phuse·connect
connect·share·advance
5-8 November
ICC Birmingham
EU 2023
The Clinical Data Science Conference

Paper DH04

Look to the Sky: Introducing COMET - the COncomitant MEdication review Tool

Nev Cope, AstraZeneca, Cambridge, UK

ABSTRACT

The review and evaluation of concomitant medication use by medical teams can be a time-consuming manual task. Study teams frequently create their own programs and procedures, often in isolation, leading to wheel-reinvention and inconsistent cross-study approaches.

The outcome of these reviews is critical to the identification and reporting of disallowed medication use and potential protocol deviations.

COMET is a SAS® tool that compares current data to the previously reviewed iteration, and presents the results in a clear, concise, and easy to use spreadsheet.

Spreadsheet functionality includes filters, persistent comments, new/amended record identification, highlighting of changed values and required re-review indicators – which together make the review process quick and efficient.

This paper will show the life cycle of a COMET report and discuss some of the tricks and hacks utilised in order to fully customise an Excel® spreadsheet created by SAS.

INTRODUCTION

A common procedure carried out during the conduct of a clinical trial is the review and documentation of protocol deviations related to the use of concomitant medications (conmeds). While most of these can be identified programmatically, ultimately the final decision on whether a conmed is actually a protocol deviation or not is made by a qualified person (or more typically a group of people) within the study team and documented in some sort of spreadsheet. If the database is being updated continuously, and potential conmed protocol deviation reports are being generated at regular intervals, it can be difficult to keep track of all the historical decisions.

As part of the Data Automation group, I was approached to see if a solution could be developed to help facilitate this decision tracking procedure.

When I was first contacted, the team already had a specification in place for the underlying data (a data set named “DISMED” that was primarily the SDTM.CM domain, with a few extra date calculation and reference variables added) and an example of the output Excel file that the medical review team were used to working with.

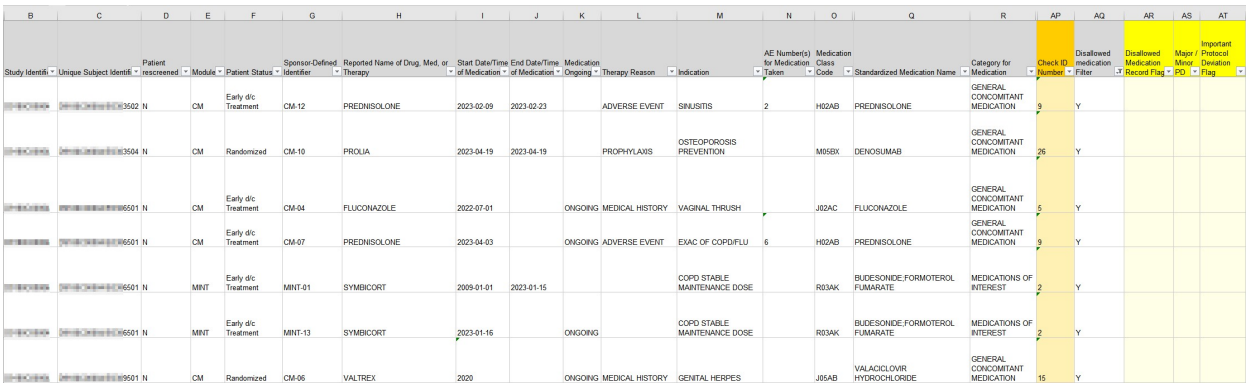
With a start and end point defined, it was just a matter of coming up with a way of generating these reports cyclically and having a mechanism for keeping track of ongoing decisions – and thus COMET was born.

EARLY DEVELOPMENT

The initial development of the COMET tool took only a couple of months, and after the first release the code continued to be fine-tuned over a period of around 6 months.

The input DISMED data set had been preprogramed, and included selection criteria for potentially disallowed concomitant medications (e.g., “where ATC code = XXX” or “where medication XXX was taken within Y days of visit Z”, etc).

The initial version of COMET took the DISMED data set, generated an output Excel file (See Figure 1 below) and had a rudimentary mechanism for reading back in the reviewed version of the Excel file - and the plan was that process would be repeated at intervals until the end of the study.



