Flat Pack Data: Converting and ZIPping SAS® Data for Delivery

Sarah Woodruff, Westat, Rockville, MD

ABSTRACT

Clients or collaborators often need SAS data converted to a different format. Delivery or even storage of individual data sets can become cumbersome, especially as the number of observations or variables grows. The process of converting SAS data sets into other forms of data and saving files into compressed ZIP storage has become not only more efficient, but easier to integrate into new or existing programs. This paper describes and explores various methods to convert SAS data sets to other formats as well as effective strategies to ZIP data sets along with any other files that might need to accompany them.

PROC IMPORT and PROC EXPORT have been long standing components of the SAS toolbox, so much so that they have their own wizards. However, understanding their syntax is important to effectively use them in code being run in batch or to include them in programs that may be run interactively but "hands free". The syntax of each is described with a particular focus on moving between SAS, STATA and SPSS, though some attention is also given to Excel. Once data sets and their attendant files are ready for delivery or need to be put into storage, compressing them into ZIP files becomes helpful. The process of using ODS PACKAGE to create such ZIP files is laid out and can be connected programmatically to the creation of the data sets or documents in the first place.

Keywords: PROC EXPORT, ZIP, ODS PACAKGE

INTRODUCTION

The data has been cleaned. The calculations have been checked. The formats have been applied. The code has been reviewed. You are done with your SAS data sets, right? Probably not. Often, data we work with in SAS needs to be converted to another type of file in order for the recipient of that data to be able to use it. In many cases, we are not in the same location as the recipient and thus the data needs to be efficiently transmitted. Even if a project has ended, data often needs to be efficiently stored and retained for a period of time. SAS provides an expanding toolbox of options in order to accomplish these goals.

OPENING THE BOX

SPSS and STATA

Compiling a complete data record often means bringing in files from other sources and getting everything into SAS even if you are not manipulating or adjusting the data. PROC IMPORT provides flexible tools making this task straightforward, often needing no more than the appropriate file extension. Such is the case when brining in SPSS data sets.

```
PROC IMPORT datafile = "C:\Project\User\DataLocation\SPSSData.sav"
            out= SASData.SPSSdata ;
RUN;
```

The .SAV file extension specifies to SAS that the data being read is from SPSS. So long as you know the location of the data, you can then set whatever name is most useful for the output data set. The OUT statement can send it to the WORK folder if specified, but if you have a LIBNAME established for an alternate location it can be saved there directly. Given the potential for differences in formatting and data composition, it is good practice to run a PROC CONTENTS on the new SAS data set to ensure that the records are coming through as expected and the data is comprehensible.

A similar process is possible when importing from STATA, but with some additional limitations. STATA is currently in version 13, but SAS is only able to read files in directly up through those created with version 9. Doing so requires the same syntax as importing an SPSS data set with the use of the .DTA file extension as opposed to .SAV. If data sets are from a version beyond 9, then Stat/Transfer can be used to convert them. Use of Stat/Transfer has the advantage of being able to check on data and make decisions about which variables to keep or eliminate as part of

the conversion process. Stat/Transfer (www.stattransfer.com) is a separate software package that allows for small or large scale movement of statistical data and data sets between platforms. Its GUI is user-friendly and it can also be run in batch should that better facilitate project work.

Excel

With some additional options utilized, PROC IMPORT becomes as adept at bringing in data from Excel as well. But, it is helpful to know something about the structure of the sheet(s) within the workbook before beginning the import. Let's review the pieces one by one.

```
PROC IMPORT OUT = SASData.ExcelData_Sheet1
```

The OUT statement allows you to name the SAS data set as well as specify its location. As discussed above, the WORK folder can be the output destination specified or an alternate LIBNAME can be established to which to send the data.

```
DATAFILE = "C:\Project\User\DataLocation\ExcelData.xlsx"
```

The DATAFILE statement allows you to specify the location of the Excel workbook to be read. XLS and XLSX files can both be utilized here without further modification or additional options.

```
DBMS = EXCEL REPLACE /* use of REPLACE is optional */ ;
```

