11 Pre-editing tools

Here are a few macros, some of which you might want to use before you actually start the sentence-by-sentence reading of your text. The idea is that if you can get macros to change a lot of the obvious and repetitious things, you will be better able to concentrate on the really skilled part of the job – making sure that the text says what it is meant to say.

The most powerful single macro is *FRedit*, and there's only a brief introduction to it in this book – it has its own documentation. Then this section also covers ways in which you can add typesetting codes to your book (<A>, <B>, <C> etc – although this can also be done with *FRedit*), ways of combining and dividing up the files that form your book, a macro to create a list of all the acronyms in a file, macros to pull all the tables and/or figures out into a separate file, and other macros to 'do things with' tables, frames and textboxes.

These are then followed by macros that 'do things globally' with footnotes and endnotes, bookmarks, comments and styles. And there's a miscellany of other things that you might want to do before you start to read the script.

FRedit hint – switch track changes on

On the final (long) chapter of my last job, I forgot to switch on track changes before running *FRedit*, and I didn't notice until I got well into the editing. That caused me a *lot* of hassle because important changes went untracked. Arrgghhh!

Not any more! *FRedit* has a facility such that I can add a line at the top of the *FRedit* list:

| Track = yes

If track changes is off, it opens a window to warn me that it's off, and it won't let *FRedit* go ahead until track changes is switched on.

FRedit hint – using FRedit on multiple selections

(Video: https://youtu.be/oDst93YCLC0)

I've discovered a new Word feature on the Home tab: *Home—Editing—Select*. This throws up a menu including the action: *Select All Text With Similar Formatting (No Data)*. Not a snappy title, but if you click, say, in the heading above, and enact this feature, then it and ***all*** the similar level headings will be selected, i.e. all selected at the same time.

You could then, say, apply a different font colour.

Unfortunately, if you get ambitious and try to run FRedit to make changes to all those selected headings, it doesn't work – it only works on the final selection in the document.

So, I added a feature to FRedit (see the FRedit Instruction file for details) so that you could make your multiple selections, then use the Font window to apply a strikethrough to those selections, and FRedit would then work the opposite way round from normal, i.e. it would ONLY edit text that had a strikethrough, as opposed to NOT editing any text that had a strikethrough.

It works, but it's a bit tortuous! (But see below)

Instead, this macro will do the following, after you have made your multiple selection:

1) Apply ST to the whole text
2) Remove ST from the multiple selection
3) Run FRedit
4) Remove ST from the whole text

```
Sub FReditSelect()
```

This macro does the same as the previous macro, but all you have to do is open your FRedit list, click in a heading (assuming it's the headings you want to apply the FRedit list to) and run the macro. It does automatically what you had to do manually above, i.e. within the macro, it does the *Select All Text With Similar Formatting*.

It's called 'FReditHeadingsOnly' because headings seemed the most likely target for FRedit-ting 'all similarly formatted text', but it can be used for any text. For example, you could edit only the text that's in Normal style.

```
Sub FReditHeadingsOnly()
```

FRedit hint – checking your FRedit list

(Video:youtu.be/Z7cjf446JWM)

It's all too easy when creating a FRedit list to introduce unintended styles and font size/name changes. They might not be obvious to the naked eye, so this macro checks the list for any funny styles and font sizes/names. They may be deliberate on your part, which is fine, but if so just click and move on.

The macro also warns you about lines that don't have a pad character. That, too, can be deliberate, if you're using two-line F&Rs for style changes, but again, at least you're prompted to check.

The macro starts its checking from the current line, and stops at the first possible problem.

```
Sub FReditListChecker()
```

Scripted F&R – simplified version of FRedit

(It's a bit like a *FRedit* trainer – a flight simulator for a trainee pilot, though you can actually use it to do some useful jobs.)

For this simplified version of *FRedit*, all you need is a list like this, of the words ***you*** want to change:

favor|favour
Favor|Favour
color|colour
Color|Colour
center|centre
Center|Centre

That has to be in one Word file, open on screen, then open the file on which to make these global changes; then run the macro.

That's it! That's all you need to know.

(However, if you're feeling brave and want to try other things, keep going, and I'll drop in some learning points [LP].)

Copy the following list and paste it into your 'list' file. Then find an old file of text to "play with" and then try to change it by using this list and see what happens:

that|that
which|which
the|the

[LP: Changing something with the ***same thing*** has no effect, except to add a highlight.]

Then you could try and see what this does:

¬the|the

Can you see what that did? It did something that the|the ***didn't*** do.

[LP: The "¬" character says "change text regardless of its case; upper or lower".]

Still feeling brave? OK, try copy-and-pasting this list:

| Double space to single space
^32^32|^32

| spaced hyphen to spaced en dash
- | ^=^32

| Double return to single return
^p^p|^p

[LP: The "^32" is exactly equivalent to " ", but it's easier to see! (32 is the ASCII code for a space.)]
[LP: Any line starting with a vertical bar is ignored by the macro.]
[LP: The "^p" is one of many Word codes. For more, click Ctrl-H: Find and Replace — More >> — Special.]

Feeling positively heroic? Then try this:

| A wildcard search for number ranges: hyphen to en dash
~([0-9])-([0-9])|\1^=\2

[LP: The "~" says the rest of this line is a wildcard F&R.]
[LP: If you find wildcards a bit scary, don't worry, just copy other people's – see the *FRedit* library.]

One final thing to try, but this time, switch track changes ON before running the macro:

| Double space to single space
~~| ^32~~

| spaced hyphen to spaced en dash
- | ^=^32

| Double return to single return
^p^p|^p

| A wildcard search for number ranges: hyphen to en dash
~~~([0-9])-([0-9])|\1^=\2~~

[LP: Any line with the single strikethrough attribute added will ***not*** be tracked, even if track changes is ON.]

```
Sub FReditSimple()
```

Scripted F&R – a simple FRedit-like tool

(Video:youtu.be/DfAGD9RCpNQ)

This is an even simpler system than *FReditSimple*, and is aimed at doing a useful job, very quickly. However, it's ***not*** aimed at training you to use *FRedit*, because it has a different way of working.

At the end of the file you want to work on, type a hash (#) followed by two or three blank lines. Select from the hash to the end of the file and make sure it is in Normal style and pure text. (I use the macro *NormaliseText*, but I assume there's a way of doing it off the toolbar, though I can't see where, sorry!)
~~~

On lines following the hash, you could do things like the following. See if you can guess what each will do, when I run the macro.

#
toolbar
FRedit
et al
-ise
-isi-
-our
-ment

The first highlights every occurrence of ‘toolbar’, the next two change the words to italic, but note that the third line will ***not*** do this: “The parap*et al*igns with the fe*et al*ong the edge.” it ***only*** applies the attribute to ‘et al’ as a separate pair of words.

Then, it’s highlighting words ***ending*** in ‘ise’, and ‘isi’ anywhere in the middle of a word, and ‘our’ and ‘ment’ at the ends of words.

Other features you can use are bold and font size, and they can be combined:

hello

and I’ve also added strikethrough:

bonjour

so that would stop ‘bonjour’ being thrown up as a spelling error by my spelling macros.

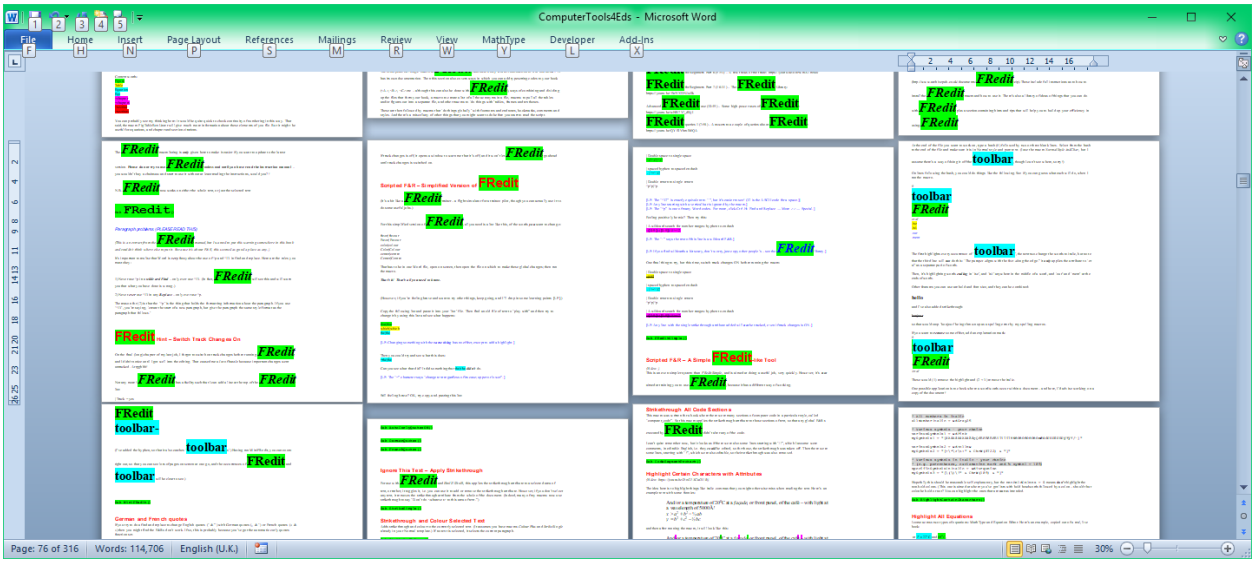
If you want to ***remove*** some effect, add an exclamation mark:
#
!toolbar
!FRedit
!et al

These would (1) remove the highlight and (2 + 3) remove the italic.

One possible application is to check where a word/words occur within a document – and here, I’d advise working on a copy of the document!

FRedit
toolbar-

(I’ve added the hyphen, so that it also catches ‘toolbars’.) Having run MiniFRedit, you can zoom right out, so that you can see lots of pages on screen at one go, and the occurrences of FRedit and toolbar will be clear to see.)



Programmer's extra: If you want to add other attributes, such as superscript, it's not difficult. Add a line to read the attribute into this list:

```
' Check the attributes on this item
myBold = tst.Font.Bold
myItal = tst.Font.Italic
mySize = tst.Font.Size
myStrike = tst.Font.StrikeThrough
mySuper = tst.Font.Superscript
```

and the add an item to apply it, such as:

```
If myStrike Then
  .Replacement.Font.StrikeThrough = True
  If doUndo Then .Replacement.Font.StrikeThrough = False
End If

If mySuper Then
  .Replacement.Font.Superscript= True
  If doUndo Then .Replacement.Font.Superscript= False
End If
```

(In fact, I thought I might as well add this anyway! So it now also does sub/superscript and underline!)

