Paper 16-28

Fancy MS Word Reports Made Easy: Harnessing the Power of Dynamic Data Exchange

— Against All ODS, Part II —

William W. Viergever, Viergever & Associates

Koen Vyverman, SAS Netherlands

Abstract

For many years, the SAS System for OS/2 and Windows has had the ability to talk to other PC applications by means of Dynamic Data Exchange (DDE). DDE enables the creation of fully customized Microsoft Word documents from within a Base SAS program, and is therefore ideal for automating high quality output from the SAS System. Using a set of easy-to-use SAS macros, we show the beginner-to-advanced SAS user how to insert and format SAS data, graphs, and tables into a Word report. As a further demonstration of the myriad possibilities, a DDE-driven Word mail-merge application is constructed

0 Introduction

Notwithstanding the SAS System's prodigious capacity for reading and writing files of virtually any pedigree, the generation of output in the form of MS Word documents has always been a bit of a pain—to say the least. A number of mechanisms come to mind:

- intermediate file formats like flat-file layouts and HTML in particular can be easily produced by means of some Base SAS code, and may be converted or incorporated into MS Word documents by means of VBA macros.
- OLE automation from within a SAS/AF application can create Word documents.
- the rather recent addition of the RTF destination to the Output Delivery System (ODS) makes it particularly easy to deliver output that can be read by the MS Word application.
- and then there's Dynamic Data Exchange (DDE), to which the present paper is devoted.

Which of these techniques should you choose? Which one is easier? Which one allows the highest level of control over the output? All seemingly valid questions, but in fact, they are largely nuncupatory since the answer must always be 'it depends'…

It depends on the level of complexity you're trying to achieve: do you simply wish to see a piece of proc tabulate output as a table in a Word document? Or did you have a complex page lay-out in mind, with several items of tabular and graphical information sitting at precise locations and being formatted in some very specific manner?

It depends on your programming experience: how familiar are you with VBA? Are you an OLE and SCL whiz-kid? To get the most out of the ODS, the proc template formalism must be mastered. Are you considered a proc template guru? DDE employs Microsoft's old MS Word macro language from before the days of VBA: WordBasic. Ever heard of it?

The point we want to make here is that none of the available techniques is simple and straightforward, each using its own particular lingo that needs to be mastered, each having a definitely mountainous learning curve.

The purpose of this paper is to take some of the steepness out of the DDE learning curve by providing numerous examples of how to build up a Word document step by step, and to introduce a set of SAS macros that encapsulate some particularly useful and frequently recurring DDE/WordBasic functionality.

The first section introduces the concept of DDE. It is followed by three sections in which the ground-work is laid for building a Word document from scratch by means of simple Base SAS code. This includes inserting and formatting of text, tables, and graphics, as well as how to add some nifty finishing touches to a document, like headers, footers, and hyperlinks. As an example, we build a personalized credit card statement based on data from a SAS data set.

Then we show how to simplify the process of document creation by means of a template document that already contains all the properly formatted static elements, together with place-holders for the variable bits of output coming from SAS. We reproduce the card statement using this technique.

Finally we introduce the set of SAS macros, and show how to use them in order to, once more, generate our credit card statement letter.

All of the code shown below—as well as the template document—will be available online as a set of '.sas' source files numbered in correspondence with sections one through eight. Visit www.vyverman.com and follow the link to SAS-related material to download the package…

The code has been developed and tested on a Windows2000 Professional system, running release 8.2 of the SAS System and MS Office97.

1 Introducing DDE from SAS to Word

A rather detailed discussion of what DDE is and how to access it from Base SAS code is given in Vyverman (2002). No need to repeat all that here, so we'll merely paraphrase and summarize the essential bits: Dynamic Data Exchange is a communication protocol available on Windows and OS/2. It enables DDE-compliant applications on these platforms to talk to each other in a client/server fashion.

The SAS System is only DDE-compliant in the sense that it can act as a client but not as a server. That is, SAS can contact Word and ask it to do certain things, but not the other way around. DDE can then be seen as a common language that both applications use for their communications. See also the Technical Support document TS325 (SAS Institute, 1999) for a more extensive introduction to DDE.

Considering the client/server picture, it is clear that a prerequisite for using DDE is that both client and server applications are up and running, otherwise they couldn't have a chat.

