Using Dynamic Data Exchange to Pour SAS® Data into Microsoft® Excel®

Koen Vyverman, Fidelity Investments Luxembourg

Abstract

Version 6.12 of the SAS® System under Windows® provides a number of methods to export data from a SAS® data set into a Microsoft® Excel® worksheet. To name but a few: the point-and-click SAS® Export Wizard is satisfactory if one only needs it occasionally and if one doesn't care too much about how the data will be formatted in the resulting worksheet. Improved results may be obtained by taking the programming approach and tinkering with `proc dbload`*, although one will still run into limitations easily. In contrast, the use of Dynamic Data Exchange (DDE) allows full programmatic control over where exactly in the worksheet data are being inserted, and how the worksheet-cells are formatted. The* `%sastoxl` *macro pours SAS® data into either a new blank Excel® workbook, or into an existing one. Furthermore, it demonstrates some key DDE functionality by allowing the user to specify the name of the workbook-worksheet that is to receive the data, choose the upper-left cell of the data-block, and decide whether to apply a certain set of common formatting options like font, font-size, column-width auto-fit, and freeze-panes for the label row.*

1 Introduction

One of the great strengths of the SAS® System is without any doubt its ability to access data on almost any imaginable platform, and in a multitude of file formats. However, depending on the particular platform and file format, reading and writing external data may prove to be anything between dead easy and … well, a pain! As is often the case though within the SAS® System, one will generally be offered the choice between a number of possible strategies to achieve a certain goal. Some solutions will work with only the plain Base SAS® product licensed while others will require one or more extra SAS® Software modules.

In an excellent review paper, Kuligowski (1999) discusses the various methods that are available for importing external data into the SAS® System. To name but a few: the SAS® Import Wizard, CSV-files, Dynamic Data Exchange, the various SAS/ACCESS® engines, ODBC, … Some of these are platform-dependent, some will only allow reading data, others also provide the functionality to write data externally, or have a counterpart that does so.

In this paper we shall focus on Dynamic Data Exchange, hereafter referred to as DDE. A *de facto* consequence of this choice is that the operating systems under consideration are limited to IBM®'s OS/2®, and Microsoft®'s Windows® – 95 and up, as well as NT. All the code in what follows has been developed and tested under Microsoft® Windows95®. More specifically, we will investigate DDE as a method for exporting data from the 6.12 release of the SAS® System to the Microsoft® Excel® version included in the Microsoft® Office97® suite of applications.

2 %SASTOXL

Among the various methods available for exporting SAS® data to Excel®, DDE is certainly not the easiest one. For a good overview, see Mumma (????), who describes and compares a number of methods for data transfer between SAS® and Excel® specifically. Without resolving to third party software solutions, one basically has the choice between the following: intermediary file formats, SAS/ACCESS®, and DDE.

The use of intermediary file formats like CSV or HTML is of course a comparatively simple solution, but it does not lend itself greatly to automation. Using `proc dbload` from the SAS/ACCESS® product is a code-based solution and hence easily automated, but its syntax offers only limited control over the actual output. DDE on the other hand allows full programmatic control over the created worksheet. Most of the actions that one can perform while interactively using Excel®, *i.e.* through use of mouse-clicks and the menu-commands, can be done from within a SAS® program by using DDE to pull the strings of Excel®.

So, how to harness the power of DDE without being inconvenienced by its rather complex syntax? SAS® macro language provides the answer. In this paper we will examine the inner workings of a custom made macro `%sastoxl`. The macro depends on a number of parameters that provide a certain measure of user-control over the data flow to Excel®, as well as the formatting of the worksheet cells. Some of the functionality that is offered by `%sastoxl` is only possible through the use of the DDE formalism. The full macro specification is as follows:

```
%macro sastoxl(
               libin=,
               dsin=,
               cell1row=1,
               cell1col=1,
               nrows=,
               ncols=,
               tmplpath=,
               tmplname=,
               sheet=Sheet1,
               savepath=c:\temp,
               savename=SASTOXL Output,
               stdfmtng=
               );
```

The meaning and usage of the macro parameters is thus:

`libin` (required): the name of the SAS® library where the input data set lives.

`dsin` (required): the name of the input SAS® data set.

`cell1row` (optional): the row number of the first cell of the worksheet where data should be inserted, *i.e.* the upper left corner of the data block. Default value = 1

`cell1col` (optional): the column number of the first cell of the worksheet where data should be inserted, *i.e.* the upper left corner of the data block. Default value = 1

`nrows` (optional): the first `nrows` observations of the input data set will be inserted into the worksheet. If no value is specified, an attempt will be made to insert all observations. Needless to say, this number needs to be smaller than the maximal number of rows supported by Excel®.

`ncols` (optional): the first `ncols` variables of the input data set will be inserted into the worksheet. If no value is specified, an attempt will be made to insert all variables. Again, this number needs to be smaller than the maximal number of columns supported by Excel®.

`tmplpath` (optional): the full path to the directory where the Excel® workbook resides into which the data need to be written. To be used in conjunction with `tmplname`. If no value is specified, a standard new workbook and worksheet will be created. Do *not* end the path with a backslash character.

`tmplname` (optional): the filename of the Excel® workbook in the directory specified by `tmplpath` into which the data need to be written. To be used in conjunction with `tmplpath`. If no value is specified, a standard new workbook and worksheet will be created.

`sheet` (optional): the name of the worksheet within the Excel®

workbook – either a new blank one, or the one specified by `tmplpath` and `tmplname` – into which the data will be written. Default value = Sheet1 (being the name of the first worksheet in any new blank Excel® workbook)

`savepath` (optional): the full path to the directory where the finalised Excel® workbook needs to be saved. May be used independently from `savename`. Do *not* end the path with a backslash character. Default value = c:\temp

`savename` (optional): the filename by which the finalised Excel® workbook should be saved in the directory specified by `savepath`. May be used independently from `savepath`. Default value = SASTOXL Output

`stdfmtng` (optional): the standard formatting flag. Is turned off by default. Set it to a value of 1 to turn standard formatting on. Doing so will apply a common set of formatting options to the inserted data. The font of the label row will be emboldened. The font of the entire inserted range of data will be set to Courier. The column width will be set to best fit. Furthermore, freeze panes will be turned on to keep the label row visible while scrolling through the worksheet.