B	C	D	E	F	G	H	I	J	K	L	M	N	O	Q	R	AP	AQ	AR	AS	AT
Study Identifi	Unique Subject Identifi	Patient rescreened	Module	Patient Status	Sponsor-Defined Identifier	Reported Name of Drug, Med, or Therapy	Start Date/Time of Medication	End Date/Time of Medication	Medication Ongoing	Therapy Reason	Indication	AE Number(s) for Medication Taken	Medication Class Code	Standardized Medication Name	Category for Medication	Check ID Number	Disallowed medication Filter	Disallowed Medication Record Flag	Major / Minor PD	Important Protocol Deviation Flag
	3502	N	CM	Early d/c Treatment	CM-12	PREDNISOLONE	2023-02-09	2023-02-23		ADVERSE EVENT	SINUSITIS	2	H02AB	PREDNISOLONE	GENERAL CONCOMITANT MEDICATION	9	Y			
	3504	N	CM	Randomized	CM-10	PROLIA	2023-04-19	2023-04-19		PROPHYLAXIS	OSTEOPOROSIS PREVENTION		M05BX	DENOSUMAB	GENERAL CONCOMITANT MEDICATION	26	Y			
	6501	N	CM	Early d/c Treatment	CM-04	FLUCONAZOLE	2022-07-01		ONGOING	MEDICAL HISTORY	VAGINAL THRUSH		J02AC	FLUCONAZOLE	GENERAL CONCOMITANT MEDICATION	5	Y			
	6501	N	CM	Early d/c Treatment	CM-07	PREDNISOLONE	2023-04-03		ONGOING	ADVERSE EVENT	EXAC OF COPD/FLU	6	H02AB	PREDNISOLONE	GENERAL CONCOMITANT MEDICATION	9	Y			
	6501	N	MINT	Early d/c Treatment	MINT-01	SYMBICORT	2009-01-01	2023-01-15			COPD STABLE MAINTENANCE DOSE		R03AK	BUDESONIDE,FORMOTEROL FUMARATE	MEDICATIONS OF INTEREST	2	Y			
	6501	N	MINT	Early d/c Treatment	MINT-13	SYMBICORT	2023-01-16		ONGOING		COPD STABLE MAINTENANCE DOSE		R03AK	BUDESONIDE,FORMOTEROL FUMARATE	MEDICATIONS OF INTEREST	2	Y			
	9501	N	CM	Randomized	CM-06	VALTREX	2020		ONGOING	MEDICAL HISTORY	GENITAL HERPES		J05AB	VALACICLOVIR HYDROCHLORIDE	GENERAL CONCOMITANT MEDICATION	15	Y			

Figure 1 - Early Spreadsheet of DISMED data set

This reviewed Excel file had a handful of columns added to hold the decisions that the reviewers had made. These consisted of:

- Confirmation of the programmed deviation flag
- Major/Minor protocol deviation indicator
- Important protocol deviation flag
- Text comment(s)

When reading back the Excel files into SAS I encountered my first major problem. I was using the newer XLSX file format, and when reading these into SAS (as with so many input procedures and mechanisms in SAS), assumptions and guesses are made by the SAS procedure as to the length and formatting of the variables. This caused some issues when trying to merge different versions of the data together as variable types and lengths could change.

For example – when being read into SAS using the LIBNAME XLSX engine, a blank column in the Excel file is assigned as a character variable of length 1. Therefore, if there are no comments yet, the comment variable would set too small for use at future iterations.

Also, if the original data had a character variable that just happened to only contain numeric values in the early runs, when this was read the resulting SAS variable would be numeric. If the same column in a subsequent run contained a character string, then the data type would change.
Situations like this meant that sometimes merging reviewed data with a previous data set would potentially fail.

I experimented with having various forms of metadata describing the structure of the data, and using that to influence the way the Excel was read in - but I could not find a satisfactory solution.

During one such test, I had a hidden sheet within the XLSX file that contained a list of the columns and their variable types/lengths, and had a program read this first and use it to generate the input statements for the rest of the file. This proved to be too complicated in the long run and was abandoned (although the metadata sheet in the excel file remained and was used in a later version to double-check some variable attributes).

