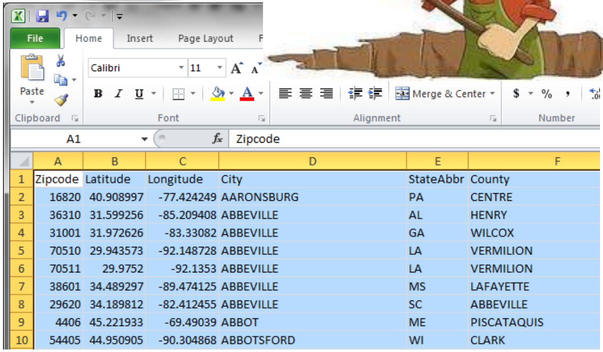
Mining Excel 2.0



	A	B	C	D	E	F
1	Zipcode	Latitude	Longitude	City	StateAbbr	County
2	16820	40.908997	-77.424249	AARONSBURG	PA	CENTRE
3	36310	31.599256	-85.209408	ABBEVILLE	AL	HENRY
4	31001	31.972626	-83.33082	ABBEVILLE	GA	WILCOX
5	70510	29.943573	-92.148728	ABBEVILLE	LA	VERMILION
6	70511	29.9752	-92.1353	ABBEVILLE	LA	VERMILION
7	38601	34.489297	-89.474125	ABBEVILLE	MS	LAFAYETTE
8	29620	34.189812	-82.412455	ABBEVILLE	SC	ABBEVILLE
9	4406	45.221933	-69.49039	ABBOT	ME	PISCATAQUIS
10	54405	44.950905	-90.304868	ABBOTSFORD	WI	CLARK

Introduction to Excel/OOXML

Excel at BCA Research...

Global Economic Analysis

BCA Edge
Reduce the time it takes to discover and integrate actionable insights into your investment workflow.

BCA Analytics
Integrate charts and analytics into your decision-making and presentations.

BCA Indicators
Leading indicators from the market leaders.

BCA Online
Access BCA's market-leading content from anywhere at any time.

bcaresearch.com > Solutions > BCA Indicators

BCA Indicators

Analytics. Insight. Integrated.

BCA Indicators directly converts our in depth research into quantitative insight to inform and drive investment decision-making.

Leading indicators from the market leaders.

GLOBAL COVERAGE

Access over 200 proprietary indicators across every asset class and region, built by our respected strategists and analysts.

INTEGRATED

View or download indicator data in order to integrate into investment decision-making frameworks and models.

UP-TO-DATE

Use our alerting engine to receive real-time alerts when data changes, so you can quickly integrate new insights into trades.

"By integrating BCA Indicators into our investment framework,

- **Data sources - (Bloomberg, ThomsonReuters...) make data available as .xlsx or .csv**
- **Lists - keeping track of... user profiles, data retrieval codes, publication files, etc.**
- **Interfaces for data collection - downloads and analytical tools are driven by Excel Add-ins**
- **Charts - if no other way to produce**
- **Statistics - if no better way to calculate**
- **Reports - presentation of analyses; lists of things that need attention, etc.**
- **Process control - "table-driven" tasks - determine what to do based on worksheet contents**

In short, Excel is extremely important, and EVERYWHERE

Excel and XML

Office Open XML = "OOXML"

- **2000-2006 - standardization process**
- **MS Office 2007 - adopts the format**
- ***.xlsx, not *.xls**

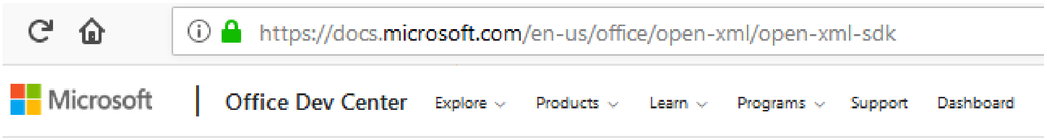
Office Open XML

From Wikipedia, the free encyclopedia

Not to be confused with Open Office XML

Office Open XML (also informally known as OOXML or OpenXML) is a <u>zipped, XML-based</u> file format developed by Microsoft for representing spreadsheets, charts, presentations and word processing documents (Excel, PowerPoint, Word).

The Office Open XML SDK


https://docs.microsoft.com/en-us/office/open-xml/open-xml-sdk

The SDK is built on the System.IO.Packaging API and provides strongly-typed classes to manipulate documents that adhere to the Office Open XML File Formats specification. The Open XML file formats are useful for developers because they are an ***open standard*** *and are* ***based on well-known technologies: ZIP and XML.***

The Open XML SDK 2.5 simplifies the task of manipulating Open XML packages and the underlying Open XML schema elements within a package. The SDK encapsulates many common tasks that developers perform on Open XML packages, so that you can perform complex operations with just a few lines of code.

