SESUG Paper 213-2018

Accessing Password Protected Microsoft® Excel Files in A SAS® Grid Environment

Brandon Welch, Rho® Inc.; Travis Mason, Rho® Inc.

ABSTRACT

Microsoft® Excel continues as a popular choice for data storage and manipulation. Extracting data from Excel files is often challenging. If a file is password-protected, the challenge intensifies. DDE (Dynamic Data Exchange) is the classic approach to reading password- protected Excel files. The DDE approach fails in a SAS grid environment. This paper illustrates an alternative using a Visual Basic Scripting (VBScript) file. The script is built within a SAS program and executed with a X command, then the text file is accessed via PROC IMPORT. The methods presented in this paper highlight SAS techniques that will educate programmers at all levels. In addition, the reader will learn how to build a simple VBScript file.

INTRODUCTION

This paper introduces a simple approach to opening a password-protected Excel file. We assume the user is working in a Windows-based SAS grid environment. At the time of writing this paper, we operated in a Windows 7 environment.

The program is organized into three parts. In the first part, we build the VBScript in a DATA _NULL_ step and output a VBScript file using the FILE statement. In this script we explicitly reference the password used to unlock the file. The script also saves the Excel file as a text file – which is not easily password-protected. Secondly, we use the X command to execute the script, which unlocks the file and saves it as a text file. Finally, PROC IMPORT is used to read the text file into a SAS data set.

BUILDING THE VBSCRIPT

In its raw form the script looks like this:

```
On Error Resume Next
Set objExcel = CreateObject("Excel.Application")
objExcel.Application.DisplayAlerts = False
objExcel.Application.Visible = False
set objSourcebook =
objExcel.Workbooks.open("input_path\input_file.xlsx",,,,"password")
Set objSourceSheet = objExcel.ActiveWorkbook.Worksheets(1)
numRows = objSourceSheet.UsedRange.Rows.Count
numCols = objSourceSheet.UsedRange.Columns.Count
i = 1
Do While i < (numRows + 1)
  j = 1
  Do While j < (numCols + 1)
    CellValue = objSourceSheet.Cells(i,j).Value
    If InStr(CellValue,Chr(13)) or InStr(CellValue,Chr(10)) Then
      objSourceSheet.Cells(i,j).Value = Replace(CellValue,Chr(13)," ")
      objSourceSheet.Cells(i,j).Value = Replace(CellValue,Chr(10)," ")
    End If
    j = j + 1
    Loop
  i = i + 1
Loop
objSourcebook.SaveAs "output_path\output_file.txt", -4158
objSourcebook.Close
objExcel.Application.Quit
```

Our goal is to build this using a DATA _NULL_ step, but first we break down the important pieces of the script.

PART 1: CREATING OBJECTS AND USING PROPERTIES

```
On Error Resume Next
Set objExcel = CreateObject("Excel.Application")
objExcel.Application.DisplayAlerts = False
objExcel.Application.Visible = False
```

The first line ensures that if an error occurs, the script will continue to execute. This is important to include, as it prevents the script from locking up or 'hanging' on the SAS grid.

To further prevent potential hang-ups, we run the script invisibly and suppress any pop-up windows; to accomplish this we create an Excel application object and use the properties: `DisplayAlerts` and `Visible`. These properties prevent any pop-ups from appearing (e.g. "Do you want to save the changes you made to…") and make sure Excel does not visibly open and run. We want Excel to run in the background.

PART 2: MORE OBJECTS AND COUNTING ROWS AND COLUMNS

```
set objSourcebook =
   objExcel.Workbooks.open("input_path\input_file.xlsx",,,,"password")
Set objSourceSheet = objExcel.ActiveWorkbook.Worksheets(1)
numRows = objSourceSheet.UsedRange.Rows.Count
numCols = objSourceSheet.UsedRange.Columns.Count
```

We create two more objects: one for the workbook, and one for the worksheet. For the workbook object, we use the `open()` method to open the file and apply the password. Note: we assume the first (left-most) worksheet with `Worksheets(1)`.

When data are manually entered into a spreadsheet, unwanted hidden characters are likely to appear. The most common culprits are carriage return and line feed. When read into SAS data sets, these characters are a nuisance. We replace these characters with white space in a while loop. However, to loop through each cell, we first need to determine the number of rows and columns in the Excel file. We use the `UsedRange` and `Count` properties to do just that.

PART 3: REMOVING UNWANTED CHARACTERS AND SAVING/CLOSING

```
i = 1
Do While i < (numRows + 1)
  j = 1
  Do While j < (numCols + 1)
    CellValue = objSourceSheet.Cells(i,j).Value
    If InStr(CellValue,Chr(13)) or InStr(CellValue,Chr(10)) Then
      objSourceSheet.Cells(i,j).Value = Replace(CellValue,Chr(13)," ")
      objSourceSheet.Cells(i,j).Value = Replace(CellValue,Chr(10)," ")
    End If
    j = j + 1
    Loop
  i = i + 1
Loop
objSourcebook.SaveAs "output_path\output_file.txt", -4158
objSourcebook.Close
objExcel.Application.Quit
```

Once we know the number of rows and columns, we loop through each cell. Within the loop, we use the `InStr` function to detect a carriage return (ASCII collating sequence = 13) or a line feed (ASCII collating sequence = 10). The function `InStr` operates similar to the INDEX function in SAS. Once a hidden

character is found, we use the `Replace` function to render the character to a space – similar to TRANWRD in SAS.

In the final step, we save the workbook as a text file, close the workbook, and quit the Excel application. Note that -4158 is the XlFileFormat enumeration for xlCurrentPlatformText. We include this value to save the file as text.