These issues all led to a big decision in the early design of COMET – that the storage mechanism for the ongoing data would be a *pairing* of a SAS data set and an Excel file.

DATA INTEGRITY
Having decided to store COMET data as a pairing of SAS data set and Excel file I ran through some simulated reviews and quickly encountered the second major problem – that of retaining data integrity.

I was storing the current iteration of the medications data as a SAS data set and merging this with the flags and comments from the reviewed spreadsheet at the subsequent run. This worked well providing the format of the excel file stayed the same. However, as experienced programmers will know: when bouncing data around in excel files it is easy for data to be altered unintentionally!

When generating Excel files in SAS you do have the option to lock cells, but this can only be done as an option on the ODS Excel statement, and it affects the *entire* worksheet. I wanted to only lock certain cells, leaving a handful of columns editable so that the reviewers can enter their decisions.

After researching this problem, I recalled that the newer Excel file format (XLSX) is a ZIP file that contains many XML files. These XML files not only hold the worksheet data itself, but also all the details of the structure and formatting of them.

EXAMINING THE EXCEL XML

All the modern Microsoft® Office 365® files (DOCX, XLSX, PPTX) are actually ZIP containers. Within them is a rigid structure, consisting of "docProps", application (either "word", "xl" or "ppt") and "_rels" folders.
Within these folders, nearly all the data are stored as XML files.

On the right is an example of the contents of a new blank XLS file.

The files in "docProps" contain overall properties such as the document creator, word count, default zoom level, etc.
The "rels" files contain URLs to the XML schemas and other internal links that Excel apparently needs to function correctly (although the URLs themselves appear to be invalid)

- docProps
  - app.xml
  - core.xml
- xl
  - theme
    - theme1.xml
  - worksheets
    - **sheet1.xml**
  - _rels
    - **workbook.xml.rels**
  - **styles.xml**
  - **sharedStrings.xml**
  - workbook.xml
- _rels
  - .rels
- [Content_Types].xml

PICKING APART THE EXCEL FILE

The application folder ("xl" in the case of XLSX files) is where the main body of the file lies.
The "xl/theme/theme1.xml" file contains the overall colour theme for the document, which if created from SAS may be governed by the ODS style used.

The "xl/styles.xml" file is where things start to get interesting. Every unique combination of options for a particular cell are recorded and given a name within the XML. Each cell can then be assigned one of these style options.

The "xl/worksheets/sheet1.xml" contains descriptions for all the data for the "sheet1" worksheet. However, it is not a straight dump of all the data, as I will discuss below.

