



Quanticate

The Clinical Data Experts

No XSLX Engine in SAS? No XLSX problem!

Will Greenway



15th-19th November

EU Connect 2021

The Clinical Data
Science Conference

Our Values: Quality • Trust • Care • Respect • Honesty • Transparency

Our Services: Biostatistics • Clinical Programming • Clinical Data Management • Medical Writing • Pharmacovigilance • Consultancy

Agenda

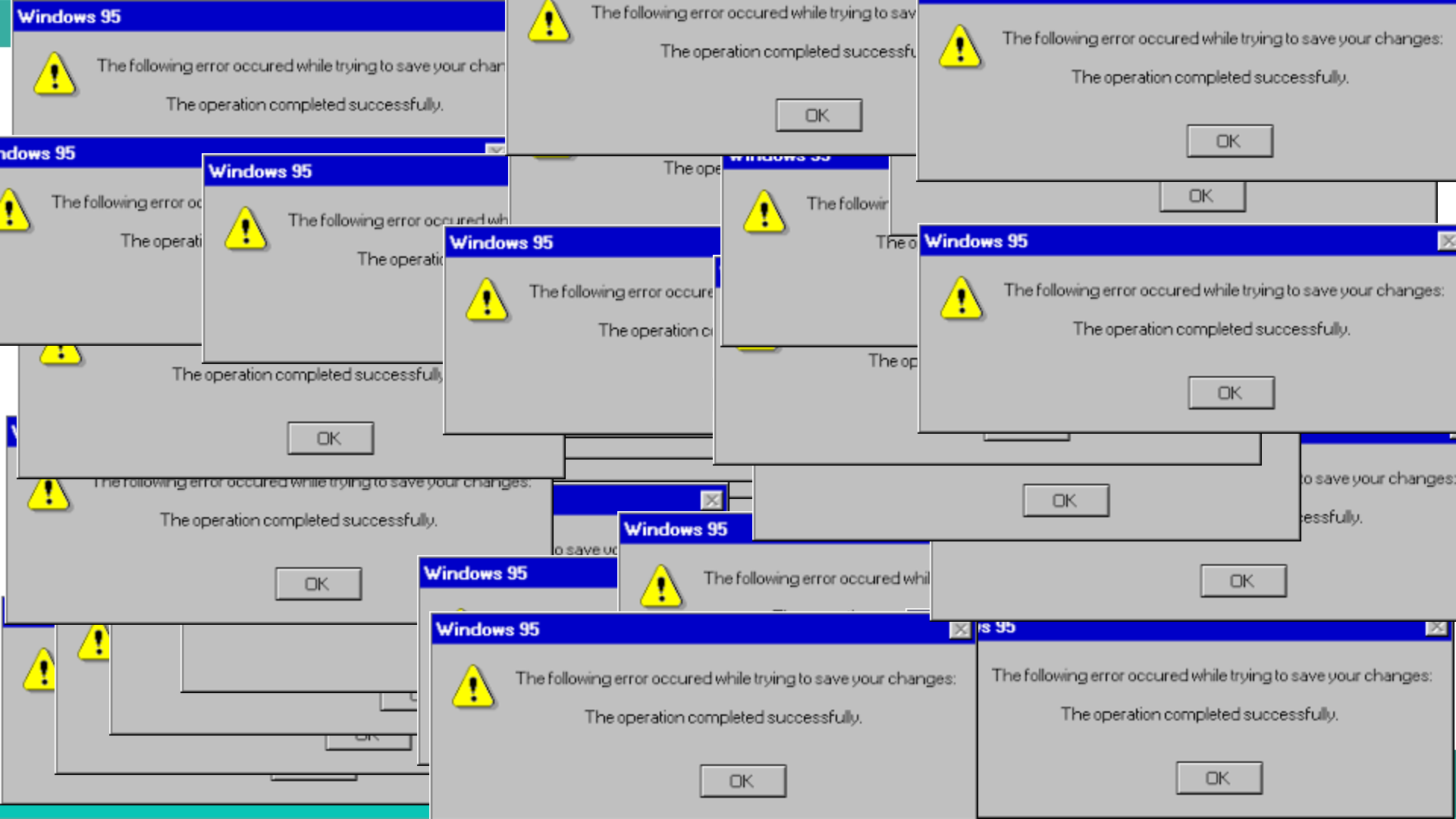
- Reading in XLSX Files
- But wait, what is an XLSX?
 - XLSX Structure
- Alternate Method for XLSX files
- Thanks / Q&A details