Once both a SAS session and Word are running, a client/server DDE-communication is initiated by means of a special form of the SAS `filename` statement. It comes in two flavours:

```
filename <fileref> dde
  'winword|<document>!<bookmark>';
```

and

```
filename <fileref> dde 'winword|system';
```

In the first of these, the expression `winword|<document>!<bookmark>` is known as the DDE-triplet. It tells the SAS session that it needs to talk to Word, that the conversation is about a certain document, and what part of it to read/write data from/to. Once a SAS fileref has been defined in this manner, it can be used on `file` and `infile` statements just like any other SAS fileref. The usual `put` and `input` statements will then write to and read from the specified bookmarked location in the document.

We'll see more of these Word bookmarks and how to use them in section 6 and beyond. For the time being, we merely use the second form of the DDE `filename` statement, which for naming consistency we shall refer to as the DDE doublet, or system-doublet. Contrary to a triplet-style DDE-fileref which is used to read/write from/to a specific place in a specific document, the system-doublet allows the SAS session to send commands to Word, telling it what to do. These commands must be formulated in WordBasic, the old MS Word macro language from before the VBA days.

It is oft noted that the main frustration with using DDE between SAS and Word stems from the fact that one cannot find the proper WordBasic function to get something specific done. However, there is a WordBasic help-file available from the Microsoft site: download and run the file 'wrdbasic.exe' from http://www.microsoft.com/downloads/release.asp?ReleaseID=26572 This installs the help-file, containing full syntax for hundreds of WordBasic functions, including some examples.

This said, we're ready to get started. Well, almost… We'll finish the preliminaries by firing up the Word application, and creating a demo data set. We can start Word using the method described by Roper (2000):

```
options noxsync noxwait xmin;
filename sas2word dde 'winword|system';
data _null_;
  length fid rc start stop time 8;
  fid=fopen('sas2word','s');
  if (fid le 0) then do;
    rc=system('start winword');
    start=datetime();
    stop=start+10;
    do while (fid le 0);
      fid=fopen('sas2word','s');
      time=datetime();
      if (time ge stop) then fid=1;
      end;
    end;
  rc=fclose(fid);
run;
```

This will start an instance of Word if none is running yet, and keeps trying for at most ten seconds to establish a DDE-conversation via the system-doublet `sas2word`.

```
data cards;
  length
    card_holder
    dept
    product     $ 30
    card_type
    signature   $ 20
    order_date
    amount_usd  8
    ;
  format
    order_date date9.
    ;
  infile datalines4 dlm='#';
  input
    card_holder    :$char30.
    card_type      :$char20.
    product        :$char30.
    order_date     :date9.
    amount_usd     :8.
    dept           :$char30.
    signature      :$char20.
    ;
  datalines4;
You SAS Programmer#corporate#Belgian Beers
  #12apr2002#4566.99#the Finance Dept#Jim
You SAS Programmer#corporate#
  Camembert Supplies#23may2002#314.15#the
  Finance Dept#Jim
You SAS Programmer#corporate#Armadillo
  Droppings#01jul2002#35.00#the Finance
  Dept#Jim
Bill Viergever#private#Petite Syrah#
  20jun2002#899.99#your Significant Other#
  The Wife
Bill Viergever#private#Blaauwklippen
  Zinfandel#15sep2002#1499.50#your
  Significant Other#The Wife
Bill Viergever#private#560Gb RAID disk
  array#20jun2002#2995.00#your Significant
  Other#The Wife
;;;;
run;
```

The data set `work.cards` has some fictitious credit card expense data. Any resemblance to actual persons, living or dead, must be seen as sheer coincidence. Note that the data-lines have wrapped to fit the columnar lay-out, so don't copy/paste this into a SAS program editor, but rather use the SAS code files downloaded from www.vyverman.com

We recommend doing this in any event: although we will describe what the code is doing as we proceed through the following sections, there is nothing like seeing it happen in front of your own eyes. The ten-page limitation of this paper does not allow a Word screenshot for every command sent from SAS…

2 Building a Document: Handling Text

As explained in the introduction, we will start laying the foundations for some serious DDE work by showing how to handle text, tables, and graphics. There is a lot of nitty-gritty coding here: all the strange instructions you'll see enclosed within square brackets are in fact WordBasic commands that our SAS session is handing over to the Word application by means of the `sas2word` DDE system-doublet fileref.

It is worth noting that most of the WordBasic functions we'll be using here, have many more parameters than we are actually showing. To find out what more these functions can do for you, and what the default settings are for the parameters we're not explicitly passing through, please see the 'wrdbasic.hlp' help file…

So, we've just booted up Word. The following shows how to do some really elementary stuff, like opening, closing and saving documents:

```
data _null_;
  file sas2word;
  put '[FileClose 2]';❶
  put '[FileNew.Template="Normal.dot"]';❷
  put '[ChDefaultDir "c:\p16-28\
       output",0]';❸
  put '[FileSaveAs.Name="0201 Elementary
        Stuff",.Format=0]';❹
run;
```

❶ We get rid of the default document that Word presents after it's started up. Of course we could keep it and use it, but for the sake of demonstrating functions… The parameter 2 means no saving before the active document is closed.