Sub MiniFRedit()

IZ to IS spelling and vice versa

(See video: youtu.be/SXmAJrUCZ_I)

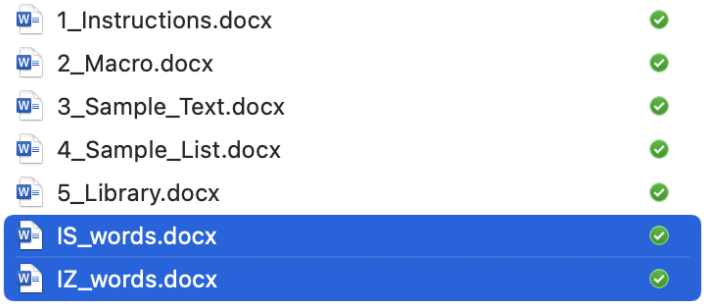
The main instructions are different for Mac and PC, so here's Mac version first:

(N.B. Late addition: if you run these macros from within *FRedit*, using, say, `DoMacro|IStoIZ`, you have to be on hand to click Yes, when it asks if you want to edit the text. I've now added an option at the beginning of the macro to avoid this. This is especially useful if you're using *MultiFileFRedit*. To stop the prompt being generated each time *IStoIZ* is run then change `promptForConfirmation = True` into `promptForConfirmation = False`.)

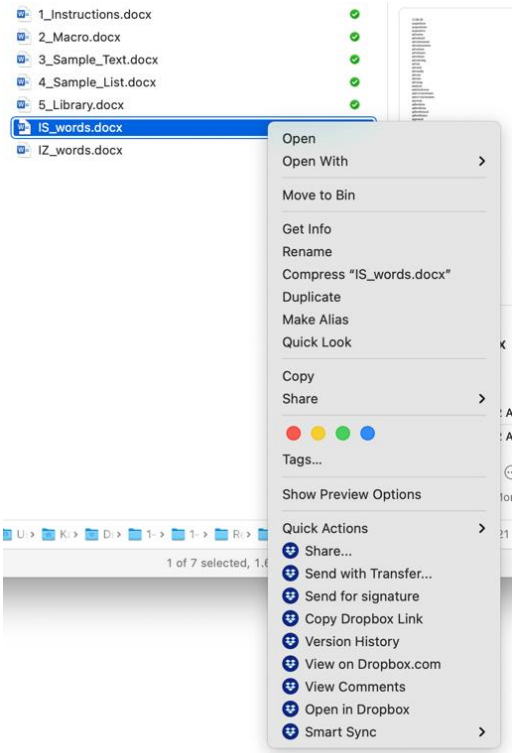
The following two macros allow you either to highlight words that need changing or to actually change them. When you run the macro, it asks which you want to do.

MAC VERSION

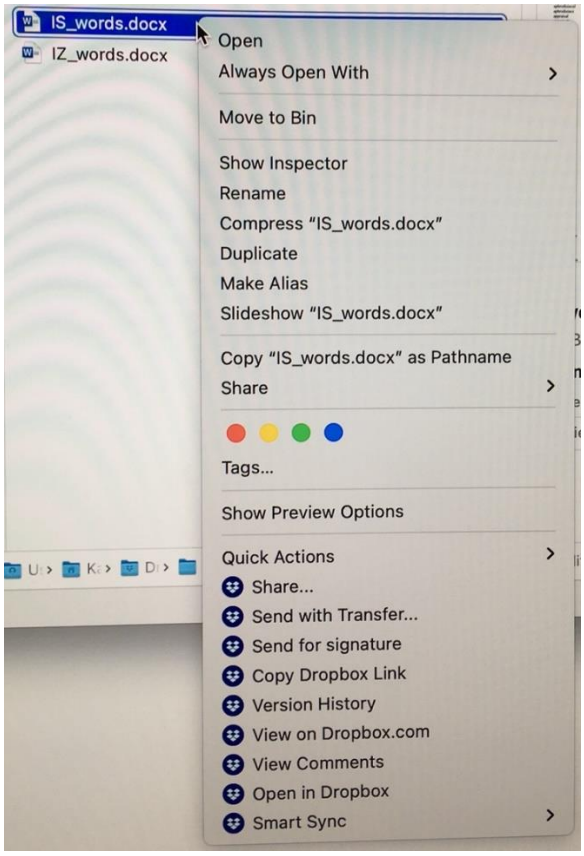
The lists of exceptions need to be held in files (one for each macro) called ‘IS_words’ and ‘IZ_words’. (And for the IZIScount macro, you need both.) You can set up each macro so that it automatically loads the relevant file from your hard disc, but the macro needs to know where on your computer to find it. You therefore have to put the full filename of the exception file into each macro (and both into the IZIScount macro). To do this, navigate in a Finder window to the folder (directory) where these two files are held:



To obtain the pathname, first right-click the file:



Then, with this dropdown box visible, depress and hold the option key. ‘Copy’ will become ‘Copy “[filename]” as Pathname’. Still holding down the option key, click ‘Copy “[filename]” as Pathname’.



The pathname will be copied to your clipboard. Paste it into the line near the beginning of the macro following the line 'Address where … exceptions file is held', replacing the pathname that is there at the moment. Suppose the pathname of my exceptions file is:

```
C:\VirtualAcorn\VirtualRPC-SA\HardDisc4\MyFiles2\WIP
```

So now the line near the beginning of the macro:

```
mySFile = "C:\Documents and Settings\Paul\My Documents\IS_words.docx"
```

has to be changed to (shaded so you can see what I've added):

```
mySFile = "/Users/Paul/My Documents/Macro stuff/IS_words.docx"
```

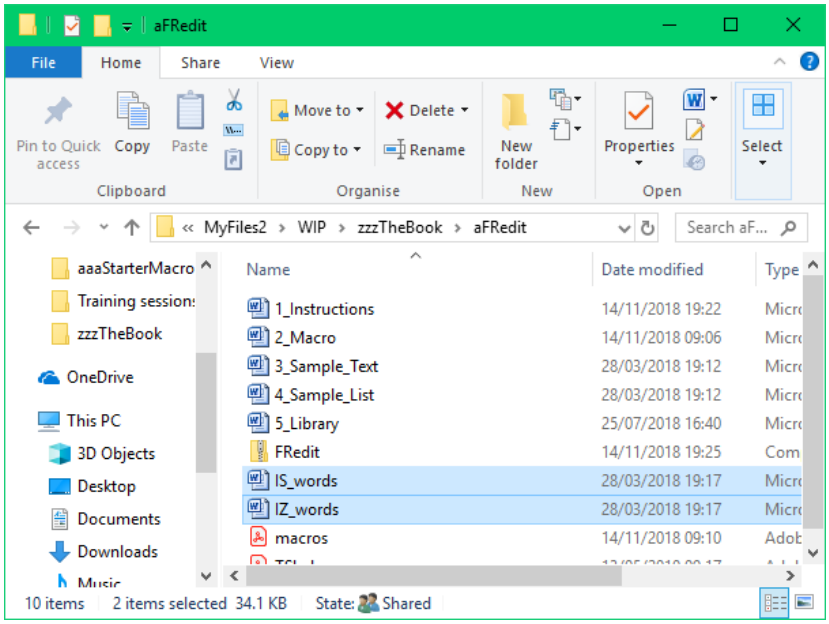
(or whatever it is on your computer).

The two lists of exceptions are among the IS/IZ macros in the TheMacros file.

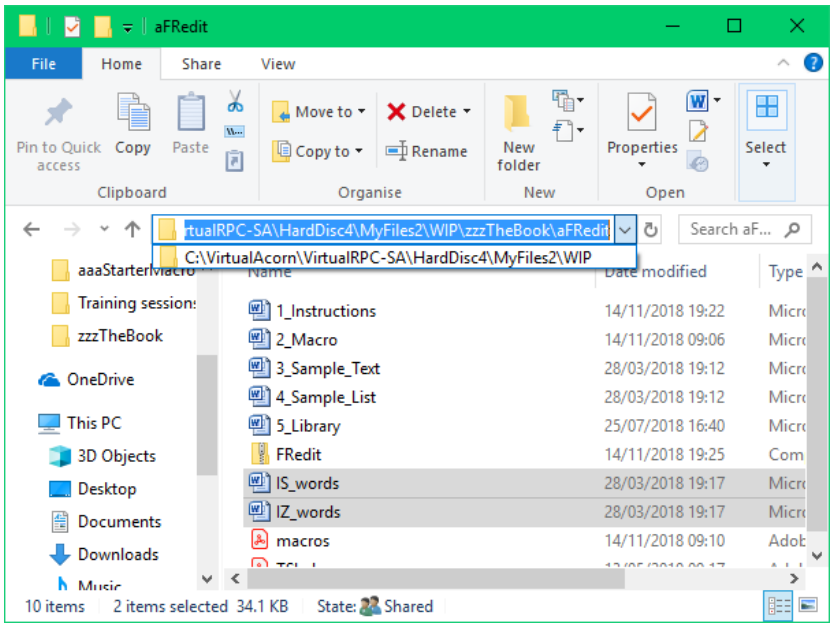
(Now jump to after the PC version for the final bit...)

PC VERSION

The lists of exceptions need to be held in files (one for each macro) called 'IS_words' and 'IZ_words'. (And for the IZIScount macro, you need both.) You can set up each macro so that it automatically loads the relevant file from your hard disc, but the macro needs to know where on your computer to find it. You therefore have to put the full filename of the exception file into each macro (and both into the IZIScount macro). To do this, navigate to the folder (directory) where these two files are held:



If you click on the down menu arrow to the right of the line showing the string of folder names, the full path name appears:



Click Ctrl-C to copy this and add it into the line at the beginning of the macro. Suppose mine is:

```
C:\VirtualAcorn\VirtualRPC-SA\HardDisc4\MyFiles2\WIP
```

So now the line at the beginning of the macro:

```
myFile = "C:\Documents and Settings\Paul\My Documents\IS_words.docx"
```

has to be changed to (shaded so you can see what I've added):

```
myFile = "C:\VirtualAcorn\VirtualRPC-SA\HardDisc4\MyFiles2\WIP\IS_words.docx"
```

(or whatever it is on your computer).

Finally for both Mac and PC

The two lists of exceptions are on the website: IS_words IZ_words.

IZ words

IS words

If you discover other words that are exceptions, please email them to me so that I can update these as central lists. I have dated the lists so that you can check if you've got the latest version. I've put a yellow highlight on the proper nouns, because they may look a little funny; the macro requires the words to be in lowercase.

N.B. You don't need words like 'disabled' and 'misapprehension' in the list (and there are a ***lot*** of them!) because the macro ignores 'isa' if it's too near the beginning of the word.

The *IStoIZ* macro takes account of the fact that, in ***UK*** English, analyse, catalyse, paralyse and hydrolyse keep the 'ys' form, but not in ***US*** English. It senses what the main language of the text is, and acts accordingly.

You can select the highlight colour at the beginning of the macro:

```
highlightColour = wdTurquoise
```

And/or you can select the font colour at the beginning of the macro:

```
textColour = wdTurquoise
```

If you don't want the is/iz words changing in certain parts of the file (e.g. quotations and/or references lists) you can 'protect' the text (a) by using the strikethrough font feature (this is the same feature as is used with *FRedit*) and/or (b) by specifying the style names using the line near the beginning of the macro: `nonoStyles = "Display Quote,References List"`, so just include your particular style name(s) in between the quotes.