INVESTIGATING STYLES

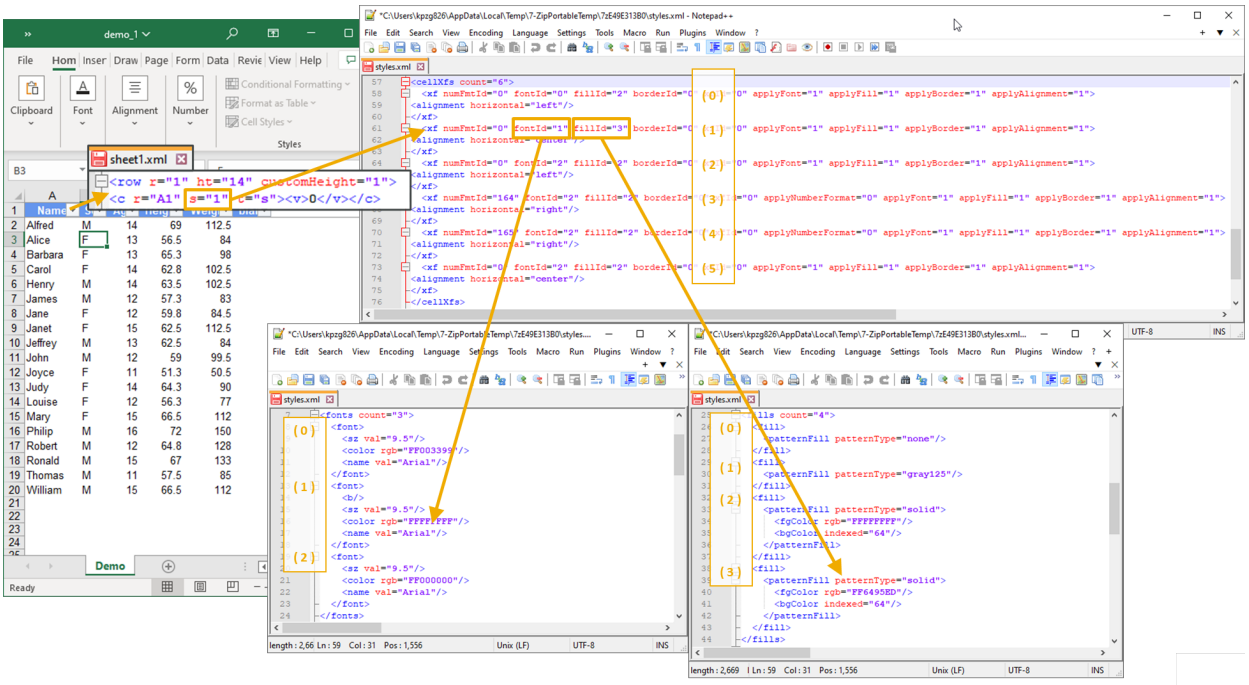


Figure 2 - Examining style definitions in Excel.

Figure 2 (above), shows a simple print of SASHELP.CLASS rendered as an Excel file using ODS EXCEL.
Cell A1 contains the column header "Name" and is white text on a blue background.
Within Excel, this cell is recorded in the **sheet1.xml** file under row 1, column 1. Within this XML entry the cell is also given a style reference, in this case `s="1"`.

If we look in **styles.xml**, style 1 (note that the first entry is labelled 0, the second is 1, etc) has a reference for the font used (`fontId="1"`) and the type of fill used (`fillId="3"`).

Further down the XML we can see that font definition 1 has a colour of “FFFFFFFF” which is white, and fill definition 3 has a colour of “FF6495ED” (which translates to “cornflower blue” [1]).

SHARED STRINGS AND UNIQUE VALUES

Within the **sheet1.xml**, the definition for each cell informs Excel the style associated with the cell, and the value it contains. However, sometimes the value is a reference to a “shared string”.

This allows Excel to store a data point once, and reuse it multiple times, thus saving space in large spreadsheets.

