22 October 2007

MSOOXML
Technical considerations of spreadsheet functionalities

MSOOXML is the office file format derived from Microsoft Corporation's Office 2007 program. Work on it began at Ecma International in December 2005 and it was approved as Ecma 376 a year later. It is currently being fast-tracked through JTC1 as DIS 29500. It will be considered at a Ballot Resolution Meeting starting on Monday 25 February 2008 in Geneva. If it is approved, it will become an ISO/IEC standard in 2008 or 2009.

Summary

1. DIS 29500 (OOXML) is an unusual draft standard. ISO/IEC 26300 (Open Document Format) already exists covering exactly the same area, but the dominant vendor, Microsoft Corporation (MSFT), refuses to support it. Most existing documents are written in neither standard, but in earlier proprietary formats of MSFT . Currently most new documents are still being produced in those earlier formats.

2. From the viewpoint of anyone interested in crafting high-quality standards, DIS 29500 is a disaster. It is absurdly long (over 6000 pages), badly designed, badly written and incomplete. It has already been through the standards process at Ecma International, but it is hardly to believe, that most participants in the process have read the whole documentation. A cursory review by National Bodies in the ISO/IEC process unearthed over a thousand obvious minor errors, which would have been hard for any careful reader to miss.

3. But then DIS 29500 does not appear to have been drafted with a view to anyone being able to make sense of it, let alone implement it. It was drafted by MSFT, which has already implemented it. Indeed the cynic would say that MSFT has a strong interest in making it difficult for anyone else to understand and implement it. And, to make it even worse, MSFT has not even given assurances that it will continue to support the draft if it is significantly amended in the ISO/IEC process.

4. The important practical conclusion is that if DIS 29500 is approved, it will not be a real standard. It will be a somewhat unreliable guidebook to help other vendors understand the file format used by MSFT software.

5. After all, why would anyone implement the published standard except to be able to interoperate with documents in MSFT formats? If MSFT does not support it on a particular point or goes beyond it, then no other vendor will be able to stick to the published standard, it will have to do its best to track the MSFT software. Conflicting standards do not help. The attempt of MSFT to get OOXML approved by ISO/IEC can only be interpreted as attempt to minimize and harm the value of the already existing, official standard for electronic document format, ODF while pushing a second MS-controlled standard forward.

6. MSFT is scared that governments will mandate the ISO/IEC standard and refuse to accept documents which do not comply with it. In other words, there is a point to the ISO/IEC exercise: to help protect MSFT's monopoly position. But it is hard to see how this benefits anyone else - except for MSFT associates and partners who believe that what is good for MSFT is good for them.

7. Turning to more technical matters, it is worth noting that a major benefit of xml-based file formats compared with the older binary formats, is that it is supposed to be easier to view and edit content within the document, even for applications which fall far short of being complete "office suites".The OOXML format contradicts this benefit by appearing to enforce the use of MSFT office products on all users, irrespective of their real needs..."

8. The attached analysis by Stephane Rodriguez, a leading file format expert, uses a variety of examples to explore what is involved in practice in working with the new MSFT Office 2007

formats (which supposedly follow DIS 29500/Ecma 376). It is written from the viewpoint of a developer, but it is surprisingly easy to read. Anyone with any technical background is urged to read it. His conclusion is that this is not a set of file formats fit for the twenty-first century.

OOXML is defective by design

Tuesday August 28, 2007

Microsoft is trying to push new file formats that are using ZIP and XML. Are those new file formats any good for Office developers ? In other words, should anyone feel safe to make direct access to file parts, and start getting free of running instances of Microsoft Office and its COM object model, usually through VBA ?

Microsoft does not run out of teasing. There is ton of videos, see
http://channel9.msdn.com/ShowPost.aspx?PostID=73329, and
http://channel9.msdn.com/Showpost.aspx?postid=313246 for example,
screencasts, articles and blog posts (self-serving Microsoft blog posts mostly) about how much they are opening up. It boils down to the following, taken from the Microsoft Office 12 introduction white paper at http://blogs.msdn.com/brian_jones/archive/2005/06/01/424085.aspx

> ... The use of XML offers the benefits of **greater transparency and openness** than were possible with the previous binary file formats. The new formats allow Office documents to easily integrate with existing and future line-of-business systems, as the contents are now open and accessible. The new formats are also designed with long-term robustness and accessibility in mind.
>
> ... The binary file formats in use currently were designed in 1994—before the advent of XML and before widespread exchange of documents and data that is common today. These file formats, .doc, .xls, and .ppt, were introduced with the release of Microsoft Office 97, at a time when it was important to optimize the files for storage on slow hard drives and “floppy” disks; it was not as crucial to focus on easy access to data within the files for better content reuse, document generation, and seamless integration of the documents into business processes.
>
> ... The new XML-based file formats in these programs enable broader integration and interoperability between Office documents and enterprise applications. Additionally, “Office 12” files are all wrapped using ZIP technologies, which allows for easy access to the content parts as well as standard compression, reducing file sizes and improving reliability and data recovery. ... Because documents stored in the Open XML Formats are machine-readable and editable by any text editor or XML processor, **solutions need not use Microsoft Office programs to view or edit content within the documents.** Enterprise business solutions can access document contents easily and efficiently. Technology providers can utilize the Microsoft Office System and Office authoring applications within their solutions, reuse Microsoft Office documents as other Office documents, or open and **act on Office documents on other platforms and in other applications**.

They insist on the fact that, provided you make a valid use of the XML, pretty much changing the content of anything in an existing document can be achieved by sequentially 1) unzipping the content 2) making appropriate changes to one or more XML parts that are compatible with the

provided XML schemas and open packaging relationships 3) zipping the content back.

Let's see if that's true.

1) Self-exploding spreadsheets
2) Entered versus stored values
3) Optimization artefacts become a feature instead of an embarrasment
4) VML isn't XML
5) Open packaging parts minefield
6) International, but US English first and foremost
7) Many ways to get in trouble
8) Windows dates
9) All roads lead to Office 2007
10) A world of ZIP+OLE files
11) Document security is a (bad) joke
12) BIFF is gone ... not!
13) Document backwards compatibility subject to neutrino radio-activity
14) ECMA 376 documents just do not exist
15) How the ISO/IEC 26300 OpenDocument format (ODF) compares?

1) Self-exploding spreadsheets

To reproduce the scenario :

- start Excel 2007 and create a new spreadsheet
- insert a value 10 in a cell
- insert a value 20 next to it
- select the two cells and click on the "sum" button to create a sum of the two cells
- save the spreadsheet (xlsx file)
- close it and unzip it

The relevant XML in the corresponding part xl/worksheets/sheet1.xml is :

```
<row r="2" spans="3:5">
  <c r="C2">
   <v>10</v>
 </c>
 <c r="D2">
   <v>20</v>
 </c>
 <c r="E2">
   <f>SUM(C2:D2)</f>
   <v>30</v>
 </c>
</row>
```

Pretty simple XML. Now say we want to edit cell E2 and set a constant value of 40 in place of a formula. But instead of doing that with Excel 2007 interactively, we are going to do it manually :

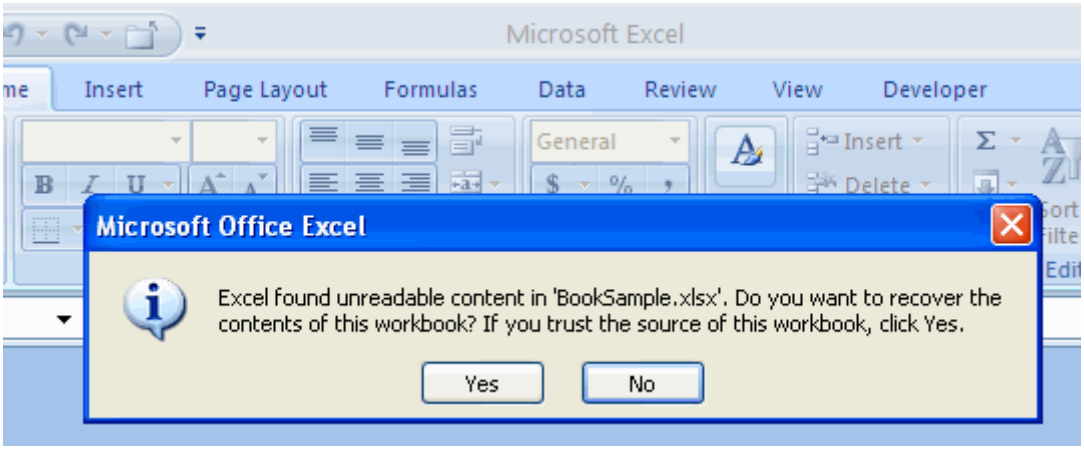
- unzip the file
- grab a zip part known as xl/worksheets/sheet1.xml

- make the edit described below
- put the updated zip part back in the zip package

The corresponding valid (and carefully changed) XML for setting the constant value of 40 in cell E2 is:

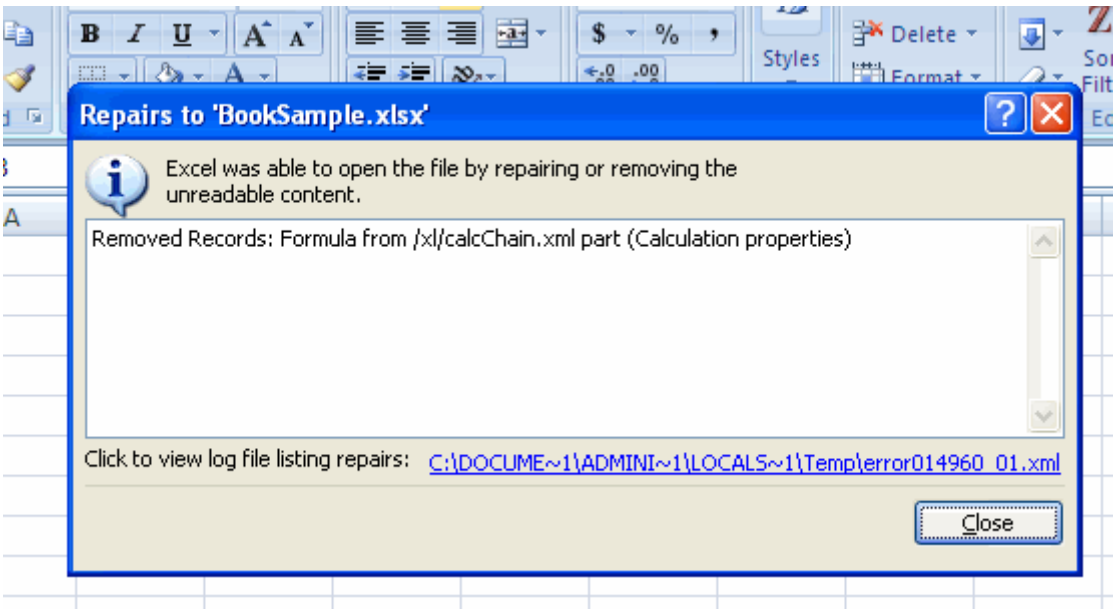
```
<row r="2" spans="3:5">
 <c r="C2">
    <v>10</v>
  </c>
  <c r="D2">
    <v>20</v>
  </c>
  <c r="E2">
    <v>40</v>
  </c>
</row>
```

Now open the file in Excel 2007. You get a blocking error message which says :



Excel 2007 cannot open the file we have manually updated

Followed by another even more frightening message:



According to Excel 2007, the problem is that the calculation chain is now corrupt

Interestingly enough, we thought parts of a spreadsheet file were updatable as long as we did not touch elements that are, according to the ECMA 376 documentation (the official paper from Microsoft), indexes to other parts.