3 Examples

Before delving into the technical details on DDE and the macro code, let us consider a few examples of how `%sastoxl` may be used.

Suppose the entire data set `work.somedata` needs to be exported to an Excel® worksheet, and saved as 'c:\temp\Some Data.xls' with the standard formatting turned on. To accomplish this, we submit the following macro call:

```
%sastoxl(
         libin=work,
         dsin=somedata,
         savepath=c:\temp,
         savename=Some Data,
         stdfmtng=1
        );
```

Suppose only the first 125 rows of the data set `work.somedata` need to be exported to an existing Excel® worksheet, and saved as 'c:\temp\Some More Data.xls'. Suppose the full path and name of the Excel® workbook into which the data need to be inserted is 'n:\sasok\data\blank dox\Serious Fun.xls', and the block of data is wanted at row 37, column 3 of a worksheet named 'Stuff from SAS'. To accomplish this, we submit the following macro call:

```
%sastoxl(
         libin=work,
         dsin=somedata,
         cell1row=37,
         cell1col=3,
         nrows=125,
         tmplpath=n:\sasok\data\blank dox,
         tmplname=Serious Fun,
         sheet=Stuff from SAS,
         savepath=c:\temp,
         savename=Some More Data,
         stdfmtng=1
        );
```

4 DDE to Excel® Basics

Most of what one really needs to know in order to start doing fancy things with DDE is detailed in one of the Institute's technical support documents, TS325 (SAS Institute, 1999). However, if one decides to focus on the specific topic of conversation between the SAS® System and Excel®, it is advisable to go for Bodt (1996) and Schreier (1998) as an initial read. Both provide ample code examples, and Schreier in particular discusses some of the more advanced intricacies related to DDE-ing between the SAS® System and Excel®…

No harm though in listing the DDE basics necessary to understand what goes on in `%sastoxl`. First of all, what is DDE? Dynamic Data Exchange is a communication method available on the Windows® and OS/2® platforms. On these platforms, applications that are DDE-compliant can talk to each other in a client/server fashion for accessing and transferring data, as well as for executing application-specific commands. Applications that are DDE-compliant to varying degrees include Borland® Paradox®, Lotus® 1-2-3®, Microsoft® Access®, Microsoft® Excel®, Microsoft® FoxPro®, Microsoft® Word®, and Corel® Quattro Pro®.

The SAS® System for Windows® and OS/2® became DDE-compliant as of the 6.08 release, with the severe limitation that in the client/server paradigm, the SAS® System can only act as a client. This means that it is possible to initiate a connection from the SAS® System (client) to another DDE-compliant application (server), and tell the other application what to do by sending it intelligible commands through the DDE-connection. On the other hand, a DDE-compliant application cannot start up a DDE-link (as a client) to the SAS® System (server) and make it dance to its tunes.

A prerequisite for initiating a DDE-conversation is that the server application, in our case Excel®, is up and running. As we shall see later on, `%sastoxl` contains a bit of logic for determining whether an instance of Excel® is running or not. When not, it will launch Excel® before proceeding.

If an instance of Excel® is running, how does one initiate a DDE-connection? This is done by using the `filename` statement with `DDE` as the device-type keyword. Such a `filename` statement can take two basic forms:

```
filename <fileref> dde '<server app>
                              |<topic>!<item>';
```

and

```
filename <fileref> dde '<server app>|system';
```

In the first of these forms, the expression `<server app>|<topic>!<item>` is known as the infamous DDE-triplet. It simply tells the `filename` statement which application to talk to, which document to work with, and which part of the document to read/write data from/to. In the context of the SAS® System talking to Excel®, this amounts to the following:

```
<server app> = Excel
<topic> = [<workbook.xls>]<worksheet>
<topic> = a cell-range like RiCj:RmCn
```

A concrete example? Suppose we have a workbook 'Some Data.xls' open in the Excel® application. It is a container document holding several sheets, one of which is a worksheet named 'Stuff from SAS'. To allow the SAS® System to communicate with the cell-range R1C1 to R5C5, we need the following statement:

```
filename sasstuff dde 'excel|[Some Data.xls]
                       Stuff from SAS!r1c1:r5c5';
```

The `fileref` defined in this manner can then be used as usual in a data step, in conjunction with the `file`, `infile`, `put` and `input` statements, depending on whether one wants to write to or read from the specified range off cells.

The second form of the `filename` statement utilises the special value `system` as the DDE-topic, and no DDE-item at all. Let us call it the DDE-doublet. A real-life example in our case will therefore look like the following:

```
filename xlsystem dde 'excel|system';
```

The functionality of a DDE-doublet differs profoundly from the DDE-triplet. Whereas a DDE-triplet opens a link to a specific cell-range in an Excel® workbook/worksheet, the doublet will allow the client application to send commands to the server application as if they were entered interactively via mouse-clicks or menu- and keyboard-commands. Obviously, commands need to be sent in a language that the server application is able to understand. In the case of Excel® as a server, commands can be sent in the Excel® version 4.0 Macro Language. More recent versions of Excel® use Visual Basic for Applications (VBA) as a macro coding language. The current Excel® macro facility is nicely backward compatible though and is still able to interpret the old-style v4.0 commands.

