Paper 121-2013

# Reading Data from Microsoft® Word Documents: It's Easier Than You Think

John E. Bentley, Wells Fargo Bank

## ABSTRACT

SAS® provides the capability of reading data from a Microsoft Word document, and it's easy once you know how to do it. Using the FILENAME statement with the DDE engine makes it unnecessary to export to Excel or work with an XML map. This paper goes through the steps of having SAS read a Word document and shares a an example that demonstrates how easy it is and how you can leverage the capability. All levels of SAS users may find this paper useful.

## INTRODUCTION

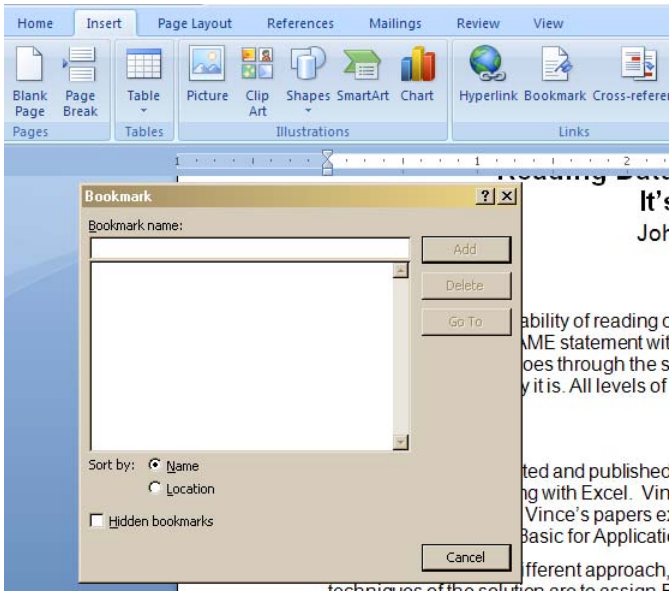
Much has been presented and published at SAS Conferences about SAS and Microsoft Office working together but most focus on SAS and Excel. Vince DelGabbo is especially prolific as an author and presenter when it comes to SAS and MS Office. His papers extensively cover using XML and ODS techniques, and other authors cover OLE DB, DDE, Visual Basic for Applications, the Excel LIBNAME Engine, and the SAS Add-In for Microsoft Office.

This paper looks at a different approach, one that the author found covered by only one other Conference paper back in 1998. The basic techniques of using SAS to read from (or write to) Word are to assign Bookmarks to specific locations in the document and then use FILENAME statements with the DDE device-type key to point to those bookmarks. With this done, the bookmark can be accessed as a data set with a single-variable or a tab-delimited text file. The focus of this paper is reading tables in a Word document and then using the data from those tables to generate SAS SQL code.

## WORD BOOKMARKS

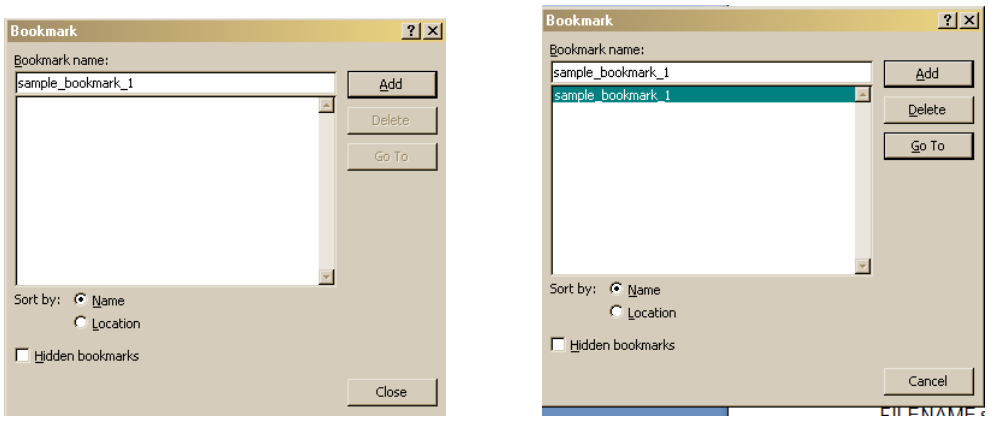
Bookmarks are a great way to quickly navigate large complex documents. They allow quick movement to specific sections of the document or items of information and do not affect the readability or printability of the document. In this author's experience they are very underutilized. Although they are intended to be navigational tools, they're also the key to letting SAS easily read from Word.