Now that's an interesting issue. The ECMA 376 documentation says that the calculation chain is the graph of formulas, sorted wrt to their dependencies. Suddenly, the little change we would like to make looks way more expensive. Rebuilding the graph of formulas ourselves is what Excel itself does, and sure enough it involves parsing the entire spreadsheet, discovering formulas, and applying formula parsing algorithms to induce a graph of the dependencies. It certainly sounds like we are going to have to rewrite a portion of Excel itself. Not every employer can afford waiting that long ... Or perhaps that's Microsoft's way to tell us : *you shall not touch this without our approbation.*

We can perhaps get rid of the calculation chain, if we carefully delete the part and associated relationship. But then, there are three problems:

- Parts are intertwined together through implicit and explicit relationships (relationships are separate zip entries). It gets even more grainy, and the risk of corruption increases. Again, we have to further take into account a number of details that are out of our scope, like how the calculation chain is defined as per the workbook part.

- If I were a programmer, the issue would be: Microsoft gives no library that I can use, at least not a library that I could use no matter the execution environment. Microsoft provides an API which works in a recent .NET run-time, and installs on Windows XP SP2 and Windows Vista. There goes the platform independence.

- If I delete the calculation chain, I am admitting that the resulting spreadsheet is being degraded, something that Excel 2007 does not suffer from since it takes care of that thanks to its infrastructure. In other words, making changes outside of Excel 2007 doesn't look as first-class citizen and safe and robust as it sounded first, and in either case either the application takes care of a lot of details, replicating what Excel itself does (there is no known non-Microsoft Excel 2007 implementation available out there), or Excel 2007 will

have to do that for me next time the spreadsheet is opened. If the changes are made on a server without Excel 2007 installed and the resulting spreadsheet is distributed to employees throughout the organization, then every single employee will have to get through a full spreadsheet recalculation (arbitrarily lengthy) next time they open the spreadsheet. The application is a second-class citizen compared to Excel 2007, that's what everybody thinks: me and the employees.

What this shows pretty clearly is that either we lack the tools, or Microsoft does not think we should be doing that in the first place. With that being said, if making a simple change to a cell is too much to ask, then what is this new format good for? The prospect of getting our recipients facing those dreaded message boxes is not exactly a change compared to the well known ugly stories with corrupted binary file formats.

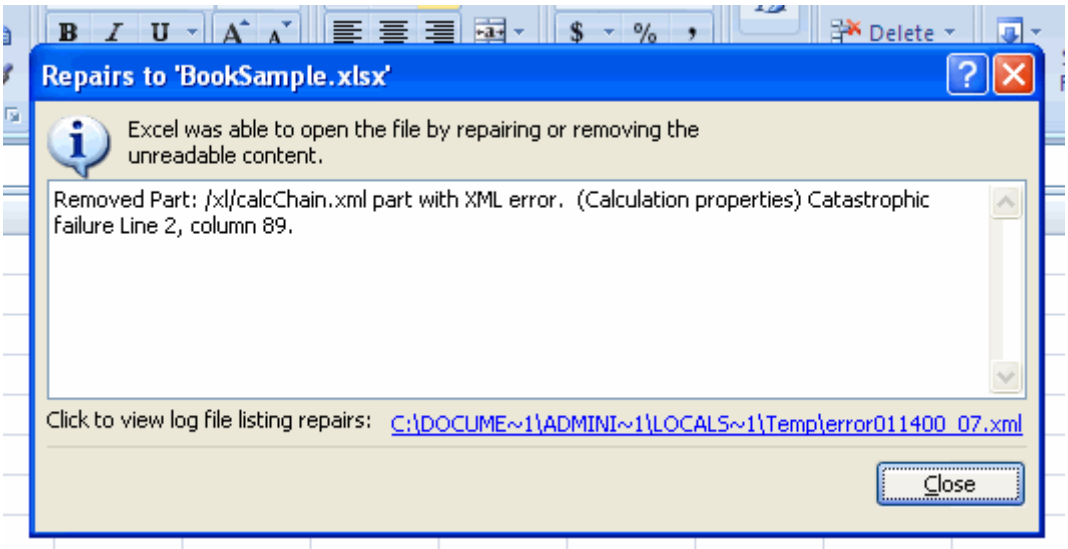
Let's play the devil's advocate now and see how far it will take us. Here is the contents of the calculation chain, xl/calcChain.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<calcChain  xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" >
  <c r="E2" i="1"/>
</calcChain>
```

The reference to formula in cell E2 is what seems to be causing the problem. All right, then since it's just XML, let's remove that reference. Edit the calculation chain xl/calcChain.xml so it looks like this:

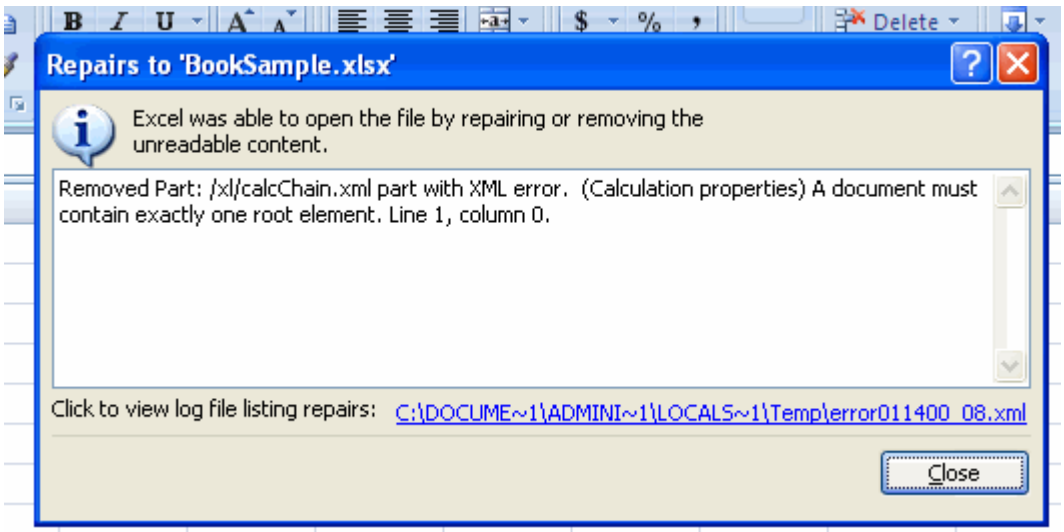
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<calcChain xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" >
</calcChain>
```

After the modification, if we open the file in Excel 2007, it still complains :



Excel 2007 cannot open the file even if the calculation chain is cleaned up manually

I particularly like the wording : *catastrophic failure*. Now it looks like a lot of blood is spilling. Well, then let's stop the hemorrhage, and remove the contents of this part altogether. Make the update, and open the file in Excel 2007, it continues to complain :



Excel 2007 cannot open the file even if the calculation chain is made empty

I guess it's time to take a look at the ECMA 376 documentation. In Part 4, page 2087, it says (emphasis mine) :

```
3.6.2 calcChain (Calculation Chain Info)
This element represents the root of the calculation chain.

<complexType name="CT_CalcChain">
  <sequence>
    <element name="c" type="CT_CalcCell" minOccurs="1" maxOccurs="unbounded"/>
    <element name="extLst" minOccurs="0" type="CT_ExtensionList"/>
  </sequence>
</complexType>
```

I guess there we have it, we can't have a calculation chain part with no cell reference in it. Excel guts spilling through the specs here, aren't they? Wait a minute, is this supposed to make it into an international standard?

Let's do a quick summary: we had Excel 2007 complain that the calculation chain was left with a reference to a formula that did not exist any more, but in fact Excel 2007 complains even with this manually fixed. The solution is to delete the physical calculation chain part and the relationship it has with the workbook (defined in the workbook relationship part xl/_rels/workbook.rels), and to update some other parts as well.

Let's get a visual diff of what it takes to make a "proper" change in a cell :

BookSample.xlsx (original)	**DIFF**	**BookSample.xlsx (updated)**
[Content_Types].xml	1	[Content_Types].xml
_rels/.rels		_rels/.rels
docProps/app.xml		docProps/app.xml
docProps/core.xml		docProps/core.xml
xl/workbook.xml		xl/workbook.xml
xl/_rels/workbook.xml.rels	1	xl/_rels/workbook.xml.rels
xl/styles.xml		xl/styles.xml

xl/theme/theme1.xml		xl/theme/theme1.xml
xl/worksheets/sheet1.xml	2	xl/worksheets/sheet1.xml
xl.calcChain.xml	4	

Original versus updated file : 40% of the parts need a change!

- all we wanted to do was to make a change in a cell and we end up butchering the package.
- remember, we don't want to have to deal with formulas or anything even further far away from our concern, we have no notion of formulas nor should have, the idea is to change a value in a cell. If the file format was properly designed, all it would take is an in-place replacement of the cell contents.
- with VBA, to make a change in a cell, we do : `Range("E2").Value = 40`. Period. End of story.
- 40% of parts have to change in order to take into account our minor change. That certainly looks like having to care of the format's own dirty laundry, this is not acceptable by any stretch of the imagination for a modern format reportedly (Microsoft source) aimed to improve interoperability across platforms and applications. This is unfortunately exactly the situation in which we are with binary formats, something the new formats were supposed to fix!
- deleting a physical part is not a minor operation. Most ZIP libraries can't delete a physical part without uncompressing everything in a temp folder and rebuilding the package (thereby simulating a delete function). If your spreadsheet is big, this will take some time and space.
- a full recalculation in the case there were formulas elsewhere.
- can you imagine making more complex changes like replacing a pivot table with another? Pivot tables have many (like hundreds) ties with everything in the spreadsheet.

2) Entered versus stored values

We all take for granted that when we type a value such as 1234.1234 in a cell of a spreadsheet, that's what actually gets stored. Excel has this auto-number format matching capability where it tries to make sense of what is manually entered in order to deduce if that's a string, a number, a boolean or a date and applies a number format accordingly, but what's being stored as a value is what is entered. By the way, if you hit Alt+F11 in Excel, and enter something like `Range("C3").Value` and run the macro, you'll get the entered value in cell C3; if you enter something like `Range("C3").Text`, you'll get the formatted value in cell C3, where the number format has been applied to the value. Note that the return value takes advantage of the locale and number formatting which means you may get `1234,1234` (note the comma) instead of `1234.1234`.

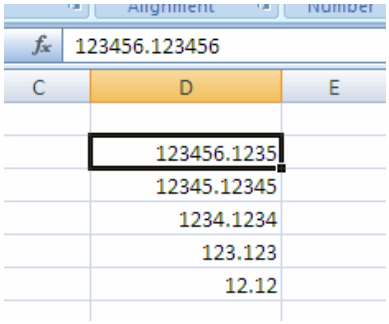
Is this storage neutrality true with the new formats?