Usually, *the* problem for the DDE-novice lies in finding out exactly which commands are available in this old Excel® v4.0 Macro Language, and what their precise syntax is. There are two solutions to overcome this problem: one could organise a hunt through the attic, or cellar, or

wherever it is that old software manuals tend to accumulate, and look for a copy of the out of print Excel® v4.0 manual titled "Function Reference". A perhaps less dusty alternative is to point a web-browser to the Microsoft® site, and search the technical support area for a file named 'macrofun.exe'. Downloading this file and executing it will install the "Microsoft Excel 4.0 Macro Functions" help file Macrofun.hlp' somewhere on a local disk. It contains largely the same information as the manual mentioned above, but some may find the book more useable.

Since this v4.0 Macro Language was replaced by VBA in more recent versions of Excel®, it should be obvious that *e.g.* the Office 97® version of Excel® offers certain functionality that is simply not covered by the commands that were available in v4.0. Consider the 'autofilter' command from the 'data' menu. There is no v4.0 Macro Language function that will toggle the autofilter on and off. This does not mean that it cannot be done via DDE. One could create and store a VBA macro that toggles the autofilter. Since there actually *is* a v4.0 Macro Language function to run a stored macro, this VBA macro can then be called via DDE, with the desired effect. Running stored macros through DDE is outside the scope of this paper. The interested reader may want to have a look at a recent paper by Stetz (2000) who considers the problem in the context of Microsoft® Word®. The technique applies equally to Excel® however…

Finally, the necessity of using the `notab` option on the DDE `filename` statement when exporting data to Excel® has been demonstrated elsewhere with great clarity (Schreier, 1998). Suffice it to say that it prevents Excel® from interpreting each and every blank character in the data-stream as a signal to move to the next cell.

5 Macro Design Methodology

A few notes concerning general macro design. The `%sastoxl` macro is reproduced in its entirety at the end of this document. A first cursory glance at the code will reveal at the very least that the code is amply interlaced with comments. Ideally one should be able to just read these comments and understand what happens in subsequent blocks of code. The following sections here will only expand on some of the important issues without going through the macro-code step by step. Please note that for the sake of this paper, some sections of the code are rendered a bit more verbose and smeared out than is technically necessary or even aesthetically desirable. The intended effect being ease of understanding and improved legibility rather than programmatic efficiency and elegance…

Since `%sastoxl` was expressly written for the purpose of being called from within any executing SAS® code, all its macro variables are defined within the local symbol table in order to prevent conflicts with extant macro variables in the running SAS® session. Moreover, each of the local macro variables is given an initial value – which can be a _null_ value – just to make absolutely sure that they all have the proper value.

As `%sastoxl` will be called upon in a plethora of potentially complex programs, just for the purpose of dumping SAS® data into an Excel® format, it is undesirable to have it cause hundreds of lines in the log window interrupting the flow of the log of the host-program. If we are confident that `%sastoxl` functions properly, we can therefore safely turn off those SAS® System options that generate most of the log content:

```
options
  nonotes
  nosource
  nosource2
  nomlogic
  nosymbolgen
  noprint;
```

Before doing so however, the current values of these options are captured in a series of macro variables that allow restoring them to their original values at the end of `%sastoxl`.

Also still part of the overall macro design is the value-checking of the required parameters. A bit of conditional logic will ensure that `%sastoxl` exits peacefully when any of its required parameters is missing. An appropriate error message is written to the log.

A final word on macro coding: one thing that may seem odd at first is the use of the `%str` macro function to mask empty strings in `%put` statements, as in

```
%put ER%str()ROR: The SASTOXL macro bombed!;
```

The reason for doing this is to avoid occurrences of the word 'ERROR' in the SAS® System log unless there really *is* an error! Making it a habit to store macros in autocall libraries evades this issue, but as soon as a macro source code gets `%include`-d by its host-code, not using `%str()` to interrupt strings like 'ERROR', 'NOTE', 'WARNING' and the likes may make scanning the log a tedious exercise.

6 Preamble

The first thing that happens in what can be considered the actual macro-body, is filling in the blanks. Some of the optional macro parameters are given a default value. Also, some of these may have inadvertently lost their default values by being mentioned in the macro call but failing to get a value. This may happen *e.g.* when the host-program generates a `%sastoxl` call dynamically and – due to errors – sets a parameter to a null value. Should anything like this happen, the parameter in question is reset to its default value and the macro continues executing after writing a note to the log.

Later on, when we will write the `filename` statements with the appropriate DDE-triplets defining the cell-ranges for the variable labels as well as for the actual data, we will add a logical record length option `lrecl` to ensure that the write-buffer is large enough to accommodate all the formatted as well as unformatted values to be pushed out to a single worksheet row. Since `%sastoxl` was designed for release 6.12 of the SAS® System, allowing 200 bytes for every output variable will do the trick. Calculating the `lrecl` value terminates a section of the macro where various items are being initialised.

As mentioned in section 4, opening a DDE-link requires the server application to be running. If no instance of Excel® is running on the PC, the code

```
filename sas2xl dde 'excel|system';
data _null_;
  file sas2xl;
run;
```

will obviously fail to access Excel® (even if it is for doing nothing really) and cause the automatic macro variable `syserr` to be set to a non-zero value. `syserr` can then be used to fire up the Excel® application.

Note that the Excel® application is started by issuing an `x`-command with the absolute path of the executable file hard-coded in the macro source. As there is generally no certainty that on different PCs software gets installed in the exact same locations, the path should be checked and if necessary adjusted. To ease the pain of doing so, the construct with the single and double quotes allows the use of long directory names in the `x`-command, so at least there is no need to figure out the DOS path.