Adding a bookmark is basically the same in Word 2003, 2007, and 2010. First move the cursor to where you want the bookmark to be or highlight the section of text or table that you want bookmarked. Then select the 'Insert' tab on the tool Ribbon. On the Insert Ribbon towards the middle is the 'Links' group. Click on 'Bookmark' and the bookmark dialog box appears.




**Display 1. Word Bookmark Dialog Box**

Give the bookmark a name using only letters and numbers and the underscore to separate words. Click on 'Add' and the bookmark is written to the document and added to the permanent bookmark list. As soon as you click on 'Add' the bookmark dialog box disappears but clicking on 'Bookmark' again will verify that it was placed.




**Display 2. Word Bookmark Dialog Boxes, Completed**

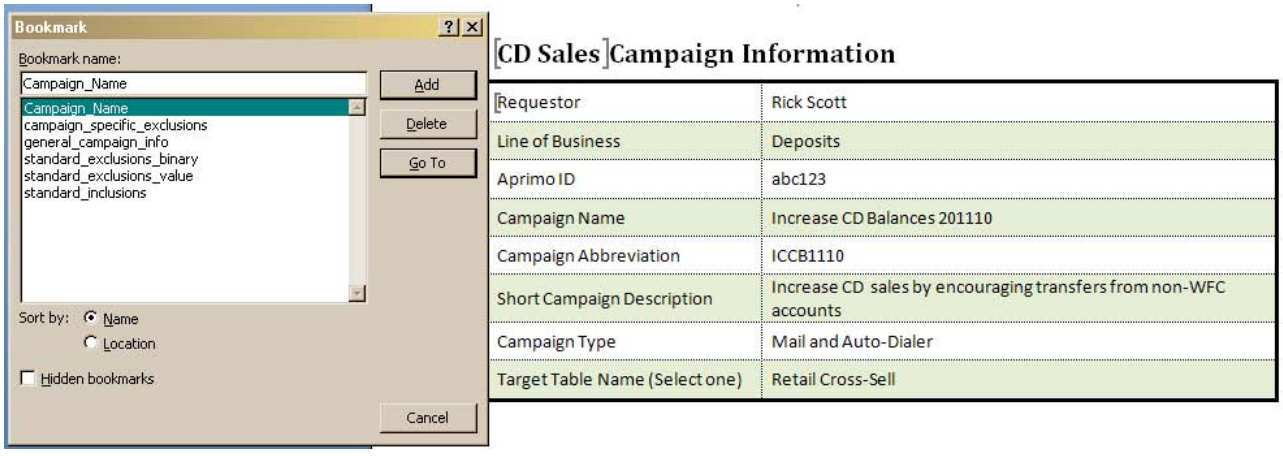
Bookmarks by default are hidden in the document but can be viewed with a few quick clicks to turn on that option. Click on the Office icon in the very upper left corner of the screen. At the bottom of the window choose 'Word Options' and then 'Advanced'. Scroll down to the 'Show document content' section and check the 'Show bookmarks' box. Click OK. Positional bookmarks then show up in the document as an `I`-beam character as in Excel Vince and sectional bookmarks delimit the section with square brackets, [ and ].

Removing a bookmark is as simple as opening the Bookmark dialog box, highlighting an entry in the list, and clicking on 'Delete'. Alternatively, deleting both words on either side of the bookmark deletes it. Copying text around the location of the bookmark does not copy the bookmark.

## SAS AND BOOKMARKS

Word bookmarks allow SAS to treat the bookmarked section as an input data source. Then we can use a FILENAME statement with the DDE device-type key to read the data source. Sounds simple, right? It is.

For our solution here, we use bookmarks to mark an entire tables but you can also bookmark individual words, a sentence or a paragraph. When the 'Show bookmarks' option is turn on, the bookmarked selection is enclosed in square brackets, [ ]. In this screenshot, the string 'CD Sales' is bookmarked as 'campaign_name' and the table itself is bookmarked as 'general_campaign_info'. Look closely and you can see the square bracket. In a bookmarked table, adding or deleting rows doesn't affect the bookmark. Deleting the table entirely of course deletes the bookmark too.




