

Help, I've Received a Spreadsheet File from StarOffice Calc...!

Author: Philip R Holland, Holland Numerics Ltd

Abstract

We have been conditioned to accept spreadsheet files from Excel, which require SAS/ACCESS to read (unless they have been saved as delimited files). When a spreadsheet file is sent to us from StarOffice, or OpenOffice.org, Calc, we are likely to panic, but we should not even break into a sweat, as Base SAS is more than enough to read all the data from this compressed XML format.

This paper will show you how to read text, numbers, percentages, currency values, dates and times from this spreadsheet file format. If you want you will even be able to read the formulae too!

Introduction

The real discussion for this paper centres on how to deal with XML layouts that are easier to read using **data** steps than with the XML engine. When the data value is located in different places in the XML structure for each type of data item, the most flexible way of reading data into SAS tables is probably the **data** step.

Opening the Spreadsheet File

All proprietary data layouts associated with the StarOffice, and OpenOffice.org, program suite are based around the same compressed XML file layout. In each file type, whether it is a text (*.sxw), spreadsheet (*.sxc), drawing (*.sxd) or presentation (*.sxp) document, the data is stored in a file called **content.xml** inside the compressed file. This file can be extracted from the sample StarOffice Calc file, **oosample.sxc**, by unzipping it using a suitable utility, e.g. WinZip, pkunzip, unzip, etc.

```
x cd "C:\OOo";
```



*The "." notation allows the **filename** and **libname** statements to refer to current folder allocated with the **x cd** statement.*

```
filename sxmdir ".";  
libname sasdata ".";
```



*The **filename pipe** option directs all the output from the command line statement between the quotes through the **fileref**, as if it were a normal text file.*

```
filename sxc pipe 'PKUNZIP.EXE -o oosample.sxc content.xml';  
data _null_;  
  infile sxc truncover;  
  input;  
  put _infile_;  
run;
```

The **content.xml** file contains a continuous stream of XML tags and data values, which must be split into separate lines of tags and data prior to analysis.

The sample text from **content.xml** below shows the beginning of the XML data.

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE office:document-content PUBLIC "-//
OpenOffice.org//DTD OfficeDocument 1.0//EN" "office.dtd"><office:document-content xmlns:office="
http://openoffice.org/2000/office" xmlns:style="http://openoffice.org/2000/style" xmlns:text="
http://openoffice.org/2000/text" xmlns:table="http://openoffice.org/2000/table" xmlns:draw="
http://openoffice.org/2000/drawing" xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:xlink="
http://www.w3.org/1999/xlink" xmlns:number="http://openoffice.org/2000/datastyle" xmlns:svg="
http://www.w3.org/2000/svg" xmlns:chart="http://openoffice.org/2000/chart" xmlns:dr3d="
http://openoffice.org/2000/dr3d" xmlns:math="http://www.w3.org/1998/Math/MathML" xmlns:form="
http://openoffice.org/2000/form" xmlns:script="http://openoffice.org/2000/script" office:class="
spreadsheet" office:version="1.0"><office:script/><office:font-decls><style:font-decl style:name="
Andale Sans UI" fo:font-family="&apos;Andale Sans UI&apos;" style:font-pitch="variable"/
><style:font-decl style:name="Tahoma" fo:font-family="Tahoma" style:font-pitch="variable"/>
```

Extracting the XML

The splitting of the **content.xml** file into separate lines of tags and data prior to processing is done to make the analysis more straightforward. As the XML tags all begin with “<” and end with “>”, it is possible to start a new line before “<”, or after “>” to separate the lines.

```
data _null_ ;
  infile sxcdir(content.xml) recfm=n lrecl=32760 end=eof;
  file sxcdir(temp1.xml) lrecl=5120;
  length char1 $1;
  do until (eof);
    input char1 $char1.;
    if char1='>' then do;
      put char1;
    end;
    else if char1='<' then do;
      put;
      put char1 +(-1) @;
    end;
    else put char1 +(-1) @;
  end;
  stop;
run;
```

The sample text from **temp1.xml** below shows some of the data tags from the middle of the XML data.