You don't have to have the is/iz words both track-changed ***and*** highlighted. However, if you ***do*** want them both track-changed and highlighted, change the option line to:

```
bothTCandHighlight = True
```

The text of 'IZ_words' file is in among the macros on the website.

IZwords

Similarly for the 'IS_words' file:

ISwords

You will have to put them each in a Word file and save them with the names 'IZ_words' and 'IS_words'.

The actual macros are:

Sub IZtoIS()

Sub IStoIZ()

German and French quotes

If you try to do a find and replace to change English quotes (" & ") with German quotes („ & ") or French quotes (« & ») then you might find the F&Rs don't work. If so, this is probably because you've got the automatic curly quotes function set.

Three ways round the problem:

1) (Obviously, you could) turn off curly quotes, do the F&Rs, turn curly quotes back on.

2) Run the appropriate French or German macro below.

3) Use *FRedit*:

| German quotes
DoMacro|*AutoCurlyQuotesOFF*
“|„
”|„
DoMacro|*AutoCurlyQuotesON*

Or...

| French quotes
DoMacro|*AutoCurlyQuotesOFF*
“|«
”|»
DoMacro|*AutoCurlyQuotesON*

But number 3) assumes you’ve got the two macros loaded:

```
Sub AutoCurlyQuotesOFF()
```

```
Sub AutoCurlyQuotesON()
```

```
Sub GermanQuotes()
```

```
Sub FrenchQuotes()
```

Ignore this text – apply strikethrough

(Video: https://youtu.be/AGyrZbgoTD0)

For use with *FRedit* and *IStoIZ*/*IZtoIS*, for example, (indeed, many of my macros now use strikethrough to say “Don’t do <whatever> to this area of text”) this applies the strikethrough attribute to a selected area of text, or rather, it toggles it, i.e. you can use it to add or remove the strikethrough attribute. However, if you don’t select any text, it removes the strikethrough attribute from the whole of the current paragraph.

```
Sub StrikeSingle()
```

Strikethrough and colour selected text

Adds strikethrough and colour to the currently selected text. (It assumes you have macros *ColourPlus* and *StrikeSingle* already in your Normal template.) If no text is selected, it selects the current paragraph.

```
Sub StrikeAndColour()
```

Strikethrough all URLs

This macro applies the strikethrough attribute to all URLs in the text. The idea here is to ensure that no URLs are edited by *FRedit* etc.

N.B. If there are URLs it misses, it will be because it contains a character that I haven’t thought of. Please let me know, and I’ll add it to the list at the beginning of the macro.

```
Sub StrikeThroughAllURLs()
```

Strikethrough all equations

This macro applies the strikethrough attribute to all equations in the text. The idea here is to ensure that no equations are edited by *FRedit* etc.

```
Sub EquationsStrikeThroughAll()
```

Strikethrough all code sections

This macro was written for a book where there were many sections of computer code in a particular style, called "computer_code". So this macro applies the strikethrough attribute to those sections of text, so that any global F&Rs executed by FRedit didn't alter any of the code.

I can't quite remember now, but it looks as if there were also some lines starting with "//", which I assume were comments, in editable English, i.e. they ***could*** be edited, so for those, the strikethrough was taken off. Then there were some lines, starting with "!", which were also editable, so their strikethrough was also removed.

```
Sub CodeSegmentProtect()
```

Highlight certain characters with attributes

(Video: youtu.be/DnG1XCuOUlk)

The idea here is to highlight things like italic commas that you might otherwise miss when reading the text. Here's an example text with some funnies:

> And/or a temperature of 20°C at a *façade*, or front panel, of the café – with light at a wavelength of 5000Å*!*
> $x > a^2 + b^2 – ½ab$
> $y = b^2 + c^2 – ½bc$

and then after running the macro, it will look like this:

> And/or a temperature of 20°C at a *façade*, or front panel, of the café – with light at a wavelength of 5000Å*!*
> $x > a^2 + b^2 – ½ab$
> $y = b^2 + c^2 – ½bc$

If you want to highlight specific characters regardless of whether they are bold/italic, etc then you can do it easily in FRedit, e.g. the diacritics:

~[áÁàÀâÂäÄãÃåÅçÇéÉèÈêÊëÉíÍìÌîÎñÑóÓòÒôÔöÖõÕøØßúÚùÙûÛüÜýÝÿŸ]|^&

But to highlight, say, **only** the *italic* superscripted numbers, as in those equations, or the italic comma after 'façade', then it's not so easy in FRedit, hence this macro (although it also

You can set up the macro (and you could have, say, two different version of the macro, with slightly different names and for different purposes/clients) using the variables at the beginning of the macro:

```
superscriptZeros = wdBrightGreen
italicCommas = wdBrightGreen
boldColons = wdPink
notBoldColons = 0
subSuperscriptSpace = wdGray50

' Just super and subscript numbers in italic
subSuperscriptNumberItalic = wdYellow
```

```
' All numbers in italic
allNumberItalic = wdGray25

' Various symbols - your choice
variousSymbols1 = wdPink
mySymbols1 = "[áÁàÀâÂäÄãÃåÅçÇéÉèÈêÊëËíÍìÌîÎñÑóÓòÒôÔöÖõÕøØßúÚùÙûÛüÜýÝÿŸ/-]"

variousSymbols2 = wdYellow
mySymbols2 = "[=\*\>\<+" & ChrW(8722) & "]"

' Various symbols in italic - your choice
' (e.g. parentheses, exclamation mark and ½ symbol = 189)
specificSymbolsInItalic = wdTurquoise
mySymbols3 = "[\(\)\!" & ChrW(189) & "]"
```

Hopefully this should be reasonable self-explanatory, but the `notBoldColons = 0` means **don't** highlight the non-bold colons. (This one is aimed at where you've got lists with bold headwords followed by a colon – should that colon be bold or not? You can highlight the ones that are **not** as intended.

```
Sub HighlightCertainCharacters()
```

Highlight all italic text

This a simple macro to do a find and replace simply to add a highlight to all italic text.

```
Sub HighlightAllItalic()
```

Highlight all equations

I come across two types of equations: MathType and Equation Editor. Here's an example, copied out of a real, live book:

at $T = 22^{\circ}\mathrm{C}$ and $60^{o}C$,

The first is MathType and the second is Equation Editor. (And yes, well spotted, both use superscripted letter-o's, and yes, italic in the second case.)

The highlighting you see above is what this macro adds. The point is that, without the highlight it looks like this:

at $T = 22^{\circ}\mathrm{C}$ and $60^{o}C$,

(a) it's not clear that the two use the different equation formats
(b) it's less obvious that there's no space in front of 'and'. (But I don't know if that matters – do typesetting packages like InDesign automatically pick up on that sort of thing?)

(In case it's useful, I've added the option to highlight the other method of creating (bits of) equations: Symbol font. And don't forget that Symbol font Greek characters can be changed to Unicode characters by using *FRedit* – see the *FRedit* library.)

```
Sub EquationsHighlightAll()
```

Highlight all 'equations' that are actually now bitmaps

Unfortunately, sometimes 'equations' do lose their editableness and become bitmap images.

But before you freak out totally, just check the filetype of the file. Sometimes, I've been able to rescue uneditable equations, where someone has saved a .docx file as the older .doc filetype, which doesn't support equations in the same way. So sometimes, if you do a Save As, and change the filetype from 'Word 97-2003 Document' to 'Word Document' the equations will jump back to life. Phew!

If not then this macro will highlight all the 'equations' that are now actually uneditable bitmap images.

Space out MathType equations in running text

If, having highlighted the MathType equations, you can see that some of them need spaces adding, you can add them manually, of course, but this macro checks all the MathType equations in the current paragraph and, where necessary, adds spaces.

In the example, "if the temperature $T_1 = 22°C$ or $T_2 = 29°C$, at the surface ($Q = 295K$)", only the first one needs a space, as the second is next to punctuation, and the third is in parentheses. The macro knows when to add a space and when not.

```
Sub SpaceEquationsInPara()
```

Space out MathType equations in whole text

This does the whole file at one go. (I forgot I'd done this when I wrote the macro above!)

```
Sub EquationSpacer()
```

Convert all Equation Editor items to text

In one job, the author used MathType for the actual equations, but Equation Editor in the running text for, e.g. "where *t* is the length of time and *k* is the coefficient of whatever". So, rightly or wrongly, I decided to convert all the Equation Editor items to pure text. This macro does that, highlightling them, as it goes; they come out in roman, so they need to be italicised.

```
Sub EquationsConvertAll()
```

Mark all quotations

(Video: youtu.be/PB0hXA_1tRo)

This is a multi-purpose macro that finds all the text in quotation marks (single and/or double) and all displayed text (i.e. text that has a left margin indent), and marks it by using any or all of:

a) strikethrough (or double strikethrough)
b) highlighting (your choice of colour)
c) font colour (your choice of colour)

The original reason for the macro was that if the quotations are struck through, then *FRedit* and *IStoIZ*/*IZtoIS* will ***not*** then make any changes to them. But then I added the font colouration because, once you've used *FRedit* and *IStoIZ*/*IZtoIS*, you'll probably want to remove the strikethrough so that you can more easily read the text. If those quotations are also in a different font colour then you will be able to see which bits of the text have indeed been protected from the effects of global changes.

Then the other major application of this macro is that you can get it to ***only*** mark long quotations, i.e. more than a certain number of words. This is to enable you to see which quotations ought to be displayed rather than being left

inline. This is why I added the option to highlight the quotes because now, as you read through the text, and are alerted to which are the long quotes, and you can use the macro *DisplayQuote*. So you just click somewhere in the highlighted text, and run *DisplayQuote* – it will then add carriage returns to make the quotation a separate paragraph, and the delete the quotation marks and (optionally) add whatever formatting and/or style you want for your displayed text.

And then the other thing that this macro can do is to add coding (<DIS> and </DIS>, or whatever), and the latter code can optionally be on the same line or the following line.

All the optional features are set up within the first few lines of the macro, but if you want to use this macro for two distinctly different applications, remember that you can make a copy of the macro, calling it, say QuotationMarker2.

For example, you could have one version of the macro that marked *all* the quotes and then another version that marked just the *long* quotations in a different colour.

N.B. Working out how to tell the difference between a close quote mark and an apostrophe was an interesting challenge. Take for example:

'Aren't the boys' books on the 'phone table?' he said.

compared with...

rounded 'pebbles' scattered individually. It turned out that we had discovered dinosaur eggs – a dinosaurs' nesting ground!

The quotation ('pebbles') *looks* like a plural possessive but isn't, and then it's immediately followed by a plural possessive. Compare that with…

He talked about 'the pebbles' various colours that showed that we had discovered the eggs of dinosaurs,' so it was a nesting ground!