❷ Then we make a new blank document, based on the 'Normal.dot' Word document template.

❸ This changes the default directory where documents are saved to and opened from. When working within a single directory, it is more practical to use this rather than to specify the full directory path on each open or save command.

❹ The active document is saved in the default directory. The `Format=0` means it is saved as a normal Word document. Other options include various flat-file formats, and RTF.

For faster processing, we minimize the Word application, which saves big time on CPU cycles during a DDE output job by avoiding continuous screen redraws as SAS keeps Word busy doing all manner of things.

```
data _null_;
  file sas2word;
  put '[AppMinimize]';
run;
```

Then we go inserting some text:

```
data _null_;
  file sas2word;
  put '[Insert "Godspeed You SAS
       Programmer!"]';❶
  put '[InsertPara]';❷
  put '[InsertPara]';
  put '[Insert "Please find below some
       items recently charged to your"]';
  put '[Insert " corporate credit card.
       Kindly forward the appropriate"]';
  put '[Insert " receipts to the Finance
       Dept."]';
  put '[InsertPara]';
  put '[InsertPara]';
  put '[Insert "Product"+Chr$(9)+"Order
       Date"+Chr$(9)+"Amount [USD]"]';❸
  put '[InsertPara]';
  put '[Insert "Belgian Beers"+Chr$(9)
       +"12-Apr-2002"+Chr$(9)+"4566.99"]';
  put '[InsertPara]';
  put '[Insert "Camembert Supplies"+
       Chr$(9)+"23-May-2002"+Chr$(9)+
       "314.15"]';
  put '[InsertPara]';
  put '[Insert "Armadillo Droppings"+
       Chr$(9)+"01-Jul-2002"+Chr$(9)+
       "35.00"]';
  put '[ParaUp 4,1]';❹
  put '[TextToTable.ConvertFrom="1",
       .NumColumns="3",.NumRows="4",
       .InitialColWidth="Auto",.Format="9"
       ,.Apply="63"]';❺
  put '[EndOfDocument]';❻
  put '[InsertPara]';
  put '[InsertPara]';
  put '[Insert "Kind Regards,"]';
  put '[InsertPara]';
  put '[Insert "Jim."]';
run;
```

❶ The `insert` function inserts text at the insertion point, just as if you'd type into the document.

❷ Where necessary we insert a paragraph break.

❸ Here we insert some lines of text that will be turned into a table. To facilitate the process, we build a concatenated string where all items are separated by a tab-character.

❹ It is extremely important to keep track of where the insertion point is at all times. Some functions like `ParaUp` will help you moving it around the document. Since we've inserted four lines to become a table, we go 4 paragraphs up. The second argument tells Word to make it a selection. So we've now effectively selected the lines we wish to turn into a table.

❺ The `TextToTable` function turns the selection into a Word table object. The cool thing is that we can have it apply some predefined format, saving us further manual labour.

❻ Before continuing to write our document, we need to release the selection again and place the insertion point at the end of the document..

Then we can apply some formatting to the text we've just inserted. The following selects the entire document and changes the font to 18 point Verdana:

```
data _null_;
  file sas2word;
  put '[EditSelectAll]';
  put '[Font "Verdana"]';
  put '[FontSize 18]';
run;
```

We then select the title only, make it a bit larger, and give it a bold attribute:

```
data _null_;
  file sas2word;
```

```
  put '[StartOfDocument]';
  put '[ParaDown 1,1]';
  put '[FontSize 24]';
  put '[Bold]';
run;
```

Note the use of the `StartOfDocument` function to place the insertion point at the beginning of the text, followed by the `ParaDown` which selects everything until the first paragraph marker encountered in the downward direction.

We exercise some more cursor control as we want to make the 'Finance Dept' appear in italics:

```
data _null_;
  file sas2word;
  put '[StartOfDocument]';
  put '[Paradown 3,0]';
  put '[CharLeft 2,0]';❶
  put '[WordLeft 2,1]';❷
  put '[Italic]';
run;
```

❶ `CharLeft` is equivalent to hitting the left arrow key. This brings us just behind the two words we wish to italicize.