```
<table:table-row table:style-name="ro1">
<table:table-cell>
<text:p>
Date
</text:p>
</table:table-cell>
<table:table-cell>
<text:p>
Time
</text:p>
</table:table-cell>
<table:table-cell>
<text:p>
Datetime
</text:p>
</table:table-cell>
</table:table-row>
```

Now that the data has been separated into XML tags and data values it is easier to filter out the unnecessary tags, knowing that the data items are stored in a hierarchical data structure:

```
• <?xml>
•   <office:body>
•     <table:table>
•       <table:table-row>
•         <table:table-cell>
•           data item.....
•         </table:table-cell>
•       </table:table-row>
•     </table:table>
•   </office:body>
```

This means that we can safely discard everything that is not associated with this hierarchy, such as any XML tags concerned with style and appearance, to leave the 'raw' data.

```
data sasdata._xml (keep=_record _level _first _last _text);
  infile sxcdir(temp1.xml) lrecl=5120 truncover;
  length _record _level _first _last 8
         _text $200
         ;
  retain _level 0;
  input _text $char200.;
  _record=_n_;
  select;
    when (_text='<?xml') do; _level=0; _first=1; end;
    when (_text='<office:body') do; _level=1; _first=1; end;
    when (_text='</office:body') do; _level=1; _last=1; end;
    when (_text='<table:table ') do; _level=2; _first=1; end;
    when (_text='</table:table>') do; _level=2; _last=1; end;
    when (_text='<table:table-row') do; _level=3; _first=1; end;
    when (_text='</table:table-row') do; _level=3; _last=1; end;
    when (_text='<table:table-cell') do; _level=4; _first=1; end;
    when (_text='</table:table-cell') do; _level=4; _last=1; end;
    when (_text='<office:') return;
    when (_text='</office:') return;
    when (_text='<style') return;
    when (_text='</style') return;
    when (_text='<number') return;
    when (_text='</number') return;
    when (_text='<table:') return;
    when (_text='</table:') return;
    when (_text='<text') return;
    when (_text='</text') return;
    when (_text=' ') return;
    otherwise;
  end;
  if _text='<?xml' or _level>0 then output;
run;
```

The sample data from the `_xml` SAS table below shows the structure of the XML tags and data items from the beginning of the filtered XML data.

_xml SAS table:

<u>record</u>	<u>level</u>	<u>first</u>	<u>last</u>	<u>text</u>
2	0	1	.	<?xml version="1.0" encoding="UTF-8"?>
231	1	1	.	<office:body>
233	2	1	.	<table:table table:name="dates and times" table:style-name="ta1">
241	3	1	.	<table:table-row table:style-name="ro1">
243	4	1	.	<table:table-cell>
246	4	.	.	Date
249	4	.	1	</table:table-cell>
251	4	1	.	<table:table-cell>
254	4	.	.	Time
257	4	.	1	</table:table-cell>
259	4	1	.	<table:table-cell>
262	4	.	.	Datetime
265	4	.	1	</table:table-cell>
267	3	.	1	</table:table-row>
269	3	1	.	<table:table-row table:style-name="ro2">
271	4	1	.	<table:table-cell table:value-type="date" table:date-value="2000-02-01">
274	4	.	.	01/02/2000
277	4	.	1	</table:table-cell>
279	4	1	.	<table:table-cell table:style-name="ce2" table:value-type="time" table:time-value="PT01H23M00S">
282	4	.	.	01:23:00
285	4	.	1	</table:table-cell>

Counting the Columns in each Sheet

Now that we have information about the data structure in each worksheet in the spreadsheet (i.e. from level 2 onward), it is straightforward to count the rows in the overall data for each sheet to create a lookup table which will help us to identify the source worksheet of each data item read.

```
data sasdata._xmltable (keep=_record_first _record_last _table);
  set sasdata._xml;
  where _level=2;
  length _table $32;
  retain _record_first _record_last . _table ' ';
```



Using the *translate()* function allows worksheets with names that include blanks to be converted to valid SAS table names containing underscores.

```
if _first=1 then do;
  _table=translate(trim(scan(_text,2,'"')), '_',' ');
  _record_first=_record;
  _record_last=.;
end;
else if _last=1 then do;
  _record_last=_record;
  output;
end;
run;

proc sort data=sasdata._xmltable;
  by _table;
run;
```

_xmltable SAS table:

<u>table</u>	<u>record_first</u>	<u>record_last</u>
currencies	591	667
dates_and_times	233	589
formulae	669	1317

The level 4 information, which relates to the columns in each worksheet, can now be used to determine the maximum number of columns that need to be created in the final SAS table from each worksheet.