Subsequently, we can open a doublet-style DDE-link to Excel® for the sending of system commands in the v4.0 Macro Language:

```
filename sas2xl dde 'excel|system';
```

These system commands can then simply be passed on to Excel® directly from a data step with the use of `put` statements, as in

```
data _null_;
  file sas2xl;
  put '...<v4.0 ML commands>...';
run;
```

It was decided to forego elegance and make use of dummy ASCII-files into which SAS® code is written dynamically before being executed with an `%include` statement:

```
filename execit1 'c:\temp\execit1.txt';
etc...
```

The reason for doing things this way is to be sought in the fact that we are doing DDE in a SAS® macro environment. In principle, that should cause no trouble. In reality, errors do occur, and since the system commands contain quite a lot of SAS® macro variable references, debugging gets nigh impossible if one cannot see what exactly Excel® was told to do! Writing the system commands to a dummy ASCII-file

has the advantage that all macro references will be resolved, and if things don't work out as expected it is possible to check afterwards to see what has happened.

An example: if `%sastoxl` gets called with values for the `tmplpath` and `tmplname` parameters, the template document needs to be opened. The data step that writes the to-be-executed code to our first dummy file `execit1` may look a bit convoluted:

```
data _null_;
  file execit1;
  put 'data _null_;';
  put '  file sas2xl;';
  put "  put '[error(false)]';";
  put "  put '[open(" '"' "&tmplpath" '\'
                    "&tmplname" '")]' "';";
  put 'run;';
run;
```

However, looking into the file execit1.txt, we will see the exact system commands that were passed on to Excel, *e.g.*:

```
data _null_;
  file sas2xl;
  put '[error(false)]';
  put '[open("c:\templates\Monthly")]';
run;
```

By the way, the command `[error(false)]` that appears in all output to the doublet-style DDE fileref `sas2xl` helps to prevent Excel® from interrupting the autonomous program-flow by prompting for user-input to typical Microsoft® questions like 'Are you sure you want to do this?'

7 Data Feed

The section of the `%sastoxl` source code that concerns getting the data across to Excel® starts with the gathering of a little meta-data about the input data set. A small data set `work.____meta` will contain important information like a variable's label, type and its position on the data set.

We distinguish two cell-ranges. The first, actually a one-row range, will represent the block of cells to which the variable labels will be written. If a variable has no label, its name will be used instead. The second range, the size of which depends on how many observations are to be output, represents the block of cells that will receive the actual data. To define a cell-range it suffices to identify its upper-left and its lower-right cells. Hence, eight macro variables are initialised with the row- and column-numbers of those four corner-cells.

When inserting data into a worksheet-cell, Excel® will try to be clever and impose one of its own formats on the incoming data. The exact format it chooses depends on what it assumes the data to represent. This semantic interference is a real pain. `%sastoxl` makes a compromise by pre-formatting all cell-ranges that will receive character-variables from the SAS® System with Excel®'s internal free-form "@" format. The cell-ranges that will receive numerical data from the SAS® side are left alone. This strategy has its pro and cons and is open for adaptation since mileages are known to vary… As an example, consider a SAS® date variable on the input data set. If it has no SAS® date-format associated with it, Excel® will treat it as a numeric integer value. Which is what one would expect, but it is arguably very practical since the origin of the SAS® System's internal calendar differs from Excel®'s. On the other hand, if the variable has, say, the `date9.` format on the input data set, Excel® will recognise the formatted string as a date and treat it as such. This is a consequence of the fact that DDE tends to pass formatted values on to the server application. Hence, if one wants to make sure that actual values are being transferred rather than formatted values, it is a good idea to remove all formats form the input data set before feeding it to `%sastoxl`. A simple data step will do:

```
data ...;
  set ...;
  format _all_;
run;
```

After pre-formatting, two triplet-style DDE-filerefs are defined, one for the range of variable labels (`xllabels`), one for the actual data (`xlsheet`.) Thereafter, writing the labels and the data is pretty straightforward. Note however the use of the double trailing @ to get all the labels output on one single row while stepping through the `work.____meta` data set. The use of the `notab` option on the `filename` statements (*supra*) necessitates the explicit coding of a tab-character between successive variables on a `put`-statement. This is done in the form of the `'09'x` hex string which is stored in a macro variable for convenience.

8 Formatting the Cells

After all labels and data have been inserted into the Excel® worksheet, what remains to be done is the formatting if such is requested. An example of what commands get sent to Excel via the execit4.txt:

```
data _null_;
  file sas2xl;
  put '[error(false)]';
  put '[workbook.activate("Stuff from SAS",
                                    false)]';
  put '[Select("r1c1:r31349c4")]';
  put '[Format.Font("Courier New",10,false,
       false,false,false,0,false,false)]';
  put '[Select("r1c1:r1c4")]';
  put '[Format.Font("Courier New",10,true,
       false,false,false,0,false,false)]';
  put '[Column.Width(0,"c1:c4",false,3)]';
  put '[Select("r2c1:r2c1")]';
  put '[Freeze.Panes(true,0,1)]';
run;
```

Finally, the finished workbook is saved once more.

9 Coda…

All said and done? A few closing remarks… First, `%sastoxl`, like most DDE-related macros, is a piece of work that is constantly under construction. DDE is such a powerful technique for controlling Excel® output straight from a SAS® program, that new functionality suggests itself once in a while. The most recent version of the `%sastoxl` source code – as well as some other macros – will be available from the author's web site http://www.sas-consultant.com

What's next? A long hard look at recent releases of the SAS® System? In order to function with release 7.0 and later, certain parts of `%sastoxl` will need to be rewritten in a more robust way. One obvious issue is the fact that as of version 7.0, the length of a character variable is no longer restricted to 200 bytes. The calculation of the `lrecl` value will therefore need to be a bit more sophisticated.

Some of the conventions used in the code, *e.g.* the formatting of the cells for character variables, may not be suitable for any given user. The reader is encouraged to use, abuse, and modify the code to suit her/his own needs! Any fancy enhancements that make `%sastoxl` more widely useful will be greatly appreciated at the e-mail address mentioned below.