❷ And then we select the next two words to the left of the insertion point. Note once more the second argument of all these functions for moving around: a zero merely moves the cursor, one extends a selection.

We're done here for now, so we save and close our document. Just for the sake of showing the function, we'll also kill the Word application with a `FileExit`:

```
data _null_;
  file sas2word;
  put '[FileSave]';
  put '[FileClose 2]';
  put '[FileExit 2]';
run;
```

3 Building a Document: Handling Tables

Now some more about working with tables. Since we've just closed Word down, we need to start it up again using the bit of code from Section 1.

Then we open the document we made in Section 2 and save it under a different name:

```
data _null_;
  file sas2word;
  put '[AppMinimize]';
  put '[FileClose 2]';
  put '[ChDefaultDir "c:\p16-28\
       output",0]';
  put '[FileOpen.Name="0201 Elementary
       Stuff"]';
  put '[FileSaveAs.Name="0302 Table
       Formatting",.Format=0]';
run;
```

First we tweak the column widths:

```
data _null_;
  file sas2word;
  put '[StartOfDocument]';
  put '[EditGoto.Destination="t"]';❶
  put '[TableSelectTable]';❷
  put '[TableColumnWidth.Autofit]';❸
  put '[StartOfDocument]';
run;
```

❶ The `EditGoto` function is useful for jumping to named regions within the active document. A named region can *e.g.* be a bookmark. Word also maintains certain automatic bookmarks. Specifying `Destination="t"` takes us to the next table in the document, which is where we want to be.

❷ Unsurprisingly, this selects the entire table the insertion point is currently positioned into.

❸ We first try an autofit on the column widths. Not entirely satisfying, so we tweak the column widths individually. Note the use of functions like `StartOfRow` and `NextCell` to move around in the table, shuffling the insertion point from column to column:

```
data _null_;
  file sas2word;
  put '[StartOfDocument]';
  put '[EditGoto.Destination="t"]';
  put '[TableSelectColumn]';
  put '[TableColumnWidth.ColumnWidth="5.19
       cm"]';
  put '[StartOfRow]';
  put '[NextCell]';
  put '[TableSelectColumn]';
  put '[TableColumnWidth.ColumnWidth="5.00
       cm"]';
  put '[NextCell]';
  put '[NextCell]';
  put '[TableSelectColumn]';
  put '[TableColumnWidth.ColumnWidth="5.25
       cm"]';
  put '[StartOfDocument]';
run;
```

Or we can apply another autoformat to the table. The format number can be determined by selecting 'Table Autoformat' from the 'Table' menu in Word, and then counting: '(none)' corresponds to zero; the next one down is number one; and so forth.

```
data _null_;
  file sas2word;
  put '[StartOfDocument]';
  put '[EditGoto.Destination="t"]';
  put '[TableAutoFormat.Format=6]';
  put '[StartOfDocument]';
run;
```

Before proceeding, we save and close our document:

```
data _null_;
  file sas2word;
  put '[FileSave]';
  put '[FileClose 2]';
run;
```

So far, we managed to make a decently looking Word table from rows of stuff that we inserted manually into the document. Let's have a look now at how to create a Word table starting from the SAS data set `work.cards` that we made earlier on. First, a new document, give it a name and save it:

```
data _null_;
  file sas2word;
  put '[FileNew.Template="Normal.dot"]';
```

```
  put '[ChDefaultDir "c:\p16-28\
       output",0] ';
  put '[FileSaveAs.Name="0307 Table from a
       SAS Data Set",.Format=0] ';
run;
```

Then we prepare the data we wish to send to Word by stuffing all the desired variables into a concatenated text string `table_row` separated by tab-characters. Sounds familiar?

```
%let tab='09'x;

data _data_for_word;
  set cards;
  length table_row $ 300;
  table_row=trim(left(product))
            ||&tab
            ||trim(left(put(order_date,
                            date9.)))
            ||&tab
            ||trim(left(put(amount_usd,
                            10.2)));
  keep table_row;
run;
```

The following then sends the `table_row` string to the Word document, selects everything, and turns it all into a formatted table:

```
data _null_;
  file sas2word;
  set _data_for_word end=last;
  length ddecmd $ 300;
  ddecmd='[Insert '!!
          quote(trim(table_row))!!'] ';
  put ddecmd;
  if not last then put '[InsertPara] ';
  else do;
    put '[EditSelectAll] ';
    put '[TextToTable.ConvertFrom="1",
         .NumColumns="3",.NumRows="3",'
         .InitialColWidth="Auto",
         .Format="35",.Apply="87"] ';
    end;
run;
```