What about COM (OLE, ActiveX...), and CSV?

COM continues to be used, but issues include:

- deployment; cost
- resource footprint
- performance
- automation challenges
- other limitations

```
'EX'⎕WC'OLEClient' 'Excel.Application'
'EX.WB'⎕WC EX.Workbooks
'WBO'EX.WB.Open⊂BOOK
'SHEET1'⎕WC EX.WB.WBO.ActiveSheet
'RNG1'⎕WC SHEET1.UsedRange
SDATA←RNG1.Value2
.......
```

CSV

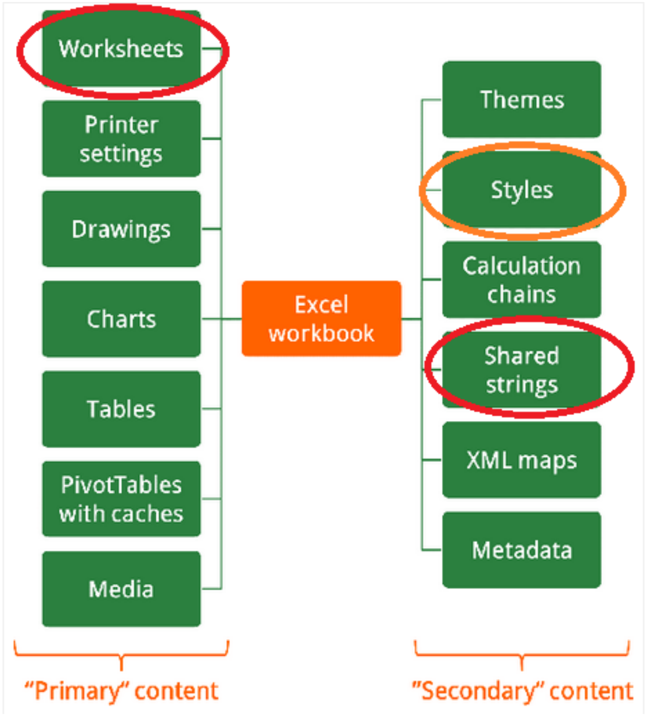
- still a very common interchange format
- esp. for spreadsheet creation / tabular data management
- so, still an essential tool for numerous tasks; now we have - ⎕CSV
- datatype concerns?

Open XML - Structure

Key components:

- Worksheets
- Shared Strings
- Styles (in part)

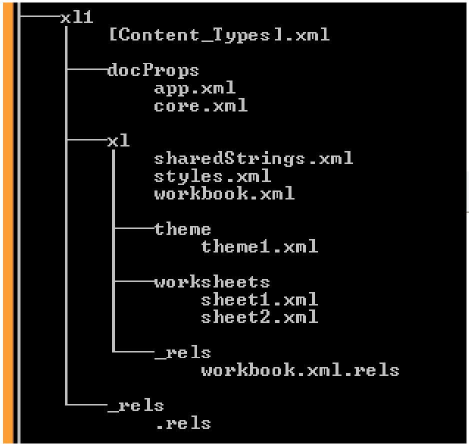
(similar for Word, PowerPoint)



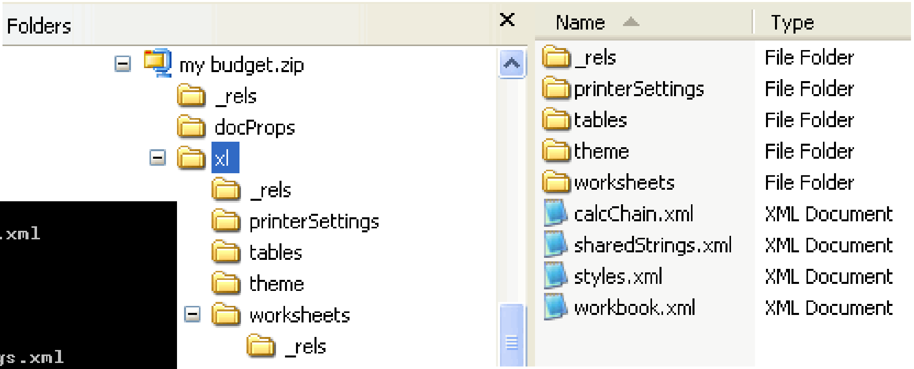
(https://professor-excel.com/xml-zip-excel-file-structure/)

So what exactly is in that ZIP?