```
data _xmlcolumn (keep=_table _columns);
  set sasdata._xml;
```



We are only interested in the first occurrences of level 2 (table) and level 4 (cell) data, but both first and last occurrences of level 3 (row) data, so that we find the maximum column count for each table.

```

where (_level in (2,4) and _first=1) or _level=3;
length _table $32
      _columns 8
      ;
retain _table ' ' _columns .;
if _level=2 then do;
  _table=translate(trim(scan(_text,2,'"')), '_',' ');
  return;
end;
else if _level=3 and _first=1 then do;
  _columns=0;
  return;
end;
else if _level=4 then do;
  _columns+1;
  return;
end;
else if _level=3 and _last=1 then do;
  output;
end;
run;

proc summary data=_xmlcolumn nway;
class _table;
var _columns;
output out=sasdata._xmlcolumn (drop=_freq_ _type_) max=;
run;

proc sort data=sasdata._xmlcolumn;
by _table;
run;

```

_xmlcolumn SAS table:

<u>_table</u>	<u>_columns</u>
currencies	1
dates_and_times	3
formulae	6

Reading the Cell Values or Formulae into SAS

To make it simpler to identify the original worksheet for each data item, SAS formats are generated to act as lookup tables.



*Generating SAS formats by creating **cntlin** files from input data is a way of ensuring that the format matches the data to be formatted, and is useful, as in this case, for generating fast lookup tables as an alternative to joining 2 tables.*

```

data cntlin (keep=fmtname start end label hlo type);
length fmtname $7
      start end label $32
      hlo type $1
      ;
set sasdata._xmltable;
by _table;
hlo=' ';
fmtname='rngtab';
type='N';
start=trim(left(put(_record_first,16.)));
end=trim(left(put(_record_last,16.)));
label=_table;
output;
run;

proc sort data=cntlin nodupkeys;
by fmtname start;
run;

proc format cntlin=cntlin;
run;

```

The SAS code to create a SAS table for each worksheet in the spreadsheet file is then generated automatically into 4 SAS catalog entries, which can be stitched together to create the final Data Step.

```
filename _src catalog 'sasdata._xml';
data _null_;
  set sasdata._xmlcolumn;
```



The static code should be generated when reading the first record from the input table (i.e. `_n_=1`).

```
if _n_=1 then do;
```



The `_data.source` code includes the `data` statement, which will then have all the generated table names appended to it.

```
file _src(_data.source);
put 'data';
```



The `_process.source` code takes each type of cell data and converts it to a suitable formatted text value.

```
file _src(_process.source);
put ' ';
put ' set sasdata._xml;';
put ' where _level in (3,4);';
put ' length var1-var256 $200 _type $16 _temp $200;';
put ' array v {*} var1-var256;';
put ' retain var1-var256 " " _count . _repeat . _type ' ';';
put ' if _level=3 and _first=1 then do;';
put '   _count=0;';
put '   _repeat=.;';
put '   do _i=1 to dim(v);';
put '     v(_i)="";';
put '   end;';
put ' end;';
put ' else if _level=4 and _first=1 then do;';
put '   if _repeat>1 then do _i=1 to (_repeat-1);';
put '     _count+1;';
put '     v(_count)=v(_count-1);';
put '   end;';
put '   _type="";';
put '   _count+1;';
```



Repeated cells are a space-saving feature of the layout, so the number of repeated cells must be store for later use.

```
put '   _offset=index(_text,"table:number-columns-repeated=");';
put '   if _offset>0 then _repeat=input(scan(substr(_text,_offset+
length("table:number-columns-repeated=")+1),1,""),best.);';
put '   else _repeat=.;';
put '   _offset=index(_text,"table:value-type=");';
```



The formula to create the value can be found in the table:formula= parameter, e.g. "`=39+39`" would be the same in Excel, but "`=IF([.J67]-[.I67]>0;[.J67]-[.I67];0)`" would be "`=IF(J67-I67>0;J67-I67;0)`" in Excel.

```
put '   if _offset>0 then do;';
put '     _type=scan(substr(_text,_offset+length("table:value-type=")+1),1,"");';
put '     select (_type);';
```



The time data is stored in the table:time-value= parameter in the format "`PthhHmmMss,ddS`", where `hh`=hours, `mm`=minutes and `ss,dd`=seconds. Note that the decimal separator is a comma, and not a full stop.