We save this, but leave the document open for the next section:

```
data _null_;
  file sas2word;
  put '[FileSave] ';
run;
```

4 Building a Document: Handling Graphics

Now, suppose we wish to show a SAS graph in our Word document. To this effect we make a `proc gchart` 3-dimensional pie-chart based on some of the variables in the `work.cards` data set, and save it to a temporary SAS catalog `work.graphs` as a GSEG-entry:

```
proc build cat=work.graphs batch;
quit;

proc gchart data=cards
            gout=work.graphs;
  format amount_usd dollar8.2;
  pie3d product / sumvar=amount_usd
                  missing
                  noheading
                  slice=arrow
                  name="pie"
                  coutline=black;
  run;
quit;
```

Using `proc greplay` and the necessary `goptions` settings, we export our GSEG-entry as a GIF graphic:

```
filename pie "c:\p16-28\temp\pie.gif";

goptions
  device=gif373
  gsfname=pie
  gsfmode=replace
  display;

proc greplay nofs igout=work.graphs;
  replay pie;
quit;

filename pie clear;

goptions reset;
```

Then we insert this GIF file into our Word document, and do some re-scaling:

```
data _null_;
  file sas2word;
  put '[EndOfDocument] ';
  put '[InsertPara] ';
  put '[InsertPicture.Name="c:\p16-28\
       temp\pie.gif"] ';❶
  put '[CharLeft 1,1] ';❷
  put '[FormatPicture.ScaleX="150%",
       .ScaleY="150%"] ';❸
  put '[CharRight 1,0] ';
  put '[InsertPara] ';
run;
```

❶ The `InsertPicture` function is equivalent to choosing 'Picture From File' from the Word 'Insert' menu.

❷ Selects the graphic, equivalent to hitting the left arrow key while holding the shift-key down.

❸ Blowing up the size of this GIF image is clearly not a fabulous idea. It looked pretty pixelated to start with, and now it has become downright ugly.

Of course, we picked a rather small GIF size by using the `GIF373` graphics device driver. We might use a larger one, like `GIF570` or even `GIF733`, but the fact remains that the GIF format does not scale well, because essentially it is a bitmapped graphic format.

To get decent results in Word, we need to resort to some sort of scaleable graphics format, or vector-graphics. We've had good results with the Computer Graphics Metafile (CGM) format. Let's run a slightly modified version of the `proc greplay` code above, and export our pie-chart GSEG-entry as a CGM file:

```
filename pie "c:\p16-28\temp\pie.cgm";

goptions
  device=cgmof97l
  gsfname=pie
  gsfmode=replace
  display;
```

```
proc greplay nofs igout=work.graphs;
  replay pie;
quit;

filename pie clear;

goptions reset;
```

Then insert the CGM file into the Word document:

```
data _null_;
  file sas2word;
  put '[EndOfDocument]';
  put '[InsertPicture.Name="c:\p16-28\
       temp\pie.cgm"]';
  put '[CharLeft 1,1]';
  put '[FormatPicture.ScaleX="150%",
       .ScaleY="150%"]';
  put '[CharRight 1,0]';
  put '[InsertPara]';
run;
```

This looks acceptable now. The graphic's dimensions can effectively be re-scaled to any size, without a noticeable loss of image quality. We save and close the document…

```
data _null_;
  file sas2word;
  put '[FileSave]';
  put '[FileClose 2]';
run;
```

5 Building a Document: Finishing Touches

To finish off the foundation work in building a decently looking Word document from scratch, all from within Base SAS code, we present a few nice touches to make professionally looking Word output.

We open the credit card statement/letter once more, as we made it in Section 3:

```
data _null_;
  file sas2word;
  put '[ChDefaultDir "c:\p16-28\
       output",0]';
  put '[FileOpen.Name="0302 Table
       Formatting"]';
  put '[FileSaveAs.Name="0501 Headers
       Footers Hyperlinks",.Format=0]';
run;
```

In the header of the Word document, we wish to display a date-time stamp. First we populate a SAS macro variable `reptdate` with some wordy date-time string:

```
data _null_;
  reptdate=put(today(),worddatx.)||' at '
           ||put(time(),time.);
  call symput('reptdate',
              trim(left(reptdate)));
run;
```

Still writing to the `sas2word` system-doublet, we open the header, insert text, and close it again:

```
data _null_;
  file sas2word;
  put '[ViewHeader]';
  put '[FormatFont.Font="Courier New",
       .Points="10",.Bold=1]';
  put '[Insert "Report created by the
       SAS System, ' "&reptdate" '"]';
  put '[CloseViewHeaderFooter]';
run;
```

