

SESUG Paper AD-109-2017

A SAS macro replacement for Dynamic Data Exchange (DDE) for use with SAS grid

Saki Kinney, David Wilson, and Benjamin Carper, RTI International

ABSTRACT

The ability to write to specific cells of Excel workbooks is an important feature of DDE that users lose when migrating to grid computing. SAS Add-In for Microsoft Office requires programming Visual Basic and complicates quality control. To replace DDE functionality we have developed a set of macros which use PROC IML to integrate functionality of R packages XLConnect and openxlsx for working with Excel workbooks. This paper describes the macros, and their performance and implementation with SAS grid computing.

INTRODUCTION

Dynamic Data Exchange (DDE) is a Microsoft protocol for exchanging information between applications on the same Windows machine. For SAS users, it allows programs to interact with Excel workbooks and other Microsoft Office documents; however, it is gradually becoming obsolete as it is incompatible with modern grid computing. For many years DDE had served as a critical tool in our organization for generating high quality and consistent Excel tables on a large scale; however, when our organization made the decision to move from individual SAS PC licenses to Linux SAS grid computing, we needed an alternative approach (Wilson, Green, and Terminiello, 2017).

The replacement for DDE we were offered by SAS was SAS Add-in for Microsoft Office; however, we estimate about half of our 400+ SAS users have a need for DDE functionality, and that few of them are familiar with Visual Basic for Applications (VBA). Switching to SAS Add-In would require a substantial investment into VBA development and training in addition to rewriting hundreds of programs. Further, the use of VBA would complicate our quality control and replicable production processes by preventing the use of SAS macros and creating additional code and logs. We produce numerous tables using SAS code written by SAS programmers, many of whom have primary roles as statisticians and analysts in other fields.

This paper describes the development of macros that allow SAS programs to modify Excel workbooks without DDE. While SAS now provides several other ways to interact with Excel, which we also use, it does not allow writing data to specific ranges in formatted Excel templates. Many of the tables RTI produces for clients incorporate a variety of disparate pieces of information from multiple SAS procedures and DDE provided a convenient method of sequentially adding output to an Excel table. This is the key functionality we sought to replace, while also replicating some of the features of DDE that our programmers were used including in their SAS code, such as copying template sheets. We did not attempt to replicate DDE functionality in entirety, only the features that were important to our programmers. Many of these features had already been coded into macros which some programmers used as an alternative to learning DDE syntax. These macros served as the starting point for the DDE replacement macros, and we strove to limit the edits that would be required for macro users to migrate their code to the grid.

APPROACH

For our replacement macros we utilized SAS/IML functionality to interact with R. R is an open-source software package with numerous independently-developed packages freely available to download (See <https://www.R-project.org/>.) and several packages are available that modify Excel workbooks. The R packages we chose to work with are *openxlsx* (Walker, 2017) and *XLConnect* (Mirai Solutions, 2017). XLConnect is more fully developed than openxlsx; however, it uses Java which can have performance issues. We found openxlsx could provide most of the functions we needed and ran about 30 percent faster, so we have used XLConnect only where openxlsx did not have the needed functionality.

It is possible to utilize any installed R package within PROC IML though openxlsx and XLConnect store workbooks in different types of R objects that cannot be read by each other so it would not be

computationally efficient to use both within the same session. Users of our macros do not need to write R code; however, some users choose to write their own IML/R code to perform similar functionality.

A fundamental difference from implementing SAS Add-In is that the process is all executed entirely by SAS on the grid, whereas SAS Add-In is executed from Excel. This is illustrated in Figure 1. Figure 2 describes the process of using R as an intermediary between SAS and Excel.

