANDARD TEXT

At different places in the Word document standard text has to be inserted. Sometimes this is completely fixed, like the headers for different parts of the report, sometimes they depend on the circumstances. This is for instance the case when the name of the user is inserted, and the date and time of production.
The most obvious way to insert text for the ODS RTF destination is to use the statement devised for that:

```
ods rtf text = "insert some text" ;
```

One disadvantage is that the text to be inserted is hard-coded, and dispersed through the code (and by now there are quite a lot lines of code!). This makes maintanance difficult. For those text that have to be determined at run time macro variables have to be used. One could use macro variables for *all* the different pieces of text, collecting the assignment of the macro values for the fixed texts in one file. That would ease the maintenance.
There is another disadvantage, however. In SAS version 8 the text inserted into the RTF stream in this way sometimes end up as part of the following table, occupying the first cell, instead of forming a block of text of its own. This behaviour is rather unpredictable and therefore difficult to prevent.
Therefore we decided to use PROC REPORT here as well. This way we had complete control over where text were going to be placed, and how they should be formatted.

PROC REPORT AS PROC PUBLISH

Many years ago their was talk about a SAS product for publishing: a PROC PUBLISH. That product never even reached the experimental stages, not outside Cary anyway. But PROC REPORT together with some ODS enhancements gives you already a lot of power. Some of those were used in this application.
All the pieces of text were collected in a SAS data set. This data set has four variables:

- *para* contains the text to be inserted, one paragraph per observation;
- *style* defines the style in which it should be typeset;
- *subject* indicates the block of text to which the paragraphs belong;
- *line* is a sequence number to ensure that the paragraphs are typeset in the right sequence within the subject.

The following table gives an excerpt from that SAS data set (called *sfemeta.wordtext*).

Para	Style	Subject	Line
Performance Tracker	style=[font_weight=bold font_size=18pt just=left]	voorblad	10
Gebruiker: |S={font_style=italic}&userid	style=[]	voorblad	20
Inputindicatoren: |S={font_style=italic}&beginDatumi t/m &afsluitDatumi	style=[]	voorblad	30
Outputindicatoren: |S={font_style=italic}&beginDatumo t/m &afsluitDatumo	style=[]	voorblad	40
	style=[font_size=2pt]	voorblad	50

Before explaining some of the things in this table let's first see what is being done with it.
The observations are extracted from that table and fed to PROC REPORT by the following macro call:

```
%report(text=sfeMeta.wordText(where=(subject eq "voorblad")), just=c);
```

This leads output in the resulting Word document similar to the following (without the lines which outline the cells).

Performance Tracker
Gebruiker: *HNEL*
Inputindicatoren: *1 januari 2004 t/m 31 december 2004*
Outputindicatoren: *1 januari 2004 t/m 31 december 2004*

The macro is defined as follows (I have added comments, without bothering to use `/*` and `*/`).

```
%MACRO report ( text = printit , just=l , width = 100 ) ;
PROC REPORT data = &text nofs noHeader spacing = 0
```

All the parameters are keyword parameters with defaults, if I fill a temporary data set *printit* with some data and call %REPORT without parameters it will typeset the content of that table.

In the case described here the data set *sfemeta.wordtext* is used, with a where clause on the variable subject.

There will be no headers above the column, and no added spacing around the table.

```
      style = [
            rules = none
            frame = void
            just = &just
            cellspacing = 0 cellpadding = 0 borderwidth = 1
            protectspecialchars = off
            font_face = Arial
            outputwidth = &width %
      ]
;
```

The PROC REPORT statement also has the STYLE option which brings the output down to its bare minimum: no horizontal of vertical lines in or around the table (rules and frame), and no extra space around the cells. It also sets the default font face, justifies the table as a whole with respect to the left and right margins (by default 'l' for left, but in this case the macro was called with 'c' for center). The width available for the table as a whole is set to 100% by default. The protection of special characters is turned off. This is because we want to be able to insert escape sequences into the text.

```
COLUMNS style para ;
DEFINE style / noPrint ;
DEFINE para      / display "" ;
```

Use only the columns *style* and *para*, but do not show *style.*

```
COMPUTE para ;
      para=resolve(para);
```

The content of the variable *para* is subjected to the function RESOLVE. This invokes the macro compiler, and thus resolves all references to a macro variable that it finds. This is how the macro variables that you see in the table above (e.g. *&userid*) are being replaced by values at run time (*HNEL* in this case).

```
      CALL DEFINE ( _COL_ , "style" , style ) ;
```

Now CALL DEFINE is being used to change the style of the cell. As new value for the style the content of the (non visible) variable *style* is taken. From the table above you see that sometimes no explicit choices are made, but for the first line font weight and size are set.

```
ENDCOMP ;
RUN ;
%MEND ;
```

This leaves two things unexplained.

The content of the variable *para* sometimes contains things like `|S={font_style=italic}` which do not show up in the output. Instead, the text following it is set in italic. These are ODS escape sequences. The vertical bar has been

used as the escape character (by specifying `ods escapechar '|' ;`) and through S= you can then change style attributes for part of the text.
The other thing still to explain is the last line which only sets the font size to the smallest possible value. Since normally there are no lines around the cells in the output, this is a hardly noticeable extra line. But SAS inserts for some unknown reason a number of empty lines before the next block of output is started. Often these lead only to unwanted white space. It appears that the last specified font size is still in force when these empty lines are inserted, so setting the font size as low as possible reduces the white space.

CONCLUSION

The use of ODS as a general engine to produce output has made it possible to effectively use the same code for the different screen of an intranet application and for the different parts of a Word document. The flexibility of ODS in general and of the REPORT procedure made it possible to accommodate the different requirements for the different destinations. This flexibility also made it possible to overcome some deficiencies found, particularly in producing RTF output and the treatment of graphics.
The RTF output has made it possible to generate a report according to a pre defined format, enabling the users to enter their own comments and additions on the tables and graphs.

CONTACT INFORMATION

If you have any comments, suggestions or questions I would like very much to hear from you. You can find my contact information below.

Frank Poppe
PW Consulting
postal address
Laan van Meerdervoort 961
2564 AJ The Hague
The Netherlands
email: Frank.Poppe@PWcons.com
web: www.pwcons.com