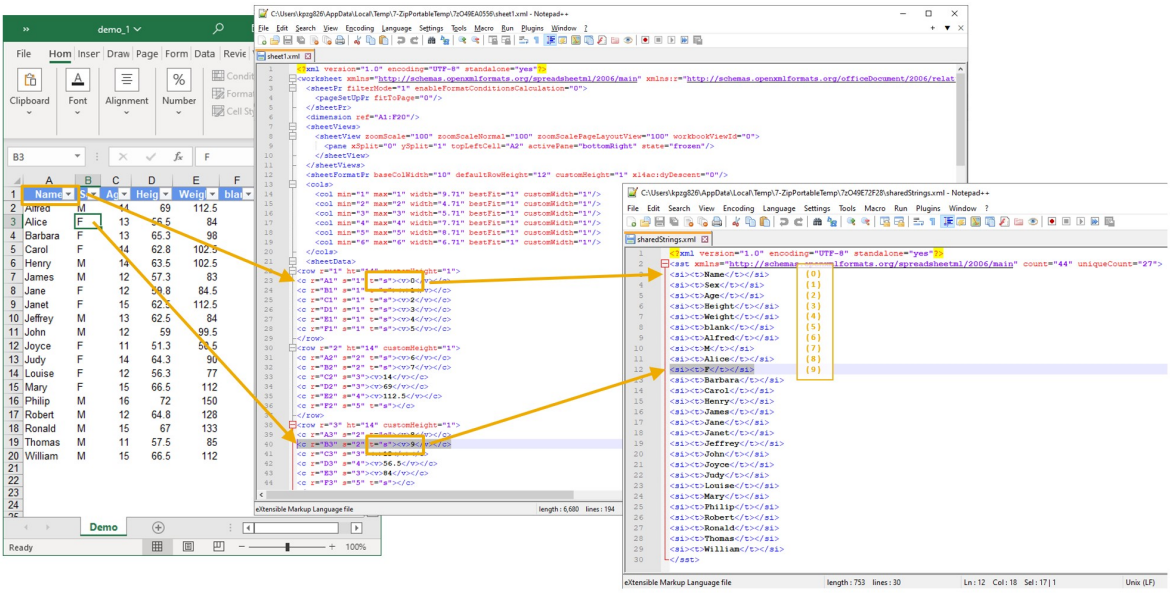


Figure 3 - Shared Strings in action

In Figure 3 (above) we can see that in **sheet1.**xml cell A1 has a type set as `t="s"` (meaning shared), and given a value (using the `<v>` tags) of 0. Cross referencing this in **sharedStrings.xml** we can see that entry 0 has the value of “Name”.
Likewise, cell B3 has a value definition of 9, which maps to a value of “F”.

TRIAL AND ERROR

By editing this simple Excel file, and then examining the changes to the XML afterwards, I discovered how many different features of Excel are defined.

For example, one of the features I was keen to include in the COMET spreadsheet was having a worksheet *partially* locked for editing – allowing a handful of columns to still be written to. The changes to the XML after setting the protection status for column A manually in Excel are shown in Figure 4 (below).

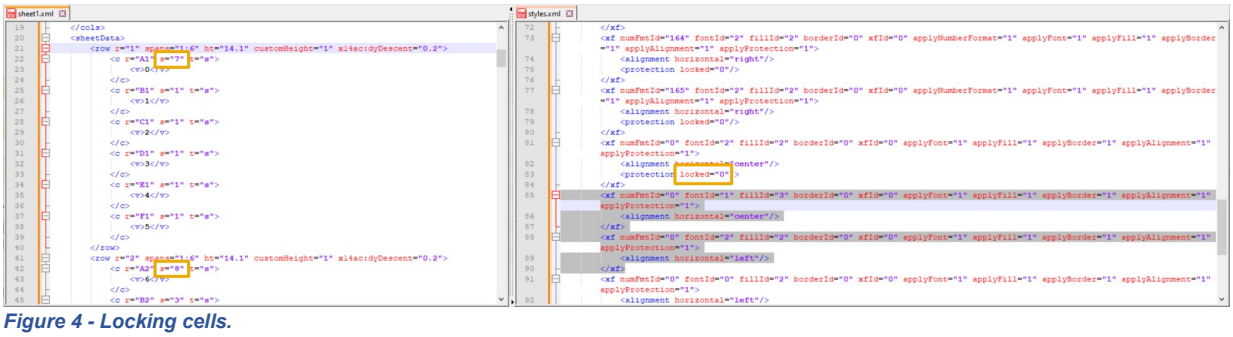


Figure 4 - Locking cells.

In **sheet1.xml**, shown on the left, the header for column A is defined as style 7, and the rest of the cells in that column are style 8. The right panel shows **styles.xml** and the corresponding styles are shown highlighted in grey. What is key here is that styles 7 and 8 DO NOT contain the `"locked=0"` protection option (so by default a cell **is** locked when sheet protection is turned on, which is done in a separate `<sheetProtection>` tag earlier in **sheet1.xml**).

Another feature I was interested in was cell validation, which allows some cells to have a drop-down selector that only allows a predetermined set of values. This would go a long way to helping to maintain data integrity during a concomitant medication review! I know from past experience that something as simple as a "Y" in a Y/N column could have many possible entries, such as: Y, y, yes, yes, etc…