…and if you can see a logical way of telling me the difference between those two, I'd like to hear from you!

Anyway, I've now got the macro to check for a situation like this, and mark it with a font colour:

rounded 'pebbles' scattered individually. It turned out that we had discovered dinosaur eggs – a dinosaurs' nesting ground!

It also sets up the find and replace so that you can search through and check them all.

More importantly, you *will* have problems if the author has got unpaired quotation marks, or if some of the quotation marks are the wrong way round – the macro simply won't cope. These have to be sorted before you run the macro, but here are some suggestions.

For unpaired quotation marks, what I suggest you do is, before running the macro properly on the working file:
– copy the whole text
– create a new blank document
– paste as pure text
– run the macro
– wind the display out to 20% or whatever, so that you can see lots of page at once

If you can then see any large areas of marked text, these are likely to be where the author has missed a close quote mark, so correct these and try again.

One trick to avoid the wrong-way-round quotation mark problem is to do a quick *FRedit* F&R:

'|'
"|"

and Word will correct any back-to-front quotation marks – but it only works if you've got the auto-curly quotes option in Word switched on.

And, of course, the macro relies on there being curly quotes, so it won't work *at all* with straight quotes. But the hint above will also encurl your straight quotation marks for you.

Finally, if the text has footnotes and/or endnotes, the macro will mark them too, unless you tell it not to do so.

Oh, and to remove single strikethrough, use the macro *SingleStrike*. Select all text and run it.

To remove all colour, select the whole text and run ColourMinus. To remove just that colour, and leave any other font colouration intact, I think you can use UnHighlightAndColour.

So here are all the options at the beginning of the macro with a bit of explanation:

```
' Do you want displayed text marked?
markDisplayed = True
```

If you make it False, it will then ignore text with an indented left margin.

```
' Add coding to existing displayed quotes?
addCodes = True
```

Do you want codes adding?

These are the codes…

```
preCode = "<DIS>"
postCode = "</DIS>"
```

Do you want the final code at the end of the final line or at the beginning of the following paragraph?

```
codeOnNextLine = False

' Minimum length of quotes (words)
minLength = 4
```

If you want ***all*** quotations to be marked, regardless of length, set this equal to zero.

```
' Minimum indent of quotes (cm)
minIndent = 0
```

This is helpful if there is text that is indented but that is not displayed text. Hopefully the displayed text is the most indented text (otherwise you're scuppered!). So if the text with a 1.0 cm indent is not displayed text, but the displayed text is, say, 2.0 cm then set this variable to, say:

```
minIndent = 1.1

' Colour the font of quotations
colourFont = True
myColour = wdColorLightBlue
```

True or False decides if this feature is used. Other colours you might want to use include: wdColorRed, wdColorBlue, wdColorPink, wdColorBrightGreen.

```
' Colour of the possible plural possessive problems
possessiveColour = wdColorRed
```

The colour for any possible possessive plural problems.

```
' Add a highlight
highlightText = True
myHighlight = wdTurquoise
```

True or False decides if this feature is used. Other colours you might want to use include: wdYellow, wdTurquoise, wdBrightGreen, wdPink, wdRed, wdGreen, wdDarkYellow, wdGray25, wdGray50.

```
' Strike it through
strikeSingle = True
strikeDouble = False
```

Take your pick, but you can't have both true.

```
' What kind of quote?
doDoubleQuotes = True
doSingleQuotes = True
```

You can mark either single quotes or double quotes or both.

```
' Do you want the notes checked?
doFootnotes = True
doEndnotes = True
```

If there aren't any foot/endnotes it doesn't matter, so you only need to make either of these `False` if you definitely ***don't*** want it to mark the notes.

`Sub QuotationMarker()`

Run a FRedit list

If there's a *FRedit* list that you run regularly then you can set up a macro so that it loads the *FRedit* list, runs *FRedit* and then closes the *FRedit* list. The `listFile =` line sets the full filename of the *FRedit* list, including the file path.

To find the full filename of the *FRedit* list, open the list file and then run this macro:

`Sub FullNameType()`

And here's the *FRedit*-calling macro:

`Sub FReditListRun()`

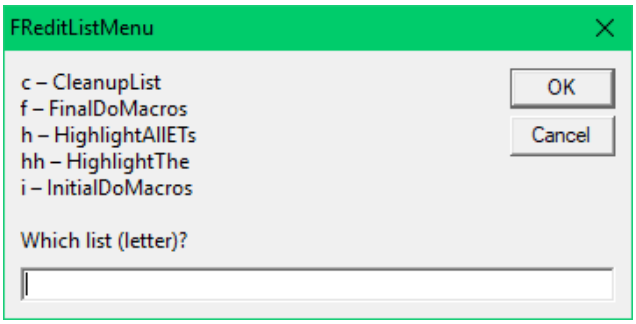
Run a FRedit List from a menu of lists

(Video: youtu.be/1bVduGAFrhU)

If you have a repertoire of different FRedit lists that you want to use, this macro will allow you to select a FRedit list from a menu. Place all your different FRedit lists in one directory (preferably using names with significant initial letter – you'll see why in a minute), and add the address of that directory to the head of the macro. If the directory name is MyLists, the line might be:

```
MyDir = "C:\Documents and Settings\Paul\My Documents\MyLists\"
```

When you run the macro, it scans that directory and creates a menu such as:



Note that it uses the initial letter of the filename as its input device. If there are two FRedit lists with the same initial letter, the second one is given a double letter as input. If you try to have three lists with the same initial letter, it says, "You can't do that!" :-)

So type in the relevant letter(s) and press <Enter> or click OK; it will then load up that FRedit list, run FRedit and then close the FRedit list file.

Sub FReditListMenu()

Multifile FRedit

Now, if you think *FRedit* is dangerous, this is ***ultra-mega-dangerous***! It will open every single file in turn in your file list (see below), edit them all, using the currently open *FRedit* list, and ***resave*** each edited file – so as the edited files have now been saved, there's no undoing what you've just done.

As with my other multifile macros, to get you to identify the folder containing the files, the macro brings up the Open File window, so you navigate to the required folder and then click Cancel. The macro then generates a list of all the files in the folder and asks whether you want to work on all the files in the list. If you say 'Yes', it uses the complete list of files that it has created and works through them all one by one.

If you say you *don't* want to work on all the files, the macro just stops. The list will look then something like this:

C:\Program Files\VirtualAcorn\VirtualRPC-SA\HardDisc4\MyFiles2\WIP
Macro Jobs.doc
Roman cats.doc
Stats.docx

The point is that you can then edit this list, either deleting files you don't want included, or putting a vertical bar ('|') in front of any you want it to ignore.

If you now run the macro again, it recognises that this Word document is a file list and so it proceeds to work through the listed (and not ignored) files, opening each one and, using the current *FRedit* list, editing them and resaving them. ***The terrible deed is done!***

Don't say I didn't warn you! :-)

MultiFileFRedit also gives you the option to accept all existing tracked changes in each file before doing you new set of F&Rs. To do this, set:

```
acceptAllRevs = True
```

This is probably a good idea, anyway, because doing global F&Rs on a file that already has track changes in it is fraught with danger, anyway.

IDEA! If you have a macro that you want to run on a whole list of files, resaving the file after running it, then MultiFileFRedit is your friend. All you would have to do is, if the macro is called *MyMacro*, then create a one-item FRedit list:

DoMacro|MyMacro

and run MultiFileFRedit.

But make sure that *MyMacro* really is a properly functionaing macro, bfore you try to use it on lots of files! (And make sure that you take a back-up copy of the files first – well, you would have done that anyway, wouldn't you?!)

Sub MultiFileFRedit()

Quicker creation of FRedit lists

(Video re spelling: youtu.be/W-JX3P1hZF8)
(Video re hyphenation: youtu.be/olyCyDzCDe8)
(SpellingSuggest gets a mention in video: youtu.be/FVt2ggFXf4A)

If you have a list of words (such as that generated by *SpellingErrorLister*) and you want to 'do something' with each of the various words, to prepare your *FRedit* list then there are several macros to help. Here's a spoof sample list:

bubblechamber
envirnment
evaluted
flangoscope
intermal
managment
mnthly
projcet

Beveley
MTRQ
Pingson
Pongson
Propsing

Correcting the spelling automatically – For words like 'envirnment' and 'evaluted', you can use *SpellingSuggest* which, if run twice, will give you:

envirnment|environment
evaluted|evaluated

Simple as that! It uses Word's spelling checker and accepts the first alternative suggestion, which in fact is usually correct.

(*SpellingSuggest* also appears in the Editing section below, because it also corrects spellings in the body of a paragraph. e.g., if you clck in 'clck' in this sentence, and it will change it to 'click'.)

Correcting spelling manually – If you use *FReditCopy*, it copies the word:

Beveley|Beveley

and then you can add the missing 'r'.

Highlighting – If you want to use *FRedit* to highlight the word, using *FReditSame* twice would produce:

Pingson|^&
Pongson|^&

And once you've got the '^&' in place, you can add various forms of 'highlighting', as per normal with *FRedit*: italic, bold, text colour, coloured highlights etc.)

However...

High speed highlighting – To speed things up even further, you can use *FReditListProcess*. The idea here is to work your way down the list, using *SpellingSuggest* or *FReditCopy* to correct the spellings, but if you just want to 'highlight' a word, just apply your chosen 'highlight' to the single word (italic, bold, text colour, coloured highlight and underline) and carry on down the list. And you don't need to waste time deleting those words that are actually correct.

All you need to do, after doing the 'highlighting' and error corrections, is to go back up the list and use *FReditListProcess*. To illustrate, if this is your 'highlighted' and corrected list:

What does *FReditListProcess* do, and why?

bubblechamber
envirnment|environment
evaluted|evaluated
flangoscope
intermal|internal
managment|management
centr|centre
projcet|project

Beveley
MTRQ
Pingson
Pongson
Propsing|Proposing

then running the macro will produce:

envirnment|environment
evaluted|evaluated
intermal|internal
managment|management
~~centr|^&~~
~<centr>|centre
~~projcet|^&~~
~<projcet>|project

~~~<Beveley>|^&~~
~~~~***~<Pingson>|^&***~~~~
~~~<Pongson>|^&~~
Propsing|Proposing

Actually, I had to run it twice because it starts at the cursor but it stops when it gets to a blank line – useful if you only want to process a section of your list.

So you can see that it has deleted 'bubblechamber', 'flangoscope' and 'MTRQ' because these had no attribute applied to them.

Now, one of the commonest errors I make with *FRedit* is to forget about the danger of word-in-word occurrences, e.g. doing etc|etc. and then ending up with 'Please fetc.h some etc.hings and sketc.hes'. Therefore, for each of 'centr|centre' and 'projcet|project', you will see that it has created two separate F&Rs. This is because, certainly in the case of the first F&R, it could generate false corrections: it would change 'use a centring device', to 'use a centreing device'.

OK, so we could just use:

~<centr>|centre

which is fine, except what this would miss the error, 'the BEVERLEY CENTR was opened...' because wildcard F&R is case sensitive. So there is a second F&R introduced into the list: ~~centr|^&~~ which would therefore highlight 'the BEVERLEY CENTR was opened...' and it would highlight, 'use a centring device'.
Doing this doubles the number of F&Rs, so I decided not to do this on longer words, such as 'intermal|internal'. The limit is set by:

```
minLength = 7
```

It has also added the wildcard codes to the 'copy' items (Beveley, Pingson and Pongson) so that it only searches on whole words – maybe that's a bit OTT, but it makes sure that, say, 'McPongson' doesn't get erroneously highlighted.
~~~

Whether it adds a strikethrough to the copying items (to stop them being track-changed) is set at the beginning of the macro:

```
makeCopyingNotTracked = True
```

Hyphenation changes – If you have a *HyphenAlyse* table of frequencies, you will need to use *FRedit* to make any changes you need. *HyphenationToFRedit* allows you to create the *FRedit* list items at the bottom of the table. Then you can copy them into your main list.