(eg. http://officeopenxml.com/SScontentOverview.php)

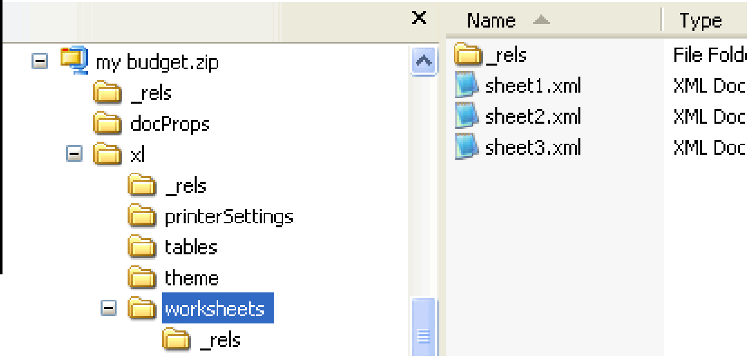


```
──xl1
  │     [Content_Types].xml
  │
  ├──docProps
  │       app.xml
  │       core.xml
  │
  ├──xl
  │   │   sharedStrings.xml
  │   │   styles.xml
  │   │   workbook.xml
  │   │
  │   ├──theme
  │   │       theme1.xml
  │   │
  │   ├──worksheets
  │   │       sheet1.xml
  │   │       sheet2.xml
  │   │
  │   └──_rels
  │           workbook.xml.rels
  │
  └──_rels
          .rels
```



```
Folders
⊟ my budget.zip
    _rels
    docProps
  ⊟ xl
      _rels
      printerSettings
      tables
      theme
    ⊟ worksheets
        _rels
```

Name	Type
_rels	File Folder
printerSettings	File Folder
tables	File Folder
theme	File Folder
worksheets	File Folder
calcChain.xml	XML Document
sharedStrings.xml	XML Document
styles.xml	XML Document
workbook.xml	XML Document



```
⊟ my budget.zip
    _rels
    docProps
  ⊟ xl
      _rels
      printerSettings
      tables
      theme
    ⊟ worksheets
        _rels
```

Name	Type
_rels	File Fold
sheet1.xml	XML Doc
sheet2.xml	XML Doc
sheet3.xml	XML Doc

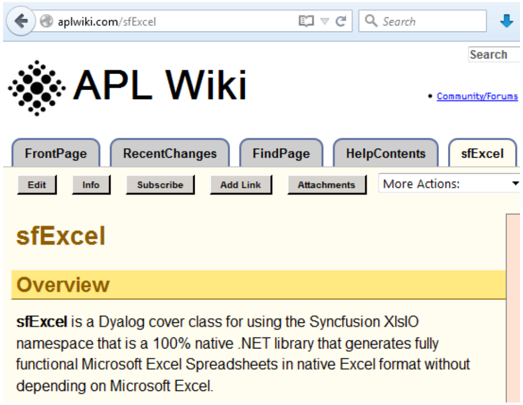
Current Dyalog APL OOXML-based Utilities

Syncfusion XlsIO

Essential XlsIO is a native **.NET** class library that can be used to create and modify **Microsoft Excel** files by using C#, VB.NET and managed C++ code. It is a non-UI component that provides a full-fledged object model that facilitates accessing & manipulating the spreadsheets without any dependency of Microsoft Office COM libraries & Microsoft Office.

sfExcel

- **excellent toolkit for Dyalog APL and XlsIO**
- **"DataTable" feature for speedups**
- **well documented and tested**



sfExcel

Overview

sfExcel is a Dyalog cover class for using the Syncfusion XlsIO namespace that is a 100% native .NET library that generates fully functional Microsoft Excel Spreadsheets in native Excel format without depending on Microsoft Excel.

Using the OOXML SDK Directly

Google It - to see examples of typical C# code to extract cell values...

```
// create a new string array
string[] theArray = new string[values.Length];

// loop through the 2-D System.Array and populate the 1-D String Array
 for (int i = 1; i <= values.Length; i++)
  {
   if (values.GetValue(1, i) == null)
    theArray[i-1] = "";
  else
   theArray[i-1] = (string)values.GetValue(1, i).ToString();
  }
  return theArray;
```

ie. get individual cell values - <u>one at a time in a loop</u>

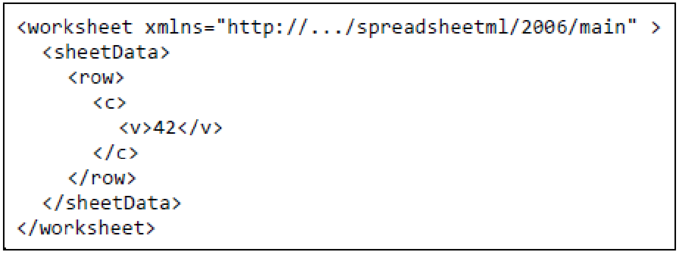
OOXML and APL - *what is the best strategy?*