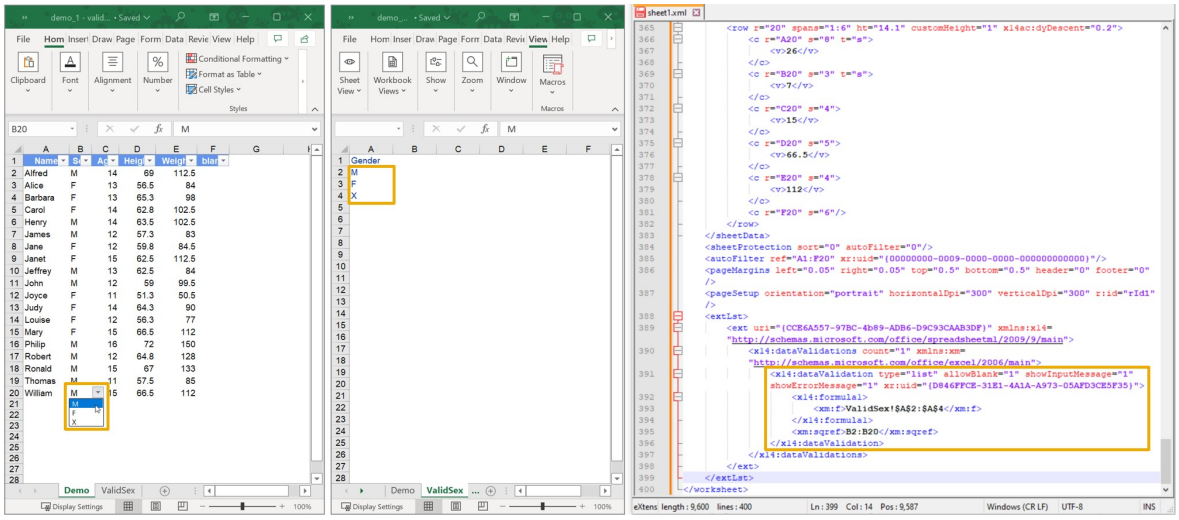


Figure 5 - Cell validation

Figure 5 (above) shows the XML behind a cell validation. A separate worksheet holds the valid values, and the XML for the sheet that is to contain the validation (in this case the tab "Demo" which Excel defines in **sheet1.xml**) has a reference to the range of possible values within a `<x14:dataValidation>` section at the bottom (in this example rows A2 to A4 from the "ValidSex" sheet)

EDITING THE XML

Other elements I wanted to introduce to the Excel file were the shading of certain columns to indicate they are editable, individual cell highlighting to indicate a value changed from the previous review and "tooltips" that would show the previous values for these changed data points when the mouse is hovered over the cell. All of these are possible in SAS using various techniques within the ODS EXCEL and PROC REPORT statements.

However, adding cell validation and individual cell protection would require physically changing the XML within the XLSX file, based on my investigations shown above.

Fortunately, SAS can read in XML files easily, as they are simple flat text files. SAS can also manipulate ZIP files using the FILENAME statement with the ZIP option.

This, ironically, uncovered a significant problem– that of XML data integrity. When manipulating Excel XML data, any syntax errors immediately render the whole XLSX file corrupt when you try to open it. This is incredibly sensitive, and even having completely valid options within XML tags just being in the wrong order will trigger a corruption message when opening the file. Therefore, this needed careful attention.

The changes that I wanted to make were relatively few, and the areas of the XML that required these changes were straightforward to discover. To find the changes I would go through the following process:

1) Save the XML files from a simple Excel file (like the SASHELP.CLASS printout shown earlier) out to area of my PC.
2) Open the Excel file, make the update and then save it with a new filename.
3) Compare the XMLs from the updated file against the original to see what changed.
4) Take the COMET Excel file from SAS and manually edit the XML file with the same changes.
5) Open the file in Excel (and see if the file is corrupted!).
6) Check if the changes were implemented as expected.

Steps 4-6 inevitably failed the first time, and it took a little while to get to know some of the nuances and implied rules that are the difference between a valid and a corrupt XML (e.g., a space may be required in a particular place, or the order of certain options needs to be swapped around, etc).

Once I was confident that I had the code required to implement a feature, I would add it to one of my XML editing programs. This program would in the XML file (via the ZIP engine) as a simple flat text file with a single text variable (as opposed to reading the file as an XML and attempting to process it that way), and then a series of

flagging variables were used to identify the place where the new code needs to be injected, before going on to inject this XML code fragment and then saving the XML back out to the Excel file again.
This involved a lot of trial and error, and many corrupt Excel files until the features I wanted to add were stable and repeatable.

THE FINISHED PRODUCT

Once the routines to edit the Excel file were in place, the whole COMET process could be tied together into a user-facing macro.

Figure 6 (below) shows a very high-level view of the COMET workflow.