**Display 3. Word Bookmarks in a Requirement Document**

SAS will see a bookmark like 'Campaign_Name' as a data set with a single character variable, but a bookmarked sentence or paragraph can be parsed into multiple variables using the SCAN or SUBSTR function. A bookmarked table is read as a tab-delimited file with one observation per row and one variable per column. In the above table, there are eight observations each with two variables. Variable names are assigned by the INPUT statement used to read the data.

Here is the FILENAME statement and DATA Step to read the bookmarked table 'General_campaign_info'.

```
filename campInfo dde "winword|&_document!general_campaign_info";

data campInfo;
      informat item $35. value $100.;

      infile campInfo dsd dlm='09'x notab missover;
      input item value;

      %** Create macro variables we might need. ;
      if item='Requestor' then call symput('_projManager',trim(value));
      else if item='Line of Business' then call symput('_lob',trim(value));
      else if item='Aprimo ID' then call symput('_aprimoID',trim(value));
run;
```

We will cover the FILENAME syntax in the next section. In the DATA Step, the informat statement sets the maximum length to read for the input variables. Values a cell that wrap are still read as a single string, but strings longer than the length specified are truncated. For the INFILE statement,

- The `dsd` and `dlm=` options to tell SAS that the source data is delimited by tabs (hex value '09'x).
- `notab` causes SAS to ignore embedded tabs and blanks within the variables, so it will read each value as a character string of words or a number. This is important.
- `missover` option prevents the INPUT statement from going to the next input line if it does not find a value in the current line for each specified variable. Without it, the input statement would wrap around to find a value for each variable. With it, at the end of the line any variable without a value is set to missing. Missover is optional but its good insurance to use it.

## BASICS OF DDE—DOUBLETS AND TRIPLETS

Dynamic Data Exchange (DDE) allows Windows applications to exchange information. DDE establishes a client-server relationship that enables a client application (SAS) to communicate with a server application (Word). SAS is always the client and either requests data from server applications, sends data to server applications, or sends commands to server applications. You can use DDE with the DATA step, the SAS macro facility, SAS/AF applications, or any other part of SAS that requests or generates data. Many good Conference papers have been written about using DDE, mostly for communicating with Excel. Once you understand having SAS use DDE for one Windows application, the others work the same way.

A specially formed FILENAME statement is what SAS uses to control DDE and there are two forms for it—

```
FILENAME fileref DDE 'DDE-doublet';
FILENAME fileref DDE 'DDE-triplet' <options>;
```

In both forms, DDE is the keyword that tells SAS to use dynamic data exchange. DDE is a device-type name that enables SAS to read and write data from a device rather than a file and in concept performs the same function as an Engine name. PIPE, EMAIL, and FTP are other common device types.

A DDE-doublet is a *two*-part specification by which the client sends system-level commands to the server application.

```
filename word DDE 'Winword|System';
```

In this DDE-doublet, all commands issued through the fileref WORD are routed to the winword executable that controls Microsoft Word. Text inside the quote marks of a doublet or triplet is not case sensitive,and the pipe ( | ) here is a delimiter, not an 'or' symbol.

With a fileref containing a DDE-doublet we can do things like open and close Word documents. In this code example, we open a document name stored as a macro variable. After we do something, we close the active document and exit Word. It's not required that the DDE-doublet be referenced inside a DATA _NULL_ step but to make the code more readable and intuitive a best practice is to give it its own step.

```
data _null_;
  file word;
  put '[FileOpen.Name = "'"&_document"'"]';
run;

< do something >

data _null_;
  file word;
  put '[fileclose 2]';
  put '[fileexit]';
run;
```

A DDE-triplet is a three-part specification that opens a link to a specific location in an external data file. It takes the form of 'executable-file-name|data-file-location!Data-range-within-the-file'. A DDE-triplet requires that the data source be open before the fileref is used… that's why we use a doublet to first open the Word document and then close it when we're finished.

```
filename campInfo dde "Winword|&_document!general_campaign_info" notab;
```

In this DDE-triplet, Winword specifies that the MS Word executable will handle references to the fileref. The macro variable &_document following the pipe contains the fully-defined path and filename with extension of the Word document we want to read, and the location within the document—the bookmark—follows the bang ( ! ) delimiter. The notab options performs the same function as it does when used with an INFILE statement.