Here's an example:

ni-au. . . . 1		
nitride-based. . . . 1	nitride based. . . . 5	
node-to-ground. . . . 1		
noise-induced. . . . 1		
no-load. . . . 1		
non-coaxial. . . . 1		
non-communication. . . . 1		
non-conducting. . . . 1		
non-device. . . . 3		
non-flammable. . . . 1		
non-isolated. . . . 10		
non-linearity. . . . 2		
non-linear. . . . 10		nonlinear. . . . 1
non-linearity. . . . 2		
off-state. . . . 3	off state. . . . 2	
on-state. . . . 9		

So, clicking in the cell 'nitride based. . . . 5' and running the macro generates:

¬nitride-based|nitride based

Clicking in 'nonlinear. . . . 1' produces

¬non-linear|nonlinear

i.e. you click in the cell of the word-format you want.

However, if you double-click on 'nonlinear', to select it (or just drag-select a bit of the text in that cell), the macro takes that to mean that you want ***both*** options:

¬non linear|nonlinear
¬non-linear|nonlinear

But you might want, say, to standardise on 'off state' and 'on state'. For the former, you just click in the cell that says 'off state. . . . 2' and run the macro, but for 'on state', you just have to click in the empty cell under 'off state. . . . 2', and run the macro.

After it has placed the *FRedit* list item(s) at the end of the file, you can jump back to the line you came from by using my *BookmarkTempFind* macro.

Bringing text from other files – If I have some text in another file (such as *ProperNounAlyse* list) that needs to go into the *FRedit* list, just copy the text, click somewhere (anywhere) in the *FRedit* list, and run *FReditCopyPlus*. It creates a new line, pastes in the text, strips off any frequency data, adds the '|' character and copies the text.

For example, suppose I had spotted an errant name in my *ProperNounAlyse* list:

Brosseau . . . 3
Brousseau . . . 2

If I click select, say, the first line and copy it, I can go into the *FRedit* list and run *FReditCopyPlus*, which would produce:

¬Brosseau|Brosseau

I can either change leave it as it is to just highlight the word, or change either the Find or the Replace word to effect a spelling correction.

Whether the macro adds the strikethrough (to stop it being track-changed) and/or the '¬' (to make it case insensitive) and/or highlights it (and in what colour) are all set at the beginning of the macro – adjust to taste!

```
myColour = wdTurquoise
makeItCaseInsensitive = True
makeItNotTracked = True
```

Whole word search and replace – (*FReditListProcess* has probably made this redundant, but...) For some jobs such as automatic spelling correction, you may feel safer with whole-word F&R. So using *FReditCopyWholeWord* with the list:

this
that
tother

click, click, click gives you:

~<this>|this
~<that>|that
~<tother>|tother

Inverting items – If you have a *FRedit* item the wrong way round, then *FReditSwap* switches it back:

```
hello|goodbye
```

becomes

```
goodbye|hello
```

Sub SpellingSuggest()

Sub FReditCopy()

Sub FReditSame()

Sub FReditListProcess()

Sub HyphenationToFRedit()

Sub FReditCopyPlus()

Sub FReditCopyWholeWord()

Sub FReditSwap()

Create a FRedit list from a proper noun query list

(Video: https://youtu.be/4Ln95a1Cqyc)

If you have one of the new-style proper nouns query lists, it's very easy to generate items for your FRedit list. For example, this extract from a list:

6 = Educaion . . . 1
6 = Education . . . 1

1 = Even . . . 3

1 = Eves . . . 2

* Hernandez . . . 1
* Hernández . . . 3
* Hernendez . . . 1

6 = Institute . . . 1 = F
6 = Instituto . . . 1 = F

Jean . . . 1 = A
Joan . . . 1 = A

* 6 Jose . . . 2
* 6 José . . . 8
* 6 Jusé . . . 1

If you click in ‘Hernández’ and run this macro, it looks for your FRedit list (i.e. an open file containing vertical bar characters: ‘|’) and types in:

Hernandez|Hernández

and you’d put the cursor in ‘José’, this time the macro would have generated two items:

Jose|José
Jusé|José

```
Sub ProperNounToFRedit()
```

Create a FRedit list from a text list

(Video: youtu.be/AqREu_iJ2Yg)

This is a general tool: it takes a list, adds some text in front of, and after, every item, and then adds formatting to the list. You could use it to add a dash and a tab before, and a colon after:

this
that
the other

could become

– this:
– that:
– the other:

To explain it, I'll give a specific example for a FRedit list:

Suppose you have a list of words that you want highlighted in the text, but you only want whole words, so you have to use a wildcard for each one, e.g.

~<color>|^&

So you start with the list:

color
colors
favor
favors
labor
labors

The start of this macro sets up the text *before* each word (or phrase), the text *after* each, and the kind of colouration/highlighting/italic/bold you want:

```
txtBefore = "~<"
txtafter = ">|^&"

doItalic = False
doBold = False

addColour = 0

addHighlight = wdYellow
```

The result is:

~<color>|^&
~<colors>|^&
~<favor>|^&
~<favors>|^&
~<labor>|^&
~<labors>|^&

Another example, might be if you want to *colour* each of the words if it's followed by a comma, colon or apostrophe. If so, use:

```
txtBefore = "~<"
txtafter = "[,':]|^&"

doItalic = False
doBold = False

addColour = wdColorBlue

addHighlight = wdNoHighlight
```

The result would be:

~<color[,':]|^&
~<colors[,':]|^&
~<favor[,':]|^&
~<favors[,':]|^&
~<labor[,':]|^&
~<labors[,':]|^&

```
Sub FReditListCreate()
```

Lines of text into paragraphs

If you have text, say from emails, or from PDFs, with lots of individual lines, and you want to make them up into paragraphs, you can, of course, use *FRedit* (see the *FRedit* library), but this is a single macro that does the necessary global F&Rs.

It may be that some lines have line breaks instead of paragraphs (use Show Formatting to see) – the macro deals with those. And/or the paragraphs may be delineated by double returns, so the macro deal with both.

It's just the macro equivalent of:

^11|^32
^p^p|zczc
^p|^32
zczc|^p

```
Sub EmailFormatter()
```

Text exported from PDFs (or from OCR)

When text is exported from a PDF into Word (or has been OCRred), there are usually some 'issues', though not always the same issues. *FRedit* is your friend here. You can make obvious changes for errors in the text such as:

¨u|ü
¨o|ö

Ligatures (fi/ff/fl/ffi) can come out in all sorts of fun formats.

Interestingly, I had one PDF in which all the 'fl's did actually came out as 'fl' but the 'fi's and the 'ff's were converted to 'W' and 'V' respectively!

So here's the *FRedit* list that I used (though I can't quite work out now how it's supposed to work!):

~W([bcdfgjklmnpqstvwxz])|fi\1
~V([bcdfgjklmnpqstvwxz])|ff\1
~([a-z])W|\1fi
~([a-z])V|\1ff
~Wr([!io])|fir\1

(But I now have a macro solution to these funny characters in place of ligatures – see below.)

The other major issue I find is hyphenation. The problem comes two ways round:

1) Where words have been soft-hyphenated, the hyphens are still there, even when they shouldn't be. For example, if 'preparation' has been split with a hyphen as 'pre-paration' or 'prepar-ation', it will need to be rejoined.

2) Occasionally, I've had PDFs where ***all*** the line-end hyphens have been deleted whether soft or hard. So, on the plus side, that means that 'pre-<newline>paration' would correctly appear as 'preparation', but then 'two-<newline>dimensional' (wrongly) becomes 'twodimensional'.

But there are macros to help you – see below.

Rejoining hyphenated words (1)

(The following macro is newer, and probably better, especially when combined with PDFHyphenChecker.)

This macro looks through a file, checks every paragraph (i.e. every chopped-up PDF line) that ends with a hyphen, and tries to link it with the word at the start of the next line. If the two part-words, when joined, form a valid word (e.g. 'pre-paration' becomes 'preparation'), it is joined; however, the word 'twodimensional', being a spelling error, remains hyphenated.

The macro highlights the words that it has joined, so that you can check them. If you don't want them highlighted, use `myColour = 0`.

```
Sub PDFsoftHyphenRemove()
```

Rejoining hyphenated words (2)

(Video: youtu.be/iESM6OaGBm4)

If you select all the text in a PDF, copy it and paste it into a new Word file, the text will be there, but probably in individual lines:

```
Our aim is to compute jelly invariants of three-dimensional spaces. This chap-¶
ter begins with a few basics about jelly and then introduces the class of which¶
this is an example of the meaningless text I'm writing here, but I need an non-¶
linear example for the purposes of this illustration.¶
```

All this macro does is rejoin (apparently) split-by-hyphen words, so it will produce:

```
Our aim is to compute jelly invariants of three-dimensional spaces. This chapter¶
begins with a few basics about jelly and then introduces the class of which¶
this is an example of the meaningless text I'm writing here, but I need an nonlinear¶
example for the purposes of this illustration.¶
```

Then you can use *PDFHyphenChecker* to check the resultant file (see below).

```
Sub PDFHyphenRemover()
```

Correcting wrongly un-hyphenated words (global)

Rejoined hyphenated words can produce errors, so first here's a global macro, which may be a bit too dangerous, and then a selective one, which may be a bit slow – take your pick!

This macro looks through a file, checks the last word of every paragraph (i.e. every chopped-up PDF line) and tests to see if it's a spelling error. If so, it tries to divide the word at various places to see if it can make it into two separate words. For example, it would divide 'twodimensional' into 'two' and 'dimensional', and then it puts back the hyphen which has presumably been deleted.

The macro highlights the words it has hyphenated in a colour of your choice. If you don't want it highlighted, use `hyphColour = 0`.

```
Sub PDFhardHyphenRestore()
```

Correcting wrongly un-hyphenated words (selective)

(Video: youtu.be/iESM6OaGBm4)

This macro starts from the current cursor position, and checks every paragraph until it finds one that ends with a spelling error. It then tries to split it up into two separate, correctly spelt words. Then it asks for your view on whether to continue. You can say yes, and it keeps the hyphenation, or no, and it restores the unhyphenated version. So you are saying it's a correctly spelled word (perhaps a specialist word for the book's subject). Or you can stop and edit the text by hand, if necessary.