To reproduce the scenario:

- start Excel 2007 and create a new spreadsheet
- type value 123456.123456
- resize the column manually so that the value entirely shows

- hit return and type 12345.12345
- hit return and type 1234.1234
- hit return and type 123.123
- hit return and type 12.12
- save the spreadsheet (xlsx file)
- close it, unzip it

Here is a screenshot of what you should see at this point :



Typing a few numeric values in Excel 2007

The corresponding XML in the main part xl/worksheets/sheet1.xml is :

```
<sheetData>
  <row r="2" spans="4:4">
    <c r="D2">
      <v>123456.123456</v>
    </c>
  </row>
  <row r="3" spans="4:4">
    <c r="D3">
      <v>12345.123449999999</v>
    </c>
  </row>
  <row r="4" spans="4:4">
    <c r="D4">
      <v>1234.1233999999999</v>
    </c>
  </row>
  <row r="5" spans="4:4">
    <c r="D5">
      <v>123.123</v>
    </c>
  </row>
  <row r="6" spans="4:4">
    <c r="D6">
      <v>12.12</v>
    </c>
  </row>
</sheetData>
```

The problem is that Excel 2007 does not store what we entered. If we read the XML, we are going to grab numbers that have rounding errors compared to the actual numbers we typed.

It is absolutely lost on me how implementers are expected to deal with this mess. The spreadsheet does not reflect the proper values, and you can easily see where it goes. Imagine non-Microsoft applications used in healthcare and critical systems relying on the spreadsheet data. Not only the

rounding error seems arbitrary (one would have to go back and study the artefacts of IEEE floating-point values, several decades of work), but it changes. There is no way we can possibly take advantage of this, with one notable exception: if we are able to be in an execution environment for which reading those floating-point values does not produce those artefacts, and returns the proper entered values, then we are good.

Problem: Microsoft does not document the execution environment. We can fairly assume its Windows, but what else? And if I am using Linux, how do I work with this?

It's important to understand that if we open the spreadsheet in Excel 2007, we see the proper values. No loss (based on the values entered) seem to have occurred, the problem is that the data in XML just cannot be used as is.

As an aside, the stored value does not use the locale (it always uses the dot as decimal separator), therefore we have to assume this is all US English. If we wrote software in Excel VBA that grabs the value in cells, then processes it, there is no way we could migrate our VBA code to work with this XML part without substantial rework. We are left with Excel's own international implementation artefacts, undocumented.

This is almost certainly related to the extraordinary 65535 bug in Excel 2007 whereby Excel displays six floating point numbers close to 65535 and another six close to 65536 as 100,000! Even worse, in some cases it actually treats the numbers as equal to 100,000.

3) Optimization artefacts become a feature instead of an embarrassment

Historically, the BIFF file format used in Excel spreadsheets was designed to be small and fast. But this design decision goes all way back to early 90s, when the Pentium CPU did not exist yet. Regular desktop computers we use everyday are at least several orders of magnitude faster and memory-friendly that those early computers were. Yet, Microsoft chose to keep those optimization artefacts as is, with the side effect that they are now exposed to the surface as part of the XML.

Among interesting optimizations is Excel insistence in trying to factor formulas as much as possible. This happens with Excel when you create a formula, then drag the cell to replicate it in other cells. That's shared formulas. Excel chooses to declare the formula itself only once, and then creates a mechanism to infer the formula in other cells (the relative position counts as an offset).

Shared formulas are supposed to be transparent for developers. Is it true?

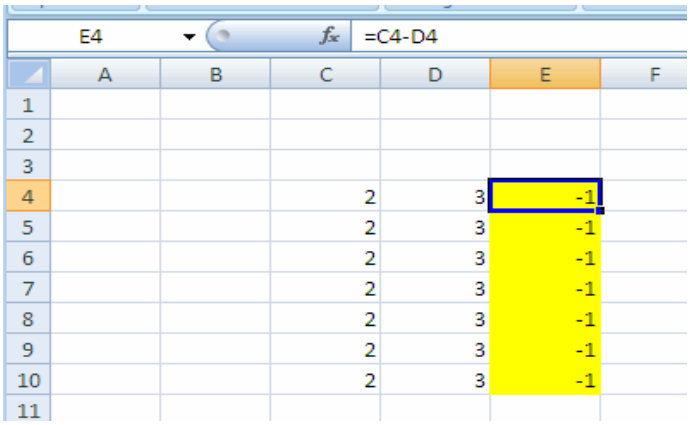
To reproduce the scenario :

- start Excel 2007 and create a new spreadsheet
- enter value 2 in cell C4
- click on cell C4, click on the bottom-right corner, and drag cell C4 down to C10 to replicate the content
- enter value 3 in cell D4
- click on cell D4, click on the bottom-right corner, and drag cell D4 down to D10 to replicate the content
- enter formula =C4-D4 in cell E4
- click on cell E4, click on the bottom-right corner, and drag cell E4 down to E10 to replicate

the formula

- save the spreadsheet (xlsx file)
- close it, unzip it

Here is a screenshot of what you should see at this point :



Typing a few numbers and formulas in Excel 2007

The corresponding XML in the main part xl/worksheets/sheet1.xml is :

```
<sheetData>
  <row r="4" spans="3:5">
    <c r="C4">
      <v>2</v>
    </c>
    <c r="D4">
      <v>3</v>
    </c>
    <c r="E4" s="1">
      <f>C4-D4</f>
      <v>-1</v>
    </c>
  </row>
  <row r="5" spans="3:5">
    <c r="C5">
      <v>2</v>
    </c>
    <c r="D5">
      <v>3</v>
    </c>
    <c r="E5" s="1">
      <f t="shared" ref="E5:E10" si="0">C5-D5</f>
      <v>-1</v>
    </c>
  </row>
  <row r="6" spans="3:5">
    <c r="C6">
      <v>2</v>
    </c>
    <c r="D6">
      <v>3</v>
    </c>
    <c r="E6" s="1">
      <f t="shared" si="0"/>
```

```
      <v>-1</v>
    </c>
  </row>
  <row r="7" spans="3:5">
    <c r="C7">
      <v>2</v>
    </c>
    <c r="D7">
      <v>3</v>
    </c>
    <c r="E7" s="1">
      <f t="shared" si="0"/>
      <v>-1</v>
    </c>
  </row>
  <row r="8" spans="3:5">
    <c r="C8">
      <v>2</v>
    </c>
    <c r="D8">
      <v>3</v>
    </c>
    <c r="E8" s="1">
      <f t="shared" si="0"/>
      <v>-1</v>
    </c>
  </row>
  <row r="9" spans="3:5">
    <c r="C9">
      <v>2</v>
    </c>
    <c r="D9">
      <v>3</v>
    </c>
    <c r="E9" s="1">
      <f t="shared" si="0"/>
      <v>-1</v>
    </c>
  </row>
  <row r="10" spans="3:5">
    <c r="C10">
      <v>2</v>
    </c>
    <c r="D10">
      <v>3</v>
    </c>
    <c r="E10" s="1">
      <f t="shared" si="0"/>
      <v>-1</v>
    </c>
  </row>
</sheetData>
```

In cell E5, we see a `ref` attribute where the shared formula range applies, a `si` attribute which identifies the shared formula range (itself redundant with the `ref` attribute), and the actual formula for that cell. But in cell E6, the cell below E5 in the grid, we see a definition with just a `si` attribute. In other words, cell E6 is linked to cell E5.

Here is the problem, let's say we make a manual change to cell E5 and remove the formula. We've seen in the first section of this article that the calculation chain is left unsynched, but an additional

problem is that cell E6 is also left unsynched because its `si` attribute now points to nowhere. Note that the situation isn't any better if we merely update the formula, by doing so cell E5 is fine, but linked cells will reference the new formula, not the old one. So a simple change in cell E5 actually spreads unintentionally. It goes without saying that it's very expensive to make a simple change.

That is a direct result of the optimization artefact. Someone willing to make a change cannot proceed without taking care of depending cells if the cell defines a shared formula range. The problem gets only bigger since we have to either remove the shared formula altogether in all cells, or translate the shared formula definition accordingly, which implies parsing the formula (the goal was only make a change in a cell!) and making a number of offset changes, many of which are left for the user to discover (formula tokens are complex). To remove the shared formula, the actual formula definitions in linked cells have to be built, and that's where an algorithm has to find a way to create these, essentially rolling back Excel's optimization as a preliminary step.

We have a case where an optimization artefact becoming an embarrassment, not a feature.

Let's see how the situation compares to the old Excel binary file format (BIFF internal format is stored inside an OLE container). Here are the corresponding BIFF records :

```
// BIFF : a shared formula, and cells linked to the shared formula [SHRFMLA
0015] 03 00 08 00 03 03 00 06 0B 00 4C 00 00 FE C0 4C 00 00 FF C0 04
[FORMULA 001B] 04 00 03 00 3E 00 00 00 00 00 00 00 00 40 08 00 05 00 03 FE 05 00
01 03 00 03 00
[FORMULA 001B] 05 00 03 00 3E 00 00 00 00 00 00 00 00 00 08 00 06 00 03 FF 05 00
01 03 00 03 00
[FORMULA 001B] 06 00 03 00 3E 00 00 00 00 00 00 00 F0 BF 08 00 07 00 03 FF 05 00
01 03 00 03 00
[FORMULA 001B] 07 00 03 00 0F 00 00 00 00 00 00 00 F0 3F 08 00 08 00 03 FF 05 00
01 03 00 03 00
[FORMULA 001B] 08 00 03 00 0F 00 00 00 00 00 00 00 00 C0 08 00 03 00 03 FF 05 00
01 03 00 03 00

// BIFF : same than above, made understandable
[SHRFMLA 0015] (ref = 03 00 08 00 03 03) (formula = [currentrow][currentcolumn-
2]-[currentrow][currentcolumn-1]) [FORMULA 001B] (cell row = 5 col = E) (formula
is a link to the shared formula)
[FORMULA 001B] (cell row = 6 col = E) (formula is a link to the shared formula)
[FORMULA 001B] (cell row = 7 col = E) (formula is a link to the shared formula)
[FORMULA 001B] (cell row = 8 col = E) (formula is a link to the shared formula)
[FORMULA 001B] (cell row = 9 col = E) (formula is a link to the shared formula)
```

With the Excel binary format, the design is right. The shared formula is defined outside a cell, so you can very simply remove the formula in cell E5 without breaking other cells. From a programming perspective, **the new XML format qualifies as a regression compared to the binary file format.**

4) VML isn't VML

Contrary to what the ECMA 376 documentation says in many places, VML drawing parts are not deprecated at all. VML is in fact very pervasive in Word, Excel and Powerpoint documents, so it's even more a blatant problem.