References

Bodt, M. Talking to PC Applications Using Dynamic Data Exchange. *Observations®*, vol. 5, no. 3, pp. 18–27, 1996.

Kuligowski, A. T. Advanced Methods to Introduce External Data into the SAS® System. *Proceedings of the Twenty-Fourth Annual SAS® Users Group International Conference*, paper 53, 1999.

Mumma, M. T. The Redmond to Cary Express – A Comparison of Methods to Automate Data Transfer Between SAS® and Microsoft Excel®. *No reference given*.

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

Schreier, H. Getting Started with Dynamic Data Exchange. *Proceedings of the Sixth Annual Southeastern SAS® Users Group Conference*, pp. 207–215, 1998.

Stetz, M. Using SAS® Software and Visual Basic for Applications to Automate Tasks in Microsoft Word: an Alternative to Dynamic Data Exchange. *Proceedings of the Twenty-Fifth Annual SAS® Users Group International Conference*, paper 21, 2000.

Trademarks

Contacting the Author

The author welcomes any questions, corrections, remarks, both on- and off-topic via e-mail at:

ddepapers@vyverman.com

```
%*****************************************************************************;
%*                                                                           *;
%* SOURCE   : sastoxl.sas                                                    *;
%*                                                                           *;
%* PURPOSE  : To pour the contents of a SAS data set into a MS Excel spread- *;
%*            sheet using DDE. The resulting spreadsheet receives some basic *;
%*            formatting and is saved to a specified location. Values are    *;
%*            entered with their formats applied, whenever there is one      *;
%*            defined on the data set.                                       *;
%*                                                                           *;
%* CREATED  : 08-DEC-99 by Koen Vyverman                                     *;
%*                                                                           *;
%* MODIFIED : -                                                              *;
%*                                                                           *;
%* USAGE    : Macro parameters :                                             *;
%*                                                                           *;
%*            libin    (required): SAS library where the input data set     *;
%*                                 lives.                                    *;
%*                                                                           *;
%*            dsin     (required): Name of the input data set.              *;
%*                                                                           *;
%*            cell1row (optional): The row number of the first cell of the  *;
%*                                 spreadsheet where data should be inserted.*;
%*                                 i.e. the upper left corner of the data.   *;
%*                                 Default = 1.                              *;
%*                                                                           *;
%*            cell1col (optional): The column number of the first cell of the*;
%*                                 spreadsheet where data should be inserted.*;
%*                                 i.e. the upper left corner of the data.   *;
%*                                 Default = 1.                              *;
%*                                                                           *;
%*            nrows    (optional): The number of rows/observations to be in- *;
%*                                 serted. If none is specified, an attempt  *;
%*                                 will be made to insert all observations.  *;
%*                                 Needless to say, this number needs to be  *;
%*                                 smaller than the maximal number of rows   *;
%*                                 supported by Excel...                     *;
%*                                                                           *;
%*            ncols    (optional): The number of columns/variables to be in- *;
%*                                 serted. If none is specified, an attempt  *;
%*                                 will be made to insert all variables.     *;
%*                                 Needless to say, this number needs to be  *;
%*                                 smaller than the maximal number of columns*;
%*                                 supported by Excel...                     *;
%*                                                                           *;
%*            tmplpath (optional): The full path to the directory where the  *;
%*                                 Excel spreadsheet resides that the data   *;
%*                                 needs to be written to. To be used in con-*;
%*                                 junction with TMPLNAME. If none is speci- *;
%*                                 fied, a standard new spreadsheet will be  *;
%*                                 used. Do _not_ end the path with a back-  *;
%*                                 slash character.                          *;
%*                                                                           *;
%*            tmplname (optional): The filename of the Excel spreadsheet     *;
%*                                 in the directory specified by TMPLPATH to *;
%*                                 which the data needs to be written. To be *;
%*                                 used in conjunction with TMPLPATH. If none*;
%*                                 is specified, a standard new spreadsheet  *;
%*                                 will be used.                             *;
%*                                                                           *;
%*            sheet    (optional): The name of the worksheet within the Excel*;
%*                                 document to which the data will be        *;
%*                                 written. Default = Sheet1 (being the name *;
%*                                 of the first worksheet in any new blank   *;
%*                                 Excel document...)                        *;
%*                                                                           *;
%*            savepath (optional): The full path to the directory where the  *;
%*                                 finalized Excel spreadsheet needs to be   *;
%*                                 saved. May be used independently from     *;