Note the `FormatFont` function, which combines in one call several font attributes that we had hitherto set separately using WordBasic functions like `Font`, `FontSize`, `Bold`, …

In the document footer, we insert a 'page x of y' indicator:

```
data _null_;
  file sas2word;
  put '[ViewFooter]';
  put '[FormatFont.Font="Courier New",
       .Points="10",.Bold=1]';
  put '[Insert Chr$(9)]';
  put '[Insert "Page "]';
  put '[InsertField.Field="PAGE
       \* Arabic"]';
  put '[Insert " of "]';
  put '[InsertField.Field="NUMPAGES
       \* Arabic"]';
  put '[CloseViewHeaderFooter]';
run;
```

Here we've used the `InsertField` function to insert a `page` and a `numpages` field. The `\* Arabic` modifier tells Word to display the calculated page numbers as Arabic numerals. Similarly, `\* Roman` can be used for Roman numerals.

These are two among a long list of predefined fields that can be embedded in similar vein into a Word document. A lot of fancy stuff can be done with fields, but exploring this further would lead us too far astray now. Again, we refer to the 'wrdbasic.hlp' help-file for more details and inspiration.

However, since this is definitely a frequently asked question, we'll give one more example of how to use the `InsertField` function to create a hyperlink within the document. We insert a hyperlink at the bottom of the letter which, when clicked, jumps back to the top of the document:

```
data _null_;
  file sas2word;
  put '[StartOfDocument]';
  put '[EditBookmark.Name="Back_to_top",
       .Add]';❶
  put '[EndOfDocument]';
  put '[InsertPara]';
  put '[InsertField.Field="HYPERLINK
       \l Back_to_top"]';❷
run;
```

❶ Using the `EditBookmark` function we place a Word bookmark at the start of the document and call it `Back_to_top`.

❷ At the end of the document we insert a `hyperlink` field. The `\l` modifier followed by a bookmark name links it to the top of the page.

Using these internal hyperlinks by means of bookmarked places in a long report allows *e.g.* the creation of a handy table of contents so that the reader can easily jump to any section of interest.

We save and close the current document:

```
data _null_;
```

```
  file sas2word;
  put '[FileSave]';
  put '[FileClose 2]';
run;
```

6 Working with Template Documents

So far we've been building Word documents from scratch, painstakingly inserting various bits of text, tables and graphics, and then applying formatting manually as well. Looking at our latest credit card letter though, there is a good deal of static matter in there that remains the same for any letter we could produce from our `work.cards` data set.

As explained in Section 1, instead of writing WordBasic commands to the DDE system-doublet, it is also possible to define triplet-style DDE filerefs that point to specific bookmarked locations in a Word document. The download package contains such a 'template' document. It contains all the static bits of text, the page x of y footer, and the hyperlink. It has a number of bookmarks defined at those locations where variable bits of data should be inserted. We will now re-create our card statement based on this 'Card Report Template' document.

To see the existing bookmarks in a Word document, choose 'Options' from the Word 'Tools' menu, and make sure the 'Bookmarks' checkbox is marked. Bookmarks are then visible as pairs of square brackets.

First we open the existing 'Card Report Template' Word document and save it under a different name:

```
data _null_;
  file sas2word;
  put '[ChDefaultDir "c:\p16-28\
       templates",0]';
  put '[FileOpen.Name="Card Report
       Template"]';
  put '[ChDefaultDir "c:\p16-28\
       output",0]';
  put '[FileSaveAs.Name="0601 Letter from
       Template",.Format=0]';
run;
```

Then we prepare the various variable data elements to feed into the bookmarked areas:

```
data _data_for_bookmarks;
  set cards;
  where upcase(card_holder)='YOU SAS
                             PROGRAMMER';
  length table_row $ 300;
  table_row=trim(left(product))
            ||&tab
            ||trim(left(put(order_date,
                            date9.)))
            ||&tab
            ||trim(left(put(amount_usd,
                            10.2)));
run;
```

We need to define as many DDE triplet-style filerefs as there are bookmarked areas to send SAS data to:

```
filename card_hol dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  card_holder' notab;
filename card_typ dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  card_type' notab;
filename dept dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  dept' notab;
filename signatur dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  signature' notab;
filename table dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  table' notab;
```

Note the `notab` option on each `filename` statement. This prevents spaces in the SAS data to be translated into tab-characters, which is some kind of mostly undesirable default behaviour that we need to turn off. Then we write the various data elements to the filerefs defined above:

```
data _null_;
  set _data_for_bookmarks(obs=1);
  file card_hol;
  put card_holder;
  file card_typ;
  put card_type;
  file dept;
  put dept;
  file signatur;
  put signature;
run;
```