Reading in XLSX Files

There are numerous ways of reading Excel (.xlsx) files into SAS®:

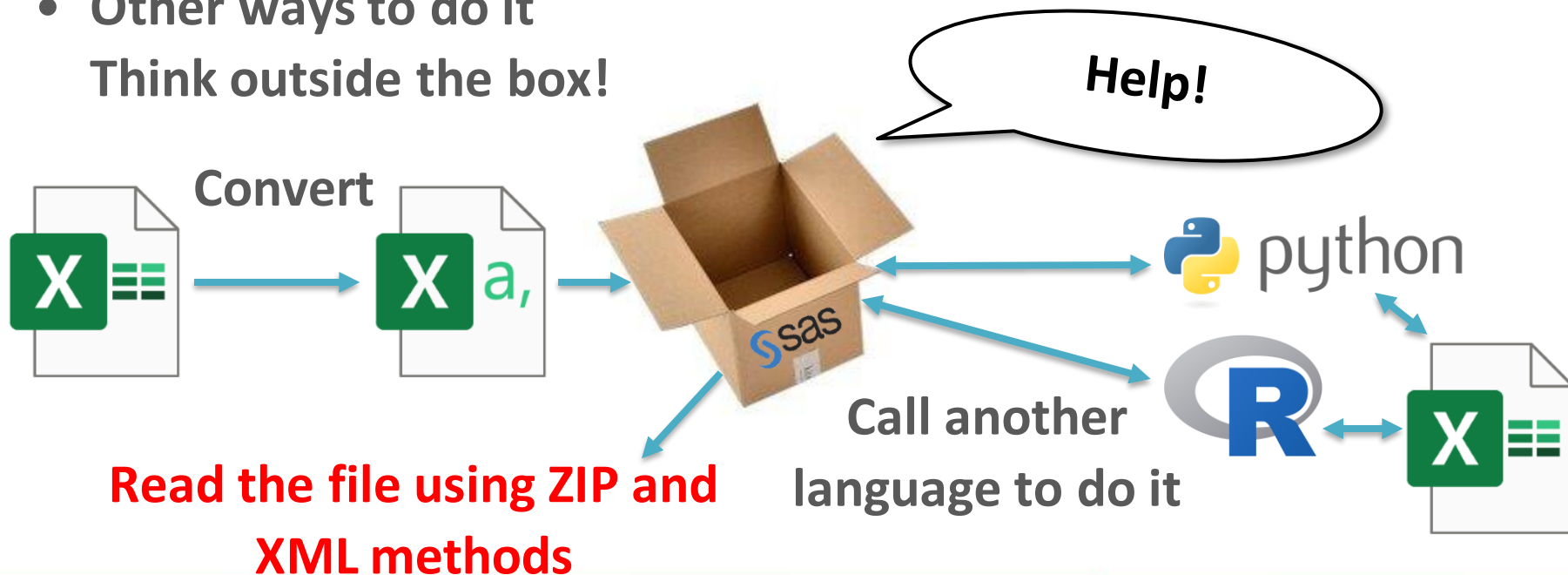
- **XLSX engine** (`libname` statement or `proc import`)
License for SAS/ACCESS to PC Files, limited control
- **Import Wizards**
Often not flexible/automated enough, can't call from code
- **DDE**
Microsoft has not used DDE internally since the release of **Windows 95!**