The DBMS option needs to be set to Excel for these purposes, but REPLACE specifies the data set to be rewritten if one with the same name already exists in the specified location. While you are in the development phase with your code, it is a useful option to have in place so that you can automatically update the data set that you are creating based on the current status of the workbook. Use of the WORK folder may be a better option than writing to a permanent location if there is any concern about the potential to overwrite an existing file. A later step in the program could check for the presence of a matching data set name and provide alternative for the current one to be saved if such a match is found. Alternatively, REPLACE can be omitted to avoid this risk entirely and it can be dealt with manually.

```
RANGE = "Sheet1$" /* use appropriate sheet name, leave $ in place */ ;
```

RANGE specifies the sheet within the workbook to be read into the data set. The default in Excel is to number the sheets consecutively (ex. Sheet1, Sheet2, etc.), but the $ sign is a necessary follow up for proper SAS syntax. Knowing if the sheets within the workbook have been given more specific names is helpful in appropriately setting up this code. In addition, an established PROC IMPORT step could be turned into a simple macro if there are a large number of sheets to be read into separate data sets where the macro could pass in the name of the sheets (or changes to any other options) via multiple macro calls as opposed to having to recreate the code multiple times in the program.

```
GETNAMES = YES /* use values in first line as variable name */ ;
```

GETNAMES is useful for when the first line of the Excel file specifies the variables that are being imported. This option should be set to NO if the first line contains data instead of variable names. Should the values contain characters that are valid for SAS then a default will be substituted. For example, since SAS variable names cannot contain a space, any spaces will be converted to underscores.

```
MIXED = YES /* evaluate data type -- default is NO */ ;
```

MIXED provides a "mixed" set of advantages, so to speak. If MIXED is set to NO, as is the default, then the determination as to whether a column contains numeric or character data is made by assessing what the majority data type is in that given column and setting it accordingly. If character data is then found in a column that has been determined to be numeric, it is set to missing and vice versa. This can create unexpected results or have unintended consequences if it is not possible or practical to review all data values. However, if MIXED is set to YES, then an

instance of character data in a numeric column will cause it to be set as character while an all numeric column will be maintained as numeric. This helps to prevent a loss of data, but may then require certain conversions to the data set.

This raises the question of how much of a data sample SAS should look at to determine whether a column's data type is really numeric or character. A registry edit may be necessary in order for the data to come in correctly. Here are the steps, but **as with any RegEdit, consider saving a copy of your current settings before making any changes**, particularly if you are new to the process.

1. Run RegEdit.
2. Navigate to HKEY_LOCAL_MACHINE -> Software then Wow6432Node -> Microsoft then Office -> 14.0 -> Access Connectivity Engine -> Engines.
3. Expand Engines, click on Excel and in the right-hand pane, look at the value for TypeGuessRows.
4. If you want SAS to assess all the data in a column before making the character versus numeric data type, set the value for TypeGuessRows to 0 (the default is 8).

```
SCANTEXT = YES /* only relevant for character data */ ;
```

SCANTEXT only applies if any of the columns could end up being considered character data. Specifying YES means that SAS is evaluating the data to determine the longest value and thus set the variable width. This has the advantage of helping to avoid data truncation that might otherwise occur, but the disadvantage of establishing a connection to the spreadsheet which means that the data cannot be updated while interactively working with the code. Specifying NO means the engine behind the import sets the variable width. Understanding the content of the spreadsheet will help to determine which value is more advantageous for a particular import scenario

```
USEDATE = YES /* default format is DATE9. */ ;
```

SAS date and datetime values are a marvelous way to store what can be long or complicated values, but in their native form, they are not readily comprehensible. By specifying YES for the USEDATE option, the DATE9. format is applied to a column that is being read in as holding date values. While DATE9. is not my preferred date format (I am partial to the more visually pleasing MMDDYY10.), changing the format once the data are read in is easy to do. The important piece of the import step is that all the dates comes through in an easy to understand way, in part to help anyone looking at the data set and in part to make it easier to verify that all data are appearing as expected. If this option is set to NO then a date format is not applied (though one can certainly be applied in a later step if so desired).

```
SCANTIME=YES;
```

SCANTIME works like USEDATE only for time values. They are included here only for completeness. If your data will not be including times, then it is not necessary, though its inclusion may be valuable if the contents of the sheet or workbook are not yet known or are too large for easy visual assessment.

```
RUN;
```

Whatever mix of options is deemed appropriate, PROC IMPORT does conclude with a RUN statement. The PROC IMPORT wizard can be used to generate the "baseline" code which can be saved for placement in a larger program and the options can be tweaked from there in order to establish the best fit for a project.