And since all the variable bits of text in the letter are now enclosed within a bookmark, applying formatting is very easy:

```
data _null_;
  file sas2word;
  put '[EditGoto.Destination=
       "card_holder"]';
  put '[FormatFont.Font="Verdana",
       .Points="24",.Bold=1]';
  put '[EditGoto.Destination=
       "card_type"]';
  put '[FormatFont.Font="Verdana",
       .Points="18"]';
  put '[EditGoto.Destination="dept"]';
  put '[FormatFont.Font="Verdana",
       .Points="18",.Italic=1]';
  put '[EditGoto.Destination=
       "signature"]';
  put '[FormatFont.Font="Verdana",
       .Points="18"]';
run;
```

So no more careful tiptoeing around the document to move the cursor to the items that need formatting, since the `EditGoto` function automatically selects the content of each bookmark.

Then we write the table rows at the `table` bookmark:

```
data _null_;
  file table;
  set _data_for_bookmarks end=last;
  if _n_=1 then do;
    put 'Product' &tab 'Order Date'
        &tab 'Amount [USD]';
    end;
  put table_row;
run;
```

Since the entire table is now also sitting within its own bookmarked area, applying table formatting is also easy:

```
data _null_;
  file sas2word;
  put '[EditGoto.Destination="table"]';
  put '[FormatFont.Font="Verdana",
       .Points="18"]';
  put '[TextToTable.ConvertFrom="1",
       .InitialColWidth="Auto",
       .Format="9",.Apply="63"]';
run;
```

To right-align the last column though, we still need to step through the table from left to right with as many `NextCell` commands as necessary:

```
data _null_;
  file sas2word;
  put '[StartOfRow]';
  put '[NextCell]';
  put '[NextCell]';
  put '[TableSelectColumn]';
  put '[RightPara]';
  put '[StartOfDocument]';
run;
```

A limitation of Word is that one cannot define bookmarks in header and footer areas. So if we want to have the date-time stamp, we need to repeat the step from Section 5 that opens the header via the system-doublet `sas2word` and insert it that way…

Save and close the document:

```
data _null_;
  file sas2word;
  put '[FileSave]';
  put '[FileClose 2]';
run;
```

7 Introducing some SAS Macros

We've come quite far by now, and looking back at the code we've been writing, it should be apparent that using dozens of WordBasic functions within a Base SAS program to create customized Word output does not make the code very readable… Luckily, all those WordBasic functions with their many arguments lend themselves excellently to being wrapped inside a SAS macro with named parameters.

This has the advantage of hiding some of the underlying ugliness and making the SAS code more readable. The download package contains a file 'DDE Word Macros.sas' that bundles a series of SAS macros we have developed over time to perform oft recurring tasks on some Word output document. Each macro comes with a full description of usage and parameters, which we will not repeat here for reasons of available space. Instead, here's a list of them together with a brief description:

- `%closedoc` closes the active document.
- `%colwidth` adjusts the width of some column in some table.
- `%findrepl` finds and replaces formatted strings.
- `%fmtbkmk` applies formatting to the contents of a bookmark.
- `%fmtcol` applies formatting to some column in some table.
- `%fmtpar` applies formatting to some paragraph.
- `%fmtpict` applies formatting to some inserted graphic.
- `%fmtrow` applies formatting to some row in some table.
- `%fmttable` applies formatting to an entire table.
- `%opendoc` opens a specific document.
- `%putpict` inserts a picture from some file.
- `%rowheigh` adjusts the height of some row in some table.
- `%savedoc` saves the active document in some format.