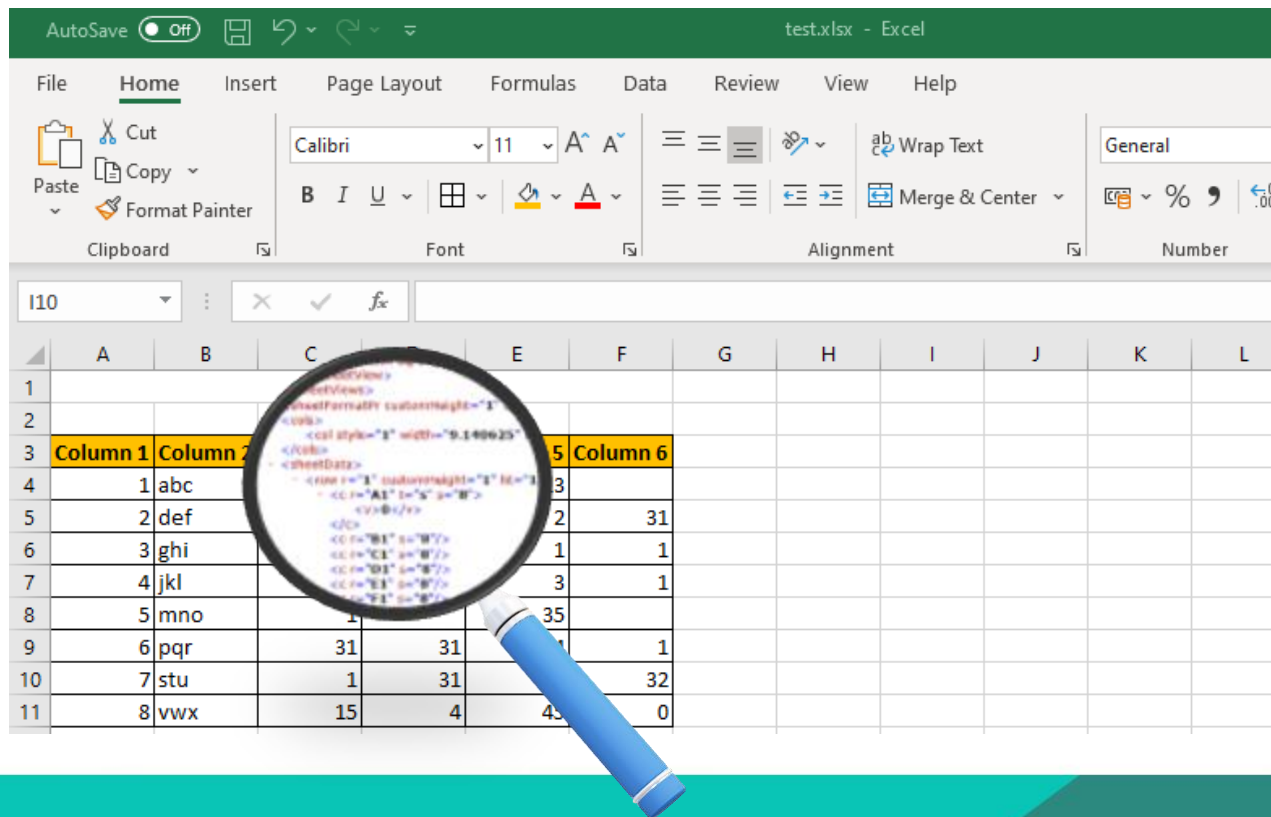


Reading in XLSX Files

- Other ways to do it
Think outside the box!



But wait, what is an XLSX?

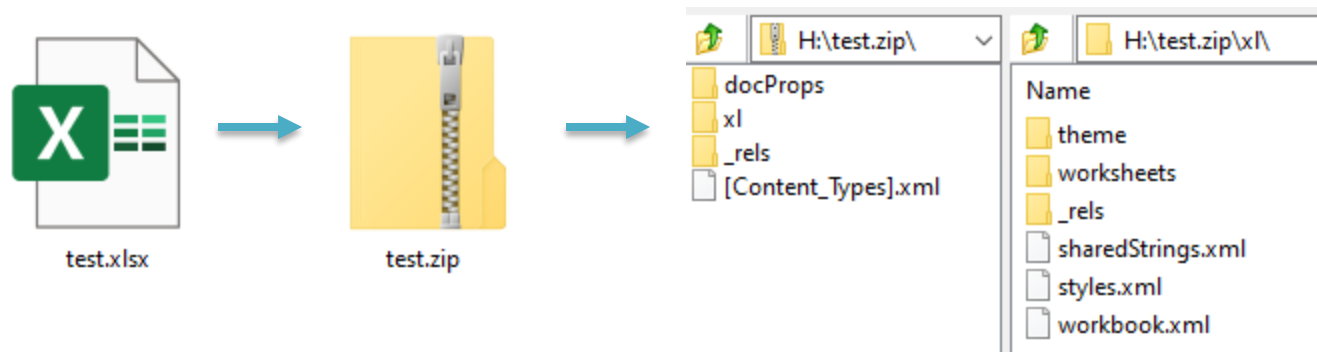


But wait, what is an XLSX?

