Paper SM17

Programmatically Modifying a Document Created by ODS Excel: Adding a Password and Concatenating Multiple Worksheets into One

Jeffrey Meyers, Regeneron Pharmaceuticals, New Jersey, USA

ABSTRACT

The Microsoft Excel destination was added to the SAS output delivery system (ODS) in version 9.4M3 and offered an improvement over the EXCELXP tag sets destination. The Excel destination allows SAS users to create workbooks in the XLSX format instead of the XML format which reduces the file size and unlocks additional functionality. This paper focuses on two advanced tricks when using the Excel destination: password protection and clickable navigation. The Excel destination can create a "protected" workbook but does not include the option to include a password which allows any user to simply remove the protection from within Excel. This paper will show techniques to add password protection to the Excel file and how to selectively choose which worksheets or columns are protected. The second focus of this paper, clickable navigation, allows the user to click on a cell within a worksheet to jump to another cell anywhere in the workbook leading to very fluid and dynamic reports.

INTRODUCTION

Microsoft Excel workbooks are fantastic destination for outputting detailed reports. Excel workbooks have multiple worksheets, are nearly unlimited in height and width, and are generally smaller in file size. SAS added the EXCEL output delivery system (ODS) destination in version 9.4M3 as an upgrade to the EXCELXP tag set destination. The EXCELXP tag set output an XML file renamed with an XLS file type and the EXCEL destination outputs a full XLSX file. This was a great improvement as XLSX files are typically less file size than XML files. The EXCEL destination comes with a number of options for formatting the worksheets, but one option is glaringly absent: the ability to password protect the workbook. The ODS destination has an option to protect the worksheets, but this does not include adding a password so is easily removed. This paper's first topic is to programmatically add this functionality to the Excel destination as password protected workbooks are needed within the industry.

The second topic of this paper addresses a common problem with the EXCEL destination: running out of memory. Outputting a table to Excel is very memory intensive in SAS, and adding any formatting, traffic highlighting, and other style modifications all require additional memory. Upon getting the error that SAS has run out of memory there are few options available to avoid this. The first is to output less rows which defeats the purpose of outputting to Excel in the first place, and the second is to split the table into multiple worksheets as each worksheet is output individually and smaller sheets take less memory to print. The issue with splitting into multiple worksheets is that the consumer wanted the table or listing as one unit, and now there is manual work to combine the tables back together. This paper will walk through several attempts at combining these worksheets back together programmatically in a way that preserves all of the style modifications.

PASSWORD PROTECTING WORKBOOKS

Password protecting a workbook is necessary in many industries to prevent accidental or intentional changes to data by internal users or when sending to an external site. Workbooks can be protected at three different levels. The first is at the worksheet level which disables the ability to edit cells within the worksheet. The second is the workbook level which disables the ability to rename, add, or remove worksheets in the workbook. This also prevents some worksheet actions such as moving columns and resizing columns. The third is the file level which prevents the workbook from being opened without the correct password.

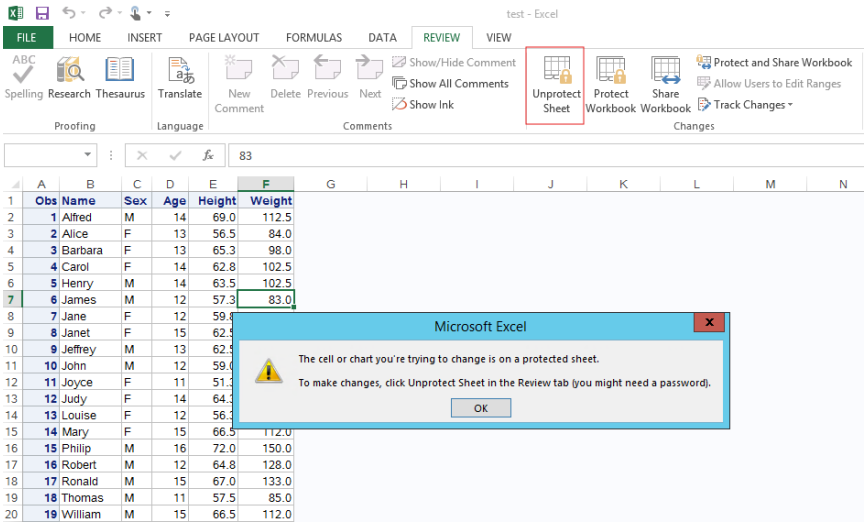
This paper provides methods to password protect at the first two levels listed above. The third level of password protecting the entire file changes the structure of the XLSX file and the methods within this paper do not currently work to apply a password to opening a file.

SAS DEFAULT OPTION

The ODS Excel statement has the option to turn on sheet protection making the worksheet read only upon creation. The following code will create a simple read only worksheet:

```
ods excel file='test.xlsx' options(protect_worksheet='on');

proc print data=sashelp.class;

run;

ods excel close;
```

Opening the Excel document displays the data from the CLASS data set from the default SASHELP library. Attempting to modify any of the cells causes the following message to appear:



	Obs	Name	Sex	Age	Height	Weight
1	1	Alfred	M	14	69.0	112.5
2	2	Alice	F	13	56.5	84.0
3	3	Barbara	F	13	65.3	98.0
4	4	Carol	F	14	62.8	102.5
5	5	Henry	M	14	63.5	102.5
6	6	James	M	12	57.3	83.0
7	7	Jane	F	12	59.8	
8	8	Janet	F	15	62.5	
9	9	Jeffrey	M	13	62.5	
10	10	John	M	12	59.0	
11	11	Joyce	F	11	51.3	
12	12	Judy	F	14	64.3	
13	13	Louise	F	12	56.3	
14	14	Mary	F	15	66.5	112.0
15	15	Philip	M	16	72.0	150.0
16	16	Robert	M	12	64.8	128.0
17	17	Ronald	M	15	67.0	133.0
18	18	Thomas	M	11	57.5	85.0
19	19	William	M	15	66.5	112.0

Attempting to modify the sheet displays the worksheet is protected, and within the Review tab gives the option to "Unprotect Sheet" also indicating the sheet is protected. The ODS Excel option successfully creates a worksheet with the protected status enabled but is limited by one critical issue. There is no password protection available to stop a user from simply clicking the "Unprotect Sheet" button from the Ribbon (highlighted in red above) to remove this protection at any time. The user can manually add a password to the workbook after it is already created, but this is manual work where a programmatic method would be preferred.

CREATING A PASSWORD PROTECTED WORKBOOK PROGRAMMATICALLY

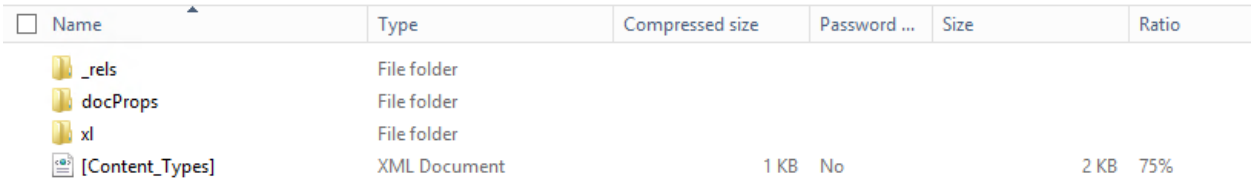
Modifying the Excel file to have a password programmatically involves entering and modifying the XML code. However, as it was stated in the introduction, one of the changes from the EXCELXP tagsets to the EXCEL destination was that files are now created in XLSX format instead of the XML format. XLSX files cannot be pulled into a data step the same way as an XML file. Digging deeper into what the XLSX file type is led to the solution for this issue.