Each bookmark needs its own fileref, so a document with four bookmarks would need four FILENAME statements—

```
filename campInfo dde "winword|&_document!general_campaign_info" notab;
filename si dde "winword|&_document!standard_inclusions" notab;
filename seb dde "winword|&_document!standard_exclusions_binary" notab;
filename sev dde "winword|&_document!standard_exclusions_value" notab;
```

## LEVERAGING SAS AND WORD—A CODE GENERATOR FOR ORACLE QUERIES

In database marketing it's common for a campaign to have a Requirements Document and within that document should be a section specifying criteria for which customers to include or exclude from the population. Essentially, the inclusion/exclusion rules define the profile of the targeted customer base. Instructions for sampling and assigning test and control cases are in separate sections.

The selection criteria can quickly get complicated and custom coding can take a long time what with keying the code, testing, and debugging. Fat-fingering happens. Code generators dramatically speed up the process and reduce risk of miss-keying constraining values. In this section we look at a SAS code generator that uses on Word and Excel.

The Requirements Document is stored as a Word template and contains one general information table and three tables with specific selection criteria written in phrases the business users will understand—not pseudo-code. Each table is bookmarked, and each item in a table has a unique ID number composed of a table identifier and the row number.

To make comprehension and completion easier, one table presents rules written as Inclusion Criteria and two tables present rules for Exclusion Criteria. The user simply marks a 'Y' for the items to apply to their campaign and, if appropriate, a value that constrains application of the rule. In the examples below, Exclusion Criteria as broken into one table for yes or no answers and another table that requires constraining values. The three tables offer about 120 unique criteria, each referencing a specific target field in a Base Audience Table.

## *Inclusion Criteria – Standard.*

This criteria defines the campaign's starting population using the Targeting Table as the data source. Code for standard inclusions is generated from metadata. "and" is applied when multiple criteria are selected. All inclusion criteria must be met for a customer to be included in the starting population.

If no inclusion criteria is provided here then the starting population is the Targeting Table. Exclusion criteria will be applied to customers who meet the inclusion criteria.

| ID | Select | Target table field name | Include If. . . | Value(s) |
|---|---|---|---|---|
| IS001 | Y | cust_xsell_cnt | Customer owns at least X number of accounts that qualify as a cross sell by MDSS. | 3 |
| IS002 | | bus_acct_cnt | Customer owns at least X number of Business accounts. Specify X. | |
| IS003 | | addr_typ_cd | Customer address type code is a specific value. Specify the value(s) separated by comma. | |

## *Exclusion Criteria – Standard binary (yes/no)*

Exclusion criteria are applied to the starting population. Code for standard binary exclusions is generated from metadata. Conceptually, an "or" is applied when multiple criteria are selected. A record will be excluded if at least one exclusion criteria is met.

| ID | Select | Target table field name | Exclude If. . . |
|---|---|---|---|
| ESB001 | Y | dcsd_in | Customer is deceased. |
| ESB002 | Y | unmailable_in | Customer's address is flagged as unmailable per NCOA. |
| ESB003 | Y | blank_uak | Address key (ADDR_KEY) is blank. |
| ESB004 | | no_ssn_in | Customer does not have a Social Security Number on record. |

## *Exclusion Criteria – Standard Value-Driven*

Value-driven exclusion criteria are applied after the binary exclusion criteria. Code for the exclusions is generated from metadata. Conceptually, an "or" is applied when multiple criteria are selected. A record will be excluded if at least one exclusion criteria is met.

| ID | Select | Target table field name | Exclude if. . . | Values |
|---|---|---|---|---|
| ESV001 | | wfb_age_drv | Customer is less than a specific age. Specify the age. | |
| ESV002 | Y | wfb_age_drv | Customer is greater than a specific age. | 35 |
| ESV003 | | dlq_10_curr_days_qty_he | Customer has been 10-29 days delinquent at least X number of times on a Home Equity account. Specify X. | |