In summary, here is a complete code block for this process with the options as shown:

```
PROC IMPORT OUT = SASData.ExcelData_Sheet1
      DATAFILE = "C:\Project\User\DataLocation\ExcelData.xlsx"
      DBMS = EXCEL REPLACE /* use of REPLACE is optional */ ;
      RANGE = "Sheet1$" /* use appropriate sheet name, leave $ in place */ ;
      GETNAMES = YES /* use values in first line as variable name */ ;
      MIXED = YES /* evaluate data type -- default is NO */ ;
      SCANTEXT = YES /* only relevant for character data */ ;
      USEDATE = YES /* default format is DATE9. */ ;
      SCANTIME=YES;
RUN;
```

ZIP

There may have been a whole collection of files to be read in that have come to you conveniently packaged in a ZIP file. Those can be accessed using a FILENAME statement that specifies the ZIP method. SAS is treating the ZIP file like a directory and thus provides that level of functionality.

```
FILENAME seeZIP ZIP "ProjectData.ZIP" ;
```

With that connection established, the contents of the ZIP file can be explored, including through the direct reading of text files.

```
DATA ProjectTable01 ;
      INFILE seeZIP(ProjectFile01.txt) firstOBS = 3 DSD DLM = '09'x ;
      /* set options and delimiters as needed to explore file */
      INPUT
      <SPECIFY VARIABLES>
      ...
      ...
      ;
RUN;
```

This capability can be made even more specific if only one file is needed by including the specification in the FILENAME statement.

```
FILENAME seeZIP ZIP "ProjectData.ZIP" member = "ProjectFile01.txt" ;
```

This capability is actually useful even when you do NOT know all the file names or even how many there are. The members of the ZIP file can be read and those values could be passed via a macro call into the FILENAME statement and thus the contents of the ZIP file could be handled on the fly.

BUILDING THE BOX

Once all the data sets have been converted or otherwise prepared and are ready to go, it is easier to generate one ZIP “master file” to send as opposed to having a whole collection. Doing so also facilitates easier receipt or downloading of the data for a client. Beyond the tidy packaging, ZIP files provide a level of compression (the effect of which could be enhanced by also applying SAS’ compression tools while the data sets are being created in the first place) that can be advantageous for long term storage. Particularly during the final phase of project clean up, data and files that need to be archived can be conveniently grouped by task or phase while also minimizing the amount of storage space they require.

ZIP

Utilizing SAS to create a ZIP file is done via the ODS package.

```
ODS PACKAGE ( ClinicalProjectZIP ) OPEN nopf ;
```

The call for a new ZIP file needs to be established. Since ODS Package originally allowed sharing from a SAS stored process, it required a PackageMetaData entry be created to allow for interpretation of the results. That information is not needed in this process and thus NOPF suppresses its creation.

```
ODS PACKAGE ( ClinicalProjectZIP ) ADD file =
            "C:\Project\User\DataLocation\ClinicalForm01" ;
```

The relevant files to be added are named. If the list was extensive or needed to routinely change, this would be an appropriate place to create a macro that would allow for easy modification of what to include.

```
ODS PACKAGE ( ClinicalProjectZIP ) ADD file = ClinicalForm02 ;
```

If a file is established via a FILEREF then the full location does not need to be used in order for the file to be added to the ZIP archive.

```
ODS PACKAGE ( ClinicalProjectZIP ) ADD file = AnnotatedForm01
            path = "Documentation\" ;
```

If subfolders are desired within the archive, they can be specified with the ADD statement for a particular file. Organizational schemas can be customized to fit the needs of a certain project or the requests of a client. The subfolder structure that will be needed for the archive should be mirrored in the original data location.

```
ODS PACKAGE ( ClinicalProjectZIP ) PUBLISH archive
            PROPERTIES (
                  archive_name = "ClinicalProject.zip"
                  /* specifies the name of the ZIP file */
                  archive_path = "C:\Project\User\Delivery"
                  /* specifies location of files to be ZIPped */
                        );
```

The ZIP file, or archive, is created with a PUBLISH statement that sets not only the name of the ZIP file but also the location from which the data is being drawn.

```
ODS PACKAGE ( ClinicalProjectZIP ) close ;
```

Upon closing, the process is executed and the ZIP file is created. Currently, the application of encryption to the ZIP file is not supported within the SAS process. If this is necessary due to security concerns or project demands, that step would need to be undertaken separately.