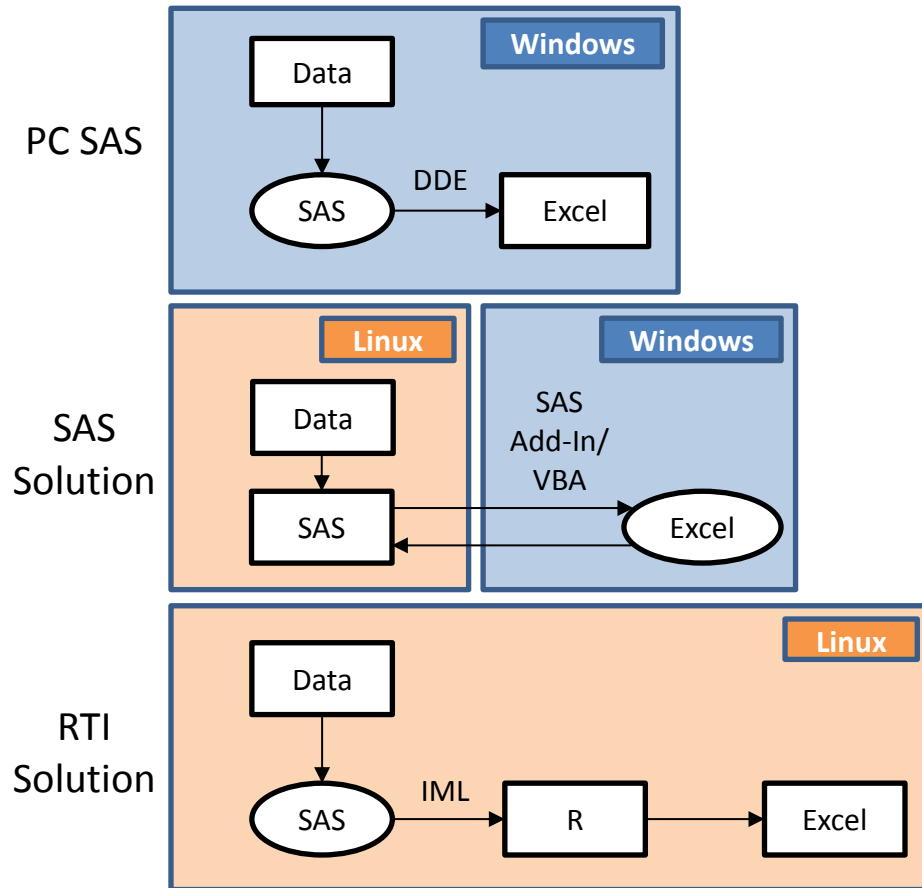


Figure 1. General Process Flow for Writing to Excel

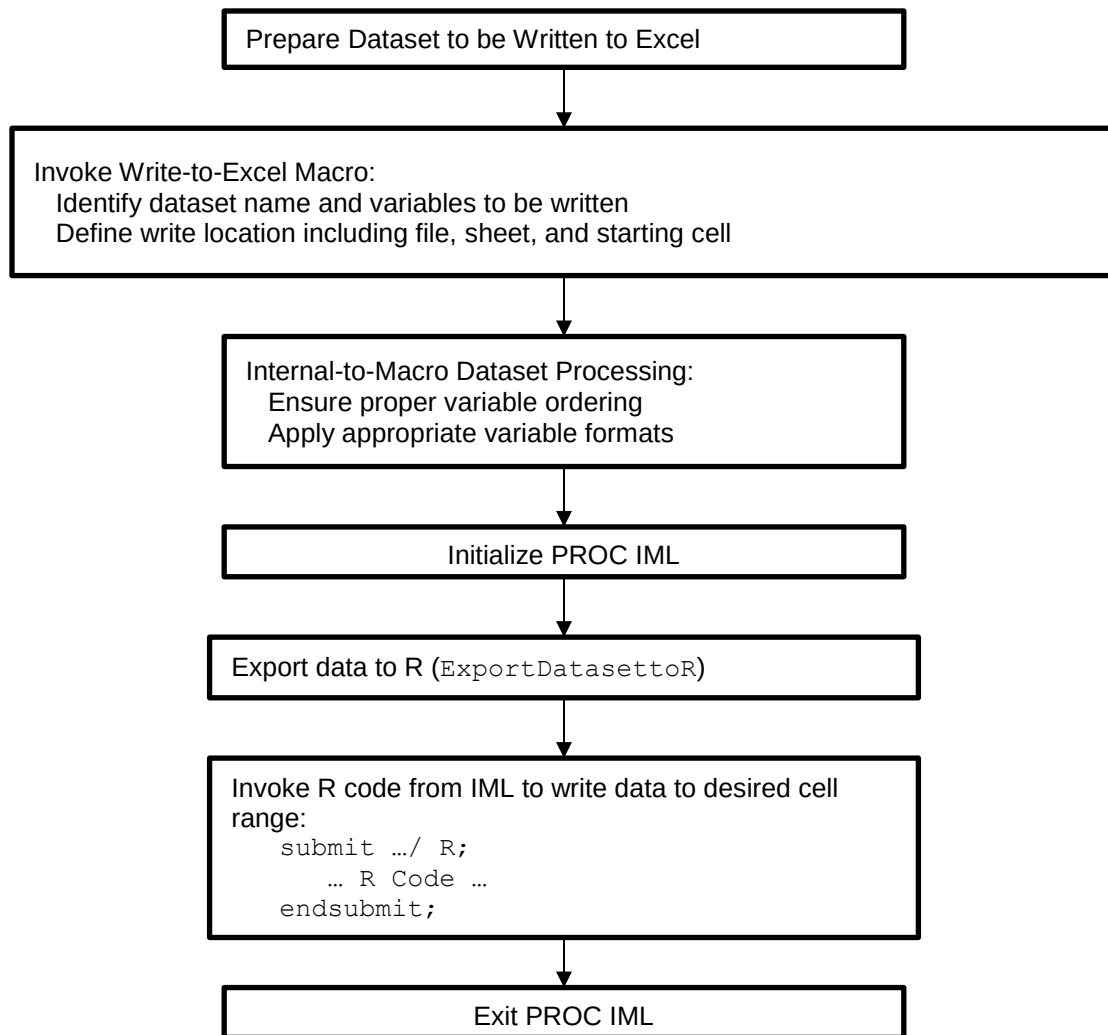


Figure 2. General Flow of DDE Replacement Macro for Writing to Excel

If writing open code, code to write data to specific cells in an Excel workbook might look something like this:

```

proc iml;
  run ExportDatasetToR("lib.mydata", "rdata");
  submit/R;
    library(openxlsx);
    wb = loadWorkbook(xlsxFile = "/myproject/mybook.xlsx");
    writeData(wb = wb, x = rdata, sheet = "Sheet1", xy = c(2,1));
    saveWorkbook(wb, file = "/myproject/mybook.xlsx", overwrite = TRUE);
  endsubmit;
quit;

```

In this case, starting PROC IML serves to initiate the session that will modify one or more workbooks. It is possible to include multiple submit blocks and to have the R session information retained between blocks. Once you quit IML, however, all R session information is lost, and all workbook edits need to be saved or they will be lost. R does not actually open Excel, rather, it imports the workbook into an R object, which can be modified with different commands. The workbook itself is not actually modified until you tell R to save changes.

There is computational overhead in repeated calls to IML and R, each time loading and saving a workbook, so that writing many small chunks of data to Excel in separate IML/R sessions is substantially slower than DDE. Combining workbook edits into fewer IML/R sessions alleviated this issue so we designed our macros to perform multiple operations within one IML/R session. Because our macros include additional SAS code for processing macro inputs, writing macros to be inserted inside of an IML procedure was infeasible as the additional code would cause IML to quit. As such, all of our macros that operate on workbooks save all changes, a shift from our pre-grid use of DDE. For sheet manipulation macros this meant vectorizing operations that had previously been performed in %DO loops. In the case of writing to cells within workbooks, this meant creating a process for allowing multiple writes within one PROC IML session. The result was a set of macros that work together to collect information and send it to R all at once. For example:

```
%xlInitMulti(workbook = &template, saveas= &populated, sheet = sheet1);
  %xlWriteS(indata = dat1, startcol = B, startrow = 2);
  %xlWriteS(indata = dat2, startcol = F, startrow = 2);
  %xlWriteS(indata = dat3, startcol = I, startrow = 2);
%xlWriteMulti();
```

In this example, %xlInitMulti copies the template workbook and stores the other inputs as global macro variables for use by the other macros. %xlWriteS does not actually interact with Excel but stores the inputs, and then all of these are collected by %xlWriteMulti and sent to Excel in a single IML/R session. An annotated example of how SAS and R can be combined in a macro is given below. Descriptions of the macros we created and their inputs are provided in the Appendix.