%*                                 SAVENAME. Do _not_ end the path with a    *;
%*                                 back-slash character. Default = c:\temp   *;
%*                                                                           *;
%*            savename (optional): The filename by which the finalized Excel *;
%*                                 spreadsheet should be saved in the        *;
%*                                 directory specified by SAVEPATH. May be   *;
%*                                 used independently from SAVEPATH.         *;
%*                                 Default = SASTOXL Output                  *;
%*                                                                           *;
```

```
%*            stdfmtng (optional): Standard formatting flag. Off by default.  *;
%*                                 Give a value of 1 to turn on. This will    *;
%*                                 apply some basic formatting to the         *;
%*                                 inserted data. The label row will be bol-  *;
%*                                 dened. Font will be set to Courier. Column *;
%*                                 width will be set to best fit. Furthermore,*;
%*                                 freeze panes will be turned on for the     *;
%*                                 label row...                               *;
%*                                                                            *;
%* EXAMPLE  : 1) Suppose the data set WORK.SOMETHNG needs to be exported to   *;
%*               an Excel spreadsheet, and saved as 'c:\temp\Some data.xls'.   *;
%*               To accomplish this, submit the following macro call:         *;
%*               %sastoxl(                                                    *;
%*                        libin=work,                                         *;
%*                        dsin=somethng,                                      *;
%*                        savepath=c:\temp,                                   *;
%*                        savename=Some Data,                                 *;
%*                        stdfmtng=1                                          *;
%*                       );                                                   *;
%*                                                                            *;
%*            2) Suppose only the first 125 rows of data set WORK.SOMETHNG    *;
%*               need to be exported to an existing Excel spreadsheet, and    *;
%*               saved as 'c:\temp\Some data.xls'. Suppose the full path and   *;
%*               name of the document in which the data need to be inserted   *;
%*               is 'n:\sasok\data\blank dox\Serious Fun.xls', and the block   *;
%*               of data is wanted at row 37, column 3 of a worksheet named   *;
%*               'Stuff from SAS'.                                            *;
%*               To accomplish this, submit the following macro call:         *;
%*               %sastoxl(                                                    *;
%*                        libin=work,                                         *;
%*                        dsin=somethng,                                      *;
%*                        cell1row=37,                                        *;
%*                        cell1col=3,                                         *;
%*                        nrows=125,                                          *;
%*                        tmplpath=n:\sasok\data\blank dox,                   *;
%*                        tmplname=Serious Fun,                               *;
%*                        sheet=Stuff from SAS,                               *;
%*                        savepath=c:\temp,                                   *;
%*                        savename=Some Data,                                 *;
%*                        stdfmtng=1                                          *;
%*                       );                                                   *;
%*                                                                            *;
%* CAVEAT   : Specifying either TMPLPATH or TMPLNAME without the other will   *;
%*            result in both values being reset to their default settings. As *;
%*            a consequence, a standard new document will be used.            *;
%*                                                                            *;
%******************************************************************************;

%macro sastoxl(
               libin=,
               dsin=,
               cell1row=1,
               cell1col=1,
               nrows=,
               ncols=,
               tmplpath=,
               tmplname=,
               sheet=Sheet1,
               savepath=c:\temp,
               savename=SASTOXL Output,
               stdfmtng=
              );

  %local
    ready
    misspar
    cnotes
    csource
    csource2
    cmlogic
    csymbolg
    cmprint
    tab
    ulrowlab
    ulcollab
    lrrowlab
    lrcollab
    ulrowdat
```

```
  ulcoldat
  lrrowdat
  lrcoldat
  lrecl
  types
  vars
  i
  colind
  ;

%let ready=0;
%let misspar=0;
%let cnotes=0;
%let csource=0;
%let csource2=0;
%let cmlogic=0;
%let csymbolg=0;
%let cmprint=0;
%let tab='09'x;
%let ulrowlab=;
%let ulcollab=;
%let lrrowlab=;
%let lrcollab=;
%let ulrowdat=;
%let ulcoldat=;
%let lrrowdat=;
%let lrcoldat=;
%let lrecl=;
%let types=;
%let vars=;
%let i=0;
%let colind=;

%* First we determine the values of certain SAS System options.             *;
%if %sysfunc(getoption(notes))=NOTES %then %do;
  %let cnotes=1;
  %end;

%if %sysfunc(getoption(source))=SOURCE %then %do;
  %let csource=1;
  %end;

%if %sysfunc(getoption(source2))=SOURCE2 %then %do;
  %let csource2=1;
  %end;

%if %sysfunc(getoption(mlogic))=MLOGIC %then %do;
  %let cmlogic=1;
  %end;

%if %sysfunc(getoption(symbolgen))=SYMBOLGEN %then %do;
  %let csymbolg=1;
  %end;

%if %sysfunc(getoption(mprint))=MPRINT %then %do;
  %let cmprint=1;
  %end;

%put;

%* We then turn all those options off, this to minimize the amount of junk  *;
%* that the usage of this macro would otherwise insert between the lines of *;
%* the LOG of the code in which it gets used...                              *;
options
  nonotes
  nosource
  nosource2
  nomlogic
  nosymbolgen
  nomprint
  ;

%* Then, we do some parameter checking...                                    *;
%if (&libin=) %then %do;
  %put ER%str()ROR: The LIBIN parameter is missing in a call to the SASTOXL macro!;