If you need to ensure that your “box” is empty before beginning this process or if you want an easy way to start over with a particular delivery, SAS also allows you to delete an existing ZIP file.

```
FILENAME checkZIP "C:\Project\User\Delivery\ClinicalProject.zip" ;
/* establish a FILENAME statement for the ZIP file */
DATA _null_;
      IF ( fexist ('checkZIP') ) then rc = fdelete( 'checkZIP' ) ;
RUN ;
```

FEXIST as a function looks for the presence of the file specified in the FILENAME statement and if that function returns true, then the follow up function, FDELETE, removes the file, in this case the prior version of the ZIP file with the same name as the one now being created.

```
FILENAME checkZIP clear;
/* clear the FILENAME here to complete the fresh start */
```

In summary, here is a complete code block for this process:

```
ODS PACKAGE ( ClinicalProjectZIP ) OPEN nopf ;
ODS PACKAGE ( ClinicalProjectZIP ) ADD file =
ODS PACKAGE ( ClinicalProjectZIP ) ADD file = ClinicalForm02 ;
ODS PACKAGE ( ClinicalProjectZIP ) ADD file = AnnotatedForm01
            path = "Documentation\" ;
ODS PACKAGE ( ClinicalProjectZIP ) PUBLISH archive
            PROPERTIES (
                  archive_name = "ClinicalProject.zip"
                  /* specifies the name of the ZIP file */
                  archive_path = "C:\Project\User\Delivery"
                  /* specifies location of files to be ZIPped */
                        );
ODS PACKAGE ( ClinicalProjectZIP ) close ;

FILENAME checkZIP "C:\Project\User\Delivery\ClinicalProject.zip" ;
/* establish a FILENAME statement for the ZIP file */

DATA _null_;
      IF ( fexist ('checkZIP') ) then rc = fdelete( 'checkZIP' ) ;
RUN ;

FILENAME checkZIP clear;
/* clear the FILENAME here to complete the fresh start */
```

Universal Exports

In preparing everything for delivery and ZIPping, you do not need to be James Bond in order to use the flexible capability of PROC EXPORT. As with PROC IMPORT, the wizard can provide a solid base of code which can be incorporated into a larger program, but understanding the fundamentals of what that code is doing is necessary to make sure it operates as needed and can be customized to fit project needs.

SPSS and STATA

Creating an SPSS data set from a SAS is just as straightforward as bringing that SPSS data into SAS in the first place.

```
PROC EXPORT data = SASData.SPSSdata
            outfile = "C:\Project\User\DataLocation\SPSSData.sav" ;
RUN;
```

In this case, the data set is specified and the OUTFILE statement needs to use the proper file extension for SPSS. For SPSS version 14 and above, SAS data sets can be read in directly, but in providing more complete service for a client, it may be better practice to deliver the data sets already converted, especially given how straightforward it can be. If compatibility issues end up interfering, use of Stat/Tranfer may be needed.

The syntax to perform this conversion for STATA is very similar, with the file extension again being the critical difference.

```
PROC EXPORT data = SASData.STATAdata
            outfile= "C:\Project\User\DataLocation\STATAData.dta";
RUN;
```

The use of Stat/Transfer is also once again possible or potentially the creation of a SAS XPORT file, which provides another format of file that STATA is able to read.

Excel

When it comes to exporting SAS data into Excel, you are highly encouraged to examine the work of Vincent DelGobbo. Whether it is creating multi-sheet workbooks, dealing with formatting issues, customizing output or exploiting the capabilities provided by XML/Excel compatibility, his work is comprehensive and highly instructional. For the purposes of this discussion packing data, we are reviewing the basics of using PROC EXPORT to parallel the opening discussion of how to implement PROC IMPORT for the reverse task. Many of us deal with clients or colleagues who either prefer Excel output or want to perform a quick review that is visually easier in Excel. Understanding the fundamentals of PROC EXPORT facilitates this type of work while DelGobbo's much more advanced and exhaustive coverage of the topic would allow in-depth reporting to take place via Excel out of SAS.

```
PROC EXPORT DATA = SASData.ClinicalForm01
```