ANNOTATED EXAMPLE

This macro writes a dataset to a specific location in a workbook:

```
%macro writetoexcel(data, workbook, startpos);
  proc iml;
    * Send dataset to R;
    run ExportDatasetToR("&data", "rdata");

    * Matrices are a good way to send numeric parameters to R;
    startpos = {&startpos};
    run ExportMatrixtoR(startpos, "startpos");

    *Put text string inputs into IML objects and list on submit line;
    book = "&workbook";

    /* NOTE: SAS does not allow submit blocks inside macros but does
    allow them to be stored in external files and read in with
    %include but for illustrative purposes the submit block is
    included here. */

    submit book /R; *you can only submit IML objects;
    # Everything between submit and endsubmit should be valid R
    # code. Note R is case-sensitive.

    # Load any non-default R packages
    library(openxlsx);

    # objects from ExportMatrixtoR are already loaded in R but
    # the matrix format is not usually what you want.
    startpos = as.numeric(startpos);

    # Items on submit line are referred to with & but real SAS
```

```
# macro variables will not be recognized. Do not use
# "&book." or the dot will be printed. You usually
# need to load a workbook to modify it.
wb = loadWorkbook(xlsxFile = "&book");

# Note nothing special is needed to identify the data sent
# from ExportDatasetToR. You can include as many edits to
# the workbook as needed, or edit multiple workbooks.
writeData(wb = wb, x = rdata, sheet = "Sheet1", xy =
  startpos));

#Always save at the end!
saveWorkbook(ActiveWB, file = "&book", overwrite = TRUE); endsubmit;
quit;
%mend writetoexcel;
```

A DIFFERENT PARADIGM

A key difference for users is that workbooks must be closed when written to, removing the interactive aspect of DDE. A benefit of this is that users retain use of their PC while workbooks are being modified, and can run jobs in parallel as long as they involve different workbooks. Multiple programs writing to the same workbook simultaneously can corrupt the workbook.

When R is submitted via SAS/IML, R errors and warnings are written to the SAS log, and R output to the SAS listing. All R messages printed to log files are prefaced with "R: WARNING". While R does not typically log successful operations it does sometimes produce notes or messages that are not warnings or errors; however, these can be suppressed within R. Another quirk of SAS functionality to submit to R is that submit blocks are not allowed within macros, but %include statements are. Hence, each SUBMIT/R block needs to be stored in a separate file.

While some programmers will not be very familiar with either IML or R, they use similar syntax. Only a rudimentary command of either language is necessary to get started, as illustrated in our sample code. The SAS IML User's Guide section on R and the documentation for XLConnect and openxlsx provide the bulk of the necessary commands. Writing fully-featured macros for multiple users and uses required a bit more R and SAS programming but not much IML.

While we already had a need for both R and IML on our grid server, it could be a disadvantage for grid administrators to have to manage R on the SAS server if they do not otherwise need it, or to purchase IML. Administrators will also need to set the RLANG option. The use of XLConnect also requires RJava. Excel does not need to be installed on the server.

LIMITATIONS

While we only sought to replicate the functionality needed by our programmers, there were some features which we were not able to replicate, impacting a small number of users. These include copying or moving sheets between workbooks and running VBA scripts. While it is possible to execute VBA macros from R, we have not investigated this as VBA does not run on Linux. As an alternative, users can run batch scripts from Windows which will send their SAS jobs to Linux and execute VBA scripts afterward. At present most of our macros are incompatible with macro-enabled Excel workbooks (XLSM files). XLS files must be converted to XLSX to use openxlsx. Anecdotally, runtimes are somewhat slower than DDE, but most users do not notice or find the ability to run their jobs remotely compensates for this. Limiting the number of IML/R sessions was essential for providing acceptable runtimes.