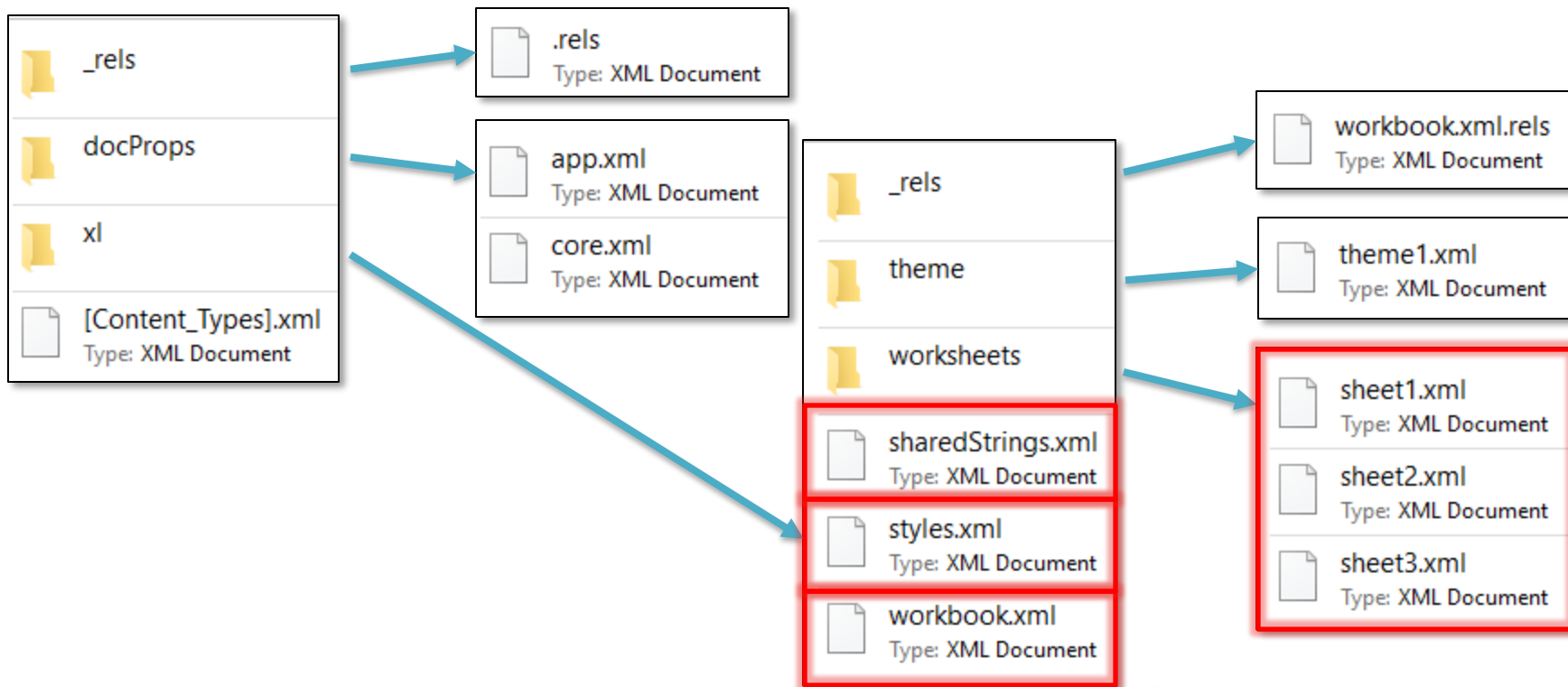
An XLSX is **zip package** containing a collection of **XML files**.

Don't believe me?

Rename an .xlsx file as .zip, Open the zip file:



XSLX Structure



Alternate Method for XLSX files

1. Unzip (if needed)
2. Create XML Maps (one time)
3. Read the XML
4. Combining the datasets
5. Transpose
6. And I get the metadata too?!

1. Unzip (if needed)

NOTE: This is not the full code. Code has been shortened and/or de-macro'd for demonstration purposes.