Revisiting what we did in Section 6, and using some of these macros, the full code to generate a card statement from the template document would then *e.g.* look as follows:

```
%opendoc(
         path=c:\p16-28\templates,
         name=Card Report Template
        );

%savedoc(
         path=c:\p16-28\output,
         name=0701 Other Letter from
              Template,
         format=0
        );

data _data_for_bookmarks;
  set cards;
  where upcase(card_holder)='BILL
                             VIERGEVER';
  length table_row $ 300;
  table_row=trim(left(product))
            ||&tab
            ||trim(left(put(order_date,
                            date9.)))
            ||&tab
            ||trim(left(put(amount_usd,
                            10.2)));
run;

data _null_;
  reptdate=put(today(),worddatx.)||' at '
           ||put(time(),time.);
  call symput('reptdate',
              trim(left(reptdate)));
run;

filename card_hol dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  card_holder' notab;
filename card_typ dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  card_type' notab;
filename dept dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  dept' notab;
filename signatur dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  signature' notab;
filename table dde 'winword|c:\p16-28\
  output\0601 Letter from Template.doc!
  table' notab;
```

```
data _null_;
  set _data_for_bookmarks(obs=1);
  file card_hol;
  put card_holder;
  file card_typ;
  put card_type;
  file dept;
  put dept;
  file signatur;
  put signature;
run;

%fmtbkmk(
         bookmark=card_holder,
         font=Verdana,
         size=24,
         bold=yes
        );

%fmtbkmk(
         bookmark=card_type,
         font=Verdana,
         size=18
        );

%fmtbkmk(
         bookmark=dept,
         font=Verdana,
         size=18,
         italic=yes
        );

%fmtbkmk(
         bookmark=signature,
         font=Verdana,
         size=18
        );

data _null_;
  file table;
  set _data_for_bookmarks end=last;
  if _n_=1 then do;
    put 'Product' &tab 'Order Date'
        &tab 'Amount [USD]';
    end;
  put table_row;
run;

%fmtbkmk(
         bookmark=table,
         font=Verdana,
         size=18,
         txt2tabl=yes,
         tablefmt=9,
         tabapply=63
        );

%fmtcol(
        bookmark=table,
        right=yes,
        colnum=3
       );

data _null_;
  file sas2word;
  put '[ViewHeader]';
  put '[FormatFont.Font="Courier New",
       .Points="10",.Bold=1]';
  put '[Insert "Report created by the
       SAS System, ' "&reptdate" '"]';
  put '[CloseViewHeaderFooter]';
run;

%savedoc(
         path=c:\p16-28\output,
         name=0701 Other Letter from
              Template,
         format=0
        );

%closedoc;
```

This is not yet entirely free of WordBasic coding since we need to fix the header with the date-time stamp manually, but nothing prevents you from adding a `%header` macro to hide that bit from sight as well.

8 DDE-driven Word Mail-Merge

Now we have all the ingredients ready for a DDE-driven mail-merge application that outputs as many personalized credit card statement Word documents as there are subjects in our SAS data set.

Indeed, we merely need to wrap a SAS macro around the code listed in the previous section, to step through the data set subject by subject, and roll out a document for each one.

Due to space limitations we are unable to include the full macro-ized version of the code here, but just like the code presented in the previous sections, it is available in the download package. Enjoy!

9 In Closing

We hope to have provided sufficient examples of what can be done with a little DDE and WordBasic to get you started… As mentioned in the introduction though, there are other techniques available for producing Word output, and sometimes you'll find a combination of them to be the most cost-effective way of getting your Word documents to look exactly like you want them to.

Without diving into `proc template` territory *e.g.*, you can easily produce some default ODS RTF output, open the resulting RTF file via DDE, and wield some WordBasic commands or one of the provided macros to fine-tune the file and save it as a real Word document.

Over the past years, there have been some SAS/DDE/Word related postings on the online SAS-L forum. For examples of what can be done beyond what's discussed here, you could use the SAS-L archive search engine at www.sas-l.com Among other topics, you'll find code there to

- concatenate multiple pieces of ODS RTF output into a single Word document.
- read a table from a Word document and import into a SAS data set
- fix `proc tabulate` output through ODS RTF to allow for arbitrary superscripts and subscripts (handy for reports involving scientific units)

Finally, if you come up with cool new DDE tricks, or have suggestions for improvements to the macro collection, we'll be delighted to hear from you!

References

Roper, C. A. “Intelligently Launching Microsoft Excel from SAS, using SCL functions ported to Base SAS”. *Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference*, paper 97, 2000.

SAS Institute, “Technical Support Document #325 – The SAS System and DDE”. *http://ftp.sas.com/techsup/download/technote/ts325.pdf*, updated 1999.

Vyverman, K. “Using Dynamic Data Exchange to Export Your SAS Data to MS Excel — Against All ODS, Part I”. *Proceedings of the Twenty-Seventh Annual SAS Users Group International Conference*, paper 5, 2002.

Acknowledgments

Our thanks are due to Deb Cassidy, Section Chair of SUGI28 Advanced Tutorials, for making this paper possible. And of course to Steve Zamparelli and the team in Cary for the inordinate amount of patience!

Trademarks

Contacting the Authors

The authors welcome and encourage any questions, corrections, improvements, feedback, remarks, both on- and off-topic via e-mail at:

`sugi28paper@vyverman.com`