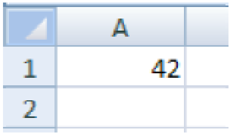
The SDK offers properties & methods to grab individual items, or...
the <u>ENTIRE worksheet</u> XML

ie. since we then "have" the raw markup, containing all the data...
...all we have to do is...???

```
      ⍴xml← {Excel/OpenXML worksheet object}...Worksheet.OuterXml
159478
```

```
<x:worksheet xmlns.......><x:v>4</x:v></x:c><x:c r="F1" t="s"><x:v>5</x:v></x:c><x:c r="G1"
t="s"><x:v>6</x:v></x:c><x:c r="H1" t="s"><x:v>7</x:v></x:c><x:c r="I1" t="s"><x:v>8</x:v></x:c><x:c
r="J1" t="s"><x:v>9</x:v></x:c><x:c r="K1" t="s"><x:v>10</x:v></x:c></x:row><x:row r="2"
spans="1:11" x14ac:dyDescent="0.25"><x:c r="A2" t="s"><x:v>11</x:v></x:c><x:c r="B2" t="s"><x:v>12
```


```
<worksheet xmlns="http://.../spreadsheetml/2006/main" >
  <sheetData>
    <row>
      <c>
        <v>42</v>
      </c>
    </row>
  </sheetData>
</worksheet>
```



Excel to APL Array -> implementation options

Processing the worksheet XML,

may depend on your objectives and the data itself:

- columns of single datatype?
- focused selections?, eg. named-ranges, particular rows/cols
- conditional selections? (ignore unnecessary segments?)
- need for meta-data? (formulas, styles, etc.)
- are strings/datatypes important? (or just the numbers?)
- batch, pre-processing?
- very large spreadsheets?

But most likely, in general...

Excel to APL Array - Typical processing (in brief)

- get XML for one worksheet
- determine result array shape; create vector of cell contents
- determine cell datatypes (strings, numerics, other)
- extract key item(s) from each cell (all still text strings at this point)
- convert numerics, dates, error items, or other numeric variations
- for (datatype = string) cells...
    - r[where shared strings]← sharedStrings.xml[cell values are the indices]
    - r[where inline]← r[strings provided in those cells]
- de-escape strings (&amp;→ &...)
- check for empty cell locations - expand if needed; reshape to 2d
- apply tests and throw exceptions as needed (missing/invalid data etc.)

so to get that worksheet XML...

Excel to APL Array - the OOXML SDK

(sort of like this...)

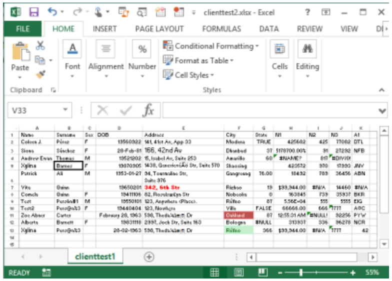
```
 ⎕USING←,⊂',DocumentFormat.OpenXml.dll' ⍝ installed in Dyalog folder

⍝ gather all necessary xml; Open Workbook obj; 0 = readonly
 d1←DocumentFormat.OpenXml.Packaging.SpreadsheetDocument.Open fname 0
 d2←d1.WorkbookPart.Workbook.Sheets    ⍝ Worksheets collection
 cn←d2.ChildElements.Count
 :For sn :In 0,⍳¯1+cn                  ⍝ sheet#s 0-origin
     flds←'SheetId' 'Name' 'Id'
     sx←flds,[1.5]⍕¨⍎¨(⊂'d2.ChildElements.Item[sn].'),¨flds
     id←3⊃sx[;2]
     sd←d1.WorkbookPart.GetPartById⊂,id     ⍝ worksheet obj
     sdx←sd.Worksheet.OuterXml              ⍝ xml
```

...essentially, navigate your way through the key worksheet items, gather contents, then process the results into APL-ready data

Excel to APL Array - the OOXML SDK

conceptually...

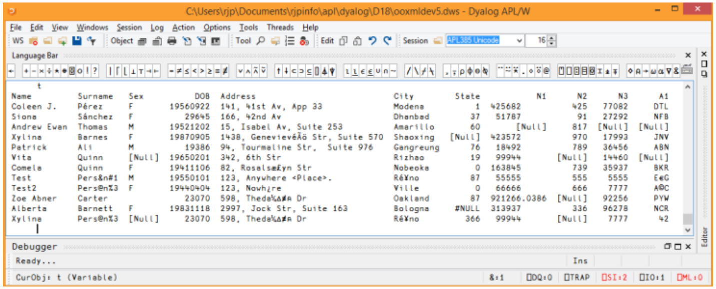


the SDK <u>unzips</u> *.xlsx file contents dynamically, exposing them to Methods and Properties we can manipulate to obtain worksheet XML contents

```
<x:worksheet xmlns......> <x:v>4</x:v></x:c>
<x:c r="F1" t="s"><x:v>5</x:v></x:c>
<x:c r="G1" t="s"><x:v>6</x:v></x:c>
<x:c r="H1" t="s"><x:v>7</x:v></x:c>
```

repeat to gather sharedStrings, style.xml, and perhaps other items...