**Display 4. Inclusion and Exclusion Criteria Tables in a Word Document**

Using the template a campaign-specific Requirements Document is produced, reviewed, and approved. When it's time to generate the query to pull the population, SAS reads the document and creates a data set for each bookmarked table. Here is the code to read the Inclusion Criteria table; the code for the Exclusion tables is very similar.

```
filename si dde "winword|&_document!standard_inclusions" notab;

data standardInclusions;
   informat id $10. selected $1. targetTabFieldName $30. businessRule $100.
        value $25.;
   length outputLabel $100;

   infile si dlm='09'x notab dsd missover;
   input id selected targetTabFieldName businessRule value;

   %** Keep the rows in the table that contain selected inclusions.
   %** Ignore empty rows at bottom.;
   if anyDigit(id) and businessRule ne '' and selected ne '';

   %** Each of the Inclusion Criteria require a constraining value. ;
   %** Check for missing data. ;
   if value='' then do;
      put ************************************************************************';
      put 'Business Rule= ' businessRule ;
      put 'VALUE is required.  Program is Aborting.';
      abort;
      put ************************************************************************';
   end;

   %** Keep only first sentence of business rule. ;
   businessRule=scan(businessRule,1,'.');

   %** Create a label to use in Output headers. ;
   if length(scan(businessRule,1,'.')) < 60 then
      outputLabel=catt(id,': ',targetTabFieldName,'--',businessRule);
   else
      outputLabel=catt(id,': ',targetTabFieldName);
run;
```

We now have data sets with the criteria identifier, selected yes or no, the source field name, business rule description, and the constraining value.

Next we need to read an Excel worksheet that has the actual SQL code for each of the criteria. Excel was selected because it's more database-like, easy to maintain, and a sheet in the workbook serves as a change log. Each of the criteria selection tables in the Requirements Document has an associated worksheet in the Rules .xls file, and each of the criteria IDs has a row in the worksheet. Here's a sample of the contents of the Inclusion Rules worksheet.

- The ID column links to the Requirements Document criteria selection tables and the data sets derived from them.
- The column named Data_Type determines whether or not the constraining value passed from the criteria selection data set is put between quote marks.
- A partial comparison statement is in the column labeled SQL_Rule Later we append the value from the criteria selection data set to the SQL_Rule to make a valid statement.

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | ID | Target_table_field_name | Business Name | if_condition | Data_Type | SQL_Rule |
| 2 | IS001 | CUST_XSELL_CT | Customer Cross-Sell Count | Customer owns at least X number of accounts that qualify as a cross sell by MDSS. | num | CUST_XSELL_CT >= |
| 3 | IS002 | BUS_ACCT_MDSS_CT | Business Account Count | Customer owns at least X number of Business accounts per MDSS. Specify X. | num | BUS_ACCT_MDSS_CT >= |
| 4 | IS003 | ADDR_TYP_CD | Address Type Code | Customer address type code is a specific value. Specify the value(s) seperated by comma. | char | ADDR_TYP_CD IN |
| 5 | IS004 | HI_VAL_TYP_CD | High-Value Type Code | Customer is assigned to a specific High Value category(s). Specify the category(s) to retain seperated by comma. | char | HI_VAL_TYP_CD IN |

**Display 5. The Excel Rules Master Workbook, Inclusion Worksheet**

This next code reads the Inclusion Rules worksheet in the Excel Rules Master workbook. A LIBNAME statement with the Excel Engine lets us reference multiple worksheets from one workbook, which the code generator needs to do because there is one sheet per Word table. After reading all three worksheets we combine the data into a single rulesBase data set.

```
libname xls excel "&_rulesExcelWorkBook";

%** Create a dataset containing inclusion rules. ;
data inclusionRules;
   attrib id length=$10 informat=$10. format=$10.;
   attrib target_table_field_name length=$30 informat=$30. format=$30.;
   attrib business_name length=$75 informat=$75. format=$75.;
   attrib if_condition length=$150. informat=$150. format=$150.;
   attrib data_type length=$4 informat=$4. format=$4. ;
   attrib sql_rule length=$100 informat=$100. format=$100.;
   attrib audit_type_code length=8 informat=3. format=3.;

   set xls.'inclusions$'n;

   if sql_rule ne '';   %** Dont read blank rows.

   keep id target_table_field_name business_name data_type sql_rule
        audit_type_code;
   rename data_type=dataType;
run;
```

So now we have five datasets.