In some versions of SAS® you may need to extract each XML file in the zip to a temporary area to read it in using an XML map. This step isn't always required.

```
filename inzip ZIP "H:\test.xlsx";
filename out "%sysfunc(getoption(work))\styles.xml";
data _null_;
  infile inzip x1/styles.xml lrecl=256 recfm=F length=length unbuf;
  file out lrecl=256 recfm=N;
  input;
  put _infile_ $varying256. length;
run;
```

← *Path/File within the ZIP (or XLSX)*



Turn the above into a macro
and call for each xml file

1. Unzip (if needed)

NOTE: This is not the full code. Code has been shortened and/or de-macro'd for demonstration purposes.

As you may have an unknown number of sheets, you can look in the zip

```
data _null_;  
  length memname $200;  
  fid=dopen("inzip");  
  memcount=dnum(fid);  
  do i=1 to memcount;  
    memname=dread(fid,i);  
    if memname='xl/worksheets/sheet' then do;  
      call execute(/*Call your extraction macro from previous slide for each sheet*/);  
    end;  
  end;  
  rc=dclose(fid);  
run;
```



Don't forget to include error handling!



For call execute, use %nrstr to ensure your macro executes as expected

2. Create XML Maps (one time)

We will read XML files via XML Maps which first need to be setup, but once setup should work for any XLSX file.

I used SAS® XML Mapper.

Don't include every XML file, nor every item in each XML, just select what you need.

I created maps for:

- sharedStrings
- workbook
- styles
- sheet



A single “sheet” map should work for any/all sheets

2. Create XML Maps (one time)

NOTE: This is not the full XML Map. Code has been shortened for demonstration purposes.

You can create multiple tables within some Maps/XML file, e.g. for “styles.xml” I use 5 tables, example of 1 table (fonts) below:

```
<TABLE description="Font info" name="__xlsx_font">
  <TABLE-PATH syntax="XPathENR">
    /styleSheet/fonts/font</TABLE-PATH>
  <COLUMN class="ORDINAL" name="fontId">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPathENR">
      /styleSheet/fonts/font</INCREMENT-PATH>
    </COLUMN>
  <COLUMN name="fontSize">
    <PATH syntax="XPathENR">
      /styleSheet/fonts/font/sz/@val</PATH>
    </COLUMN>
  <COLUMN class="ORDINAL" name="fontBold" retain="YES">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPathENR">
      /styleSheet/fonts/font/b</INCREMENT-PATH>
    <RESET-PATH beginend="BEGIN" syntax="XPathENR">
      /styleSheet/fonts/font</RESET-PATH>
    </COLUMN>
```

```
...
  <COLUMN class="ORDINAL" name="fontItalics" retain="YES">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPathENR">
      /styleSheet/fonts/font/i</INCREMENT-PATH>
    <RESET-PATH beginend="BEGIN" syntax="XPathENR">
      /styleSheet/fonts/font</RESET-PATH>
    </COLUMN>
  <COLUMN name="fontColour">
    <PATH syntax="XPathENR">
      /styleSheet/fonts/font/color/@rgb</PATH>
    </COLUMN>
  <COLUMN name="fontName">
    <PATH syntax="XPathENR">
      /styleSheet/fonts/font/name/@val</PATH>
    </COLUMN>
</TABLE>
```



Include **ORDINALs**. Make use of
INCREMENT-PATH and **RESET-PATH**

3. Read the XML

NOTE: This is not the full code. Code has been shortened and/or de-macro'd for demonstration purposes.

If you have unzipped the file (Step 1.)

```
filename style      "%sysfunc(getoption(work)) \style.xml";  
filename xlsxmap    "H:\xlsx_ style.map";  
libname  style xmlv2 xmlmap=xlsxmap access=readonly;  
proc copy in=style out=work;  
run;  
  
libname  style      clear;  
filename xlsxmap    clear;  
filename style      clear;
```



Turn these steps into a macro
and call for each xml file



You may already have the xml files set to a filename
from a previous step so may not need the first line

3. Read the XML

NOTE: This is not the full code. Code has been shortened and/or de-macro'd for demonstration purposes.