process xml strings into APL array(s)



```
      t
Name         Surname   Sex         DOB  Address                       City         State            N1      N2      N3     A1
Coleen J.    Pérez     F      19560922  141, 41st Av, App 33          Modena           1  425682           425   77082    DTL
Siona        Sánchez   F         29645  166, 42nd Av                  Dhanbad         37   51787            91   27292    NFB
Andrew Ewan  Thomas    M      19521202  15, Isabel Av, Suite 253      Amarillo        60        [Null]     817  [Null] [Null]
Xylina       Barnes    F      19870905  1438, GenevievéÃö Str, Suite 570 Shaoxing  [Null]  423572           970   17993    JNV
Patrick      Ali       M         19386  94, Tourmaline Str,  Suite 976 Gangreung      76   18492           789   36456    ABN
Vita         Quinn     [Null] 19650201  342, 6th Str                  Rizhao          19   99944        [Null]   14460 [Null]
Comela       Quinn     F      19411106  82, RosalsæÆyn Str            Nobeoka          0  163845           739   35937    BKR
Test         Pers&n#1  M      19550101  123, Anywhere <Place>.        Ré¥no           87   55555           555    5555    E¢G
Test2        Pers@n%3  F      19440404  123, Nowh¿re                  Ville            0   66666           666    7777    A©C
Zoe Abner    Carter              23070  598, Theda%Δ≠a Dr             Oakland         87  921266.0386   [Null]   92256    PYW
Alberta      Barnett   F      19831118  2997, Jock Str, Suite 163     Bologna      #NULL  313937           336   96278    NCR
Xylina       Pers@n%3  [Null]    23070  598, Theda%Δ≠a Dr             Ré¥no          366   99944        [Null]    7777     42
      |
```

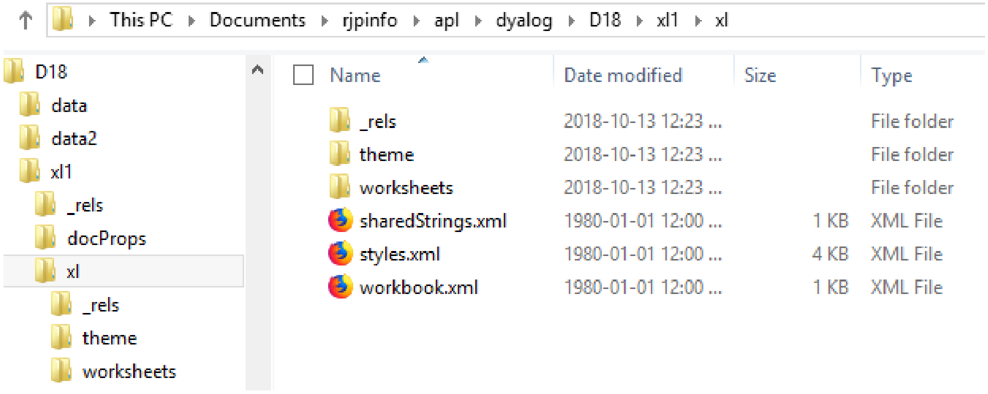
```
Ready...                                                     Ins
CurObj: t (Variable)                       &:1   ⎕DQ:0  ⎕TRAP  ⎕SI:2  ⎕IO:1  ⎕ML:0
```

Excel to APL Array - "option B"

the Excel file is a Zipped archive, so...

```
      System.IO.Compression.ZipFile.ExtractToDirectory...
```

unzips *.xlsx contents into a target folder:



This PC ▸ Documents ▸ rjpinfo ▸ apl ▸ dyalog ▸ D18 ▸ xl1 ▸ xl

- D18
  - data
  - data2
  - xl1
    - _rels
    - docProps
    - xl
      - _rels
      - theme
      - worksheets