- One general information data set from which we create macro variables;
- Three criteria selection data sets from the Requirements Document; and
- One data set from the Excel Rules workbook that contains a partial SQL statement for each of the criteria.

Now the datasets are sorted and merged by record `ID`. In the code below, the variable `whereCriteria` holds a value that appends the constraining value of the requirement to the partial comparision string from the rulesBase. The dataset `notInRulesBase` serves as a check to see that new criteria wasn't added to the Requirements Document without the Excel Rules base being updated. A later check not shown compares the count of the number of selected criteria in the Requirements Document with the number of records in the `selected` data set. If the counts don't match, the process aborts.

```
data selected notInRulesBase;
  length value $25;
  merge rulesBase(in=inBase keep=id sql_rule dataType)
    standardInclusions(in=selectedSI keep=id value rename=(value=valueSI))
    binaryStandardExclusions(in=selectedBSE keep=id)
    valueStandardExclusions(in=selectedVSE keep=id value rename=(value=valueVSE));
  by id;
  if inBase and (selectedSI or selectedBSE or selectedVSE);

  %** Assign Value based on data source. ;
  if selectedSI then value=valueSI;
  else if selectedBSE then value='';
  else if selectedVSE then value=valueVSE;

  %** Put quotes around list elements and character and date variables. ;
  if lowcase(dataType) in ('char' 'date') and value ne '' then do;
   if index(sql_rule,'in') then do;
   needsParens=1;
    comma=indexc(value,',');
    if comma=0 then value="'"||trim(value)||"'";
    else value="'"||tranwrd(compress(value),",","',"')||"'";
  end;
  else do;
   value="'"||value||"'";
  end;
```

```
  end;

  %** Derive where criteria. ;
  if value='' then whereCriteria=sql_rule;
  else if needsParens = 1 then do;
   whereCriteria=catx(' ',sql_rule,'(',value,')');
  end;
  else whereCriteria=catx(' ',sql_rule,value);

  %** Create two variables for sorting so code will be in Reqs Doc order. ;
  %** Now sorted by ID, the order of the rules will be ascending--ESB, ESV, IS. ;
  %** So we have to work with it a bit. ;
  if substr(id,3,1) in ('B' 'V') then do;
   if substr(id,1,3)='ESV' then charID='ESA';
   else charID=substr(id,1,3);
   numID=substr(id,4);
  end;
  else do;
   charID=substr(id,1,2);
   numID=substr(id,3);
  end;

  %** output to notInRulesBase if the rule isnt in Rules Master. ;
  if inBase and (selectedSI or selectedBSE or selectedVSE) then output selected;
  else if selectedSI or selectedBSE or selectedVSE then output notInRulesBase;
run;
```

Finally we generate a SAS SQL Passs-Through query that is written to a text file. Not shown is where we derive a macro value named `&_len`. That's the length of the longest `whereCriteria` in the dataset. We use it to line up a comment for each of the where criteria in the generated code showing the selection ID associated with it. With that we can tie each item back to the Requirements Document and quickly verify accuracy and completeness.

```
filename psc4106 "&_localDir\sql_select_query_&_aprimoID..txt";

data _null_; set selected end=eof;
  file psc4106;

  len=&_len;

  if _n_=1 then
    put @5 "CREATE TABLE &_aprimoID._BASE_POPULATION AS"
      / @10 'SELECT *    '
      / @10 'FROM CONNECTION TO ORACLE'
      / @15 '(SELECT * '
      / @15 "FROM &_dataSource"
      / @15 'WHERE';

  %** Add the Rule Identifier as a comment next to each criteria. ;
  if _n_=1 then put @21 whereCriteria @len+17+10 '/* ' id ' */';
  else put @17 'and ' whereCriteria   @len+17+10 '/* ' id ' */';

  if eof then
    put @10 ') ;' ;
run;
```