If you have not unzipped the file (Step 1.) and have a version of SAS that supports reading the XML directly from a ZIP file:

```
filename    xlsxfile zip      "H:\test.xlsx" member="xl/style.xml";
filename    xlsxmap          "H:\xlsx_style.map";
libname      xlsxfile xmlv2 xmlmap=xlsxmap access=readonly;
proc copy   in=xlsxfile out=work;
run;
libname      xlsxfile clear;
filename     xlsxmap  clear;
filename     xlsxfile clear;
```



Turn these steps into a macro
and call for each xml file

3. Read the XML



Watch out for your
validvarname option!

When the SAS option VALIDVARNAME = UPCASE, this may result in the values for the ORDINAL variables all being set to missing in the resulting data sets. When you change the option to set VALIDVARNAME = v7 or ANY the ordinal variables contained the correct values.

If the element is not upcased, we add the element name and append the upcased value _ORDINAL such as the below. So if the element name is not upcased then we run into this problem.

SAS upcases everything when reading this into SAS, however, this is not the case with the XML libname engine, therefore if the element that this ORDINAL is being based on is lowercase (e.g. "c_ord") then we are subject to run into the issue you have described.

```
<COLUMN class="ORDINAL" name="Interchange_ORDINAL">
  <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/Interchange</INCREMENT-PATH>
  <TYPE>numeric</TYPE>
  <DATATYPE>integer</DATATYPE>
</COLUMN>
```

4. Combining the datasets

NOTE: This is not the full code. Code has been shortened and/or de-macro'd for demonstration purposes.

Excel defaults ID references starting at 0, while SAS XMLMap ORDINALs default start at 1. Combine tables within styles via IDs to create 1 overall “style” dataset

So first, subtract 1 from all ORDINAL variables.

```
proc sql;
  update __xlsx_xf1
    set xf1Id = xf1Id - 1;
  update __xlsx_font
    set fontId = fontId - 1;
  ...
quit;
```

Then merge/join as needed:

```
proc sql;
  create table __xlsx_style1 as
  select *
  from
    __xlsx_xf1
  natural left join
    __xlsx_font;
  ...
quit;
```

4. Combining the datasets

For actual “values” of cells, numeric values are captured directly in the sheet XML files, while character strings are split off into the “sharedStrings” XML file.

This is the tricky part, but here is something that may help:

XML Code	Meaning
<pre><c r="B6" s="4"> <v>31001</v> </c></pre>	Cell B6 uses the styleID 4 (merged on from style) and the tag <v> contains the cell’s numeric value 31001
<pre><c r="A6" s="5" t="s"> <v>10</v> </c></pre>	Cell A6 uses the styleID 5 but has a type of “ s ” (string). The value in <v> is now a reference ID, not a value. The value is found in sharedString with ID 10

5. Transpose

Once you've read in all your XML files and combined the your information, you'll have one long dataset, each cell being 1 row. To convert it back to a tabular structure you will need to transpose the dataset.

There are multiple ways to get the column/rows, e.g. using the r= attribute from previous slide, or using ORDINALs, or other XML attributes.

Doing a transpose straight away will give you every cell in your Excel that contains data.

	 a	 b	 c	 d	 e	 f
1	Title					
2	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
3	1	abc	2	4	13	
4	2	def	2	1	2	31

But the interesting part comes next...

6. And I get the metadata too?!

Yes! Those “style” XML files contain fonts, colours, borders, Bold/Italics information etc.

This means, before you transpose (or after) you can make use of these.

	row	COL	CellId	styleId	type	v	value	fontBold	fillColorTheme	fillPattern	fontName
1	1	1	A1	1	s	0	Title	1	0	solid	Calibri
2	2	1	A2	2	s	2	Column 1	0	9	solid	Calibri
3	2	2	B2	2	s	3	Column 2	0	9	solid	Calibri
4	3	1	A3	3		4	1	0	5	solid	Calibri
5	3	2	B3	3		5	2	0	5	solid	Calibri

6. And I get the metadata too?!

Some ideas:

- Use background colours or font size to automatically identify column headers, or titles (good if you get a series of similar-but-different XLSX files)
- Use information on borders to decide if you've reached footnotes
- If your XLSX file flags important data using font styles (e.g. bolding) then you have this info which you can utilise either before, or after transpose
- And much, much more!

Thank you for listening!

Follow up questions:

Will Greenway
Principal Programmer II



Email: will.greenway@quanticate.com
Web: www.quanticate.com
LinkedIn: [willgreenway](https://www.linkedin.com/in/willgreenway)

{CODED TO CARE}