  %let misspar=1;
  %end;
```

```
%if (&dsin=) %then %do;
  %put ER%str()ROR: The DSIN parameter is missing in a call to the SASTOXL macro!;
  %let misspar=1;
  %end;

%if &misspar %then %do;
  %put;
  %put ER%str()ROR: The SASTOXL macro bombed due to errors...;
  %put;
  %goto mquit;
  %end;

%* If we are still there, we fill in the values of some of the optional     *;
%* parameters that were either left blank, or were inadvertently reset to   *;
%* blank in the macro call.                                                 *;
%if (&cell1row=) %then %do;
  %let cell1row=1;
  %put;
  %put NO%str()TE: The default value of the CELL1ROW parameter appears to have been;
  %put ----- overwritten by a _NULL_ value during invocation of the SASTOXL;
  %put ----- macro. It has been reset to '1' in order to allow macro execution.;
  %put;
  %end;

%if (&cell1col=) %then %do;
  %let cell1col=1;
  %put;
  %put NO%str()TE: The default value of the CELL1COL parameter appears to have been;
  %put ----- overwritten by a _NULL_ value during invocation of the SASTOXL;
  %put ----- macro. It has been reset to '1' in order to allow macro execution.;
  %put;
  %end;

%if (&sheet=) %then %do;
  %let sheet=Sheet1;
  %put;
  %put NO%str()TE: The default value of the SHEET parameter appears to have been;
  %put ----- overwritten by a _NULL_ value during invocation of the SASTOXL;
  %put ----- macro. It has been reset to 'Sheet1' in order to allow macro execution.;
  %put;
  %end;

%if (&savepath=) %then %do;
  %let savepath=c:\temp;
  %put;
  %put NO%str()TE: The default value of the SAVEPATH parameter appears to have been;
  %put ----- overwritten by a _NULL_ value during invocation of the SASTOXL;
  %put ----- macro. It has been reset to 'c:\temp' in order to allow macro execution.;
  %put;
  %end;

%if (&savename=) %then %do;
  %let savename=SASTOXL Output;
  %put;
  %put NO%str()TE: The default value of the SAVENAME parameter appears to have been;
  %put ----- overwritten by a _NULL_ value during invocation of the SASTOXL;
  %put ----- macro. It has been reset to 'SASTOXL Output' in order to allow macro execution.;
  %put;
  %end;

%if (&nrows=) %then %do;
  proc sql noprint;
    select
      trim(left(put(nobs,20.))) into :nrows
    from
      sashelp.vtable
    where
      (libname=upcase("&libin"))
      and
      (memname=upcase("&dsin"))
    ;
  quit;
  %end;

%if (&ncols=) %then %do;
  proc sql noprint;
    select
      trim(left(put(nvar,20.))) into :ncols
```

```
    from
      sashelp.vtable
    where
      (libname=upcase("&libin"))
      and
      (memname=upcase("&dsin"))
    ;
  quit;
  %end;

%* To make sure that a put-statement to one of our DDE-filenames remains on *;
%* one single row of the spreadsheet, we calculate a LRECL for the filename *;
%* that will (hopefully) be large enough to accommodate all the formatted   *;
%* values that we will want to push through. In the true SAS 6.12 spirit,   *;
%* we assume that 200 bytes per variable will do the trick:                 *;
%let lrecl=%eval(200*&ncols);

%* The parameters TMPLPATH and TMPLNAME should be used in conjunction with  *;
%* each other. Check if this is the case. When necessary, reset both to a   *;
%* _NULL_ value...                                                          *;
%if ((&tmplpath=) and (&tmplname ne)) or ((&tmplpath ne) and (&tmplname=)) %then %do;
  %let tmplpath=;
  %let tmplname=;
  %put;
  %put NO%str()TE: During invocation of the SASTOXL macro, either the parameter TMPLPATH;
  %put ----- was specified without TMPLNAME, or vice versa. The macro expects either;
  %put ----- both or none of them. They have been reset to a _NULL_ value to allow macro
             execution.;
  %put;
  %end;

%* The following either launches MS Excel, or does nothing if an instance   *;
%* of the application is already running.                                   *;
filename sas2xl dde 'excel|system';

data _null_;
  file sas2xl;
run;

options
  noxwait
  noxsync
  ;

%if &syserr ne 0 %then %do;

  x '"c:\program files\microsoft office\office\excel.exe"';

  data _null_;
    x=sleep(10);
  run;

  %end;

%* We then open a DDE link to MS Excel for the sending of system commands.  *;
filename sas2xl dde 'excel|system';

%* Define some simple text files to write the DDE commands to before they   *;
%* are executed. This will allow us to check what commands precisely were   *;
%* sent to the Excel application.                                           *;
filename execit1 'c:\temp\execit1.txt';
filename execit2 'c:\temp\execit2.txt';
filename execit3 'c:\temp\execit3.txt';
filename execit4 'c:\temp\execit4.txt';
filename execit5 'c:\temp\execit5.txt';

%* If TMPLPATH and TMPLNAME were given, we open the document they specify.  *;
%* Otherwise, we ask for a new blank document of the workbook type.         *;
%if ((&tmplpath ne) and (&tmplname ne)) %then %do;
  data _null_;
    file execit1;
    put 'data _null_;';
    put '  file sas2xl;';
    put "  put '[error(false)]';";
    put "  put '[open(" '"' "&tmplpath" '\' "&tmplname" '")]' "';";
    put 'run;';
  run;
  %end;
```

```
%else %do;
  data _null_;
    file execit1;
    put 'data _null_;';
    put '  file sas2xl;';
    put "  put '[error(false)]';";
    put "  put '[new(1)]';";
    put 'run;';
  run;
  %end;

%include execit1;

%* We then need to define a DDE link pointing to the exact location where   *;
%* data should be inserted. To do so, we need to know the filename. The     *;
%* easiest way to find the filename is to save the document at the location *;
%* specified by &SAVEPATH with the name &SAVENAME...                        *;
data _null_;
  file execit2;
  put 'data _null_;';
  put '  file sas2xl;';
  put "  put '[error(false)]';";
  put "  put '[Save.As(" '"' "&savepath" '\' "&savename" '")]' "';";
  put 'run;';
run;

%include execit2;

%* Writing the labels and the data requires gathering some information      *;
%* about the input data set:                                                *;
proc contents data=&libin..&dsin
              out=____meta
              noprint;
run;

%* If a variable does not have a label defined, use the variable name...    *;
data ____meta;
  set ____meta;
  if label=' ' then label=name;
run;

proc sort data=____meta;
  by
    varnum;
run;

%* Calculate the range of cells to which label data will be written. The    *;
%* upper left cell is obviously defined by (&CELL1ROW,&CELL1COL). The lower *;
%* right corner of the range, which incidentally is on the same row in case *;