```
put '   when ("time") do;';
put '     _offset=index(_text,"table:time-value=");';
put '     _temp=scan(substr(_text,_offset+length("table:time-value=")+1),1,
"");';
put '     v(_count) = put(hms(input(scan(_temp,1,"PTHMS"),best.)),');
```

```

put '          input(scan(_temp,2,"PTHMS"),best.);';
put '          input(translate(scan(_temp,3,"PTHMS"),".",""),
                    best.)),time11.2);';
put '      end;';

```



The date data is stored in the table:date-value= parameter in the format "ccyy-oo-aa", where ccyy=year, oo=month of year and aa=day of month. Datetime values are also stored in the table:date-value= parameter, but in the format "ccyy-oo-aaThh:mm:ss,dd" Note again, like the time data, the decimal separator is a comma, and not a full stop.

```

put '      when ("date") do;';
put '          _offset=index(_text,"table:date-value=");';
put '          _temp = scan(substr(_text,_offset+length("table:date-value")+1),
                          1,"");';
put '          if scan(_temp,2,"T") ne " " then';
put '              v(_count) = put(dhms(input(scan(_temp,1,"T"),yymmdd10.),';
put '                              0,0,input(translate(scan(_temp,2,"T"),".",""),
                              time11.)),datetime21.2);';
put '          else';
put '              v(_count) = put(input(scan(_temp,1,"T"),yymmdd10.),date9.);';
put '      end;';

```



The float and percentage data are stored in the table:value= parameter.

```

put '      when ("float","percentage") do;';
put '          _offset=index(_text,"table:value=");';
put '          v(_count)=scan(substr(_text,_offset+length("table:value")+1),1,"");';
put '      end;';

```



The currency data is stored in the table:currency= and table:value= parameters, which are concatenated together, e.g. £103.81 would be converted to "GBP103.81".

```

put '      when ("currency") do;';
put '          _offset=index(_text,"table:currency=");';
put '          v(_count)=scan(substr(_text,_offset+length("table:currency")+1),
                          1,"");';
put '          _offset=index(_text,"table:value=");';
put '          v(_count)=trim(v(_count)) !! scan(substr(_text,_offset+
                    length("table:value")+1),1,"");';
put '      end;';
put '      otherwise;';
put '      end;';
put '      end;';
put '      end;';
put '      else if _level=4 and _first ne 1 and _last ne 1 then do;';

```



Data other than time, date, float, percentage or currency data is assumed to be text, and are taken directly from the records between the <table:table-cell> and </table:table-cell> tags, which are concatenated together into a single string.

```

put '      if not (_type in ("time","date","float","percentage","currency")) then
put '          v(_count)=trim(v(_count)) !! trim(_text);';
put '      end;';

```



The **_output.source** code contains the **select** clauses which direct the records to the correct tables, based on their original worksheets.

```

file _src(output.source);
put ' if _level=3 and _last=1 then do;';

```



Repeated cells are a space-saving feature of the layout, so, when creating the SAS table, you must expand out the repeated cells into separate column values to prevent loss of data.

```

put '      if _repeat>1 then do _i=1 to (_repeat-1);';
put '          _count+1;';
put '          v(_count)=v(_count-1);';
put '      end;';

```

```
put ' select (put(_record,rngtab.));';
```



The `_run.source` code completes the `select` clauses and also finishes the `data` step.

```
file _src(_run.source);
put ' otherwise;';
put ' end;';
put ' end;';
put 'run;';
end;
j=trim(left(put(_columns,16.)));
file _src(_data.source);
put ' sasdata.' _table '(keep=var1-var' j +(-1) ')';
file _src(_output.source);
put ' when ("' _table +(-1) ') output sasdata.' _table ';';
run;
```

All the generated code is then executed to populate the tables with the formatted text extracted from every cell in all of the worksheets.