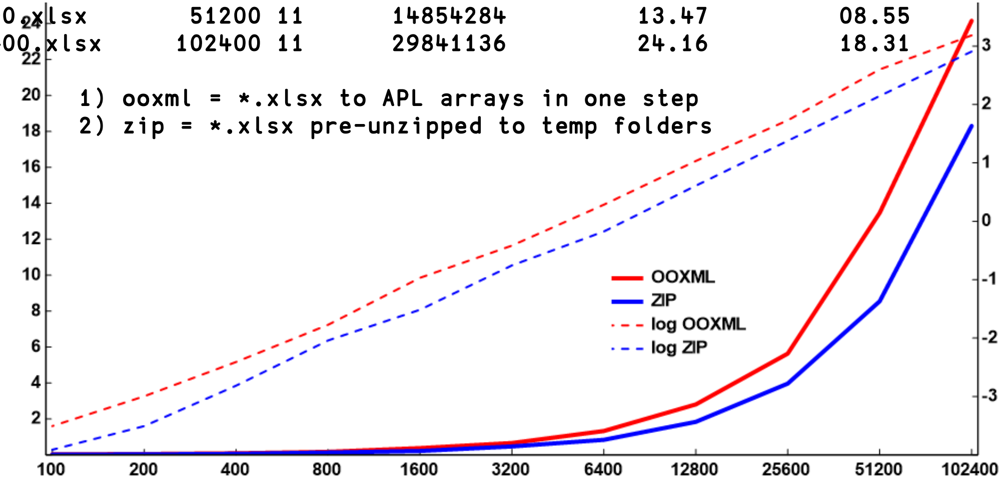
Name	Date modified	Size	Type
_rels	2018-10-13 12:23 ...		File folder
theme	2018-10-13 12:23 ...		File folder
worksheets	2018-10-13 12:23 ...		File folder
sharedStrings.xml	1980-01-01 12:00 ...	1 KB	XML File
styles.xml	1980-01-01 12:00 ...	4 KB	XML File
workbook.xml	1980-01-01 12:00 ...	1 KB	XML File

then: fetch worksheet and sharedStrings xml vectors → process...

so... OOXML or (un)ZIP - which approach is faster?

OOXML (*.xlsx as-is) vs. ZIP (pre-extracted)

```
file                          shape          bytes     ooxml(1)      zip(2)
uclients100.xlsx             100 11          28732     00.03         00.02
uclients200.xlsx             200 11          57300     00.05         00.03
uclients400.xlsx             400 11         114468     00.09         00.06
uclients800.xlsx             800 11         229008     00.17         00.13
uclients1600.xlsx           1600 11         458664     00.38         00.22
uclients3200.xlsx           3200 11         919920     00.66         00.47
uclients6400.xlsx           6400 11        1848508     01.33         00.84
uclients12800.xlsx         12800 11        3706952     02.81         01.84
uclients25600.xlsx         25600 11        7409796     05.64         03.97
uclients51200.xlsx         51200 11       14854284     13.47         08.55
uclients102400.xlsx       102400 11       29841136     24.16         18.31
```



```
1) ooxml = *.xlsx to APL arrays in one step
2) zip = *.xlsx pre-unzipped to temp folders
```

Excel to APL Array - in detail

Parse the worksheet XML strings to obtain individual cell content items...

What about ⎕XML ?

it can be used to process results, but...

```
┌→┬───┬────────────┬──────────┬─┐
↓3│x:c│            │┌→┬───┐   │3│
│ │   │            │↓r│F13│   │ │
│ │   │            │├→┼──→┤   │ │
│ │   │            ││s│13 │   │ │
│ │   │            │├→┼──→┤   │ │
│ │   │            ││t│s  │   │ │
│ │   │            │└→┴──→┘   ↓ │
├─┼──→┼───────────⊖┼─────────→┼─┤
│4│x:v│46          │┌→┬─┐     │5│
│ │   │            │⌽ │ │     │ │
│ │   │            │└⊖┴⊖┘     ⌽ │
├─┼──→┼───────────→┼─────────→┼─┤
│3│x:c│            │┌→┬───┐   │3│
│ │   │            │↓r│G13│   │ │
│ │   │            │└→┴──→┘   ↓ │
├─┼──→┼───────────⊖┼─────────→┼─┤
│4│x:f│SUM(G2:G12) │┌→┬─┐     │5│
│ │   │            │⌽ │ │     │ │
│ │   │            │└⊖┴⊖┘     ⌽ │
├─┼──→┼───────────→┼─────────→┼─┤
│4│x:v│366         │          │5│
```

alternatively: **partition, by start/end strings, using ⍷ and ⊂**

```
      ↑xmlvector stringget '<x:c ' '/x:c>'
<x:c r="F13" s="13" t="s"><x:v>46</x:v></x:c>
<x:c r="G13"><x:f>SUM(G2:G12)</x:f><x:v>366</x:v></x
```

or Regex/⎕S? - useful to a point...

Datatypes & Conversions

Numerics

A) LoadTEXT strategy: ⎕VFI everything and see what sticks ?

```
num←(⊂,1)≡∘⊃¨tmp←⎕VFI¨cols/data   ⍝ all the numeric items
:If header<∨/ncol←∧⌿num           ⍝ do we have full length columns of numbers?
```

B) ⎕CSV: column type info required

...all fields are assumed to be character fields unless otherwise specified...