OUTPUT VBSCRIPT IN DATA _NULL_

Taking what we have above, in the SAS DATA _NULL_ step:

```
%let FilePath =;
%let FileNme  =;
%let Password =;

filename tmp "output_path\PWBreak.vbs";
DATA _null_;
  file tmp;
  put 'On Error Resume Next';
  put 'Set objExcel = CreateObject("Excel.Application")';
  put 'objExcel.Application.DisplayAlerts = False';
  put 'objExcel.Application.Visible = False';
  put %unquote(%nrbquote('set objSourcebook =
    objExcel.Workbooks.open("&FilePath.\&FileNme..xlsx",,,,"&Password")'));
  put 'Set objSourceSheet = objExcel.ActiveWorkbook.Worksheets(1)';
  put 'numRows = objSourceSheet.UsedRange.Rows.Count';
  put 'numCols = objSourceSheet.UsedRange.Columns.Count';
  put 'i = 1';
  put 'Do While i < (numRows + 1)';
  put '  j = 1';
  put '  Do While j < (numCols + 1)';
  put '    CellValue = objSourceSheet.Cells(i,j).Value';
  put '    If InStr(CellValue,Chr(13)) or InStr(CellValue,Chr(10)) Then';
  put '      objSourceSheet.Cells(i,j).Value = Replace(CellValue,Chr(13)," ")';
  put '      objSourceSheet.Cells(i,j).Value = Replace(CellValue,Chr(10)," ")';
  put '    End If';
  put '    j = j + 1';
  put '    Loop';
  put '  i = i + 1';
  put 'Loop';
  put %unquote(%nrbquote('objSourcebook.SaveAs "&FilePath.\&FileNme..txt", -
    4158'));
  put 'objSourcebook.Close';
  put 'objExcel.Application.Quit';
RUN;
```

Note the only differences are the use of macro variables `FilePath`, `FileNme`, and `Password` as well as `%unquote(%nrbquote())`. We use %`unquote(%nrbquote())` so the macro variables resolve in single quotes. Single quotes are not used in VBScript for text strings.

RUN THE SCRIPT AND IMPORT THE TEXT FILE

```
x "c:\windows\System32\cscript.exe &FilePath.\PWBreak.vbs";

%let rc = %sysfunc(sleep(3,1));

libname outdat 'input_folder';
PROC IMPORT datafile="&FilePath.\&FileNme..txt"
  out = outdat.from_text
  dbms = dlm
  replace;
  delimiter='09'x;
```

```
RUN;
libname outdat clear;
```

We use the X command to run the script. The path of the executable cscript.exe may vary depending on your Windows installation. The SLEEP function is used to prevent the PROC IMPORT from running prematurely.

CONCLUSION

This paper illustrates a simple approach to reading a password-protected Excel file. This technique works well in most situations, but there is room for improvement. For example, the script loops through each cell of the Excel file and replaces control characters with white space. This is time-consuming for large files. A possible solution is using a technique similar to the `Range.Replace` method in Visual Basic for Applications (VBA). However, even with this shortcoming, we feel this paper provides the user a good starting place if they encounter a password-protected Excel file.

REFERENCES

VBScript Language Reference. 2015. Accessed August 8, 2018. https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/scripting-articles/d1wf56tt(v=vs.84).

ACKNOWLEDGMENTS

Eva J. Welch

Steve Noga

CONTACT INFORMATION

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

Brandon Welch
Rho Inc.
919-595-6592
Brandon_Welch@rhoworld.com

APPENDIX

```
%let FilePath =;
%let FileNme  =;
%let Password =;

filename tmp "output_path\PWBreak.vbs";
DATA _null_;
  file tmp;
  put 'On Error Resume Next';
  put 'Set objExcel = CreateObject("Excel.Application")';
  put 'objExcel.Application.DisplayAlerts = False';
  put 'objExcel.Application.Visible = False';
  put %unquote(%nrbquote('set objSourcebook =
    objExcel.Workbooks.open("&FilePath.\&FileNme..xlsx",,,,"&Password")'));
  put 'Set objSourceSheet = objExcel.ActiveWorkbook.Worksheets(1)';
  put 'numRows = objSourceSheet.UsedRange.Rows.Count';
  put 'numCols = objSourceSheet.UsedRange.Columns.Count';
  put 'i = 1';
  put 'Do While i < (numRows + 1)';
  put '  j = 1';
  put '  Do While j < (numCols + 1)';
  put '    CellValue = objSourceSheet.Cells(i,j).Value';
  put '    If InStr(CellValue,Chr(13)) or InStr(CellValue,Chr(10)) Then';
  put '      objSourceSheet.Cells(i,j).Value = Replace(CellValue,Chr(13)," ")';
```

```
  put '      objSourceSheet.Cells(i,j).Value = Replace(CellValue,Chr(10)," ")';
  put '    End If';
  put '    j = j + 1';
  put '    Loop';
  put '  i = i + 1';
  put 'Loop';
  put %unquote(%nrbquote('objSourcebook.SaveAs "&FilePath.\&FileNme..txt", -
    4158'));
  put 'objSourcebook.Close';
  put 'objExcel.Application.Quit';
RUN;


/*run script*/
x "c:\windows\System32\cscript.exe &FilePath.\PWBreak.vbs";

%let rc = %sysfunc(sleep(3,1));

libname outdat 'input_folder';
PROC IMPORT datafile="&FilePath.\&FileNme..txt"
  out = outdat.travis
  dbms = dlm
  replace;
  delimiter='09'x;
RUN;
libname outdat clear;
```