```
%include _src(_data.source,_process.source,_output.source,_run.source) / source2;
```

Summary

The most important factor in extracting any new data is to thoroughly research the structure of the data before starting to process it, so that specific features are known in advance. In this case the critical features are:

- Repeated cells.
- The different locations of data for time, date, float, percentage, currency and text data.
- The relevant and irrelevant XML tags.

As most of the important information about the data stored in the compressed XML file format used by StarOffice Calc is located inside the tags themselves, rather than between the tags, processing the raw XML in a data step is the most reliable method of extracting all the data items from the spreadsheet cells.

References

- The sample StarOffice Calc file used throughout this paper, **oosample.sxc**, can be downloaded from the Holland Numerics web site, at <http://www.hollandnumerics.com/SASPAPER.HTM>.
- Another sample StarOffice Calc file, **consultants.sxc**, can be downloaded from the OpenOffice.org project web site, at <http://bizdev.openoffice.org/consultants.html>.
- Full documentation of the file layouts for StarOffice and OpenOffice.org documents can be found at <http://xml.openoffice.org>.

Contact Details

The author is a consultant for Holland Numerics Ltd and can be contacted at the following address:

Philip R Holland
address: Holland Numerics Ltd
94 Green Drift
Royston
Herts. SG8 5BT
UK
e-mail: <phil.holland@bcs.org.uk>
web: <http://www.hollandnumerics.com/>
tel. (mobile): +44-(0)7714-279085

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

Appendix 1. Generated output files

_data.source SAS catalog entry:

```
data
  sasdata.currencies (keep=var1-var1)
  sasdata.dates_and_times (keep=var1-var3)
  sasdata.formulae (keep=var1-var6)
```

_process.source SAS catalog entry:

```

;
set sasdata._xml;
where _level in (3,4);
length var1-var256 $200 _type $16 _temp $200;
array v {*} var1-var256;
retain var1-var256 " " _count . _repeat . _type ;
if _level=3 and _first=1 then do;
  _count=0;
  _repeat=.;
  do _i=1 to dim(v);
    v(_i)="";
  end;
end;
else if _level=4 and _first=1 then do;
  if _repeat>1 then do _i=1 to (_repeat-1);
    _count+1;
    v(_count)=v(_count-1);
  end;
  _type="";
  _count+1;
  _offset=index(_text,"table:number-columns-repeated=");
  if _offset>0 then _repeat=input(scan(substr(_text,_offset+
    length("table:number-columns-repeated=")+1),1,""),best.);

  else _repeat=.;
  _offset=index(_text,"table:value-type=");
  if _offset>0 then do;
    _type=scan(substr(_text,_offset+length("table:value-type=")+1),1,"");
    select (_type);
      when ("time") do;
        _offset=index(_text,"table:time-value=");
        _temp=scan(substr(_text,_offset+length("table:time-value=")+1),1,"");
        v(_count) = put(hms(input(scan(_temp,1,"PTHMS"),best.),
          input(scan(_temp,2,"PTHMS"),best.),
          input(translate(scan(_temp,3,"PTHMS"),".",""),best.)),time11.2);
      end;
      when ("date") do;
        _offset=index(_text,"table:date-value=");
        _temp = scan(substr(_text,_offset+length("table:date-value=")+1),1,"");
        if scan(_temp,2,"T") ne " " then
          v(_count) = put(dhms(input(scan(_temp,1,"T"),ymmdd10.),
            0,0,input(translate(scan(_temp,2,"T"),".",""),time11.)),
            datetime21.2);
        else v(_count) = put(input(scan(_temp,1,"T"),ymmdd10.),date9.);
      end;
      when ("float","percentage") do;
        _offset=index(_text,"table:value=");
        v(_count)=scan(substr(_text,_offset+length("table:value=")+1),1,"");
      end;
      when ("currency") do;
        _offset=index(_text,"table:currency=");
        v(_count)=scan(substr(_text,_offset+length("table:currency=")+1),1,"");
        _offset=index(_text,"table:value=");
        v(_count)=trim(v(_count)) !! scan(substr(_text,_offset+length("table:value=")+1),1,"");
      end;
    otherwise;
  end;
end;
end;
end;
else if _level=4 and _first ne 1 and _last ne 1 then do;
  if not (_type in ("time","date","float","percentage","currency"))
  then v(_count)=trim(v(_count)) !! trim(_text);
end;
```

_output.source SAS catalog entry:

```
if _level=3 and _last=1 then do;
  if _repeat>1 then do _i=1 to (_repeat-1);
    _count+1;
    v(_count)=v(_count-1);
  end;
  select (put(_record,rngtab.));
    when ("currencies") output sasdata.currencies ;
    when ("dates_and_times") output sasdata.dates_and_times ;
    when ("formulae") output sasdata.formulae ;
```

_run.source SAS catalog entry:

```
    otherwise;
  end;
end;
run;
```