C) OOXML: cells contain datatype indicators (if not, it's numeric)

eg. t="s", t="e", t="b", f=..., s=...

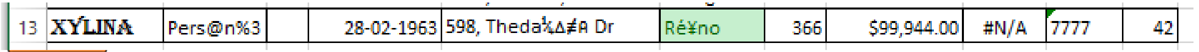
Key Tag item letters

v = value
t = type, text
c = cell, column; row = row
si = string item
f = formula
t = "s" = shared string
s = "18" = style item
r = reference, RichText items

OOXML Cell Datatypes

eg. XML for one row:

```
<row r="13" spans="1:11" ht="17.25" x14ac:dyDescent="0.3">          <- row info
<c r="A13" s="9" t="s"><v>27</v></c>                  <- sharedString #27, with style #9
<c r="B13" t="s"><v>48</v></c>                        <- sharedString #48   (note: missing "C13")
<c r="D13" s="17"><v>23070</v></c>                    <- numeric value, style #17  (date)
<c r="E13" t="s"><v>63</v></c>                        <- sharedString #63
<c r="F13" s="12" t="s"><v>45</v></c>                 <- sharedString #45, style #12
<c r="G13"><f>SUM(G2:G12)</f><v>366</v></c>               <- formula, and result value
<c r="H13" s="2"><v>99944</v></c>                     <- numeric value, style #2
<c r="I13" t="e"><v>#N/A</v></c>                      <- Excel N/A  (note: not a sharedString)
<c r="J13" s="13" t="s"><v>64</v></c>                 <- sharedString #64, style #13
<c r="K13"><v>42</v></c>                              <- numeric value
</row>
```

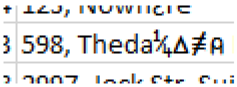


13	XYLINA	Pers@n%3		28-02-1963	598, Theda¼∆≠Ɐ Dr	Ré¥no	366	$99,944.00	#N/A	7777	42

OOXML Cell Items - sharedStrings (..."table", a.k.a. "sst")

sharedStrings.xml

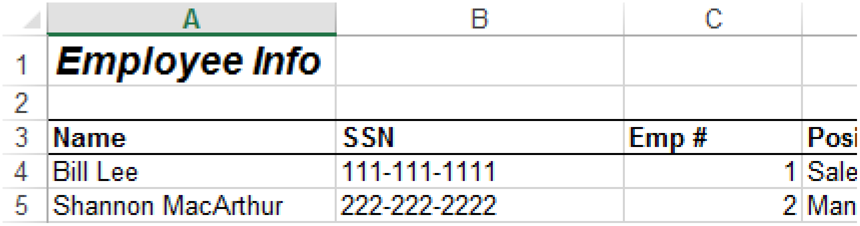
- items delimited by <si> tags
- item position = ID# (⎕IO←0)
- escaped chars (<, >, &...)
- some items may contain spans of differing fonts, styles, etc., so these text segments must be re-joined to obtain the entire string


598, Theda¼∆≢⍝

```
<sst count="81" uniqueCount="68">
 <si><t>Name</t></si>
 <si><t>Surname</t></si>
 <si><t>Address</t></si>
 <si>
  <r>
  <t>598, Theda</t>
  </r>
  <r>
   <rPr>
    <sz val="11"/>
     <color theme="1"/>
     <rFont val="APL385 Unicode"/>
     <family val="3"/>
    </rPr>
    <t>¼∆≢⍝</t>
  </r>....</si>
```

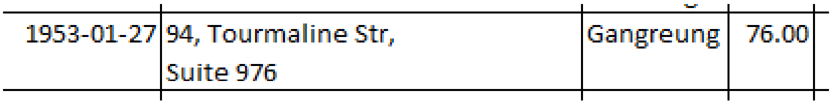
OOXML Cell Items - other translation issues

- Some cells may have style but no value items (search issue), eg.
  `<c r="A1" s="1" t="s"><v>0</v></c><c r="B1" s="1" /><c r=...`



	A	B	C	
1	*Employee Info*			
2				
3	Name	SSN	Emp #	Posi
4	Bill Lee	111-111-1111	1	Sale
5	Shannon MacArthur	222-222-2222	2	Man

- LINEFEED vs. NEWLINE, ie. Alt-Enter in cell input keystrokes
  - breaks strings into items on new lines visually
  - in Excel (visually), the character behaves like NL, but in XML = LF
  - so in APL, you may want to modify



1953-01-27	94, Tourmaline Str, Suite 976	Gangreung	76.00

OOXML Cell Items - the <x: namespace prefix

inserted by the SDK

https://social.msdn.microsoft.com/Forums/office/en-US/4fb3bb5e-ef5f-4795-837a-fae30d0ecef0/xml-prefixes-from-openxml-sdk-but-not-excel?forum=oxmlsdk