In the ECMA 376 documentation, part 4, page 4343, we learn : (emphasis mine)

> [Note: The VML format is a legacy format originally introduced with Office 2000 and is included and fully **defined in this Standard for backwards compatibility reasons**. The DrawingML format is a newer and richer format created with the goal of eventually replacing any uses of VML in the Office Open XML formats. VML should be considered a deprecated format included in Office Open XML for legacy reasons only and new applications that need a file format for drawings are strongly encouraged to use preferentially DrawingML .end note]

Here is a way to create a VML part in a **new** document :

- start Excel 2007 and create a new spreadsheet
- right-click and choose Insert Comment
- enter a comment
- save the spreadsheet (xlsx file)
- close it, unzip it

The corresponding XML in the drawing part xl/drawings/vmlDrawing1.vml is :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xml xmlns:v="urn:schemas-microsoft-com:vml"
     xmlns:o="urn:schemas-microsoft-com:office:office"
     xmlns:x="urn:schemas-microsoft-com:office:excel">

  <o:shapelayout v:ext="edit">
    <o:idmap v:ext="edit" data="1"/>
  </o:shapelayout>

  <v:shapetype id="_x0000_t202" coordsize="21600,21600" o:spt="202"
path="m,l,21600r21600,l21600,xe">
    <v:stroke joinstyle="miter"/>
    <v:path gradientshapeok="t" o:connecttype="rect"/>
  </v:shapetype>

  <v:shape id="_x0000_s1025" type="#_x0000_t202" style="position:absolute;
margin-left:203.25pt;margintop: 37.5pt;width:96pt;height:55.5pt;z-index:1;
visibility:hidden" fillcolor="#ffffe1" o:insetmode="auto">
    <v:fill color2="#ffffe1"/> <v:shadow on="t" color="black" obscured="t"/>
    <v:path o:connecttype="none"/>
    <v:textbox style="mso-direction-alt:auto">
      <div style="text-align:left"/>
    </v:textbox>
    <x:ClientData ObjectType="Note">
      <x:MoveWithCells/>
     <x:SizeWithCells/>
      <x:Anchor> 4, 15, 2, 10, 6, 15, 6, 4</x:Anchor>
     <x:AutoFill>False</x:AutoFill>
      <x:Row>3</x:Row>
     <x:Column>3</x:Column>
    </x:ClientData>
  </v:shape>
</xml>
```

From a pure markup perspective, it is XML, but there are application encoded values such as `="m,l,21600r21600,l21600,xe"` and `4, 15, 2, 10, 6, 15, 6, 4` which contradict proper

XML design, ie in a way that it is both poor XML, and cannot be used by XSLT transforms. VML allows far more complicated values in the general case, including conditionals expressed in their own language.

```
<v:shape style='top: 0; left: 0; width: 250; height: 250' stroke="true"
strokecolor="red" strokeweight="2" fill="true" fillcolor="green" coordorigin="0
0" coordsize="175 175">
<v:path v="m 8,65 l
72,65,92,11,112,65,174,65,122,100,142,155,92,121,42,155,60,100 x e"/>
 <formulas>
    <f eqn="sum #0 0 10800"/>
    <f eqn="prod #0 2 1"/>
    <f eqn="sum 21600 0 @1"/>
   <f eqn="sum 0 0 @2"/>
    <f eqn="sum 21600 0 @3"/>
    <f eqn="if @0 @3 0"/>
    <f eqn="if @0 21600 @1"/>
    <f eqn="if @0 0 @2"/>
    <f eqn="if @0 @4 21600"/>
    <f eqn="mid @5 @6"/>
    <f eqn="mid @8 @5"/>
   <f eqn="mid @7 @8"/>
    <f eqn="mid @6 @7"/>
    <f eqn="sum @6 0 @5"/>
  </formulas>
</v:shape>
```

If that is XML, then I propose writing C code the following way :

```
<v:shape style='top: 0; left: 0; width: 250; height: 250' stroke="true"
strokecolor="red" strokeweight="2" fill="true" fillcolor="green" coordorigin="0
0" coordsize="175 175">
<v:path v="m 8,65 l
72,65,92,11,112,65,174,65,122,100,142,155,92,121,42,155,60,100 x e"/>
  <formulas>
   <f eqn=" int main(int argc, char** argv) { printf("Hello World\n"); return 0;
} "/>
  </formulas>
</v:shape>
```

This is XML, right? There are angle brackets, it conforms to the XML W3C recommendation, therefore it's XML.

VML also contains application-specific markup, with no documentation associated to it, for instance in the example above, 4, 15, 2, 10, 6, 15, 6, 4. Because only Microsoft used this markup, there was no need to define it in its own namespace and so on. But now that VML is part of ECMA 376, either ISO accepts a vendor-specific markup, which defies the point of ISO standards in the first place, or it just contradicts ISO standards.

The implication for an implementer, or for someone willing to make a change is that there is no way someone can possibly edit this thing without a proper implementation of the VML library itself. The risk of corruption is extremely high. Obviously, Microsoft expects that VML parts are replaced by other VML parts as a whole, without a finer granularity. The problem is that VML too can contain references to objects and other parts, so that contradicts even a simple template scenario.

VML is an old, undocumented, library that speaks volumes of past Microsoft lock-in strategy. Mr Bill Gates in person sent in 1998 a memo to the Office product group (led by Steven Sinofsky at the time), memo disclosed to the public thanks to the IOWA consumer case :

From: Bill Gates
Sent: Saturday, December 05, 1998 9:44 AM
To: Bob Muglia (Exchange); Jon DeVaan; Steven Sinofsky
Cc: Paul Maritz
Subject: Office rendering

One thing we have got to change in our strategy - allowing Office documents to be rendered very well by other peoples browsers is one of the most destructive things we could do to the company.

We have to stop putting any effort into this and make sure that Office documents very well depends on PROPRIETARY IE capabilities.

Anything else is suicide for our platform. This is a case where Office has to avoid doing something to destory Windows.

I would be glad to explain at greater length.

Likewise this love of DAV in Office/Exchange is a huge problem. I would also like to make sure people understand this as well.

The undocumented VML library shipped in Internet Explorer 5 in 2000, and has been part of Internet Explorer ever since. The DAV protocol (Distributed Authoring and Versioning) is an international cross-platform standard, open to everybody. God only knows why Bill Gates likes so much VML, and dislikes so much DAV ...

For the record, the ECMA 376 documentation describes the VML markup, but it does not specify it. Much of application-defined behaviors are left for one to guess.

5) Open packaging parts minefield

The underlying architecture of how zip entries relate together is called by Microsoft “open packaging conventions”. What it means is that zip entries are not independent, or even related by way of a single master zip entry which would work as a directory of all zip entries of relevance. There is a logical tree of entries which uses separate zip entries to define relations between zip entries. The logical tree has nothing to do with the physical tree of zip entries in a package, despite Microsoft continuously using screenshots of Windows XP’s built-in ZIP folders to mimic a folder hierarchy.

The problem with such an architecture is that a part may or may not relate to another and there is no standard way to know. Often, there is a `r:id attribute` right in the content of some XML part that tells the application that there is a relation, but this is not standard. By the way, Microsoft’s PDF fixed format competitor called XPS is also based on the same underlying architecture, except that the team who developed XPS did not quite want to play by the same rules than the Office team. For instance the XPS main zip document entry related to one or more XPS pages with an attribute such as: `Source="Pages/1.fpage"`. In other words, they are not using the `r:id` attribute, instead relying on their own mechanism. This makes it impossible for a generic library to know which part relates to which part, and it has an unfortunate consequence.

The unfortunate consequence is being unable to know whether a part relates or not to another part makes it impossible to know, when you delete a part, if you are going to corrupt the document or not. The document becomes corrupt if it points or relates (implicitly or explicitly) to a missing part. It's unclear why Microsoft chose this way of doing things, obviously leading to an internal chaos, instead of just copying the research from the OpenOffice project, where a central directory is used (OpenOffice ZIP initiative predates Microsoft's by at least three years, despite Microsoft stealing the thunder).

When you don't know the dependencies of a part, the consequence is obvious, you leave those parts alone. If you do this enough times, it clutters up the package, and soon enough you end up with a package containing any number of parts - God only knows why they are there. Add to this you can add a part of any content type (arbitrary MIME type), and you have a recipe for disaster. Among other things, virus could proliferate.

Microsoft's deletePart() function which is available in their System.IO.Packaging library (itself part of .NET), does not solve this problem. We have a case of poor engineering, creating unnecessary problems for others to worry about.

6) International, but US English first and foremost

An important ongoing tension with Office documents is the support for locales. Microsoft historically used a number of mechanisms to address this need, but they kept evolving and Microsoft aggregated all mechanisms to keep compatibility with older versions. What was hidden is being surfaced with the new XML. Anything that gets displayed, calculated, rendered or stored depends one way or another on an complex and undocumented combination of locale settings including: the Office application language, the Office application language settings (per application), the Office document language settings (per document), the system locale of the operating system.

To save them time, Microsoft chose to store XML using the US English locale regardless of all settings above.

This has an unfortunate consequence for implementers or those willing to make a manual change. Indeed, Microsoft is imposing everybody else to adapt to US English locale options (separators, date formats, formula conventions, ...) despite the fact that when using Office interactively, this fact is hidden to the user. The Office application infrastructure manages to abstract it away from users, which is a good thing. **Office developers using VBA all over the world are used to work with localized functions, the complexity is hidden to them**. But since the XML resurfaces this US English locale, all the complexity is left for one to implement. We are talking two decades' worth of internationalization issues, for Office-related locale issues and Windows-related locale issues. To get an idea of how bad the situation is, suffice to say that a Microsoft employee part of the internationalization team in Windows has a blog where he posts daily horror stories.

Also, for Excel formulas, it means the formula names are US English formula names, which you'll never see in Excel if you are using a locale version such as French or Brazilian. It's left for one to guess how to map function names one way to another, and of course the ECMA 376 documentation does not provide those localized formula names. If you intend not to implement a mapping to a locale, ideally your customer's locale, it implies you are willing to work with US English function names (plus US English separators, ...). **If your company has invested in libraries or developed**

libraries in-house, they cannot be used any more.

Can it get any worse than that?

Unfortunately yes.

Despite Microsoft insistence to store everything using the US English locale, they still manage to store a number of contradicting country/encoding flags in the XML. Examples of that are DrawingML and VML languages. They store encoding tags for storing text chunks, but text chunks in document itself does not use any such encoding tag. It is in fact entirely possible that DrawingML and VML are implementations which involve nothing localized itself but which store localized tags in the document, while the rest of languages (WordML, SpreadsheetML, ...) are implemented otherwise : their implementation is chockful of encoding settings, but they need not store anything in the document itself. In other words, everything gets localized at run-time with WordML, SpreadsheetML, ... except DrawingML and VML.

It's clear at this point that the legacy shows ... One would have expected Microsoft to fix this once for all, provide a consistent framework. They chose not do so, and as a result, it's left for any implementer or someone willing to make a change to do the heavy lifting. What we are talking about here is entire internationalization implementation stacks which can represent years of work and stabilization. Ironically enough, you will not only have to implement this stuff (reverse engineering since it is not addressed by the ECMA 376 documentation), you will have to implement in a way that reproduces current Office flaws. No matter how correct your implementation is, you have to retrofit it to work just like Office does.

To get a flavor of non-US English within US-English (thereby violating ECMA 376's own rules), all you have to do is to insert a chart :

- start Excel 2007 and create a new spreadsheet
- enter 10, hit tab, 20, hit tab, 30
- select those 3 cells and insert a chart
- insert a chart title
- edit the chart title and give it whatever name you fancy
- save the spreadsheet (xlsx file)
- close it, unzip it

Here is an excerpt of the chart part xl/charts/chart1.xml :