XLSX File Type

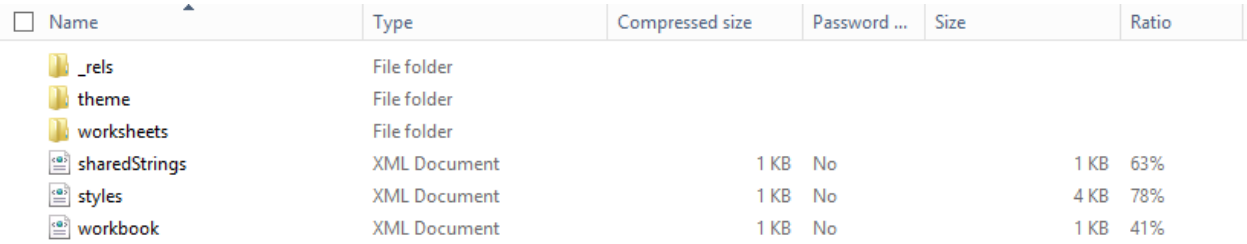
The XLSX file type is actually a Zip file with a specific organization of XML files contained within. The easiest way to see this is to rename an XLSX file to have the .zip file extension instead. Doing the rename in Microsoft Windows 11 didn't work directly, so the following code was run within SAS to do the rename:

```
data _null_;

    rc=rename('test.xlsx', 'test.zip','file');

run;
```

The zip file can then be opened in Windows Explorer to see the following contents:



The folder that contains the relevant files is "xl." Within this folder are the XML files "styles" and "workbook" and the folder "worksheets" which contains the XML file for each worksheet in the workbook:



Name	Type	Compressed size	Password ...	Size	Ratio
_rels	File folder				
theme	File folder				
worksheets	File folder				
sharedStrings	XML Document	1 KB	No	1 KB	63%
styles	XML Document	1 KB	No	4 KB	78%
workbook	XML Document	1 KB	No	1 KB	41%

The files needed include the “styles” and “workbook” XML files as well as the XML files contained in the “worksheets” folder.

Styles XML

The styles XML file contains unique values for each style element found in the workbook. Each unique font, size, color, pattern, fill, and border combination. Each of these cell elements will have its own section within the file. The different fonts included in the example test.xlsx are shown below:

```
- <fonts count="2">
    - <font>
          <sz val="9.5"/>
          <color rgb="FF000000"/>
          <name val="Albany AMT"/>
      </font>
    - <font>
          <b/>
          <sz val="9.5"/>
          <color rgb="FF112277"/>
          <name val="Albany AMT"/>
      </font>
  </fonts>
```

There are two unique fonts within the test XLSX file shown above which aligns with the screenshot of the Excel sheet shown previously. There are two different colors and only one font is bold faced. The end of the styles XML file then has a tag labeled “cellXfs” which has one set for every element combination in the file:

```
- <cellXfs count="6">
    - <xf numFmtId="0" borderId="0" fillId="2" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
          <alignment horizontal="left"/>
      </xf>
    - <xf numFmtId="0" borderId="1" fillId="3" fontId="1" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
          <alignment horizontal="right"/>
      </xf>
    - <xf numFmtId="0" borderId="1" fillId="3" fontId="1" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
          <alignment horizontal="left"/>
      </xf>
    - <xf numFmtId="0" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
          <alignment horizontal="left"/>
      </xf>
    - <xf numFmtId="0" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
          <alignment horizontal="right"/>
      </xf>
    - <xf numFmtId="164" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0" applyNumberFormat="0">
          <alignment horizontal="right"/>
      </xf>
  </cellXfs>
```

There are six unique combinations of style elements and options in the test XLSX file. The different elements are referenced with ID numbers such as “fontID.” The numbers in these references are the N+1th value listed in that elements section. When fontID=”0” that refers to the first listed font within the font elements section. Other options such as applyBorder are binary on or off values to indicate if the option is enabled or disabled. Certain options are then contained within the initial tag (“xf”) and the closing tag (“/xf”) such as the “alignment horizontal” option in Display 5’s example.

Workbook XML

The workbook XML contains any global options for the entire workbook and meta information for each worksheet in the workbook such as the sheet name:

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<workbook xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main">
    <fileVersion rupBuild="9302" lowestEdited="5" lastEdited="5" appName="xl"/>
    <workbookPr autoCompressPictures="0" showInkAnnotation="0"/>
  - <bookViews>
        <workbookView/>
    </bookViews>
  - <sheets>
        <sheet r:id="rId1" sheetId="1" name="Print 1 - Data Set SASHELP.C"/>
    </sheets>
    <calcPr concurrentCalc="0" calcId="0"/>
</workbook>
```

There is a section for global options such as “autoCompressPictures” and a section for the meta data on each worksheet.

Worksheets Folder

The worksheets folder contains one XML file for every worksheet in the workbook name sequentially sheet1, sheet2, sheet3 and so forth. These XML files includes the actual data stored in the worksheet within the “sheetData” block, column options within the “cols” block, and other various blocks for worksheet wide options. The sheet protection option that was set in the original code to make test.xlsx is found at the end of the XML:

```
<sheetProtection sheet="1" objects="1" scenarios="1"/>
<printOptions gridLines="0" verticalCentered="0" horizontalCentered="0" headings="0"/>
<pageMargins footer="0" header="0" bottom="0.5" top="0.5" right="0.05" left="0.05"/>
<pageSetup verticalDpi="300" horizontalDpi="300" orientation="portrait" scale="100" fitToHeight="1" fitToWidth="1" paperSize="1"/>
```

The sheetProtection tag shows the protect worksheet option from ODS EXCEL has been included in the XML file.

Importing the XML Data into SAS

The method described within this paper relies on pulling in the XML information, modifying it, and then pushing the changes back out to the file. There are several methods to import and XLSX into SAS including the IMPORT procedure and the XLSX libname statement, but this pulls the actual data values and not the information behind the XLSX workbook. The trick to importing the XML data is to think of the XLSX file as a .ZIP file instead. The following code assigns the XLSX file to a FILENAME with the ZIP option and then pulls the data for sheet one into a data step with the INFILE option:

```
filename inZip zip 'test.xlsx';

data sheet1;

 infile inzip("xl/worksheets/sheet1.xml") truncover;

 input code $2000. ;

 run;
```

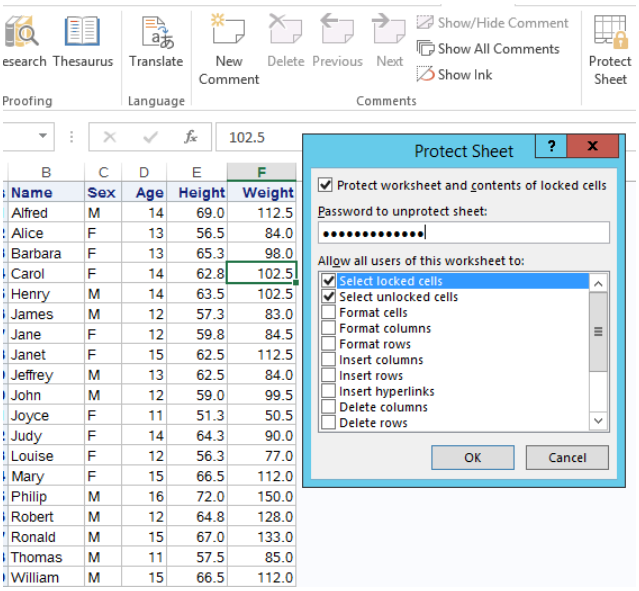
This leads to seeing the actual XML code:

```
<sheetProtection scenarios="1" objects="1" sheet="1"/> <printOptions headings="0" horizontalCentered="0" verticalCentered="0" gridLines="0"/>
<pageMargins left="0.05" right="0.05" top="0.5" bottom="0.5" header="0" footer="0"/>
<pageSetup paperSize="1" fitToWidth="1" fitToHeight="1" scale="100" orientation="portrait" horizontalDpi="300" verticalDpi="300"/>
```

The same tags from above can be seen within a data step character variable. The INFILE statement starts at the topmost folder of the referenced zip file, and any XML file within the zip file can be targeted to be pulled into a SAS data set.

Adding Password Protection

The author of this paper is not an XML programmer or expert, so the method the author used to see which XML tags are added when a password is added to a workbook is to manually add the password then go back to the XML code. The following is a screenshot of opening test.xlsx from earlier and adding a manually typed password:



Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83.0
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84.0
John	M	12	59.0	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90.0
Louise	F	12	56.3	77.0
Mary	F	15	66.5	112.0
Philip	M	16	72.0	150.0
Robert	M	12	64.8	128.0
Ronald	M	15	67.0	133.0
Thomas	M	11	57.5	85.0
William	M	15	66.5	112.0

After password protecting the worksheet the XLSX file is converted to a .ZIP file so the XML code can be viewed to find the added XML code:

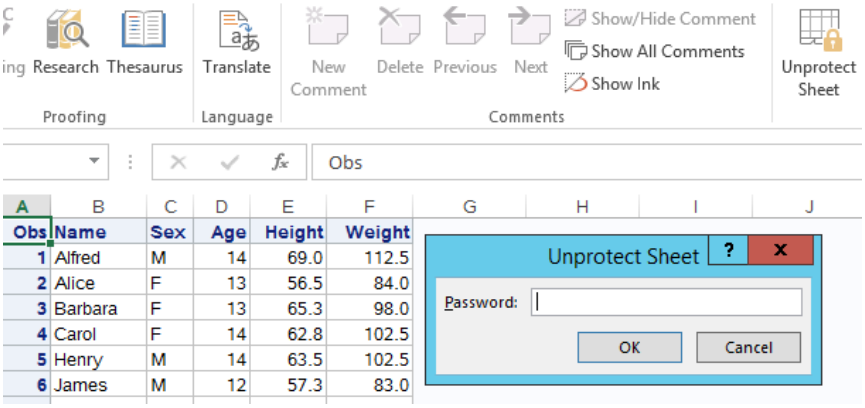
```
<sheetProtection algorithmName="SHA-512"
    hashValue="klOI1HiG3CXjIMuhxtwfwRygdOolTd0qCXYCtTLcP25uJ75TczNKZRapsHC0Wws4VApKdar8/qCMbLDBmF/pBg=="
    saltValue="HiK/bzR8cRUiYUpVMro32A==" spinCount="100000" sheet="1" objects="1" scenarios="1"/>
```

The sheetProtection code was updated with values for spinCount, saltValue, hashValue, and algorithmName parameters. The password is not readable directly from the XML file because it is encrypted, but these same encryption values can be added programmatically to create the wanted password. The following code demonstrates creating test.xlsx, importing the XML code, modifying the XML code with the password protection, and then pushing the modified XML back into test.xlsx:

```
ods excel file='test.xlsx' options(protect_worksheet='on');

proc print data=sashelp.class;

run;

ods excel close;

filename inZip zip 'test.xlsx';

data sheet1;

   infile inzip("xl/worksheets/sheet1.xml") truncover;

   input code $2000. ;

     if find(code,"<sheetProtection","i")>0 then do;

        start=find(code,'/>','i',find(code,'<sheetProtection','i'));

        code=catx(' ',substr(code,1,start-1),

              'spinCount="100000"','saltValue="HiK/bzR8cRUiYUpVMro32A==" ',


'hashValue="klOI1HiG3CXjIMuhxtwfwRygdOolTd0qCXYCtTLcP25uJ75TczNKZRapsHC0Wws4VApKdar8/qCMb
LDBmF/pBg==" ', ' formatCells="0" formatColumns="0" formatRows="0" ',

                'algorithmName="SHA-512"',

                substr(code,start));
```

```
    end;

    drop start;

run;


data _null_;

  set sheet1;

  file inzip("xl/worksheets/sheet1.xml");

  put code;

run;
```

Note that the final step to push the modified code back into test.xlsx is a null data step that sets the data set with the modified code, references the zip file with the FILE statement, and then uses a PUT statement to push the code into the file replacing it. Upon opening the file once again and clicking on the password protection button in the Review ribbon Excel now asks for a password to remove the protection:



Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5
6	James	M	12	57.3	83.0

The worksheet is now protected by a password and can be unlocked by those that know the password. The limitation with this method is that the only way to know the password is to add it manually then extract the new hash and salt values from the resulting XML file. If the worksheet should never be unlocked then a random value can be assigned.

Adding Password Protection to the Whole Workbook

Microsoft Excel has two layers of password protection available. The first is worksheet by worksheet individually, and the second is to password protect the workbook structure. Protecting the workbook structure means that users cannot add, remove, rename or move the worksheets. This does not protect the data within the worksheets alone as each worksheet must be protected separately to prevent the data from being modified. The same password can be added with the same hash and algorithm but the code needs to be added to the workbook.XML file. The password protection block needs to be completely added, and can be done with the following code:

```
data __workbook;

   infile inzip("xl/workbook.xml")  truncover ;

   input code $2000. ;

   if code='<bookViews>' then do;

      code='<workbookProtection lockStructure="1" workbookSpinCount="100000"'||

           ' workbookSaltValue="HiK/bzR8cRUiYUpVMro32A==" '||


'workbookHashValue="klOI1HiG3CXjIMuhxtwfwRygdOolTd0qCXYCtTLcP25uJ75TczNKZRapsHC0Wws4VApKd
ar8/qCMbLDBmF/pBg==" '||
```

```
            'workbookAlgorithmName="SHA-512"/>';output;

     code='<bookViews>';output;

    end;

    else output;

run;


data _null_;

  set __workbook;

  file inzip("xl/workbook.xml");

  put code;

run;
```

Bonus Trick: Allowing Individual Columns to be Modified

A recent challenge for the author was to create a report where the worksheets and workbook were password protected except for one column for adding comments. After many attempts at finding a solution for this the author discovered that the protection locked element can be added to a specific cell style in the styles.XML file. If this protection locked element is set to zero then the protected status is removed for that cell and it can be edited. The difficult part of the trick is to separate the style for the column to only apply the protection locked element to the intended target.

The solution that was found was to use a style modifier in the REPORT procedure to give that column a style that no other column in the report has. For this example the style was to make the column font italic:

```
ods excel file="test.xlsx" options(protect_worksheet="on");

proc report data=sashelp.class;

    columns name age height weight sex;

    define name / display style(column)={fontStyle=Italic};

run;

ods excel close;
```

Opening the new Excel workbook in an XML viewer the italic element can now be found in only one font ID:

```
- <fonts count="3">
  - <font>
      <sz val="9.5"/>
      <color rgb="FF000000"/>
      <name val="Albany AMT"/>
    </font>
  - <font>
      <b/>
      <sz val="9.5"/>
      <color rgb="FF112277"/>
      <name val="Albany AMT"/>
    </font>
  - <font>
      <i/>
      <sz val="9.5"/>
      <color rgb="FF000000"/>
      <name val="Albany AMT"/>
    </font>
</fonts>
```

The third font now has an element <i/> which indicates the font is italicized. The third font has a font ID of 2 because XML counts from zero and up. Looking within the cellXfs section of the same XML identifies which cell style is using font ID of 2:

```
- <cellXfs count="6">
  - <xf numFmtId="0" borderId="0" fillId="2" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
      <alignment horizontal="left"/>
    </xf>
  - <xf numFmtId="0" borderId="1" fillId="3" fontId="1" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
      <alignment horizontal="center"/>
    </xf>
  - <xf numFmtId="0" borderId="2" fillId="4" fontId="2" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
      <alignment horizontal="left"/>
    </xf>
  - <xf numFmtId="164" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0" applyNumberFormat="0">
      <alignment horizontal="right"/>
    </xf>
  - <xf numFmtId="165" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0" applyNumberFormat="0">
      <alignment horizontal="right"/>
    </xf>
  - <xf numFmtId="0" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
      <alignment horizontal="left"/>
    </xf>
  </cellXfs>
```

The third cell format is the only one that contains the font ID of two which contains the italic font. All that remains is to import the XML code into a SAS dataset, programmatically identify the correct font ID, use the font ID to find the correct cell format, and inject the protection locked element to that specific cell format. The following code performs these steps:

```
data __styles;

    infile inzip("xl/styles.xml")  truncover ;

    input code $2000. ;

    if find(code,'<fonts','i') then _s=-1;

    else if find(code,'<font>','i') then _s+1;

    else if code='</fonts>' then _s=.;

    length mark $50.;

    if find(code,'<i','i') and _s>=0 then do;

        mark='fontID="'||strip(put(_s,12.0))||'"';

        delete;

    end;

    retain mark mark2;


    if find(code,'<xf ','i') and find(code,strip(mark),'i') then mark2=1;

    if mark2=1 and find(code,'<alignment','i') then do;

        output;

        code='<protection locked="0"/>';output;

    end;

    else if code='</xf>' then do;

        call missing(mark2);

        output;

    end;

    else output;

run;
```

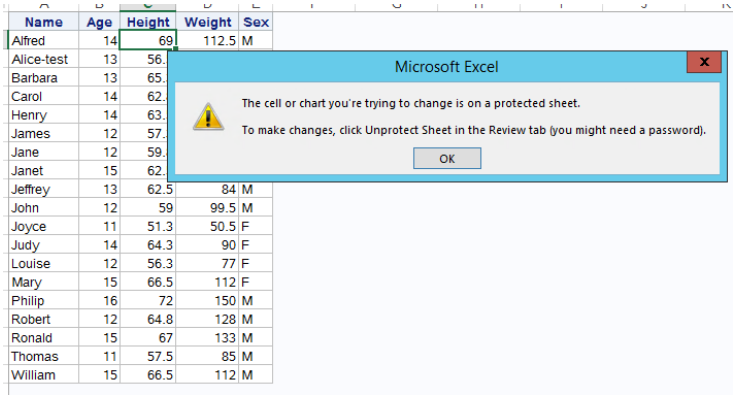
```
data _null_;

    set __styles;

    file inzip("xl/styles.xml");

    put code;

run;
```

Note that the code also deletes the italic tag as this style is not preferred in the final report. The injection of the protection status can be seen in the dataset:

```
- <cellXfs count="6">
  - <xf numFmtId="0" borderId="0" fillId="2" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
      <alignment horizontal="left"/>
    </xf>
  - <xf numFmtId="0" borderId="1" fillId="3" fontId="1" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
      <alignment horizontal="center"/>
    </xf>
  - <xf numFmtId="0" borderId="2" fillId="4" fontId="2" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
      <alignment horizontal="left"/>
      <protection locked="0"/>
    </xf>
  - <xf numFmtId="164" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0" applyNumberFormat="0">
      <alignment horizontal="right"/>
    </xf>
  - <xf numFmtId="165" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0" applyNumberFormat="0">
      <alignment horizontal="right"/>
    </xf>
  - <xf numFmtId="0" borderId="2" fillId="4" fontId="0" applyAlignment="1" applyBorder="1" applyFill="1" applyFont="1" xfId="0">
      <alignment horizontal="left"/>
    </xf>
</cellXfs>
```

The third cell format now contains the protection locked=”0” element. The final test is to open the Excel report and attempt to modify the Name column in the worksheet along with the other columns.

	Name	Age	Height	Weight	Sex
2	Alfred	14	69	112.5	M
3	Alice-test	13	56.5	84	F
4	Barbara	13	65.3	98	F
5	Carol	14	62.8	102.5	F
6	Henry	14	63.5	102.5	M
7	James	12	57.3	83	M
8	Jane	12	59.8	84.5	F
9	Janet	15	62.5	112.5	F
10	Jeffrey	13	62.5	84	M
11	John	12	59	99.5	M
12	Joyce	11	51.3	50.5	F
13	Judy	14	64.3	90	F
14	Louise	12	56.3	77	F
15	Mary	15	66.5	112	F
16	Philip	16	72	150	M
17	Robert	12	64.8	128	M
18	Ronald	15	67	133	M
19	Thomas	11	57.5	85	M
20	William	15	66.5	112	M



The Name column can be modified while the Height column still contains the protected status. The test is successful in that the Name column can be modified but the other columns remain protected in the worksheet.

CONCATENATING WORKSHEETS

The EXCEL destination is a very memory intensive when combined with printing procedures such as PRINT and REPORT. Longer, wider and/or style heavy tables can commonly run into SAS giving an error that it ran out of memory and cannot finish printing the table. An example of this was easily found on the SAS Communities website technical support threads:

```
146 run;

NOTE: Multiple concurrent threads will be used to summarize data.

ERROR: The SAS System stopped processing this step because of insufficient memory.
```

Expanding the memory is not possible for most programmers that are using SAS based on a server, so the easiest solution is to split the table into multiple worksheets. The memory limits are separated by printing each worksheet versus the entire workbook. However when the final product needs to be a single worksheet for the consumer this creates another problem to be solved. The programmer can manually go into the Excel workbook to move the

worksheet data from the extras to the main sheet, but this is very tedious, time consuming, and many reports are required to be generated on regular intervals.

EXAMPLE WORKBOOK

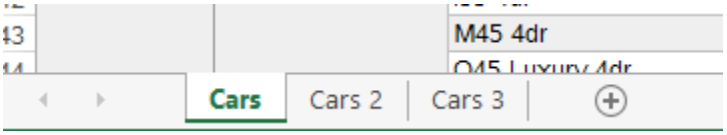
The following code will create a workbook that has one table split into three different worksheets. This example is not intended to avoid a memory limit error, but includes style modifications that must be preserved in the final product:

```
options nobyline;
ods excel file="example1.xlsx"
    options(flow='tables' frozen_headers='2' frozen_rowheaders='1'
sheet_name="Cars" sheet_interval="bygroup"
absolute_column_width='12,16,35,10,10,10,12,12,11,13,10,15,13,15,11');
proc sort data=sashelp.cars out=cars;
    by origin type;
run;
proc report data=cars nowd spanrows
    style(column)={background=white just=l vjust=top}
    style(header)={background=white color=black just=c};
    by origin;
    columns ("Car Category" origin make model type)
        ("Cost" MSRP Invoice)
        ("Engine" DriveTrain Enginesize cylinders)
        ("Performance" horsepower mpg_city MPG_Highway)
        ("Size" weight wheelbase length);
    define origin / order;
    define make / order;
    define msrp / display;
    compute msrp;
        if msrp>45000 then call define('msrp','style/merge','style={color=red}');
        else if msrp>25000 then call define('msrp','style/merge','style={color=orange}');
        else call define('msrp','style/merge','style={color=green}');
        row+1;
        if mod(row,2) then call define(_row_,'style/merge','style={background=greyef}');
        if ^missing(make) then call
     define(_row_,'style/merge','style={bordertopcolor=black bordertopwidth=10}');
    endcomp;
run;
ods excel close;
```

Running this program leads to the following workbook:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1			Car Category		Cost			Engine			Performance			Size	
2	Origin	Make	Model	Type	MSRP	Invoice	DriveTrain	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
3	Asia	Acura	MDX	SUV	$36,945	$33,337	All	3.5	6	265	17	23	4451	106	189
4			RSX Type S 2dr	Sedan	$23,820	$21,761	Front	2	4	200	24	31	2778	101	172
5			TSX 4dr	Sedan	$26,990	$24,647	Front	2.4	4	200	22	29	3230	105	183
6			TL 4dr	Sedan	$33,195	$30,299	Front	3.2	6	270	20	28	3575	108	186
7			3.5 RL 4dr	Sedan	$43,755	$39,014	Front	3.5	6	225	18	24	3880	115	197
8			3.5 RL w/Navigation 4dr	Sedan	$46,100	$41,100	Front	3.5	6	225	18	24	3893	115	197
9			NSX coupe 2dr manual S	Sports	$89,765	$79,978	Rear	3.2	6	290	17	24	3153	100	174
10		Honda	Civic Hybrid 4dr manual (gas/electric)	Hybrid	$20,140	$18,451	Front	1.4	4	93	46	51	2732	103	175
11			Insight 2dr (gas/electric)	Hybrid	$19,110	$17,911	Front	2	3	73	60	66	1850	95	155
12			Pilot LX	SUV	$27,560	$24,843	All	3.5	6	240	17	22	4387	106	188
13			CR-V LX	SUV	$19,860	$18,419	All	2.4	4	160	21	25	3258	103	179
14			Element LX	SUV	$18,690	$17,334	All	2.4	4	160	21	24	3468	101	167
15			Civic DX 2dr	Sedan	$13,270	$12,175	Front	1.7	4	115	32	38	2432	103	175
16			Civic HX 2dr	Sedan	$14,170	$12,996	Front	1.7	4	117	36	44	2500	103	175
17			Civic LX 4dr	Sedan	$15,850	$14,531	Front	1.7	4	115	32	38	2513	103	175
18			Accord LX 2dr	Sedan	$19,860	$17,924	Front	2.4	4	160	26	34	2994	105	188
19			Accord EX 2dr	Sedan	$22,260	$20,080	Front	2.4	4	160	26	34	3047	105	188
20			Civic EX 4dr	Sedan	$17,750	$16,265	Front	1.7	4	127	32	37	2601	103	175

The worksheet contains vertically merged cells, spanning headers, alternating shading, font changes, and traffic highlighting. Looking at the navigation bar, the workbook is split into three worksheets separated by origin. The names of the worksheets are kept as “Cars” as the goal is to have one final worksheet with this name:



This example will be used to test the following programmatic solutions.

PROGRAMMATIC SOLUTIONS

There are several ways to programmatically alter data in an xlsx format. The most common method would be to use the XLSX engine in a LIBNAME statement. This gives access to the worksheets like a SAS dataset and they can be manipulated and combined together:

```
libname test1 xlsx "example1.xlsx";

data test1.cars;

    set test1.cars test1.'cars 2'n test1.'cars 3'n;

run;
```

Running this code and checking the file again does show that the data was combined together but with two issues:

1. All of the formatting and spanning headers are lost:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	Car Category	B	C	D	Cost	F	Engine	H	I	Performance	K	L	Size	N	O
	Origin	Make	Model	Type	MSRP	Invoice	DriveTrain	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
	Asia	Acura	MDX	SUV	36945	33337	All	3.5	6	265	17	23	4451	106	189
			RSX Type S 2dr	Sedan	23820	21761	Front	2	4	200	24	31	2778	101	172
			TSX 4dr	Sedan	26990	24647	Front	2.4	4	200	22	29	3230	105	183
			TL 4dr	Sedan	33195	30299	Front	3.2	6	270	20	28	3575	108	186
			3.5 RL 4dr	Sedan	43755	39014	Front	3.5	6	225	18	24	3880	115	197
			3.5 RL w/Navigation 4dr	Sedan	46100	41100	Front	3.5	6	225	18	24	3893	115	197
			NSX coupe 2dr manual S	Sports	89765	79978	Rear	3.2	6	290	17	24	3153	100	174
		Honda	Civic Hybrid 4dr manual (gas/electric)	Hybrid	20140	18451	Front	1.4	4	93	46	51	2732	103	175
			Insight 2dr (gas/electric)	Hybrid	19110	17911	Front	2	3	73	60	66	1850	95	155

2. The headers are repeated at the start of each worksheet’s data:

	Origin	Make	Model	Type	MSRP	Invoice	DriveTrain	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
161	Origin	Make	Model	Type	MSRP	Invoice	DriveTrain	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
162	Europe	Audi	A4 1.8T 4dr	Sedan	25940	23508	Front	1.8	4	170	22	31	3252	104	179
163			A41.8T convertible 2dr	Sedan	35940	32506	Front	1.8	4	170	23	30	3638	105	180
164			A4 3.0 4dr	Sedan	31840	28846	Front	3	6	220	20	28	3462	104	179
165			A4 3.0 Quattro 4dr manual	Sedan	33430	30366	All	3	6	220	17	26	3583	104	179
166			A4 3.0 Quattro 4dr auto	Sedan	34480	31388	All	3	6	220	18	25	3627	104	179
167			A6 3.0 4dr	Sedan	36640	33129	Front	3	6	220	20	27	3561	109	192
168			A6 3.0 Quattro 4dr	Sedan	39640	35992	All	3	6	220	18	25	3880	109	192
169			A4 3.0 convertible 2dr	Sedan	42490	38325	Front	3	6	220	20	27	3814	105	180
170			A4 3.0 Quattro convertible 2dr	Sedan	44240	40075	All	3	6	220	18	25	4013	105	180
171			A6 2.7 Turbo Quattro 4dr	Sedan	42840	38840	All	2.7	6	250	18	25	3836	109	192
172			A6 4.2 Quattro 4dr	Sedan	49690	44936	All	4.2	8	300	17	24	4024	109	193

MODIFYING THE XML

The solution found by the author is to once again delve into the XML files in the background of the XLSX file. All of the style information is stored as well as all the cell contents. The short macro CONCAT_WORKSHEETS was developed with the following two parameters:

1. INFILE: File name and file path to xlsx document to combine together

2. HEADERROWS: Number of rows in the table that are considered header rows

The macro begins by importing the worksheet information with the methods described earlier in the paper:

```
filename inzip zip "&infile";

data __worksheets(keep=memname);

  length memname $200;

  fid=dopen("inzip");

  if fid=0 then stop;

  memcount=dnum(fid);

  do i=1 to memcount;

    memname=dread(fid,i);

    if find(memname,'xl/worksheets/','i') and find(memname,'.rel','i')=0 then output;

  end;

  rc=dclose(fid);

run;
```

The macro then uses this dataset to count how many worksheets are in the workbook. Note that if not concatenating all worksheets, then this part of the program would need to be modified to be more targeted at specific worksheets:

```
%local i lastrow nsheets lastUpdatedColumn;

PROC SQL NOPRINT;

    SELECT COUNT(*) into :nsheets separated by '' from __worksheets;

QUIT;
```

This method follows the following assumptions to function:

- Every worksheet XML has a “header” section, a “body” section, and a “footer” section
- To concatenate the worksheets together, the header, body and footer sections are only needed from the first worksheet
- Every other worksheet only requires the body section to be appended to the body section of the first worksheet while adjusting the row references for each cell
- The header, footer, and overall workbook sections need to be updated with the correct number of rows/columns/sheets

Header Section

The header section contains everything from the start of the XML through the <sheetData> starting tag. This includes information such as column widths, encoding, zoom, frozen headers. This is required for keeping the formatting of the columns in the final combined worksheet.

CARS
_WORKSHEETS
_WS1_HEAD
_WS1_BODY
_WS1_FOOT
_WS2_BODY
_WS2_MC
_WS3_BODY
_WS3_MC
_COMBINE
_WORKBOOK

	code
1	<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2	<worksheet xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" xmlns:r="http://schemas.openxmlfor...
3	<sheetPr filterMode="0" enableFormatConditionsCalculation="0">
4	<pageSetUpPr fitToPage="0"/>
5	</sheetPr>
6	<dimension ref="A1:O160"/>
7	<sheetViews>
8	<sheetView zoomScale="100" zoomScaleNormal="100" zoomScalePageLayoutView="100" workbookViewId="0">
9	<pane xSplit="1" ySplit="2" topLeftCell="B3" activePane="bottomRight" state="frozen"/>
10	</sheetView>
11	</sheetViews>
12	<sheetFormatPr baseColWidth="10" defaultRowHeight="12" customHeight="1" x14ac:dyDescent="0"/>
13	<cols>
14	<col min="1" max="1" width="12.71" bestFit="1" customWidth="1"/>
15	<col min="2" max="2" width="16.71" bestFit="1" customWidth="1"/>
16	<col min="3" max="3" width="35.71" bestFit="1" customWidth="1"/>
17	<col min="4" max="4" width="10.71" bestFit="1" customWidth="1"/>
18	<col min="5" max="5" width="10.71" bestFit="1" customWidth="1"/>
19	<col min="6" max="6" width="10.71" bestFit="1" customWidth="1"/>
20	<col min="7" max="7" width="12.71" bestFit="1" customWidth="1"/>
21	<col min="8" max="8" width="15.71" bestFit="1" customWidth="1"/>
22	<col min="9" max="9" width="11.71" bestFit="1" customWidth="1"/>
23	<col min="10" max="10" width="13.71" bestFit="1" customWidth="1"/>
24	<col min="11" max="11" width="10.71" bestFit="1" customWidth="1"/>
25	<col min="12" max="12" width="15.71" bestFit="1" customWidth="1"/>
26	<col min="13" max="13" width="13.71" bestFit="1" customWidth="1"/>
27	<col min="14" max="14" width="15.71" bestFit="1" customWidth="1"/>
28	<col min="15" max="15" width="11.71" bestFit="1" customWidth="1"/>
29	</cols>
30	<sheetData>

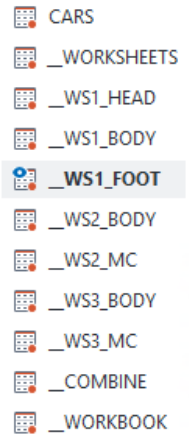
Body Section

The body section contains the data values contained in the cells of the worksheet. This includes the values for the column headers. The cells are filled with style references to the previously mentioned styles xml, numeric values, and string references. Character strings are not contained directly in the cell but are instead as references where each unique string is kept in another xml file.

1	<row r="1" ht="14" customHeight="1">
2	<c r="A1" s="1" t="s"><v>0</v></c>
3	<c r="B1" s="1"/>
4	<c r="C1" s="1"/>
5	<c r="D1" s="1"/>
6	<c r="E1" s="1" t="s"><v>1</v></c>
7	<c r="F1" s="1"/>
8	<c r="G1" s="1" t="s"><v>2</v></c>
9	<c r="H1" s="1"/>
10	<c r="I1" s="1"/>
11	<c r="J1" s="1" t="s"><v>3</v></c>
12	<c r="K1" s="1"/>
13	<c r="L1" s="1"/>
14	<c r="M1" s="1" t="s"><v>4</v></c>
15	<c r="N1" s="1"/>
16	<c r="O1" s="1"/>
17	</row>

Footer Section

The footer section starts with the /sheetData tag to close the cell value contents and finishes with the /worksheet tag to close the worksheet. Settings such as print options, page setup, data validations, and merged cell information is stored in this section.



	code
1	</sheetData>
2	<mergeCells count="20">
3	<mergeCell ref="A1:D1"/>
4	<mergeCell ref="E1:F1"/>
5	<mergeCell ref="G1:I1"/>
6	<mergeCell ref="J1:L1"/>
7	<mergeCell ref="M1:O1"/>
8	<mergeCell ref="A3:A160"/>
9	<mergeCell ref="B3:B9"/>
10	<mergeCell ref="B10:B26"/>
11	<mergeCell ref="B27:B38"/>
12	<mergeCell ref="B39:B46"/>
13	<mergeCell ref="B47:B48"/>
14	<mergeCell ref="B49:B59"/>
15	<mergeCell ref="B60:B70"/>
16	<mergeCell ref="B71:B81"/>
17	<mergeCell ref="B82:B94"/>
18	<mergeCell ref="B95:B111"/>
19	<mergeCell ref="B112:B113"/>
20	<mergeCell ref="B114:B124"/>
21	<mergeCell ref="B125:B132"/>
22	<mergeCell ref="B133:B160"/>
23	</mergeCells>
24	<printOptions headings="0" horizontalCentered="0" verticalCentered="0" gridLines="0"/>
25	<pageMargins left="0.05" right="0.05" top="0.5" bottom="0.5" header="0" footer="0"/>
26	<pageSetup paperSize="1" fitToWidth="1" fitToHeight="1" scale="100" orientation="portr...
27	<headerFooter>
28	<oddHeader></oddHeader>
29	<oddFooter></oddFooter>
30	</headerFooter>
31	</worksheet>

The macro continues by starting with worksheet one to extract the header, body and footer sections of the xml:

```
data __ws1_head __ws1_body __ws1_foot;

    infile inzip('xl/worksheets/sheet1.xml') truncover;

    input code $10000. ;

    if _n_=1 then do;

        _header_=1;_body_=0;_footer_=0;

    end;

    retain _header_ _body_ _footer_ row;
```

```
    if code='</sheetData>' then do; _body_=0;_footer_=1;end;
    if _header_ then output __ws1_head;
    if _body_ then do;
        if find(code,'row r=','i') then
           row=input(prxchange('s/.+row r="(\d+)".+/$1/',-1,code),??best.);
        else if find(code,'/row','i')=0 then
          column=prxchange('s/.+r="([a-zA-Z]+)\d+".+/$1/',-1,code);
        else if find(code,'/row','i') then
             call symput('lastRow',strip(put(row,12.)));
        output __ws1_body;
    end;
    if _footer_ then output __ws1_foot;
    if code='<sheetData>' then do; _header_=0;_body_=1;end;
run;
```

Regular expressions are used to find the cell reference row and column numbers (e.g. A10). Every other worksheet in the workbook has the body extracted and, in this example, the merged cell information as well. Other parts can be added as well for data validations (comments):

```
%do i=2 %to &nsheets;
    data __ws&i._body __ws&i._mc;
        infile inzip("xl/worksheets/sheet&i..xml") truncover;
        input code $10000.;
        retain _header_ _body_ _mc_ row;
        if _n_=1 then do;
            _header_=1;_body_=0;_footer_=0;_mc_=0;
        end;
        if code='</sheetData>' then do; _body_=0;_footer_=1;end;
        if code='</mergeCells>' then do; _body_=0;_footer_=0;_mc_=0;end;
        if _body_ then do;
            if find(code,'row r=','i') then
                row=input(prxchange('s/.+row r="(\d+)".+/$1/',-1,code),??best.);
            else if find(code,'/row','i')=0 then
                column=prxchange('s/.+r="([a-zA-Z]+)\d+".+/$1/',-1,code);
            else if find(code,"/row","i") then
                call symput("lastRow",strip(put(row-&headerRows+&LastRow,12.)));
            if ^missing(column) and ^missing(row) then
              call symputx('lastUpdatedColumn',catt(column,&LastRow+row-&headerRows));
```

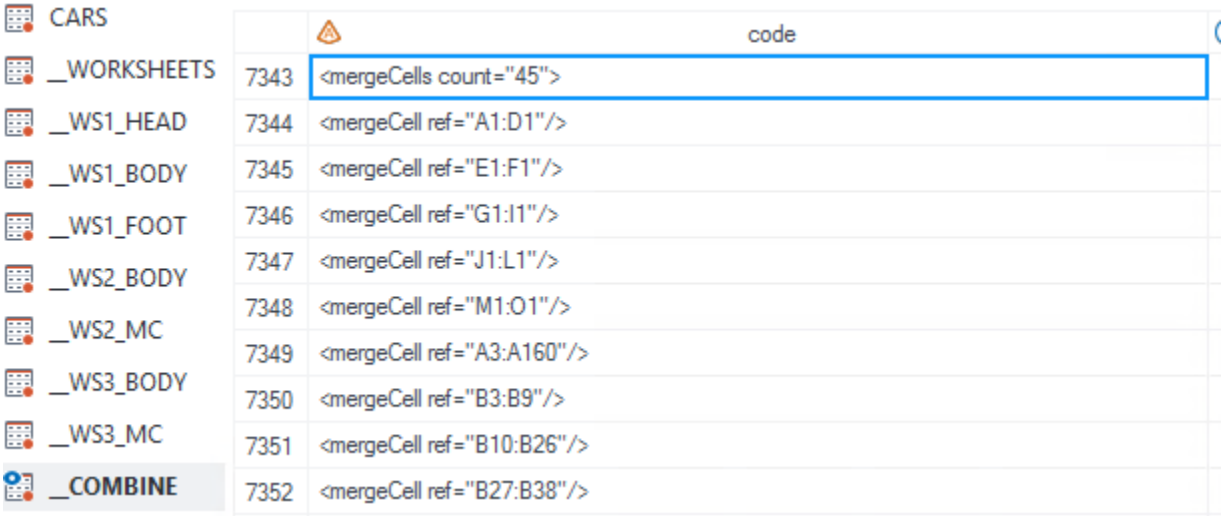
```
            code=prxchange('s/r="'||strip(catt(column,row))||
                '"/r="'||strip(catt(column,row+&lastRow.-&headerRows))||'"/',-1,code);
            if row>&headerRows then output __ws&i._body;
        end;
        if _mc_ then do;
            if find(code,'<mergeCell ref=','i') then do;
                row1=input(prxchange('s/.+ref="[a-zA-Z]+(\d+):.+".+/$1/',
                    -1,code),??best.);
                row2=input(prxchange('s/.+ref=".+:[a-zA-Z]+(\d+)".+/$1/',
                     -1,code),??best.);
                column1=prxchange('s/.+ref="([a-zA-Z]+)\d+:.+".+/$1/',-1,code);
                column2=prxchange('s/.+ref=".+:([a-zA-Z]+)\d+".+/$1/',-1,code);
                code=prxchange('s/ref="'||strip(catt(column1,row1))||':'
                   ||strip(catt(column2,row2))||'"/ref="'
                   ||strip(catt(column1,row1+&lastRow.-&headerRows))||':'||
                     strip(catt(column2,row2+&lastRow.-&headerRows))||'"/',-1,code);
            end;
            if ^(row1<=&headerRows and row2<=&headerRows) then output __ws&i._mc;
        end;
        if code='<sheetData>' then do; _header_=0;_body_=1;end;
        if find(code,'<mergeCells','i') then do; _header_=0;_body_=0;_mc_=1;end;
    run;
    data __ws&i._body;
        set __ws&i._body end=last;
        if last then delete;
    run;
%end;
```

Two macro variables are continuously carried forward: LASTROW and LASTUPDATEDCOLUMN. The macro variable HEADERROWS is from the initial macro call and is used to ignore the extra header rows from worksheets after the first. LASTROW contains the last row number from the previous worksheet so that the row value of each cell reference can be updated. For example if worksheet one ends on row 100 then 100 will be added to any row of worksheet two to avoid trying to place cells in the same locations. Worksheet two adds its row number to this macro variable so that worksheet three can be properly adjusted and so forth. LASTUPDATEDCOLUMN is to contain the last cell to be updated in order to update the header of the worksheet in the final steps.

The pieces are then combined in the next step starting with the header from worksheet one, then the body files in order, and finishing with the footer from worksheet one. The footer is added strategically in include all of the merged cell information from the second worksheet and beyond:

```
data __combine;
    set __ws1_head (in=h)
        %do i=1 %to &nsheets;  ws&i._body  %end;
        __ws1_foot (where=(code=('</sheetData>') or (find(code,'mergeCell','i')
                           and find(code,'/mergeCells','i')=0)))
        %do i=2 %to &nsheets;
            __ws&i._mc (where=(find(code,'<mergeCells','i')=0))
        %end;
        __ws1_foot (in=f where=(code^='</sheetData>' and
                     (find(code,'mergeCell','i')=0 or find(code,'/mergeCells','i'))));
    if h and find(code,'dimension ref=','i') then
           code=prxchange('s/:[a-zA-Z]+\d+/:'||"&lastUpdatedColumn"||'/',-1,code);
    if f and find(code,'autofilter ref=','i') then
           code=prxchange('s/:[a-zA-Z]+\d+/:'||"&lastUpdatedColumn"||'/',-1,code);
run;
proc sql noprint;
    %local mergeCellNRows;
    %let mergeCellNRows=0;
    select count(*) into :mergeCellNRows separated by ''
        from (select code from __ws1_foot where find(code,'mergeCell ref','i')
                %do i=2 %to &nsheets;
                    OUTER UNION CORR
                    select code from __ws&i._mc
                %end;);
    %if mergeCellNRows>0 %then %do;
        update __combine
            set code=prxchange('s/"\d+"/"'||"&mergeCellNRows"||'"/',-1,code)
            where find(code,'<mergeCells count','i');
    %end;
quit;
data _null_;
    set __combine;
    file inzip("xl/worksheets/sheet1.xml");
    put code;
run;
```

The SQL step counts how many merged cell references there are to update the footer section of the combined worksheet:



If this number is not updated correctly then the sheet will give an error message upon opening. The data _null_ step then pushes the changes into worksheet one in the Excel file.

Deleting the Extra Worksheets

The concatenated worksheets must now be deleted so that the workbook does not contain extra duplicated data. This can be done with filename statements specifically targeting the XML files in the XLSX file and using a data step FDELETE function:

```
%do i=2 %to &nsheets;

    filename inzip&i zip "&infile" member="xl/worksheets/sheet&i..xml";

    data _null_;

        x=fexist("inzip&i");

        if x then do;

            rc=fdelete("inzip&i");

        end;

    run;

    filename inzip&i clear;

%end;
```

Final Steps

The workbook XML requires more updates to ensure the file does not open corrupted or with errors. The final steps of the macro address these updates and clean up temporary datasets:

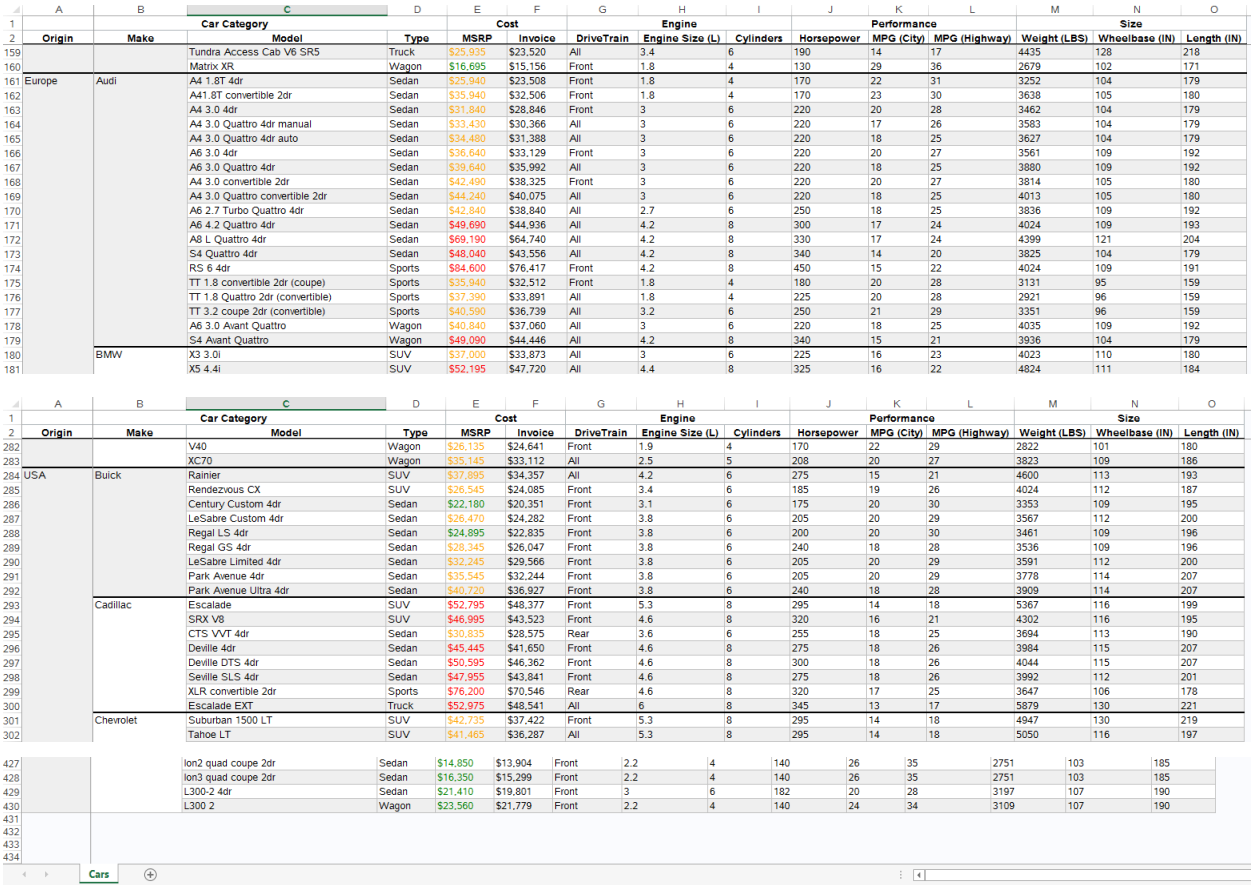
```
data __workbook;

    infile inzip("xl/workbook.xml") truncover;

    input code $10000. ;

    if find(code,'sheet name=','i') and find(code,'sheetId="1"','i')=0 then delete;