"Using a namespace prefix is only required if the XML refers to more than one namespace.... The Open XML SDK can't judge whether what you're creating will work with more than one namespace, so it's designed to always write out namespace prefixes..."

with (via SDK):

```
<x:c r="A13" s="9" t="s"><x:v>27</x:v></x:c><x:c r="B13" t="s"><x:v>48</x:v></x:c><x:c r="D13"
s="17"><x:v>23070</x:v></x:c><x:c r="E13" t="s"><x:v>63</x:v></x:c>
```

without (via ZIP):

```
<c r="A13" s="9" t="s"><v>27</v></c><c r="B13" t="s"><v>48</v></c><c r="D13" s="17"> <v>23070</v>
</c><c r="E13" t="s"><v>63</v></c
```

OOXML Cell Items - STYLES

<c r="H11" s="4"><v>921266</v></c> - linked to styles.xml items

DATES - stored as IDN, with style applied, eg.   <c r="D3" s="1"><v>29645</v></c>
(* so to identify date cells... follow the style "mapping")

style.xml - contains number formats; fonts; characteristics for fill, border, etc.

numberFormat codes: ID refer to items in a standard list (see below) or supplied definitions

```
<styleSheet xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main">
  <numFmts count="3">
    <numFmt numFmtId="164" formatCode="[$-414]mmmm\ yyyy;@" />
    <numFmt numFmtId="165" formatCode="0.000" />
    <numFmt numFmtId="166" formatCode="#,##0.000" />

<xf numFmtId="14" ... applyNumberFormat="1" />
<xf numFmtId="1" ... applyNumberFormat="1" />
```

notes:
[$-414] = "locale"

see ID table below

OOXML Cell Items - STYLES > Number Formats

eg. https://stackoverflow.com/questions/4730152/what-indicates-an-office-open-xml-cell-contains-a-date-time-value

0 = 'General';	18 = 'h:mm AM/PM';	49 = '@';
1 = '0';	19 = 'h:mm:ss AM/PM';	27 = '[$-404]e/m/d';
2 = '0.00';	20 = 'h:mm';	30 = 'm/d/yy';
3 = '#,##0';	21 = 'h:mm:ss';	36 = '[$-404]e/m/d';
4 = '#,##0.00';	22 = 'm/d/yy h:mm';	50 = '[$-404]e/m/d';
9 = '0%';	37 = '#,##0 ;(#,##0)';	57 = '[$-404]e/m/d';
10 = '0.00%';	38 = '#,##0 ;[Red](#,##0)';	59 = 't0';
11 = '0.00E+00';	39 = '#,##0.00;(#,##0.00)';	60 = 't0.00';
12 = '# ?/?';	40 = '#,##0.00;[Red](#,##0.00)';	61 = 't#,##0';
13 = '# ??/??';	44 = '_("$"* #,##0.00_);_("$"* \(#,##0.00\);_("$"* "-"??_);_(@_)';	62 = 't#,##0.00';
14 = 'mm-dd-yy';	45 = 'mm:ss';	67 = 't0%';
15 = 'd-mmm-yy';	46 = '[h]:mm:ss';	68 = 't0.00%';
16 = 'd-mmm';	47 = 'mmss.0';	69 = 't# ?/?';
17 = 'mmm-yy';	48 = '##0.0E+0';	70 = 't# ??/??';

OOXML - Very Large Spreadsheet Files?

Grabbing and manipulating the entire XML for a large worksheet (1E6 rows?) may exhaust available workspace memory, so ask yourself if the entire worksheet really has to exist as a single APL array?, and if not...

- **explore further OOXML-SDK methods/props to target smaller parts? (rows, cols, cell ranges? - see C# examples online)**
- **use ⎕NREAD to read/process chunks?**
- **read the entire worksheet xml string, but convert only parts to APL?**

Performance may also be an issue,
...so compare with COM and CSV options, or try sfExcel with DataTable?

OOXML - Automation Issues

Suppose: a list of spreadsheet files to be processed, data to be extracted

- **use OOXML directly?, or UnZip to temp folders on first pass?**
  (one UnZip advantage: facilitates ⎕nread chunks on really large xml strings?)

- **concurrency ("...cannot access the file...because it is being used by another process...") - is still a factor**

- **edge conditions - be prepared for spreadsheet files: lacking internal components; corrupted and unreadable; etc.**

Summary

- *.xlsx is a Zipped collection of xml files; some provide key content
- internal xml files obtained via OOXML-SDK, or UnZip
- parse for cell contents, datatypes, etc.; construct the APL array result

That's Reading... what about Writing - APL to Excel?

...Stay tuned!

Acknowlegements

thanks to Dyalog (esp. JD, Vince), and...

Thank you!