As it goes through, in response to your decisions to accept the hyphenation or not, it creates a list of 'OKwords', at the end of the file. Then each time it finds an end-of-line spelling error, it checks against the OKwords list, and if you've already accepted it once, it simply ignores the 'error', and moves on, thereby saving time.

This OKwords list can also used by other spelling-related macros. In particular, if you run (the latest version of) *SpellingErrorLister*, it will check the OKwords list and ***not*** include any OKwords in the spelling error list that it generates.

Sub PDFHyphenChecker()

PDFs with missing ligatures

In one PDF script that I converted to Word, all the ligatures had been converted to underline characters: *'I _nd English people di_cult to in_uence, which causes some su_ering.*', where each is missing either 'fi', 'ffi', 'fl' or 'ff'.

This macro finds each underline, then tries each of the ligatures in turn, and checks the spelling of the resulting word. If it's OK (e.g. '_nd' becomes 'find'), it changes it into the new word. If none of the ligatures gives a recognised spelling (e.g. 'John Black_eld'), it just highlights it.

Sub PDFunderlineToLigature()

PDFs with missing ligatures (2)

On the job mentioned a few paragraphs above, specific ligatures were replaced by specific characters ('fi' became 'W' and 'ff' was converted to 'V'). So this next macro deals with that. It looks for these characters, and tries to replace them by the relevant ligature, but if it makes an incorrectly spelt word, it leaves it alone.

It does make mistakes, of course. For example, if 'fi' is 'W', then the sentence, 'We went on Wednesday.' becomes 'fie went on Wednesday.' The 'Wednesday' is OK, but 'fie' happens to be a correctly spelt word.

No worries, just run the macro from within *FRedit*:

| Block off all 'We's
We|Wzczce

| Run the macro
DoMacro|*PDFfunniesToLigatures*

| Get rid of the dummy text
zczc|

You can set the characters for each ligature at the beginning of the macro:

```
fi_Code = "W"
fl_Code = "U"
ffi_Code = "Z"
ff_Code = "V"
```

However, in the case I mentioned, the 'fl' and the 'ffi' had translated OK, so you can save time by only testing the ligatures needed, use:

```
fi_Code = "W"
fl_Code = ""
ffi_Code = ""
ff_Code = "V"
```

Sub PDFfunniesToLigatures()

PDFs odd ASCII codes for ligatures

In one job, I copied and pasted the text from the PDF into Word, only to find that the ligatures had come across as funny ASCII codes: 11, 12, 13 and 14. This is a little difficult and doesn't lend itself to resolution via FRedit since, for example, 13 is the ASCII code for newline! So I had to use devious techniques.

You may never have this situation, but if you do, then try this macro, and if it doesn't work right, let me know, and send me a sample file if possible, and I'll tailor it to your situation.

```
Sub LigatureConverter()
```

For OCR/PDF, underline all spelling errors

In order to see what's wrong in a file, and how to convert it (with *FRedit* or with some of the following macros), it can be helpful to have all of the spelling errors within the file highlighted in some way. If you use underline as a way of 'highlighting' them then you can limit F&R to only the underlined text.

The two macros either underline ***all*** 'spelling errors', or try to work out which words might be proper nouns and ignore them.

```
Sub PDFspellAll()
```

```
Sub PDFspellIgnoreProperNouns()
```

Extracting formatted text from complex files (e.g. from PDFs)

This is of use when you have a complex PDF and you want to get the text out in order to analyse it. This was written for PDFs that were created by InDesign, where there were images that had a column of text alongside it, and InDesign had put every single line of text into a separate text box!

This is a complex process and even on a small file of, say, 2000 words it can take many minutes. So, after converting the PDF to a Word file, (1) load that file into Word and wait for it to 'settle' – I tend to click Ctrl-End to move to the end of the document and then back up to the top. Then (2) run the macros, and (3) DON'T TOUCH either the mouse or the pointer – preferably go and walk the dog.

```
Sub TextHoover()
```

Extracting text from textboxes

When trying to get the text out of a PDF file (perhaps to analyse it with macros), you sometimes find that the text is in a series of little text boxes. Here's a quick way to extract that text into a separate file.

Here's how you do it:

1) Open a new blank document
2) Click in the first text box/frame/area to be collected
3) Click back in the blank document
4) Set the macro running
5) Click on the **title bar** of the source document, and you should see that the text area is then selected and copied over to the blank document
6) Click in the middle of the next text area
7) Repeat until you've done enough

You can stop the macro at any time by clicking the **up-arrow key**. (A bit complex – don't ask!)

If you stop collecting text and yet leave the macro running, it will bong at you to remind you.

```
Sub TextPickup()
```

Multifile text compilation

(The latest version is demo'ed at: youtu.be/GE47DZ-ZkV0)

(***Mac users!*** You should be OK with this macro, but if it does throw up any errors, please try using *MultiFileTextForMac*, which works slightly differently, but is not as fully-featured as the main macro. Do ask if you have any problems.)

(N.B. If you have Word 365, this macros the *MultiFileText* macro can actually combine the **PDFs** into a single **Word** file for you.)

If you have a book made up of a set of separate files, it might be helpful to have a single file containing the text of the whole book. So that's what this macro does.

As with my other multifile macros, the macro gets you to identify the folder containing the files by bringing up the Open File window. Navigate to the required folder and click 'Cancel'. (If you're using *MultiFileTextForMac*, click 'Open' instead.) The macro then asks whether you want to work on *all* the Word files in that folder. If you say 'Yes', it uses the complete list of files that it has created and works through them all one by one.

If you say you *don't* want to work on all the files, the macro just stops. The list will look then something like this:

C:\Documents and Settings\Paul\My Documents\myNewBook
Chapter_1.docx
Chapter_2.docx
Chapter_3.docx
Prelims.docx

N.B. If chapters 1–9 have a leading zero, they'll come in the correct alphabetical order: Chapter_01, Chapter_02, etc.

The point is that you can then edit this list, either deleting files you don't want included, or putting a vertical bar ('|') in front of any you want it to ignore.

If you now run the macro again, it recognises that this Word document is a file list and so it proceeds to work through the listed (and not ignored) files, opening each one and creating the compilation.

It opens each of the Word files in the list, copies the text and pastes the text into a single Word file. It doesn't copy across any of the images or the text of the comments, but it does include the text of the footnotes and endnotes plus any text that appears in textboxes. However, no attempt is made to interleave the notes or the textbox text with the main text; rather, all this extra text is placed at the end of the text in a given file.

It also preserves any italic text in italic, and ditto for bold and superscripted text. This means that you can use the resulting file with *DocAlyse*, and it will correctly count how many 'et al.'s are in italic, and also how many 'funny degree symbols' there are, i.e. superscripted zeros, O's or o's. And having bold text helps you to see where the headings are.

(***Recent upgrade feature for*** *FindSamePlace*: If you have a file open that has been created by MultiFileText or MultiFileWord then if you click in a line and run this macro, it loads up the relevant original file and then finds the same line, so that you can look at the context.)

```
Sub MultiFileText()
```

```
Sub MultiFileTextForMac()
```

These macros are also available from:
http://www.archivepub.co.uk/LongMacros/MultiFileText
http://www.archivepub.co.uk/LongMacros/MultiFileTextForMac

Multifile Word compilation

If you have a book made up of a set of separate files, it might be helpful to have a single file containing the text of the whole book. So that's what this macro does. (A simpler alternative is *ChapterFileLinker*, below)

As with my other multifile macros, to get you to identify the folder containing the files, the macro brings up the Open File window, so you navigate to the required folder and then click Cancel. The macro then generates a list of all the files in the folder and asks whether you want to work on all the files in the list. If you say 'Yes', it uses the complete list of files that it has created and works through them all one by one.

If you say you *don't* want to work on all the files, the macro just stops. The list will look then something like this:

C:\Documents and Settings\Paul\My Documents\myNewBook
Chapter_1.docx
Chapter_2.docx
Chapter_3.docx
Prelims.docx

N.B. If chapters 1–9 have a leading zero, they'll come in the correct alphabetical order: Chapter_01, Chapter_02, etc.

If you now run the macro again, it recognises that this Word document is a list of filenames and so it proceeds to work through the listed files, opening each one and creating the compilation.

One thing to beware of is that if you join together lots of files, the size of the resulting file might prove a challenge for your computer system. I have therefore added an option to deletes all the embedded pictures.

The option is set at the beginning of the macro, so if you want to *keep* the images, use:

```
deleteImages = False
```

There is also now an option to add a filename at the top of each file so that you can see more easily where one section ends and the next begins.

```
addTitle = True
myFontSize = 30
titleHighlightColour = wdYellow
```

Another addition is the option to either leave the foot/endnotes as linked notes or to copy and paste each set of footnotes and/or endnotes at the end of the text of each file. This is useful because otherwise the endnotes for the whole book will be right at the end of the document.

```
insertNotesWithinText = True
```

You can also decide whether or you want it to accept the track changes before concatenating the files:

```
acceptTCs = True
```

The text within textboxes can be a bit of a pain so rather than just leaving that text within the textboxes, there's now an option to copy the text out of them boxes and embed that text as ordinary text within the file. The empty textboxes are then deleted.

```
embedTextboxText = True
```

Unfortunately, I haven't found any way of working out, where, within each file, a given textbox is. I have therefore simply had to paste the textbox text at the end of the text of each file.

Also, you can unlink field, which makes the overall file less 'complicated', i.e. if there are still fields – say, linking section numbers to their citations in the text – these can get corrupted in the process of concatenating files, so you can choose replace these citations with pure text. However, if you unlink *all* fields then equations can get turned into uneditable 'pictures', so there are two options. I suggest using:

```
unLinkAllFields = False
unLinkFieldsExceptEqns = True
```

(***Recent upgrade feature for*** *FindSamePlace*: If you have a file open that has been created by MultiFileText or MultiFileWord then if you click in a line and run this macro, it loads up the relevant original file and then finds the same line, so that you can look at the context.)

Sub MultiFileWord()

Chapter file compilation

If you have split up a book into separate chapter files and now want to recompile them into a single file, you can use *ChapterFileLinker*.

As with my other multifile macros, to get you to identify the folder containing the files, the macro brings up the Open File window, so you navigate to the required folder and then click Cancel. The macro then generates a list of all the files in the folder and asks whether you want to work on *all* the files in the list. If you say 'Yes', it uses the complete list of files that it has created and works through them all one by one.

If you say you *don't* want to work on all the files, the macro just stops. The list will look then something like this:

C:\Documents and Settings\Paul\My Documents\myNewBook
Chapter_1.docx
Chapter_2.docx
Chapter_3.docx
Prelims.docx

N.B. If chapters 1–9 have a leading zero, they'll come in the correct alphabetical order: Chapter_01, Chapter_02, etc.