%* of the labels, is defined as follows.                                    *;
%let ulrowlab=&cell1row;
%let ulcollab=&cell1col;
%let lrrowlab=&ulrowlab;
%let lrcollab=%eval(&cell1col+&ncols-1);

%* Calculate the range of cells to which the real data will be written. The *;
%* upper left cell is obviously defined by (&CELL1ROW+1,&CELL1COL).         *;
%let ulrowdat=%eval(&cell1row+1);
%let ulcoldat=&cell1col;
%let lrrowdat=%eval(&cell1row+&nrows);
%let lrcoldat=%eval(&cell1col+&ncols-1);

%* Now, before we even attempt to send any data to Excel, we select all     *;
%* target cells to which we plan to write variables of type 2 (character)   *;
%* and format them as text. Otherwise, MS Excel will try to be clever and   *;
%* interpret the entered data, which we do not want to see happen. We leave *;
%* the cells for numerical data alone, risking MS cleverness attempts. The  *;
%* reason being that the autofilter tool performs badly on numerical data   *;
%* with a text format imposed on it. Ah, well... cant have it all...        *;
%* From ____META, we generate a list &TYPES of variable types, separated by *;
%* blanks:                                                                  *;
proc sql noprint;
  select distinct
    type
  into
    :types separated by ' '
  from
    ____meta
```

```
  order by
    varnum
  ;
quit;

data _null_;
  file execit3;
  put 'data _null_;';
  put '  file sas2xl;';
  put "  put '[error(false)]';";
  put "  put '[workbook.activate(" '"' "&sheet" '"' ",false)]';";
                          %let i=1;
                          %let colind=;
                          %do %while (%length(%scan(&types,&i))>0);
                            %if ((%scan(&types,&i)=2) and (&i le &ncols)) %then %do;
                              %let colind=%eval(&ulcollab+&i-1);
  put "  put '[Select(" '"r' "&ulrowlab" 'c' "&colind" ':r' "&lrrowdat" 'c' "&colind" '")]' "';";
  put "  put '[Format.Number(" '"@")]' "';";
                              %end;
                              %let i=%eval(&i+1);
                            %end;
  put 'run;';
run;

%include execit3;

%* Now we can define the DDE link for the section of the spreadsheet where  *;
%* the variable labels will be written:                                      *;
filename xllabels dde
  "excel|&savepath.\[&savename..xls]&sheet!r&ulrowlab.c&ulcollab.:r&lrrowlab.c&lrcollab."
  notab lrecl=&lrecl;

%* For the first &NCOLS variables, we write the labels to the DDE-filename  *;
%* XLLABELS.                                                                 *;
data _null_;
  set ____meta end=last;
  file xllabels notab;
  if varnum<=&ncols then do;
    put label @@;
    if not last then put &tab @@;
    end;
run;

%* We proceed to define the DDE link to the section of the spreadsheet      *;
%* where the actual data will be written:                                    *;
filename xlsheet dde
  "excel|&savepath.\[&savename..xls]&sheet!r&ulrowdat.c&ulcoldat.:r&lrrowdat.c&lrcoldat."
  notab lrecl=&lrecl;

%* From ____META, we generate a list &VARS of variable names, separated by  *;
%* blanks:                                                                   *;
proc sql noprint;
  select distinct
    name
  into
    :vars separated by ' '
  from
    ____meta
  order by
    varnum
  ;
quit;

%* And then we actually write the data...                                    *;
data _null_;
  set &libin..&dsin(obs=&nrows);
  file xlsheet notab;
   put
                          %let i=1;
                          %do %while(%length(%scan(&vars,&i))>0);
     %scan(&vars,&i) &tab
                            %let i=%eval(&i+1);
                            %end;
     ;
run;

%* If the &STDFMTNG flag is on, we apply some standard formatting to the    *;
%* inserted range...                                                         *;
```

```
%if (&stdfmtng ne) %then %do;

  %* First select all and set the font to Courier 10pt.                    *;
  %* Then select the row of labels, and turn them bold.                     *;
  %* Then select the column-range, and apply best fit column width.         *;
  %* Finally, do a freeze panes to keep the labels visible while scrolling. *;
  data _null_;
    file execit4;
    put 'data _null_;';
    put '  file sas2xl;';
    put "  put '[error(false)]';";
    put "  put '[workbook.activate(" '"' "&sheet" '"' ",false)]';";
    put "  put '[Select(" '"r' "&ulrowlab" 'c' "&ulcollab" ':r' "&lrrowdat" 'c' "&lrcoldat" '")]'
        "';";
    put "  put '[Format.Font(" '"Courier New",10,false,false,false,false,0,false,false)]' "';";
    put "  put '[Select(" '"r' "&ulrowlab" 'c' "&ulcollab" ':r' "&lrrowlab" 'c' "&lrcollab" '")]'
        "';";
    put "  put '[Format.Font(" '"Courier New",10,true,false,false,false,0,false,false)]' "';";
    put "  put '[Column.Width(0," '"c' "&ulcollab" ':c' "&lrcollab" '",false,3)]' "';";
    put "  put '[Select(" '"r' "&ulrowdat" 'c' "&ulcoldat" ':r' "&ulrowdat" 'c' "&ulcoldat" '")]'
        "';";
    put "  put '[Freeze.Panes(true,0,1)]';";
    put 'run;';
  run;

  %include execit4;

  %end;

%* We save the document once more with its intended name and close it.      *;
data _null_;
  file execit5;
  put 'data _null_;';
  put '  file sas2xl;';
  put "  put '[error(false)]';";
  put "  put '[Save.As(" '"' "&savepath" '\' "&savename" '")]' "';";
  put "  put '[File.Close(false)]';";
  put 'run;';
run;

%include execit5;

%* Did we get this far? If so, we want to clean up and therefore turn the   *;
%* READY flag on.                                                           *;
%let ready=1;

%* Upon exiting the macro, we restore all the system options that we turned *;
%* off earlier on...                                                        *;
%mquit:;

%if &cnotes %then %do;
  options notes;
  %end;

%if &csource %then %do;
  options source;
  %end;

%if &csource2 %then %do;
  options source2;
  %end;

%if &cmlogic %then %do;
  options mlogic;
  %end;

%if &csymbolg %then %do;
  options symbolgen;
  %end;

%if &cmprint %then %do;
  options mprint;
  %end;

%* Check if the macro actually executed some code (READY=1) or if we got    *;
%* here because of lacking parameter errors (READY=0).                      *;
%if &ready %then %do;
```

```
    options
      nonotes
      ;

    %* Clean up remaining junk in the local SAS session.                    *;
    proc datasets nolist lib=work;
      delete
        ____meta  / memtype=data;
    quit;

    %if &cnotes %then %do;
      options notes;
      %end;

    %end;

%mend sastoxl;
```