And here's what the text file looks like. Many of the criteria use binary values where 1=Yes and 0=No. So criteria such as dcsd_in=0 translates as 'deceased indicator is no'.

```
 CREATE TABLE abc123_BASE_POPULATION AS
  SELECT *
  FROM CONNECTION TO ORACLE
  (SELECT *
   FROM ORCDB.CUST_TRGTING_D143
   WHERE
         CUST_XSELL_CT >= 3                /* IS001  */
```

```
      and HI_VAL_TYP_CD IN ('HV02','HV03')  /* IS004  */
      and STATE_CD IN ('CA','FL','TX')      /* IS006  */
      and OPEN_ACCT_CT_HOGAN >= 2           /* IS012  */
      and BUS_DEP_PRIM_CT >= 1              /* IS027  */
      and RET_DEP_IBT_PRIM_CT >= 1          /* IS037  */
      and RET_DEP_MRA_PRIM_CT >= 1          /* IS045  */
      and dcsd_in=0                         /* ESB001  */
      and unmailable_in=0                   /* ESB002  */
      and blank_uak=0                       /* ESB003  */
      and bad_name=0                        /* ESB005  */
      and bad_addr=0                        /* ESB006  */
      and minor=0                           /* ESB007  */
      and ncoa_suppr=0                      /* ESB010  */
      and epss_no_solicit_in=0              /* ESB011  */
      and prison_dns_in=0                   /* ESB012  */
      and comb_do_not_mail_in=0             /* ESB013  */
      and wfb_do_not_mail_in=0              /* ESB014  */
      and epss_do_not_email_in=0            /* ESB015  */
      and pcs_in=0                          /* ESB021  */
      and priv_bnk_in=0                     /* ESB022  */
      and trust_in=0                        /* ESB023  */
      and age_drv >= 35                     /* ESV002  */
   ) ;
```

Now that we have the text file, we copy it to the server where it will run against the Oracle database. A Cron job monitors the remote directory and periodically runs the queries that land there.

```
%** Copy needed macro variables to the server.;
%syslput _aprimoID=&_aprimoID;
%syslput _remoteDir=%bquote(&remoteDir);

%** Move the file to the server. ;
%** infile fileref created when text file was created. ;

RSUBMIT kwood;
  filename psc4106  "&_remoteDir/sql_select_query_&_aprimoID..txt";

  proc upload
    infile=psc4106
    outfile=psc4106;
  run;

  filename psc4106 clear;
ENDRSUBMIT;
```

To run the query we simply %INCLUDE it in a SAS program. Not shown is code that parses the file name and creates the macro variable &_aprimoID.

```
proc sql threads noprint feedback;
   connect to oracle (path='mktgprod' user="&_dbUser" pass="&_dbUserPwd"
                      readbuff=5000);
   %include &_remoteDir/sql_select_query_&_aprimoID..txt" / source2;
quit;
```

## CONCLUSION

SAS and Word are a powerful combination. This paper showed how easily you can use SAS and DDE to read a bookmarked Word document and presented a code generator that exploits that capability. Now that you know how to read from Word, with a little experimentation you can reverse engineer the technique to write to it.

## REFERENCES, RESOURCES, AND RECOMMENDED READING


- SAS Institute. "Using Dynamic Data Exchange under Windows". Available at http://support.sas.com/documentation/cdl/en/hostwin/63285/HTML/default/viewer.htm#dde.htm
- DelGabbo, Vince. 2003. "A Beginner's Guide to Incorporating SAS® Output in Microsoft® Office Applications." *Proceedings of the SAS Users Group International Conference 28*. Cary NC: SAS Institute.
- Gilmore, Jodie. 1998. "Using Dynamic Data Exchange with Microsoft Word." *Proceedings of the SAS Users Group International Conference 22*. Cary NC: SAS Institute.
- SAS Institute. 2010. *SAS® 9.2 Companion for Windows, Second Edition. Cary NC: SAS Institute.*
- Zhou, Jay. 2009. "Importing Data from Microsoft Word into SAS®." *Proceedings of the 2009 PHARMASUG Conference*. Cary NC: SAS Institute.


## ACKNOWLEDGMENTS

Thanks to Jodie Gilmore for writing her DDE with Word paper back in 1998. It was a big help getting me started. And thanks also to my enlightened bosses over the years who had the vision and foresight to send me to SAS training and conferences. .

## CONTACT INFORMATION

John E. Bentley
Wells Fargo Bank
Charlotte NC 28226
john.bentley@wellsfargo.com