If you now run the macro again, it recognises that this Word document is a list of filenames and so it proceeds to work through just the remaining listed files, opening each one and creating the compilation.

The macro saves the compilation file – in the same folder – as 'allTheBook'. N.B. it will overwrite any existing 'allTheBook' file in that folder. This name is set (and can be changed) in the line:

```
bookName = "allTheBook"
```

Sub ChapterFileLinker()

Multifile references compilation

(This can be used to collect the text of foot/endnotes, even if they aren't references.)

This works in the same multifile way as the two above, but this scrapes together all the references. One and the same macro will collect the references from all the files, whether they are in footnotes, endnotes or a section at the end of the main text.

The section containing the references has to have a heading, which you then identify at the start of the macro code. You can use a search something like:

```
refTitle = "^pReferences^p"
```

or

```
refTitle = "<H1>References"
```

(N.B. If there are footnotes or endnotes that ***aren't*** references, then change the first line of the macro to `collectNotes = False`.)

Sub MultiFileReferenceCollator()

Multifile track changes compilation

(Video: youtu.be/2hrfWRyDx18)

This macro goes through a set of files and creates a single file containing all the sentences that contain at least one track change.

```
Sub MultifileTrackChangeReport()
```

Loading multiple files from a folder

At the beginning of a job, it might be helpful to load a specific set of files from the work folder. This macro does just that. Run it once to create a file list for the folder, then edit the list to just the files you want. Then in future, just open the file list and run the macro.

```
Sub MultiFileLoader()
```

Text-only version of current document

This is a sort one-off version of MultiFileText, in that it creates a text-only version of the current open document, but it preserves bold, italic, super- and subscript.

However, I find that, with large files, the F&Rs that it needs to do can be quite slow; worse still, after the macro has finished, and beeped at to to tell you so, it still hase to reformat the whole of the new document. If you click anywhere on the screen before the cursor starts flashing again (i.e. the reformatting is complete) then Word can crash.

So for macros such as *ProperNounAlyse* and *SpellingErrorLister* – where all you are interested in is the words – then I've done an absolutely pure-text version, *CopyTextVerySimple*, which is much quicker for large files.

```
Sub CopyTextSimple()
```

N.B. This used to be called: *CopyTextWithSomeFeatures*

```
Sub CopyTextVerySimple()
```

Chopping into chapters

(Video: https://youtu.be/aRArJz6HmKI*)*

If a multi-chapter job comes to you as a single file, it can help greatly to edit it chapter by chapter, maybe using *FRedit*, as I have explained in Section 6 "Book editing – a possible workflow".

This macro allows you to chop a file up semi-automatically.

N.B. The chopped up files will be stored in the same folder as the source file.

For this macro, I decided to try using **video-only documentation**. Please let me know if you find this helpful... or not!

Thanks.

```
Sub ChapterChopper()
```

Chopping a file into sub-files, e.g. a book into chapters

This macro can be used to split any file into a set of smaller files based on where page breaks and/or section breaks occur, so it's up to you to insert the necessary breaks. You can insert page breaks by using F&R, for example using **Find**: ^pChapter and **Replace**: ^mChapter.

When the macro is run, it asks you for the filename, offering you 'Chapter' as the default, but you can change it to something else, if you prefer, then press Return. So, by default, the filenames will be Chapter01, Chapter02 etc.

If your text has some prelims followed by chapter 1, you can set:

```
firstChapterNumber = 0
```

and then the prelims will be Chapter00, and chapter 1 will be Chapter01 etc.

As it stands, it will spilt the text every time either type of break occurs, page or section. However, if, say, you want to split ***only*** at section breaks, you can set, near the beginning of the macro:

```
myBreak = "^b"
```

If you're using a Mac, you'll need to set, near the beginning of the macro:

```
myPostfix = ".docx"
```

Sub FileChopper()

List all files in a folder

(Video: youtu.be/AqREu_iJ2Yg)

If you just want to create a list of all the files in a folder – say for record-keeping, or whatever – then this macro does that. Navigate to the folder in question, then press Escape, and it will create the list. It can either create a list of just the Word files (.doc or .docx) or of absolutely all the files, as set by:

```
showAllFiles = True
```

Sub FileLister()

Acronym list with frequency

Like the following macro, this creates a list of all the acronyms that occur in the currently open file, including mixed-case acronyms such as 'SfEP'. The macro asks first if you want to list acronyms that include numbers. If you say 'Yes' then it includes acronyms such as BBC2, C2C and H2SO4.

The difference is that it also counts them, so you then know how often each acronym occurs. The other extra feature is that it highlights any acronym that occurs fewer than a certain number of times. This is useful if the client has a different way they want you to define once and/or repeatedly and/or in a separate file.

Is this the sort of thing your student wants?

The decision point as to whether to highlight an acronym or not is set at the beginning of the macro:

```
minCount = 3
```

Adjust to taste.

The final extra feature that it **ignores** any words in the source file that have a ~~strike-through~~ applied. The idea is that if, say, all headings are in full-caps, you can add strike-through to the heading styles (temporarily), and the headings will be ignored by AcronymAlyse.

N.B. Please test it initially with a dummy file with < 5000 words. And have it seeded with some known acronyms in it, so you can see if it's really doing what you expect.

The macro beeps at you every now and then, so that you know it's not given up, but, as usual, **please don't touch the mouse**, while the macro is running.

```
Sub AcronymAlyse()
```

Acronym list creator

This creates a list of all the acronyms that occur in the currently open file, including mixed-case acronyms such as 'SfEP'. The macro asks first if you want to list acronyms that include numbers. If you say 'Yes' then it includes acronyms such as BBC2, C2C and H2SO4.

It will also, optionally, create a *FRedit* list which you can then use to highlight all occurrences of each of the acronyms, so that you can see them in context. This is set by using `create`*`FReditList`* `= True` at the beginning of the macro.

On long files (20,000 words+), it can take quite a while to run, so if you want reassurance that it hasn't given up trying, set `doBeeps = True`, at the beginning of the macro and it will beep after completing each stage of the analysis, so you know something is still happening.

```
Sub AcronymLister()
```

The list is created by copying the whole text, highlighting the whole thing and then, using F&R, removing the highlighting from those items we want to keep. So, clearly, we unhighlight all the capital letters (using [A-Z]); if we want numbers, we unhighlight [0-9].

The problem comes with mixed upper/lowercase words. How do you include, say, 'SfEP' but reject words that have a single capital because they are at the beginning of a sentence? What the macro does therefore is only to unhighlight a lowercase letter if it immediately precedes an uppercase one (using [a-z][A-Z]). This will not, therefore, find 'BBCi' but will lose the 'i' and you will end up with just 'BBC'.

The original macro avoided this problem by going through the text word by word, checking each one in turn, but that was unusably slow.

Acronym finder

Having got your list of acronyms, you might need to find what they stand for. The following macro looks at the selected text, say 'TLC' and sets up a wildcard F&R to enable you to look for, 'a word beginning with t/T followed by a word beginning with l/L followed by a word beginning with c/C'. You can then look through the text to see if you can identify the right bit of text. You can then copy it and paste it into your acronym list.

Sadly, if you try this with a four-letter acronym, the wildcard search is just a bit too much for Word and it generates an error saying that the wildcard search is too complicated. So if you select a four-letter, the macro just looks for the first three letters and, the first time it finds something that matches, it selects the following word, just in case that's the right definition for the acronym, so you can just click Ctrl-C to copy it. If it's not right and you then got through to the next match, it'll only be looking for the three first words.

```
Sub AcronymFinder()
```

Create a list of acronyms and definitions

This macro assumes that, in the text, you have got things like "This is published by the British Broadcasting Corporation (BBC) and then edited my members of the Society for Editors and Proofreaders (SfEP) and concerns contacts with HM Revenue & Customs (HMRC)." It then looks through for the acronyms in parentheses and does its

best to find the definition, prior to the acronym. It errs on the side of picking up too many words, on the basis that it's easier to tidy up the list by deleting unwanted words, rather than having to look back through the text if any words are missing from the definition. From the above text, the macro creates:

Acronym list
BBC by the British Broadcasting Corporation
HMRC contacts with HM Revenue & Customs
SfEP Society for Editors and Proofreaders

```
Sub AcronymDefinitionLister()
```

Tagged uppercase words changed to small caps

The requirement here is to convert all words in a text that are in uppercase and have been tagged to be small caps. It assumes that it will be tagged as:

<sc>MY TEXT IN CAPS</sc>

And should end up as:

MY TEXT IN CAPS

But there is an option *not* to also delete the tags, which would give:

<sc>MY TEXT IN CAPS</sc>

(The person asking for this, also wanted "<th>/</th>" to be converted to a thin space, which is just a global F&R, added to the end of the macro.)

```
Sub TaggedTextToSmallCaps()
```

Make formatting tag invisible (hidden text)

If you had to "edit" a reference looking like this,

> <REF><BOOK><AU><SNM>Daley</SNM>, <GNM>J.</GNM></AU>, <AU><SNM>Wood</SNM>, <GNM>D.</GNM></AU>, and <AU><SNM>Chivers</SNM>, <GNM>C.</GNM></AU>.<YR>2017</YR>. <BTL>*Regional Patterns of Australia's Economy and Population*</BTL>. <LOC>Melbourne</LOC>:<PUB>The Grattan Institute</PUB>.</BOOK></REF>

It would be difficult, right?! And what if you wanted to do a Find for "Wood, D" – no way!

Is there a facility within Word to hide the tags? I don't know, so please tell me if there is one, but my motto is: if in doubt, write a macro!

So, if you run this macro, you get:

> Daley, J., Wood, D., and Chivers, C..2017. *Regional Patterns of Australia's Economy and Population*. Melbourne:The Grattan Institute.

Which is both readable and editable. And you can now search for "Wood, D"!

Run it again, and the tags reappear.

```
Sub TagsShowHide()
```

Acronyms to small caps

The requirement here is to convert all acronyms in a text to small caps. It works on either a selection or, if no text is selected, the whole of the file.

It defines an acronym as any complete word that consists of all capital letter, as long as it is three characters or more. This limit is set in the macro as:

```
minLength = 3
```

```
Sub AcronymsToSmallCaps()
```

Highlight incomplete paragraphs

This is difficult to describe, but I find it very useful. The macro looks for any paragraph that does not end in a suitable punctuation mark and, for any it finds, it highlights the very last character

This is in case you miss something like the fact that the previous paragraph, which didn't have a full point

And neither did that one, but at least you were alerted!

OK, it will also highlight the final character of every heading, but I decided I could live with that. But you can opt for the macro *not* to highlight any paragraph with fewer than, say, 20 words (or whatever number you want to set), which means only ***long*** heading get their final character highlighted.

But if you want ***all*** unfinished paragraphs to be highlighted whether they are bold or not then use:

```
minWords = 0
```

at the beginning of the macro.