```
<c:chartSpace
  xmlns:c="http://schemas.openxmlformats.org/drawingml/2006/chart"
  xmlns:a="http://schemas.openxmlformats.org/drawingm l/2006/main"
 xmlns:r="http://schemas.openxmlformats.org/officeDo cument/2006/
    relationships">
  <c:lang val="fr-FR"/>
  <c:chart>
    <c:title>
      <c:tx>
        <c:rich>
          <a:bodyPr/>
          <a:lstStyle/>
          <a:p>
            <a:pPr>
              <a:defRPr/>
```

```
            </a:pPr>
           <a:r>
              <a:rPr lang="fr-FR"/>
              <a:t>Some title</a:t>
            </a:r>
...
```

A reader of the article also mentions that within a strict US-English stream, you can get localized text formatting. Here is how :

- start Excel 2007 and create a new spreadsheet
- click on the Insert ribbon tab and click on Header and Footer
- type some text in the edit area
- select some of this text, click on the Home ribbon tab, and put this selection on bold and italic
- save the spreadsheet (xlsx file)
- close it, unzip it

I am using a French version of Excel 2007. Here is an excerpt of the chart part xl/worksheets/sheet1.xml :

```
<headerFooter>
  <oddHeader>&Ctrt&"-,Gras Italique"uiy tuieyrtui</oddHeader>
</headerFooter
```

Gras and **Italique** are French for **Bold** and **Italic**. Just because I am using a French version of Excel 2007, the format produced inserts French localized fragments, therefore anyone willing to read Excel 2007 files in the most general case must be ready to parse non-US English. The ECMA 376 documentation says, in section 3.3.1.36, Part 4, page 1965 (emphasis mine) :

> & "font name,font type" - code for "text font name" and "text font type", where font name and font type are strings specifying the name and type of the font, separated by a comma. When a hyphen appears in font name, it means "none specified". Both of font name and font type **can be localized values**.

What is **can be** supposed to mean? Who reviewed this documentation? And where are the localized values to expect? Reading the documentation I have the feeling that :

- **can be** is short for *it's complicated, we couldn't get this written on paper in a small space*
- the localized names are not provided, so even if I try to do the heavy lifting myself despite the fact that SpreadsheetML violates its own rules (it's supposed to be encoded using US English), where can I get the material to write this library?

7) Many ways to get in trouble

The extensiveness of the ECMA 376 documentation, over 6000 pages, is telling how much legacy Microsoft is willing to bring into the future. Taking an example of such legacy clarifies what it takes to implement even a portion of the documentation. The example is *text formatting*. Any of the 3 applications, Word, Excel and Powerpoint uses its own text formatting markup. Worse, the shared

libraries themselves (VML, DrawingML, MathML, ...) also use separate text formattings, each different. Even worse, if that's possible, Word has many own ways to do text formatting. Excel has many own ways to do text formatting. Powerpoint has many own ways to do text formatting.

By "many ways" is meant different markup, sometimes drastically different: in one you could have no country/encoding at all, and in another it's cluttered up with country/encoding markup. If Microsoft were to design a general purpose Office document model (note : ECMA 376 is a description of one specific Office document: Microsoft's), they would have factored all of this into a single text formatting markup. God only knows why they chose not to do so, keep all the legacy, and try to get away with this mess by making as little publicity as possible about it.

Now enter the implementer, or someone willing to make a change to a document. There are three scenarios :

- write a document
- read a document
- read and write the document

The third scenario is just a combination of the others, so there is nothing interesting to say about it.

The first scenario is the most simple. To write a document, of a given type, including a given set of objects (from shared languages or not), you only need to write the document in a way that is compatible with the expected XML. In other words, you can use only one text formatting markup model. It's you who decides which one, whether you implement one or more, and so on. So from a writer perspective, you don't suffer the problem very much.

Now consider the second scenario. To read a document, you cannot assume what's in that document, therefore you've got to implement all possible combinations of objects that may be part of the document. In particular, you've got to implement all ways to get text formatting markup models because that may well be the XML you face. This is a horrible scenario. To support this scenario, either you are Microsoft, or you have a number of years of work ahead on the subject with plenty of implementation done already. There is no way around, the barrier to entry to this scenario is sky high.

Of course, if you read a document, read the markup, and do nothing with it, or nothing of substance with it, it's not quite the same problem. But then, remember that even reading a small chunk of markup can be complicated because of the implicit semantics. You don't need a lot of XML markup to find yourself unable to process it in any meaningful way.

To give you an example of how bad the situation is, here is 4 different Excel text formatting markup chunks, all meant to do the same thing (not entirely accurate here, but you get the idea) :

1) regular cell formatting

```
<xf numFmtId="0" fontId="2" fillId="0" borderId="0" xfId="0" applyFont="1"/>
<font>
  <sz val="11"/>
  <color theme="6" tint="-0.249977111117893"/>
  <name val="Calibri"/>
  <family val="2"/>
  <scheme val="minor"/>
</font>
```

2) shared-string cell formatting (note that the shared-string is a technical artefact surfacing as

everyone’s problem now)

```
<r>
  <rPr>
    <sz val="11"/>
    <color rgb="FFFF0000"/>
    <rFont val="Calibri"/>
    <family val="2"/>
    <scheme val="minor"/>
  </rPr>
  <t>ruir</t>
</r>
```

3) cell formatting in a conditional alert (note: the conditional alert itself is declared elsewhere)

```
<dxf>
  <font>
    <b/>
    <i val="0"/>
  </font>
  <numFmt numFmtId="2" formatCode="0.00"/>
  <fill>
    <patternFill patternType="solid">
      <fgColor auto="1"/>
      <bgColor rgb="FFFFFFFF"/>
    </patternFill>
  </fill>
</dxf>
```

4) text formatting in charts

```
<c:rich>
  <a:bodyPr/>
  <a:lstStyle/>
  <a:p>
    <a:pPr>
      <a:defRPr/>
    </a:pPr>
    <a:r>
      <a:rPr lang="en-US"/>
      <a:t>t t</a:t>
    </a:r>
    <a:r>
      <a:rPr lang="en-US" sz="1850" u="dash" baseline="0">
        <a:solidFill>
          <a:schemeClr val="accent4">
            <a:lumMod val="60000"/>
            <a:lumOff val="40000"/>
          </a:schemeClr>
        </a:solidFill>
        <a:uFill>
          <a:solidFill>
            <a:sc hemeClr val="accent1">
              <a:lumMod val="60000"/>
              <a:lumOff val="40000"/>
            </a:schemeClr>
          </a:solidFill>
        </a:uFill>
      </a:rPr>
      <a:t>rui ry</a:t>
    </a:r>
    <a:r>
      <a:rPr lang="en-US"/>
```

```
      <a:t>t gfgfgfg</a:t>
    </a:r>
  </a:p>
</c:rich>
```

The beauty about a file format that is impossibly hard to read and update, and is decently easy to write from scratch, is that it fits perfectly in the read-only model that is exactly the reason why Microsoft has a monopoly in Office documents. As a side effect to its proprietary-ness (Office 2007 documents are extensions of ECMA 376 documents), it provides zero interoperability with anything else that exists.

From a technical marketing perspective, you can always try to start a project that will support ECMA 376, but your chances to complete the project is exactly zero. You can hear about projects starting here and there, you can hear about programs that write documents that can be opened in Office 2007 (note that the only test that can be possibly be made is whether or not Office 2007 opens it, regardless of the flaws of said program, regardless of the impedance mismatch between Office 2007 documents and ECMA 376 documents), but that is not evidence of progress in interoperability across applications and platforms, contrary to what the claims are.

For instance, to this date there isn't a computer program that perfectly mimics any of the old Microsoft Office versions 97-2000-XP-2003. It just does not exist. In addition to the impossibility of perfectly mimicking a complex and proprietary software, third parties that would go too far supporting the binary file formats would face the wrath of violating Microsoft intellectual property. Example : VBA macros. With Office 2007, Microsoft is bringing all of this forward for backwards compatibility reasons, therefore "opening up" changes nothing. What was proprietary is still proprietary, and barely referenced in ECMA 376.

8) Windows dates

Microsoft uses all kinds of date types, not just from one Office application to next, but even at a library level, there are differences. Despite storing document metadata properties of type date in an ISO compatible format (metadata properties are for instance the creation date of the document), Microsoft insists on using their legacy date types elsewhere. Unfortunately, this has consequences.

It's well documented elsewhere that the date type Microsoft uses for spreadsheets is basically flawed for a number of reasons. But what isn't often said is that the date type is actually the OLE date type. Here are the corresponding Windows OLE API functions:

- VariantTimeToSystemTime
- SystemTimeToVariantTime

Here is an excerpt of the VariantTimeToSystemTime function documentation, this may sound familiar if you have ever worked with Excel dates.

> Converts the variant representation of time to system time values.
>
> ```
> INT VariantTimeToSystemTime(
>   double vtime,
>   LPSYSTEMTIME lpSystemTime
> );
> ```
>
> A variant time is stored as an 8-byte real value (double), representing a date between

> January 1, 100 and December 31, 9999, inclusive. The value 2.0 represents January 1, 1900; 3.0 represents January 2, 1900, and so on. Adding 1 to the value increments the date by a day. The fractional part of the value represents the time of day. Therefore, 2.5 represents noon on January 1, 1900; 3.25 represents 6:00 A.M. on January 2, 1900, and so on. Negative numbers represent the dates prior to December 30, 1899.
>
> Using the SYSTEMTIME structure is useful because:
>
> - It spans all time/date periods. MS-DOS date/time is limited to representing only those dates between 1/1/1980 and 12/31/2107.
> - The date/time elements are all easily accessible without needing to do any bit decoding.
> - The National Language Support data and time formatting functions GetDateFormat and GetTimeFormat take a SYSTEMTIME value as input.
> - It is the default Win32 time and date data format supported by Windows NT and Windows 95.
>
> The VariantTimeToSystemTime function will accept invalid dates and try to fix them when resolving to a VARIANT time. For example, an invalid date such as 2/29/2001 will resolve to 3/1/2001. Only days are fixed, so invalid month values result in an error being returned. Days are checked to be between 1 and 31. Negative days and days greater than 31 results in an error. A day less than 31 but greater than the maximum day in that month has the day promoted to the appropriate day of the next month. A day equal to zero resolves as the last day of the previous month. For example, an invalid date such as 2/0/2001 will resolve to 1/31/2001.

As explained, the function makes all kinds of fixup internally, and that's exactly the problem. The date type used in Excel, which is that one, is incompatible with everything else out there, platform-dependent, and undocumented (this documentation describes behaviors, but it does not specify the date type).

If Windows dates were replaced by ISO dates, a dependency on Windows would be gone.

So, to read a cell containing a date in a spreadsheet and make sense of it implies you are using one of these two OLE API function calls. Even better, there is no cell date type. A cell is either a string, a number, ... but a date is a number with no associated type. In fact, the only way to know the cell contains a date is to lookup and parse the number format that may be associated to that cell. And that's where you enter another black hole. Here is an example of number format :

```
<numFmt numFmtId="171" formatCode="_-* #,##0.00\ _F_-;\-* #,##0.00\ _F_-;_-*
"-"??\ _F_-;_-@_-"/>
```

And another:

```
<numFmt numFmtId="171" formatCode="[$-40C][Red][>120]_-* #,##0.00\ _F_-;\-*
#,##0.00\ _F_-;_-* "-"??\ _F_-;_-@_-"/>
```

The second one is particularly interesting, it says: use the French locale (40C), if the number is greater than 120, apply red, and do formatting as follow, with plenty of padding/layout special characters (described in the ECMA 376 documentation, but not specified). Implementing those undocumented patterns requires north of 10,000 lines of code, all subject to wild guesses and platform interoperability problems.

An even more interesting bit is that the XML used in ECMA 376 departs in a number of non-

standard ways compared to the XML produced by Excel 2003 (data-only XML). Let's make a comparison. To reproduce the scenario :

- start Excel 2007 and create a new spreadsheet
- enter 11 Jan 2001 in a cell (note : translate this date according to your own regional settings)
- save the spreadsheet (xlsx)
- save the spreadsheet as Excel 2003-compatible (the exact file type option is XML Spreadsheet 2003 (*.xml))
- close it
- unzip the xlsx package and view the part called xl/worksheets/sheet1.xml
- open the Excel 2003-compatible XML file

Here is the relevant XML you'll see in xl/worksheets/sheet1.xml :

```
<row r="3" spans="3:3" ht="15" customHeight="1">
 <c r="C3" s="1">
    <v>36902</v>
  </c>
</row>
```

If you are puzzled by what the value `36902` might be, it's Excel's Windows date encoding! It is not possible to infer the cell contains a date. You have to explore the number format attached to style `s="1"` which in turn is described in another part of the package, xl/styles.xml :

```
<cellXfs count="2">
  <xf id="0" numFmtId="0" fontId="0" fillId="0" borderId="0" xfId="0"/>
  <xf id="1" numFmtId="168" fontId="0" fillId="0" borderId="0" xfId="0"
     applyNumberFormat="1"/>
</cellXfs>
```

In turn, `numFmtId` is defined in a separate collection of the same part xl/styles.xml :

```
<numFmts count="1">
  <numFmt numFmtId="168" formatCode="\D\a\t\e\ \:\ d\ mmm\ yyyy"/>
</numFmts>
```

If you parse the proprietary encoding in the number format adequately, you might be able to infer it's a date. To compute the actual date involves more effort, obviously.

Let's take a look at our Excel-2003 compatible XML :