    if find(code,'<definedName name=','i') and

         find(code,'localSheetId="0"','i')=0 then delete;
```

```
    x=prxchange('s/([a-zA-Z]+)(\d+)/\\\$$1\\\$$2/',-1,"&lastUpdatedColumn");
    if find(code,'<definedName name=','i') then
        code=prxchange('s/:\$[a-zA-Z]+\$\d+/:'||strip(x)||'/',-1,code);
run;
data _null_;
    set __workbook;
    file inzip("xl/workbook.xml");
    put code;
run;
filename inzip clear;
proc datasets nolist nodetails;
    delete __workbook __worksheets __ws: __combine;
quit;
%mend;
```

The worksheet XMLs were deleted in the previous step, but this information must also be removed from the workbook XML in the first step. The sheets tag includes all the worksheets in the workbook, so in this example sheets two and three are removed. The references for definedName are found when using data validations. This example did not have any data validations so they do not appear in the screenshot:

Upon opening the Excel file created in the initial example it can now be seen that all three worksheets have been concatenated into one with the formatting intact:

		Car Category		Cost		Engine			Performance			Size		
Origin	Make	Model	Type	MSRP	Invoice	DriveTrain	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
Asia	Acura	MDX	SUV	$36,945	$33,337	All	3.5	6	265	17	23	4451	106	189
		RSX Type S 2dr	Sedan	$23,820	$21,761	Front	2	4	200	24	31	2778	101	172
		TSX 4dr	Sedan	$26,990	$24,647	Front	2.4	4	200	22	29	3230	105	183
		TL 4dr	Sedan	$33,195	$30,299	Front	3.2	6	270	20	28	3575	108	186
		3.5 RL 4dr	Sedan	$43,755	$39,014	Front	3.5	6	225	18	24	3880	115	197
		3.5 RL w/Navigation 4dr	Sedan	$46,100	$41,100	Front	3.5	6	225	18	24	3893	115	197
		NSX coupe 2dr manual S	Sports	$89,765	$79,978	Rear	3.2	6	290	17	24	3153	100	174
	Honda	Civic Hybrid 4dr manual (gas/electric)	Hybrid	$20,140	$18,451	Front	1.4	4	93	46	51	2732	103	175
		Insight 2dr (gas/electric)	Hybrid	$19,110	$17,911	Front	2	3	73	60	66	1850	95	155
		Pilot LX	SUV	$27,560	$24,843	All	3.5	6	240	17	22	4387	106	188
		CR-V LX	SUV	$19,860	$18,419	All	2.4	4	160	21	25	3258	103	179
		Element LX	SUV	$18,690	$17,334	All	2.4	4	160	21	24	3468	101	167
		Civic DX 2dr	Sedan	$13,270	$12,175	Front	1.7	4	115	32	38	2432	103	175
		Civic HX 2dr	Sedan	$14,170	$12,996	Front	1.7	4	117	36	44	2500	103	175
		Civic LX 4dr	Sedan	$15,850	$14,531	Front	1.7	4	115	32	38	2513	103	175
		Accord LX 2dr	Sedan	$19,860	$17,924	Front	2.4	4	160	26	34	2994	105	188
		Accord EX 2dr	Sedan	$22,260	$20,080	Front	2.4	4	160	26	34	3047	105	188



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1			Car Category		Cost			Engine			Performance			Size	
2	Origin	Make	Model	Type	MSRP	Invoice	DriveTrain	Engine Size (L)	Cylinders	Horsepower	MPG (City)	MPG (Highway)	Weight (LBS)	Wheelbase (IN)	Length (IN)
159			Tundra Access Cab V6 SR5	Truck	$25,935	$23,520	All	3.4	6	190	14	17	4435	128	218
160			Matrix XR	Wagon	$16,695	$15,156	Front	1.8	4	130	29	36	2679	102	171
161	Europe	Audi	A4 1.8T 4dr	Sedan	$25,940	$23,508	Front	1.8	4	170	22	31	3252	104	179
162			A41.8T convertible 2dr	Sedan	$35,940	$32,506	Front	1.8	4	170	23	30	3638	105	180
163			A4 3.0 4dr	Sedan	$31,840	$28,846	Front	3	6	220	20	28	3462	104	179
164			A4 3.0 Quattro 4dr manual	Sedan	$33,430	$30,366	All	3	6	220	17	26	3583	104	179
165			A4 3.0 Quattro 4dr auto	Sedan	$34,480	$31,388	All	3	6	220	18	25	3627	104	179
166			A6 3.0 4dr	Sedan	$36,640	$33,129	Front	3	6	220	20	27	3561	109	192
167			A6 3.0 Quattro 4dr	Sedan	$39,640	$35,992	All	3	6	220	18	25	3880	109	192
168			A4 3.0 convertible 2dr	Sedan	$42,490	$38,325	Front	3	6	220	20	27	3814	105	180
169			A4 3.0 Quattro convertible 2dr	Sedan	$44,240	$40,075	All	3	6	220	18	25	4013	105	180
170			A6 2.7 Turbo Quattro 4dr	Sedan	$42,840	$38,840	All	2.7	6	250	18	25	3836	109	192
171			A6 4.2 Quattro 4dr	Sedan	$49,690	$44,936	All	4.2	8	300	17	24	4024	109	193
172			A8 L Quattro 4dr	Sedan	$69,190	$64,740	All	4.2	8	330	17	24	4399	121	204
173			S4 Quattro 4dr	Sedan	$48,040	$43,556	All	4.2	8	340	14	20	3825	104	179
174			RS 6 4dr	Sports	$84,600	$76,417	Front	4.2	8	450	15	22	4024	109	191
175			TT 1.8 convertible 2dr (coupe)	Sports	$35,940	$32,512	Front	1.8	4	180	20	28	3131	95	159
176			TT 1.8 Quattro 2dr (convertible)	Sports	$37,390	$33,891	All	1.8	4	225	20	28	2921	96	159
177			TT 3.2 coupe 2dr (convertible)	Sports	$40,590	$36,739	All	3.2	6	250	21	29	3351	96	159
178			A6 3.0 Avant Quattro	Wagon	$40,840	$37,060	All	3	6	220	18	25	4035	109	192
179			S4 Avant Quattro	Wagon	$49,090	$44,446	All	4.2	8	340	15	21	3936	104	179
180		BMW	X3 3.0i	SUV	$37,000	$33,873	All	3	6	225	16	23	4023	110	180
181			X5 4.4i	SUV	$52,195	$47,720	All	4.4	8	325	16	22	4824	111	184
282			V40	Wagon	$26,135	$24,641	Front	1.9	4	170	22	29	2822	101	180
283			XC70	Wagon	$35,145	$33,112	All	2.5	5	208	20	27	3823	109	186
284	USA	Buick	Rainier	SUV	$37,895	$34,357	All	4.2	6	275	15	21	4600	113	193
285			Rendezvous CX	SUV	$26,545	$24,085	Front	3.4	6	185	19	26	4024	112	187
286			Century Custom 4dr	Sedan	$22,180	$20,351	Front	3.1	6	175	20	30	3353	109	195
287			LeSabre Custom 4dr	Sedan	$26,470	$24,282	Front	3.8	6	205	20	29	3567	112	200
288			Regal LS 4dr	Sedan	$24,895	$22,835	Front	3.8	6	200	20	30	3461	109	196
289			Regal GS 4dr	Sedan	$28,345	$26,047	Front	3.8	6	240	18	28	3536	109	196
290			LeSabre Limited 4dr	Sedan	$32,245	$29,566	Front	3.8	6	205	20	29	3591	112	200
291			Park Avenue 4dr	Sedan	$35,545	$32,244	Front	3.8	6	205	20	29	3778	114	207
292			Park Avenue Ultra 4dr	Sedan	$40,720	$36,927	Front	3.8	6	240	18	28	3909	114	207
293		Cadillac	Escalade	SUV	$52,795	$48,377	Front	5.3	8	295	14	18	5367	116	199
294			SRX V8	SUV	$46,995	$43,523	Front	4.6	8	320	16	21	4302	116	195
295			CTS VVT 4dr	Sedan	$30,835	$28,575	Rear	3.6	6	255	18	25	3694	113	190
296			Deville 4dr	Sedan	$45,445	$41,650	Front	4.6	8	275	18	26	3984	115	207
297			Deville DTS 4dr	Sedan	$50,595	$46,362	Front	4.6	8	300	18	26	4044	115	207
298			Seville SLS 4dr	Sedan	$47,955	$43,841	Front	4.6	8	275	18	26	3992	112	201
299			XLR convertible 2dr	Sports	$76,200	$70,546	Rear	4.6	8	320	17	25	3647	106	178
300			Escalade EXT	Truck	$52,975	$48,541	All	6	8	345	13	17	5879	130	221
301		Chevrolet	Suburban 1500 LT	SUV	$42,735	$37,422	Front	5.3	8	295	14	18	4947	130	219
302			Tahoe LT	SUV	$41,465	$36,287	All	5.3	8	295	14	18	5050	116	197
427			Ion2 quad coupe 2dr	Sedan	$14,850	$13,904	Front	2.2	4	140	26	35	2751	103	185
428			Ion3 quad coupe 2dr	Sedan	$16,350	$15,299	Front	2.2	4	140	26	35	2751	103	185
429			L300-2 4dr	Sedan	$21,410	$19,801	Front	3	6	182	20	28	3197	107	190
430			L300 2	Wagon	$23,560	$21,779	Front	2.2	4	140	24	34	3109	107	190
431															
432															
433															
434															

Cars

CONCLUSION

The EXCEL destination is a great tool for building large and complex reports. Adding the ability to password protect the workbook from being modified is a great benefit that should have been a default feature. Becoming familiar with the XML behind the XLSX file allows the user to program around roadblocks such as hitting memory limits. These methods could be fine tuned even further to account for more situations such as data validations and to add grouping to tables.

CONTACT INFORMATION

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

Jeffrey Meyers
Regeneron Pharmaceuticals
Jeffrey.meyers@regeneron.com