One complication in writing to Excel via R is in maintaining SAS data formats. While the variable types (ie, character, numeric, date, etc) are generally written correctly, there are a few exceptions. Additional SAS code was required in order to display value formats (eg. Yes/No instead of 0/1) created with the FORMAT procedure. With DDE, numbers contained in SAS character variables would be written as numeric values in Excel; however, our programs write them all as character. A workaround is to write the numbers and text separately or to apply post-processing in VBA. One user wrote a small VBA script to convert numbers

to numeric format. Generally we rely on formatted templates for styles like fonts and borders, and specific date and number formats, though some of these can be controlled with R. Text strings containing Excel formulas are written as text strings and not automatically evaluated in Excel unless they are assigned a formula format in R. Some special or unprintable characters and foreign alphabets require running SAS in UTF-8 mode.

IMPLEMENTATION IN SAS GRID

In a SAS grid environment, computing resources are shared between all users with users not generally selecting where their work is performed. For best results, the infrastructure for interacting with R will need to be installed on all usable grid nodes. This includes base R and all relevant packages.

Our macros were designed for widespread usage by all SAS users at our organization. To avoid versioning issues and loss of functionality, we decided to maintain a centrally-located and curated copy of our macros with all modifications rolled out for testing to new locations to avoid creating problems in functioning programs. Users can make local copies of the macros and modify them without interfering with the shared copies.

CONCLUSION

This paper has described the an option for retaining a subset of DDE functionality after migration to grid computing. Providing users SAS macros to write to Excel was crucial to the success of our organization's transition to SAS grid. For our users that already were using macros instead of DDE commands the transition was easier as they required fewer edits to their code. While DDE may have become outdated, our need to execute some of its functionality from within SAS has not gone away, and is not likely to in the near future. If in the future a better mode for writing to Excel templates from SAS becomes available, the use of macros will facilitate that transition as well.

REFERENCES

- Mirai Solutions GmbH (2016). "XLConnect: Excel Connector for R". R package version 0.2-12.
<https://CRAN.R-project.org/package=XLConnect>.
- Walker, A. (2017). openxlsx: Read, Write and Edit XLSX Files. R package version 4.0.17.
<https://CRAN.R-project.org/package=openxlsx>.
- Wilson, D., Green, A., and Terminiello, K. (2017). "Getting on SAS Grid, Our Journey". SESUG 2017: The Proceedings of the SouthEast SAS Users Group, Cary, NC. Paper 86.

RECOMMENDED READING

- SAS® IML User's Guide

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Satkartar Kinney
RTI International
skinney@rti.org
www.rti.org

David Wilson
RTI International
dwilson@rti.org
www.rti.org

Benjamin Carper
RTI International

bcarper@rti.org
www.rti.org

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

APPENDIX: KEY MACROS

This section describes some of the key features of our macros.

xlWrite: Write data to a workbook.

Required arguments:

indata Name of the dataset containing the data to be written. Can include a libname and options, e.g. indata = mylib.mydata(rename = (var1 = var2)).

Optional arguments:

workbook Specify workbook with full path. If not specified, the active workbook is used. [default: active]

saveas Specify if you want to write workbook to a new location, using the full path. If not specified changes will be saved to the file specified in the workbook argument. [default: <blank>]

vars List of variables to be written. Variables will be written in the order they appear in this list. If blank, all variables from the dataset will be written in the order in which they appear in the dataset. [default: <blank>]

sheet Name of the worksheet where data values will be written. Sheet must exist. [default: active]

startrow First row in the cell range where data values will be written. [default: 1]

startcol First column in the cell range where data values will be written. The column may be specified as either a number or a letter. [default: 1]

formatvars List of variables that should be converted and stored as their formatted values prior to sending to Excel. [default: <blank>]

formula List of variables that should be written as Excel formulas. Variables must be all formulas or no formulas. The equal sign beginning a formula is optional and will be added if missing. Specify formula = yes if all variables being written contain formulas. [default: no]

autofit Columns in the worksheet to autofit. Multiple columns can be separated by commas, and multiple consecutive columns can be separated by hyphens. For example, autofit = %str(A,C-E,K) specifies that columns A, C, D, E, and K will be autofit. Columns may be specified as either numbers or letters. To autofit all columns to which data are written specify autofit = all. [default: no]

xlInitMulti: Initialize multi-step write

Required arguments:

workbook Name of the file where the new workbook will be saved. The name should include the full path (eg. /myprojectshare/excelfile.xlsx).