```
<Row ss:Index="3" ss:AutoFitHeight="0">
  <Cell ss:Index="3" ss:StyleID="s63">
    <Data ss:Type="DateTime">2001-01-11T00:00:00.000</Data>
  </Cell>
</Row>
```

This old Excel 2003 XML is not nearly as bad as the one in ECMA 376. We can infer it's a date thanks to the `ss:Type` attribute. And, goodness, the date is encoded using ISO 8601, which is definitely a good thing for interoperability purposes. The obvious question : why is the new SpreadsheetML using a proprietary date encoding, when Microsoft managed to ship Excel 2003 (this Excel 2003 XML can also be generated with Excel 2007) with dates supporting an international standard encoding, and no need to go into proprietary parsing? Why isn't SpreadsheetML an extension to the old Excel 2003 XML following the same good principles?

9) All roads lead to Office 2007

In part 2 of ECMA 376, page 96, we learn :

> Table H–1. Package model conformance requirements
>
> M1.30 The package implementer shall name relationship parts according to the special relationships part naming convention and require that parts with names that conform to this naming convention have the content type for a Relationships part

In theory, this requirement allows unattended part renaming. But this means in practice that some process in the processing chain may reshuffle an entire package on its own, which may break assumptions from other processes of the processing chain.

Interestingly enough, Excel 2007 takes it to heart to do this reshuffling. If you create a package with part names and relationships that perfectly conform to open packaging conventions as defined by ECMA 376 part 2, then this by no means provide neutrality throughout the entire processing chain, especially if you open the package in Excel 2007 and save it without making changes.

To reproduce the scenario:

- start Excel 2007 and create a new spreadsheet
- save the spreadsheet (xlsx file)
- close it
- unzip it
- edit [Content_Types].xml part manually, and replace /xl/worksheets/sheet1.xml by /xl/custom/sheet1.xml
- edit xl/_rels/workbook.xml.rels, and replace worksheets/sheet1.xml by custom/sheet1.xml
- grab xl/worksheets/sheet1.xml locally as a file
- delete the worksheets “folder”
- create “folder” xl/custom in the package
- add the worksheet part in “folder” xl/custom
- zip it
- open it in Excel 2007
- click save
- close it
- unzip it

What you should see is that our custom structure has been replaced with Excel 2007 hardcoded structure. If a business process assumes the existence of a custom structure, it won’t work. Why is Excel 2007 reshuffling the package? It’s ours!

10) A world of ZIP+OLE files

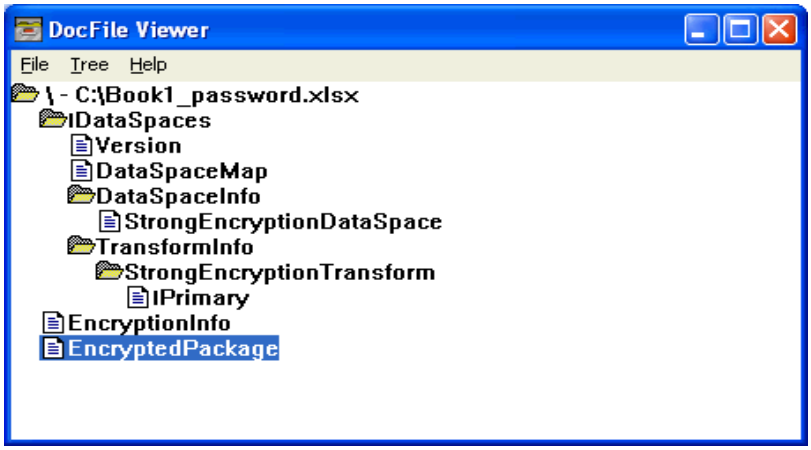
To reproduce the scenario:

- start Excel 2007 and create a new spreadsheet

- click on the Office button, select the Prepare menu option, and then Properties
- enter metadata such as author, title, ...
- click on the Office button, select the Prepare menu option, and then Encrypt Document
- enter a password, re-enter the password
- save the spreadsheet (xlsx file)
- close it

If you try to unzip it, you'll get an error. When you password-protect any Office 2007 document, it becomes an OLE document. Wait a minute, isn't Microsoft moving to ZIP files?

Here is a screenshot of what you should see in an OLE document viewer :



A password-protected MS Office 2007 document is a...OLE file, not a ZIP file

If you roll-over your mouse on the file in Windows Explorer (note: behavior not tested on Windows Vista), you should get a flying tool-tip with just minimal information that is provided by the disk file system, but not the document metadata (author, subject, title, keywords, last modification date, ...)

The OLE container contains a number of OLE streams. The document metadata is not available for consumption, meaning that you cannot retrieve this information either. This is a regression compared to binary file formats.

In fact, there are two regressions: the document type changes ; the metadata is not accessible. And the password-protection mechanism is undocumented as a whole.

Let's see how exactly it compares to an old binary password-protected Excel file.

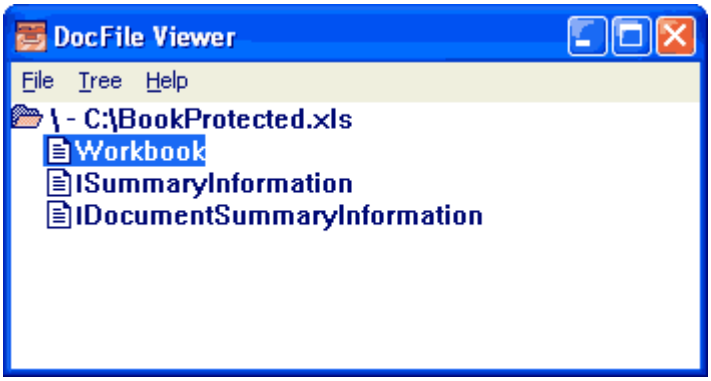
To reproduce the scenario :

- start Excel 97 or 2000 or XP or 2003 and create a new spreadsheet
- go in File / Properties
- enter metadata such as author, title, ...
- go in Tools / Options, select the Security tab
- enter a password in field *Password* to open and click OK, re-enter the password
- save the spreadsheet (xls file)

- close it

If you roll-over your mouse on the file in Windows Explorer, you should get a flying tool-tip with all the information that is provided by the disk file system plus all the document metadata (author, subject, title, keywords, last modification date, ...)

Indeed, if we open the xls file in an OLE document viewer, the file structure is kept intact, only the OLE document stream itself is encrypted (contains the encrypted BIFF content). The stream holding the metadata, *ISummaryInformation*, is left unencrypted. Here is a screenshot of a password-protected binary Excel file in an OLE document viewer :



A legacy password-protected MS Office document is still an OLE file

This has unfortunate consequences for Content Management Servers (CMS). By making the metadata of password-protected Office 2007 documents unavailable, existing CMS workflows are arbitrarily broken. By changing the content type of the document, only because it's password protected, existing CMS workflows have another reason to break.

Note : if Microsoft Sharepoint server, or Windows Vista, manages to show the metadata of a password-protected Microsoft Office 2007 document, perhaps it knows something that we don't ...

11) Document security is a (bad) joke

To reproduce the scenario:

- start Excel 2007 and create a new spreadsheet (or open an existing spreadsheet).
- right-click on the first sheet tab, and select Protect Sheet...
- enter a password, hit enter, re-enter the password, hit enter.
- try editing cells, you should get an error message.
- save, close, unzip.
- edit the zip part named xl/worksheets/sheet1.xml
- find the XML element called **sheetProtection**
- remove the element **sheetProtection** and the accompanying attributes (password, sheet, ...).
- save the changes and put that part back in the zip file.
- open the file in Excel 2007.
- try editing some cells, **it works!**

Can you think of a worse document security? We have a format that encrypts buffers into OLE containers when it shouldn't, and leaves XML password hashes right in the ZIP file in such a way that anyone can manually edit the file and remove it.

When you consider the amount of professional spreadsheet workers out there relying on sheet protection to ensure the integrity of their spreadsheets, you have no idea the impact of this flaw when they learn it ...

There goes another regression with those formats. Note: the Excel binary format had a similar PASSWORD record stored on a per worksheet basis, but the big difference is that none of the target audience is able to edit a binary Excel file. It's binary, therefore it's a form of security by obscurity. And since binary files contain pointers to the stream, removing a record using an hexadecimal editor without taking care of the dirty laundry would automatically result in a corrupt spreadsheet.

I guess it's fair to say it's a case where document security needs a calculated chain to make such manual changes less easy! (see the first section of that article to get the irony in full perspective).

12) BIFF is gone ... not!

BIFF (Binary Interchange File Format) is the name of the Excel binary file format. A BIFF buffer is contained within an OLE container, and consists of a sequence of BIFF records. A BIFF record consists of an identifier, a length to follow, and the corresponding buffer of said BIFF record. BIFF itself is platform-neutral, but of course OLE isn't. Anything stored within BIFF records referencing Windows-specific functions (such as the DEVMODE structure of printer settings) voids any platform-neutral claims. Every Excel release has its own BIFF-specific records. Those set of collective records are called BIFF8 for Excel 97, BIFF9 for Excel 2000, BIFF10 for Excel XP and BIFF11 for Excel 2003.

One of the claims from the introduction is that ECMA 376 documents are moving to ZIP and XML exclusively. This implies that BIFF is gone. Let's see if that's true.

> From: Doug Mahugh (Microsoft)
> Sent: August 21, 2006
> To: Stephane Rodriguez
>
> Stephane,
>
> I shared the link with some of the people over in the BIFF12 group today, and they liked the article too. They had some additional info for you which may be of interest. These sorts of details will all be documented in the BIFF12 documentation that will come out when Office ships, but for now there's no way you could know what's going on for sure (as you explain in the article)
>
> ...
>
> Doug Mahugh
> Office 2007
> Technical Evangelist
> 425-707-1182 | MSDN Blog | OpenXmlDeveloper.org

The subject matter? This article: http://www.codeproject.com/cs/library/office2007bin.asp.

Ironically enough, to this date, the only article shedding a light on BIFF12 available on the internet is this article. Microsoft made no such document available.

There is more than meets than eye.

Besides the point that BIFF is actually not gone at all, an even bigger shocker is that BIFF12 is a departure from BIFF11. It's not an extension of BIFF11 with new BIFF records, it's a complete rewrite of all BIFF records, new BIFF record model and new content encoding. So any person who would have invested in a BIFF library (or grabbed it elsewhere), will have to redo that work from scratch. Where is backwards compatibility gone?

According to Microsoft's Excel blog at http://blogs.msdn.com/excel/archive/2006/07/20/671995.aspx

> File Format Number 2 - Excel Binary (XLSB files)
>
> The Excel binary format is the second full fidelity format for Excel 2007. It is similar to the Office Open XML format in structure - a set of related parts, in a zip container - except that instead of each part containing XML, each part contains binary data. Even though we've done a lot of work to make sure that our XML formats open quickly and efficiently, this binary format is still more efficient for Excel to open and save, and can lead to some performance improvements for workbooks that contain a lot of data, or that would require a lot of XML parsing during the Open process.

It's hard enough to understand that Microsoft Office is moving to XML, and still away from XML at the same time, but the official justification is performance related. Unfortunately, this justification is just a lie since it is already the justification for the extreme akwardness of the SpreadsheetML XML file format itself. Someone from the Microsoft Office team has this to say about it (at http://blogs.msdn.com/brian_jones/archive/2006/10/26/performance-of-xml-file-formats.aspx):

> There are a number of things we looked into doing to help improve the performance of these files (SpreadsheetML) both using current technology as well as thinking about future approaches. Some of the approaches we took that are easiest to understand are:
>
> 1. Reduce the need to fully parse repeating data - In order to cut back on parsing large strings repeated multiple times, or formulas that are repeated down an entire column, spreadsheetML does a lot of sharing. I already talked about the huge benefits we can get from the shared formulas work in a post last spring. The shared string table can give similar results.
>
> 2. Tag Size - the size of XML elements actually does directly affect performance times. The more text you have to parse, the longer it will take.
>
> 3. Splitting the XML into multiple parts - In SpreadsheetML we break the XML content out into multiple parts in the ZIP (each worksheet, the string table, pivot table data, etc.). This can help a ton with on demand loading, or even loading multiple pieces at once. For example, it means that if you had a multi-proc machine, you could load each worksheet on a separate thread. It also means that you could decide to only load one sheet and you wouldn't have to parse through all the XML from the other sheets.
>
> 4. Relationships stored outside of content - By storing all the relationships from one part to another in separate (and much smaller) files, it makes it really easy to see what other

> parts you should load when you are loading a particular part of the file. If it weren't for the relationships, you'd actually have to parse through the content of the XML to determine what other resources that part used. For example, if you wanted to just load one slide in a presentation, you can also see what images are used on that slide before you start parsing the XML for the slide.

At this point, we are left with the obvious question, if the SpreadsheetML is made much more complex than we would have expected only to cope with performance problems, what is the rationale for the binary workbook (.xlsb) ?

Microsoft won't tell. There are two reasons however: 1) embarrassing reality of XML; 2) embarrassing reality of ECMA 376.

To illustrate the first untold reality, suffice to create a new spreadsheet, and then query external data. As you do this, Excel 2007 creates a part called the connection data source part, where **it stores the connection strings in plain text**, among other things. It should be clear by now that connection strings contain sensitive information such as server names, login and passwords. OOps!

The quick Microsoft solution to this? Security by obscurity, just turn this stuff into binary records (BIFF12), and the problem goes away in theory.

A similar problem with XML parts is that password hashes are stored in plain-text, as we've seen in the previous section, meaning that armed with a simple text editor, both password hashes and connection string passwords can be edited and/or removed. Vulnerabilities by design?

The embarrassing fact for ECMA 376 is that since this is supposed to be XML only, the new binary workbook, despite being the official answer to the "plain-text" problem, cannot be part of said documentation otherwise automatically violating the claim that those documents are made with XML.

Last but not least, a pseudo-rebuttal was posted to address the lack of availability of the BIFF12 documentation, a minor problem in comparison to the XML violation rules by the way. This pseudo-rebuttal was explaining that the documentation of binary formats can be freely obtained. Here is the corresponding article - http://support.microsoft.com/kb/840817. In which we learn:

> Microsoft makes its .doc, .xls, and .ppt binary file format specifications available under a royalty-free covenant not to sue to anyone who wishes to implement all or part of these specifications in their products. **Implementation includes the ability to use the specification documentation for analysis and forensic reference purposes**. Microsoft Office Drawing File Format for 2007 and Visual Basic for Applications (VBA) File Format for 2007 are also available under this program.

No right to create a competing product.

As for whether Microsoft responds at all to any such documentation request, it remains to be seen. Last but not least, whether the documentations contain material that are required even for just analysis or forensic purposes remains to be seen. Microsoft used to distribute those documentations as part of the MSDN Library, until February 1998 (right when Mr Gates had a pinch for Office documents viewed in Internet Explorer). Those documentations are incomplete, often just descriptive, and full of typos. In fact, just as the ECMA 376 documentation itself.

Now enter BIFF11+. Yes, you heard that right! Microsoft created not just one new BIFF file

format, it has also taken the time to extend BIFF11 (i.e. Excel 2003's BIFF file format) to include new BIFF records. What for ?

The reason is round-tripping of spreadsheets. Imagine you are creating a new Excel 2007 spreadsheet using some of the new features, such as the formatting databar (a bar chart drawn inside cells). Then save this file as a Excel 97-2003 compatible file in order to facilitate the collaboration with others (not using Excel 2007). What you still expect is that this file preserves those features. That's exactly what BIFF11+ does, just that Microsoft hasn't considered documenting those new BIFF records, not even talking about it by the way.

To reproduce the scenario:

- start Excel 2007 and create a new spreadsheet (or open an existing spreadsheet).
- enter values 10, 20, 30, 40, 50 in separate cells, then click on the ribbon on the Style button, Conditional Formatting, Data bars, and then pick a color gradient.
- save the spreadsheet as a Excel 97-2003 Workbook (*.xls)
- a prompt warns that there is going to be a significant loss. Click Continue.
- open the .xls file with any older version of Excel (97/2000/XP/2003)
- the file opens with no prompt, and the databar does not show up (which was expected).
- enter value 60 next to the other cells, and save the file as a regular .xls file, just like it is.
- open this file in Excel 2007 : the databar magically shows up. The databar however does not include the value we just entered, meaning that we have created a discrepancy.

How can the databar show up at all in Excel 2007 since we went through two independent .xls writing phases (one with Excel 2007, one with Excel 97/2000/XP/2003)?

You guessed it, new BIFF11+ records are created and preserved. None of that is documented, meaning that this round-trip scenario only works with Microsoft Office. It works as follows:

- when Excel 2007 creates a Excel 97-2003 workbook (*.xls), it actually creates a BIFF11+ file. Because BIFF11+ is an extension of BIFF11, it can be opened in Excel 97, 2000, XP and 2003.
- and because how BIFF works (any record in the header and footer of internal sections is preserved as is), it allows to do a save in Excel 97, 2000, XP and 2003 without killing those new records.
- since Microsoft created such new BIFF records to represent the new features: theme, databars, new visual candy in charts, ... but none of it is documented, it follows that interoperability across those file formats is exclusive to them.
- BIFF11+ records are not a subset of BIFF12 records. Implementers will have to implement/preserve BIFF11+ records and BIFF12 records.
- we have a discrepancy in the databar that is left for actual spreadsheet users to worry about. The value 60 is left disconnected to the formatting databar, even in Excel 2007.

To close up the discussion about BIFF, let's just take a look at an example of BIFF11+, actually just an excerpt from a file (in red, lines 16-24, the new BIFF11+ records):

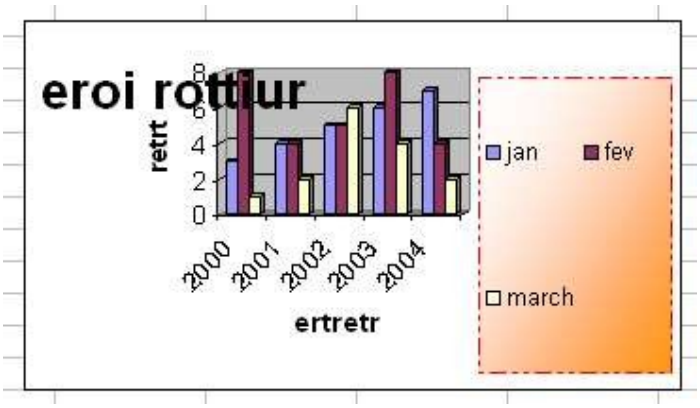
```
[DIMENSIONS 000E] 01 00 00 00 06 00 00 00 01 00 02 00 00 00
```