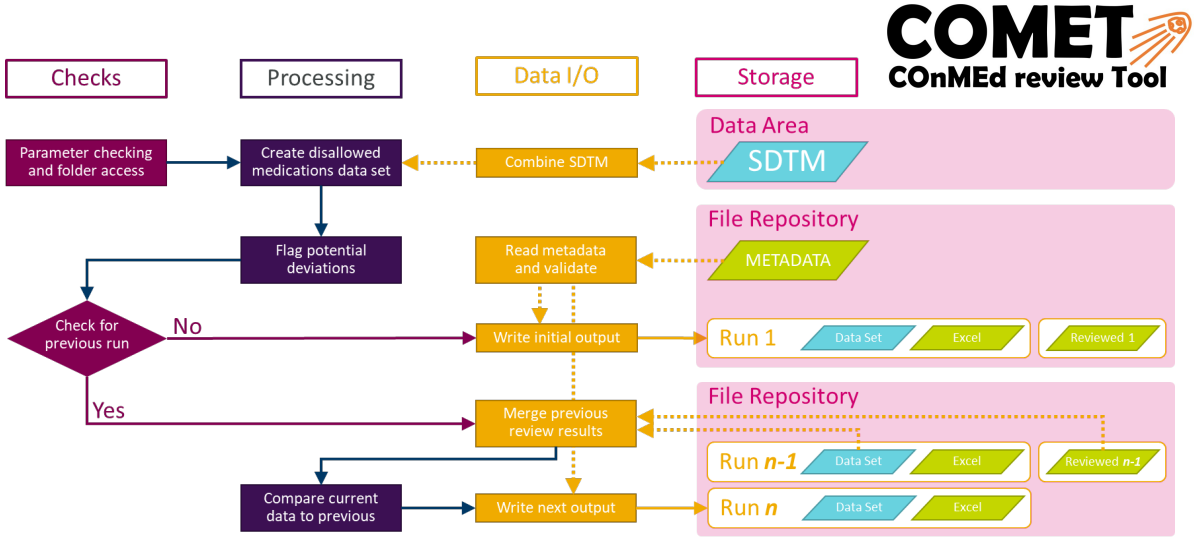


Figure 6 - COMET Workflow

The COMET macro calls a program to create the disallowed medications (DISMED) data set. It then checks to see if COMET has been run before. If it hasn't, this data set will get written out to the file system as the first COMET Excel file.
If it has been run before, the comments and flags from the previous run are added to the current data set, and specified variables are also compared, and any differences noted (for use with the tooltip/flyover mechanic).

The final Excel file is then created, with shading and layout by ODS EXCEL and tool-tips, cell locking, and data integrity routines added via XML editing.

CONCLUSION

The initial COMET codebase was quite straightforward to create – it was after all only a system for reading in and comparing data on a cyclical basis. However, it became much more complicated once the team began to review the initial output and suggest improvements and extra features.

As these features are not native to SAS (via ODS EXCEL) it would have been easy to have said "no" to the feature requests. However, because of the way Excel now structures its files and the fact that this is readable and editable by SAS, a lot of potential enhancements were unlocked.
I was fortunate to have the bandwidth to explore these potential enhancements, and although the process was both frustrating and time consuming it was ultimately a very rewarding experience – both for the team (they got their requirements granted) and personally (I learnt a whole new branch of SAS/Excel working!).

COMET was rolled out at the end of 2022 and is currently in use by several study teams, with more teams coming onboard regularly. Other than the occasional bug fix (including a recent one that has triggered a confirmed bug report to SAS themselves!) the code hasn't changed significantly since then, and the system has proven to be quite stable overall.

The COMET tool has shown an 80% reduction in the volume of work involved in reviewing concomitant medications compared to the previous manual process, and the ability to have frequent incremental reviews has significantly reduced resource bottlenecks around deliverables. In addition, COMET has minimised the risk of errors occurring in the programming and has enhanced the quality of the review via the data integrity constraints that were introduced.

For future developments, Look To The Sky!

REFERENCES

1. From https://www.schemecolor.com/sample?getcolor=6495ED

CONTACT INFORMATION

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

Nev Cope
AstraZeneca
Academy House
136 Hills Road
Cambridge, CB2 8PA
UK

T: +44 (0) 1223393484
E: neville.cope@astrazeneca.com