Optional arguments:

saveas Name of file name to save as. This creates a new file. This should be specified if file= a template file that you do not want to overwrite. The name should include the directory (like /myproject/newfile.xls). [default: <blank>]

sheet Specify a sheet to write to. If not specified, then sheet must be specified in each instance

of %xlWriteS. [default: <blank>]

xlWriteS: Similar to xlWrite but it does not actually write to Excel – it saves the inputs which are later collected by xlWriteMulti.

xlWriteMulti: Submit datasets and text strings as specified by %xlWriteS since last %xlInitMulti.

Required Arguments: None.

xlCopySheet: Copy one or more sheets in a workbook. This macro uses Java/XLConnect. If the names of any sheets involved contain spaces then you need to specify dlm = a character not in any sheet names, even for single sheets. To copy multiple sheets, you can either copy 1 to 1 or 1 to many. For example:

1) sheet = Template1 Template2 and copy = Copy1 Copy2, where Copy1 is copy of Template1 and Copy2 is a copy of Template2. Both sheet and copy must have the same number of sheet names.

2) sheet = template and copy = Copy1 Copy2, where both Copy1 and Copy2 are copies of template.

Required arguments:

sheet One or more worksheets to be copied.

Optional arguments:

position	The position(s) where the copied worksheets should end up. For example, if copy = Copy1 Copy2 and position=3 4, then Copy1 will be in the 3rd position and Copy2 in the 4th position in the workbook. If blank then the copies will be inserted in last position, in the order listed in copy=. This should have the same number of positions as there are sheets in copy. For best results, list in increasing order as they are processed sequentially. Can use : notation, eg 1:3 for 1 2 3. [default: <blank>]
workbook	Specify workbook with full path. If not specified, the active workbook is used. If specified this will become the active workbook. [default: active]
copy	Name given to the copy(ies) of the source worksheet. If blank, the macro will use the name of the source worksheet(s) with the suffix “ – Copy”. If the sheets listed already exist they will be overwritten. [default: <blank>]
active	Sheet to set active in Excel after copying is complete. [default: first sheet from copy=]
deletesheet	Sheet(s) to delete after copying is completed. [default: <blank>]
dlm	Specify a delimiter for lists of variables used in sheet/copy/deletesheet, and then use to specify sheet(s). Most any character not in any of your sheet names will do, but if it could be interpreted differently by SAS then both dlm and sheet names should be placed inside %str(). For example, dlm = %str(,) and sheet = %str(Sheet1,Sheet2,Sheet3), with no spaces unless part of a sheet name. If dlm is specified it applies to sheet, copy, and deletesheet arguments. [default: space]

xlOrder: Reorder sheets in a workbook.

Required arguments: None

Optional arguments:

workbook	Specify workbook with full path. If not specified, the active workbook is used. If specified this will become the active workbook. [default: active]
sheet	One or more sheets to be moved within the workbook. If blank, then all sheets will be moved to the position(s) specified.[default:<blank>]
position	The position(s) where the worksheets should end up. For example, if sheet = Sheet1 Sheet2 and position=3 4, then Sheet1 will be moved to the 3rd position and Sheet2 to the 4th position in the workbook. If blank then the sheets will be placed in the order listed in sheet=, starting with the first sheet in 1st position. Otherwise there should be the same

number of positions as there are sheets in sheet=. Can use : notation, eg 1:3 for 1 2 3.
[default: <blank>]

dlim Specify a delimiter for lists of variables used in sheet, and then use to specify sheet(s). Any character not in any of your sheet names will do, but if it could be interpreted differently by SAS then both dlm and sheet names should be placed inside %str(). For example, dlm = %str(,) and sheet = %str(Sheet1,Sheet2,Sheet3), with no spaces unless part of a sheet name. [default: space]

xllnit: Assigns a workbook to be 'active' which is the default workbook for other macros. If saveas is specified, then the saveas file is specified as the active workbook.

Required arguments:

workbook Name of the workbook to be set active. The name should include the full path (eg. /myprojectshare/excelfile.xlsx).

Optional arguments:

saveas Name of file name to save as. This creates a new file. This should be specified if workbook= a template file that you do not want to overwrite. The name should include the directory (like /myproject/newfile.xls). [default: <blank>]