```
[ROW 0010] 01 00 01 00 02 00 2C 01 00 00 00 00 00 01 0F 00
[ROW 0010] 02 00 01 00 02 00 2C 01 00 00 00 00 00 01 0F 00
[ROW 0010] 03 00 01 00 02 00 2C 01 00 00 00 00 00 01 0F 00
[ROW 0010] 04 00 01 00 02 00 2C 01 00 00 00 00 00 01 0F 00
[ROW 0010] 05 00 01 00 02 00 2C 01 00 00 00 00 00 01 0F 00
[FLOAT 000A] 01 00 01 00 0F 00 00 00 24 40
[FLOAT 000A] 02 00 01 00 0F 00 00 00 34 40
[FLOAT 000A] 03 00 01 00 0F 00 00 00 3E 40
[FLOAT 000A] 04 00 01 00 0F 00 00 00 44 40
[FLOAT 000A] 05 00 01 00 0F 00 00 00 49 40
[DBCELL 000E] AA 00 00 00 50 00 0E 00 0E 00 0E 00 0E 00
[WINDOW2 0012] B6 06 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00
[SELECTION 000F] 03 01 00 01 00 00 00 01 00 01 00 05 00 01 01
[00EF 0006] 00 00 37 00 00 00
[0867 0017] 67 08 00 00 00 00 00 00 00 00 00 00 02 00 01 FF FF FF FF 03 44 00 00
[089C 0026] 9C 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 3C 33 00 00 00 00 00 00 00 00
[088B 0010] 8B 08 00 00 00 00 00 00 00 00 00 00 00 00 02 00
[0879 0022] 79 08 01 00 01 00 05 00 01 00 01 00 01 00 03 00 01 00 05 00 01 00 01
00 01 00 01 00 05 00 01 00 01 00
[087A 004C] 7A 08 00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 00 00 00
00 00 00 01 01 00 03 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 2A 00
00 01 0A 5A 02 00 00 00 FF 55 5A FF 00 00 00 00 00 00 00 00 02 00 00 03 00 00
[EOF 0000]
```

We thought BIFF was gone, and in fact now we have BIFF11+ and BIFF12. Great progress on the interoperability front ...

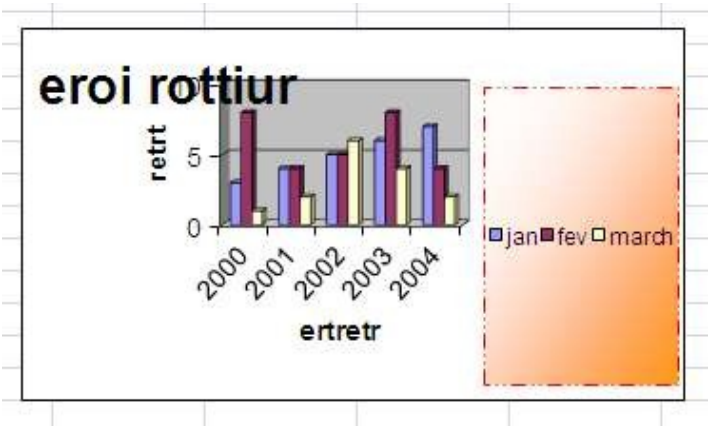
13) Document backwards compatibility subject to neutrino radio-activity

Here is a simple chart created with Excel 2003. Open it with any of Excel 97/2000/XP/2003, then open it in Excel 2007. Microsoft said they could not change the internal structure of Excel spreadsheets (and other Office document types) because they had to provide 100% full fidelity otherwise their customers would not want it. Is this true?



A simple chart in Excel 2003

Same file opened in Excel 2007



The differences are:

- vertical axis all set to automatic scale/min/max.
- also impacts the number of horizontal gridlines in the background.
- chart title font not the same weight chart title incorrectly positioned vertically.
- legend border incorrect.
- legend entries incorrectly positioned.
- spacing between the plot area and the legend.

<sarcasm>Programs used by hundreds of millions people need no special attention, that goes without saying ... </sarcasm>

Good luck programming charts. Microsoft provides no documented mapping between the Escher library (pre-Office 2007 era) and the DrawingML+VML libraries (Office 2007 era, before it gets dropped to something else in the future). Is there possibly a reason why?

Answer: Microsoft has dropped the existing chart drawing engine in favor of a new library. It is impossible in practice to rewrite a library with new source code, **a new agenda (visual candy)**, and manage to get 100% full fidelity with the past. This results in improper drawing of **existing** charts. Which means, in other words :

- Microsoft is lying, the part of ECMA 376 documentation describing charts does not 100% map the old chart drawing spec.
- The implementation itself is buggy, an automatic consequence of the first point.

14) ECMA 376 documents just do not exist

Last but not least, ECMA 376 documents just do not exist. The reasons why are manifold:

- Office 2007 documents are derived from theoretical ECMA 376 documents, to which is added binary parts, macros, OLE objects, ActiveX serialization, DRM, sharepoint metadata, ...
- Office 2007 documents are incompatible with theoretical ECMA 376 documents, since the ECMA 376 documentation says among other things that VML is deprecated, and Office 2007 documents are still using plenty of it when creating new documents.

- Microsoft strategy itself is to provide as little information as possible about the huge impedance mismatch between ECMA 376 and the actual implementation. Expect ECMA 376 to evolve only marginally, while Office 2007 next version will come with plenty more of Microsoft proprietary layers, especially those integrating the Microsoft Office suite with Windows (on the client), more undocumented integration points between Microsoft Office, Windows and Sharepoint (on the server), more undocumented client-server protocols (between the Microsoft Office client, and servers running Microsoft server software).

These are blatant elements why the ISO submission is actually a diversion. If you think the elements above are not enough, consider the following response at http://blogs.msdn.com/brian_jones/ archive/2007/07/12/spreadsheet-formula-bugs.aspx from Brian Jones, Microsoft employee and authoritative voice on Microsoft Office XML formats: (emphasis mine)

> To your last point [It would be good if Microsoft would state officially its intent to support future development and improvement of the standard in Ecma of new version of the format and that it intends those version to get similar open licensing.], **it's hard for Microsoft to commit to what comes out of Ecma** in the coming years, because we don't know what direction they will take the formats. We'll of course stay active and propose changes based on where we want to go with Office 14. At the end of the day though, the other Ecma members could decide to take the spec in a completely different direction. Now my impression is that won't happen, as the folks on the TC [Ecma Technical group Committee, in which Microsoft is the ... armchair] all have pretty similar visions for the future of the spec, but since it's not guaranteed it would be hard for us to make any sort of official statement.

15) How the ISO/IEC 26300 OpenDocument format (ODF) compares?

When I wrote this article, I wanted it to focus on Microsoft Office XML formats exclusively, but Microsoft apologists out there running out of arguments to find a justification as to why Microsoft Office XML formats are that bad, turned to the typical rhetoric, started sending piques at the ISO OpenDocument format (ODF). I thought, well, let's do a quick comparison on all the points above. Results are as follows :

- *Self-exploding spreadsheets*: I have created a visual diff of two ODF spreadsheets, before and after the modification of a cell containing a formula.

  And sure enough, there is no calculation chain, which means it's easy to make manual changes to existing ODF files, no hassles, you don't have to care about the format's own dirty laundry. And to remove the formula, you just remove the corresponding XML formula attribute in that cell. ODF ok

BookSample.original.ods	DIFF	BookSample.updated.ods
META-INF/manifest.xml		META-INF/manifest.xml
content.xml	2	content.xml
meta.xml	1	meta.xml
mimetype		mimetype
settings.xml		settings.xml
styles.xml	1	styles.xml
Thumbnails/thumbnail.png		Thumbnails/thumbnail.png
Configurations2/accelerator/current.xml		Configurations2/accelerator/current.xml
Configurations2/floater/		Configurations2/floater/
Configurations2/menubar/		Configurations2/menubar/
Configurations2/popupmenu/		Configurations2/popupmenu/
Configurations2/progressbar/		Configurations2/progressbar/
Configurations2/statusbar/		Configurations2/statusbar/
Configurations2/toolbar/		Configurations2/toolbar/
Configurations2/images/Bitmaps/		Configurations2/images/Bitmaps/

- *Entered versus stored values*: the entered values are stored as is. There is no rounding error. ODF goes as far as storing both the entered value and the displayed value (after the number format is applied) which is a wonderful thing for those willing to extract data. ODF ok
- *Optimization artefacts*: there is no such thing as a shared formula concept in ODF. I guess ODF needs not be encumbered with optimization techniques that were valid 20 years ago, when computers were a thousand times less powerful than right now. ODF ok
- *VML isn't XML*: there is no VML or anything fundamentally non- XML in ODF. If you create a note in an ODF file, the corresponding XML fragment is:

```
<office:annotation office:display="true" draw:style-name=" gr1" draw:text-style-
 name="P1" svg:width="2.899cm" svg:height="0.596cm" svg:x="7.374cm"
 svg:y="0.307cm" draw:caption-point-x="-0.61cm" draw:caption-point-y="1.5cm">
  <dc:creator>sr</dc:creator>
  <dc:date>2007-09-02T00:00:00</dc:date>
  <text:p text:style-name="P1">some comment</text:p>
</office:annotation>
```

This is proper XML, one value per attribute, the fragment is full in context in the stream. Plus, you can grab an existing SVG library to make sense of the XML, no need to implement a single vendor's soup. ODF ok

- *Open packaging parts minefield*: there is no such thing as a tree of parts. Instead, ODF has a central directory whose management is deterministic and in fact trivial. In retrospect, why did Microsoft went with such a problematic concept of a tree part with undeterministic relations? Was it because strictly copying ODF was embarrassing (Not-Invented-Here syndrome) ? ODF ok
- *International, but US English*: everything is stored uniformly in US English, there is no discrepancy. The locale is applied as per the style definitions, separate from the data, which is a good design in decoupling that implementers can take advantage of (smaller implementations, less prone to errors, especially in a global market context). ODF ok
- *Many ways to get in trouble*: ODF defines a single style concept across the entire formats. So not only any Word, Spreadsheet or Presentation document has a single style concept instead of many. But both document types share the same style concept. This translates in interoperable formatting across applications and platforms, plus small implementations. Contrary to ECMA 376, implementers don't have to implement more than 15 years of formatting legacy. ODF ok

- *Windows dates*: ODF stores ISO 8601 dates, which makes it easy for implementers to interoperate on any application or platform. Here is the following XML fragment when entering date 11/01/2001 03:45:20 : ODF ok

```
<table:table-cell table:style-name="ce1" office:value-type=" date"
 office:date-value="2001-01-11T03:45:20">
  <text:p>11/01/2001 03:45:20</text:p>
</table:table-cell>
```

- *All roads lead to Office 2007*: ODF also destroys any custom part name. ODF also wrong
- *A world of ZIP+OLE files*: After encryption, an ODF file is still an ODF file (i.e. ZIP, XML). ODF encrypts the content part (using a standard algorithm described in the manifest part). The metadata part is left unencrypted. ODF ok
- *Document security*: ODF does the same, it saves a password hash that can be manually removed. ODF also wrong
- *BIFF is gone*: ODF is ZIP and XML. If you insert or import an OLE object, it's saved as a binary part, just like ECMA 376 does, but the ODF streams themselves are still XML. ODF ok
- *Document backwards compatibility*: I've installed the oldest OpenOffice release I could get (1.0.3, released in 2003), loaded the same chart as in the example, saved as an .SXC file (this was the name extension before it became an .ODS file later), then loaded the .SXC file in the OpenOffice 2.2. The chart is exactly the same in both versions. ODF ok
- *ECMA 376 documents do not exis*t: No hidden intellectual property since it's a 100% open-source project in GPL. ODF ok

Final score: ODF is ok on 12 points, wrong on 2 points.

To better cope with those problems, ECMA 376 will have to be redesigned to something that is pretty much what ODF does. Why isn't Microsoft extending ODF instead of reinventing a bad wheel? Customers would benefit a great deal of a single standard.

-Stéphane Rodriguez, *August 2007*
Independent software vendor, file format expert
Not affiliated to any pro-MSFT or anti-MSFT party/org/ass.

Update : this article was Slashdotted on Sunday 26 of August.
Update2 : this article is taking 300,000 hits a day, and is making it all around the world in all kinds of sites. My web host provider was so angry at the peak in traffic that he threatened to cut me off, so I had to redirect to a blog site such as Google's blogger to host the article.
Update3 : wednesday august 29, added a new section on Document security
Update4 : friday august 31, added more content to sections US English and Windows dates
Update5 : sunday september 2, added a quick comparison between ODF and ECMA 376
Update6: thursday october 11, minor changes to 2) and 15)