The initial statement needs to specify the location of the data and the name of the data set. As with PROC IMPORT, the default would be to look to the WORK folder, but any established LIBNAME can be called upon in this context.

```
OUTFILE = "C:\Project\User\Delivery\ClinicalForm01.xlsx"
/* extensions like XLS or XLSM can also be used */
```

The OUTFILE statement provides three key pieces of information: the location in which the Excel workbook should be placed, the name of the workbook and the particular Excel file extension. If a client has a need for an earlier version of Excel compatibility (say XLS versus XLSX), this is where that can be set.

```
DBMS = EXCEL LABEL replace ;
/* use REPLACE with caution */
```

The DBMS option again needs to be set to Excel as was the case with PROC IMPORT, but the additional components of this statement may vary depending on your needs and the amount of control you want to have over the creation of the workbook. Specifying LABEL will cause the SAS labels to be used as the column headers rather than the SAS variable names. If a label does not exist, then the variable name will be used by default. This is an

excellent option to employ as often the people who want Excel output are not necessarily the one who are most familiar with the variables and thus the presumably more descriptive labels would be of more help to them. The use of REPLACE allows SAS to overwrite an existing Excel workbook to create the one being requested in the proc. This is useful when you are interactively refining the output for a delivery, but may be problematic if version control is an issue. Consider not using this option if overwriting would create any sort of problem.

```
SHEET = "Form01";
```

The SHEET option is used to set the name for the workbook page being generated. This is not required and the sheet will default to Excel's Sheet1 if a value is not provided. However, this is an easy place to provide more specific information about the workbook's contents and should be employed when possible. The length of this value has the same restrictions as if it were being manually entered into Excel.

```
NEWFILE = YES ;
```

NEWFILE works with REPLACE to allow the workbook to be fully recreated if it already exists.

```
RUN;
```

Finally, as with bringing data in, PROC EXPORT does require a RUN in order to execute.

In summary, here is a complete code block for this process:

```
PROC EXPORT DATA = SASData.ClinicalForm01
      OUTFILE = "C:\Project\User\Delivery\ClinicalForm01.xlsx"
      /* extensions like XLS or XLSM can also be used */
      DBMS = EXCEL LABEL replace ;
      /* use REPLACE with caution */
      SHEET = "Form01";
      NEWFILE = YES ;
RUN;
```

A quick manual inspection of any Excel workbook is always a good idea to ensure that the product created is the product intended. The behavior of the spreadsheet can be anticipated and controlled extensively through SAS, but a conversion is still being made between two very different data systems.

CONCLUSION

Even for programmers who work primarily in SAS, we need to maintain flexibility when it comes to the format of data. Clients may have a particular need, a certain project may require different types of collaboration or we may just be looking for more efficient ways to store data long term. This review of importing, exporting and ZIPping capabilities within SAS provides some basic tools for moving between formats, creating deliveries that provide an ease-of-use experience and generating some simple storage options. Flat pack is not just for dorm rooms and DIY decorating – it can also facilitate your experiences with data in SAS!

REFERENCES

- SAS. "EXPORT Procedure." Base SAS 9.3 Procedures Guide, Second Edition. 2014. Available at http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#n045uxf7ll2p5on1ly4at3vpd47e.htm
- Hemedinger, Chris. "Reading and Updating ZIP Files with FILENAME ZIP." The SAS Dummy. January 29, 2014. Available at http://blogs.sas.com/content/sasdummy/2014/01/29/using-filename-zip/.
- Hemedinger, Chris. "Using SAS and ODS Package to Create ZIP Files." The SAS Dummy. January 28, 2014. Available at http://blogs.sas.com/content/sasdummy/2014/01/28/create-zip-ods-package/.

- SAS. “IMPORT Procedure.” Base SAS 9.3 Procedures Guide, Second Edition. 2014. Available at http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#n18jyszn33umngn14czw2qfw7thc.htm

ACKNOWLEDGMENTS

I would like to thank Rick Mitchell for encouraging me to continue to write and being a SAS conference inspiration. I would like to thank Michael Raithel at Westat for his patience and editorial support. I would like to thank Westat for their institutional support of my participation in both local and regional SAS users’ groups.

CONTACT INFORMATION

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

Sarah Woodruff
Westat
1600 Research Boulevard, WB 401
Rockville, MD 20850
240-314-7562
SarahWoodruff@westat.com

DISCLAIMERS

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.