Other options that the macro has are set with:

```
mySoftColour = wdColorBlue
' or for no colouration inside tables
mySoftColour = wdColorAutomatic

addLightColour = True
myLightColour = wdGray25

underlineLineFeeds = True
underlineQuoteNoPunct = True

minWords = 10
```

The idea of the first is that for text inside tables, you don't always *want* every paragraph to have a punctuation mark, so in a table, instead of using, say, a bright green highlight (very 'in yer face'), it uses a font colour (or not even a font colour – see the option above).

The 'light colour' of the second option is that which is applied to all full stops at the ends of paragraph – but you can switch it off altogether.

The final two are if you want (1) to put at underline on either or both one of those rogue linefeed, that can cause problems (it has to be underline, because a highlight would be invisible!) or (2) underline cases where the final character is a close quotation mark, and there's not punctuation mark immediately in front of it.

Also, if you want to ***remove*** the highlighting applied – perhaps so you can make some global changes to the file and then rerun the macro to check if your changes have done the trick – simply select a bit of text, and the macro will ask you if you want to remove the existing highlighting.

Sub ParagraphEndChecker()

Sort a list and remove duplicates

(Video: youtu.be/Yx97w8XJ6iE)

Here are two functions that might be useful when dealing with acronym lists – or indeed any other sort of list. The first is not rocket science – it just sorts the selected text. Once the list is sorted, the second macro removes all identical adjacent lines, i.e. removes the duplicates from your list.

If no text is selected, each macro assumes you want to sort and/or remove duplicates from the whole text – but it does ask you first.

And there's now a combined macro that does both.

And now the DuplicatesRemove macro has the ability to do the removal case-insensitively (or sensitively), by the setting of:

```
anyCase = True
```

or False.

Added feature: I wanted to compare two lists (the macro names on my laptop and those on my desktop) and delete from the list any names that appear twice (or more). So to implement that change the variable:

```
' Make this True if you want to completely delete
' any lines that appear more than once
' removeBothDuplicates = True
removeBothDuplicates = False
```

Sub SortIt()

Sub DuplicatesRemove()

Sub SortAndRemoveDups()

Sort case-sensitively

(Video: youtu.be/Yx97w8XJ6iE)

If you have a list that you want sorting so that all the uppercase words (e.g. proper nouns) are sorted out separately, you can use this macro.

Sub SortCaseSense()

Sort blocks of text

(Video: youtu.be/Yx97w8XJ6iE)

This is for sorting blocks of text such as names and addresses or block of data on multiple lines, where a block is defined by a blank line:

```
Jones, DE
Age: 42
Height: 5'6"
```

```
Brown, PQ
Age: 92
Height: 7'6"
Comment: Crikey she’s tall!

Green ZX
No data available

Adams
etc
etc
```

The macro will sort this into:

```
Adams
etc
etc

Brown, PQ
Age: 92
Height: 7'6"
Comment: Crikey she’s tall!

Green ZX
No data available

Jones, DE
Age: 42
Height: 5'6"
```

If a number of blocks of text are selected, it sorts just those, but if not it sorts the whole file.

Sub SortTextBlocks()

Sort, ignoring first ‘word’

(Video: youtu.be/Yx97w8XJ6iE)

(The first of these two macros is probably superseded by the second, but you never know; it might be useful.)

The idea here is that you’ve got a Vancouver list of references and want to sort it, ignoring the number that comes at the beginning of the line, i.e. sort of the first author’s name:

1 Smith HG Blah blah blah
2 Anybody PQ Jabber Jabber Jabber
3 Whatsun TT Nother book
4 Jones KJ Whatever Next?

then gets sorted into:

2 Anybody PQ Jabber Jabber Jabber
4 Jones KJ Whatever Next?
1 Smith HG Blah blah blah
3 Whatsun TT Nother book

The macro sorts on the first so-many characters (set in the macro as 20). And in case the number is separated by a tab, not a space, you can use the second `myDelimiter` line at the head of the macro.

Sub SortNumberedList()

A more general form of the macro is presented next. Here, you select the list to be sorted, run the macro and, if you want the list sorted on the text after the first tab character, just press Enter. Otherwise, for the same effect as the previous macro, type a space in the input box and then press Enter. Or to sort after some other character, just type that character instead, e.g.

Beverley, Paul – macro writer word following a tab
Brown, Andrew – typesetter text following a tab
Johnson, Arthur – author this follows a tab
Williamson, James – proofreader and this follows a tab

using a comma gives:

Brown, Andrew – typesetter text following a tab
Johnson, Arthur – author this follows a tab
Williamson, James – proofreader and this follows a tab
Beverley, Paul – macro writer word following a tab

or a dash (type into the input box ^=, the F&R code for an en dash) gives:

Johnson, Arthur – author this follows a tab
Beverley, Paul – macro writer word following a tab
Williamson, James – proofreader and this follows a tab
Brown, Andrew – typesetter text following a tab

and going back to just running the macro and pressing Enter, to sort on the tab, gives:

Williamson, James – proofreader and this follows a tab
Brown, Andrew – typesetter text following a tab
Johnson, Arthur – author this follows a tab
Beverley, Paul – macro writer word following a tab

And by using ^+, you can sort of what follows an em dash.

Or how about these ideas on this list:

Williamson, James (2017) *"Sandwiches for lunch"*. (Wiley)
Brown, Andrew (2016) *"A trip to the moon".* (Hodder)
Johnson, Arthur (2045) *"My first space flight"*. (McGraw Hill)
Beverley, Paul (2015) *"Confessions of a macro junky"*. (Elsevier)

Sorted using '(' gives:

Beverley, Paul (2015) *"Confessions of a macro junky"*. (Elsevier)
Brown, Andrew (2016) *"A trip to the moon".* (Hodder)
Williamson, James (2017) *"Sandwiches for lunch"*. (Wiley)
Johnson, Arthur (2045) *"My first space flight"*. (McGraw Hill)

And sorted using '"' gives:

Brown, Andrew (2016) *"A trip to the moon".* (Hodder)
Beverley, Paul (2015) *"Confessions of a macro junky"*. (Elsevier)
Johnson, Arthur (2045) *"My first space flight"*. (McGraw Hill)
Williamson, James (2017) *"Sandwiches for lunch"*. (Wiley)

And sorted using '. (' gives:

Beverley, Paul (2015) *"Confessions of a macro junky"*. (Elsevier)
Brown, Andrew (2016) *"A trip to the moon".* (Hodder)
Johnson, Arthur (2045) *"My first space flight"*. (McGraw Hill)

Williamson, James (2017) *"Sandwiches for lunch"*. (Wiley)

Fun, isn't it?! :-)

```
Sub SortOnTextAfterDelimiter()
```

Sort list of names by surname

(Video: youtu.be/P-6VdmT2BbE)

Suppose you have a list of names that need to be put in order by surname. Suppose too that some of the names have postfixes, such as 'OBE' or 'MSc' or ', Jr.', so that the surname isn't the final word on the line. This would mean that, say, 'Paul Beverley OBE' would be sorted with the O's not with the B's.

No worries! This macro will allow you to include these postfixes in a list at the beginning:

```
myPostfixes = "| BSc| MSc| OBE|, Jr.|, Sr.|"
```

It then sorts the list on the surname. Note that for, say, 'Paul Beverley, Jr.' you have to include the comma in the 'myPostfixes' line.

```
Sub SurnameSorter()
```

Sort reference list that has 'ditto' lines

(Video:https://youtu.be/f9sSbJ9XLZM)

The scenario here was of bibliography and references lists that were out of alphabetical order. Worse, the list used the convention of not repeating the author name but adding some sort of ditto marks – initially, three em dashes:

Adams, Sathy. *Bsa Arulobeit im Sathy Orocn.* Pinbit: Siuwsbit Cessdet, 1918.

———. "Bfi Litlagbn im Depahby." Et *Depahodenc otr Ebn Lhebeln.* Taf Yihk: TYU Ghann, 1984.

This could easily be solved with FRedit (explanation follows):

^p^+^+^+|zczc
DoMacro|SortIt
zczc|^p^+^+^+

The first F&R pulls all the 'ditto' paragraphs up to become part of the paragraph to which they belong, leaving a special 'marker text' (zczc) so that it can be restored later. So those single paragraphs, along with all the others, can then be sorted, using the SortIt macro. Finally, the paragraphs are split up again, by using another F&R.

No problem!

But then I discovered that some of the other references had a different format:

Aristotle. *Telicolsaot Abseln.* Cetaido, TY: Rivah, 1998.

- *The Giabeln.* Ditrit: Colcennot otr Li., 1917.

So I extended it to:

^p^+^+^+|zczc
^p-^t|pqpq

DoMacro|SortIt

zczc|^p^+^+^+
pqpq|^p-^t

Sorted!

Well, yes, except that there were also some references such as:

"Bending Vibahn faha Cibevobar py Maoh im Dinetw bsaeh Nbobun." *Bsa Aliticenb*, 26 Oghed, 2018.

which needed to be sorted in with the B's!

So I had to write a macro after all, to do a fiddle to remove any open double quotes, do the sort, then restore the quotes.

But while I was at it, I decided to make the macro so that it would either work on the whole file or (because this list had three separate alphabetic sections) or work only on the selected text.

So the macro declares two acceptable line-starters:

```
myDitto1 = "-^t"
myDitto2 = "^+^+^+"
```

i.e. a line-starter can be either [a hyphen and a tab] OR [three em dashes].

```
Sub BibSortWithDittos()
```

Sort group of citations

(Video: youtu.be/Yx97w8XJ6iE)

In the section below are two macros, *SortListInText* and *CitationListSortByYear*. They can sort citation lists such as "(Saheem, 2013; Darwish et al., 2007; Andrews and Bloggs, 2010)" either into alphabetic order by surname, or chronologically.

Either "(**Andrews** and Bloggs, 2010; **Darwish** et al., 2007; **Saheem**, 2013)"

or "(Darwish et al., **2007**; Andrews and Bloggs, **2010**; Saheem, **2013**)".

Here are the two macros...

Alphabetic sort in-line lists within a paragraph

(Video: youtu.be/Yx97w8XJ6iE)

The first macro, *SortListInText*, can sort lists in a number of formats within a paragraph:

1) (Saheem, 2013; Darwish et al., 2007; Andrews and Bloggs, 2010)
2) (Saheem 2013, Darwish et al. 2007, Andrews and Bloggs 2010)
3) The winners are Simons T, Andrews P and Peters V.
4) The winners are Simons T, Andrews P & Peters V.
5) The winners are Simons T, Andrews P, and Peters V.
6) The winners are Simons T, Andrews P, & Peters V.

It sorts lists with or without conjunction and with or without serial comma, and using 'and' or '&' as the conjunction.

It checks if there are semicolons and, if so, takes those as defining the items that make up the list. If there are no semicolons, it uses the commas instead.

Also, to make the macro easier/quicker to use, when sorting items (1) or (2), all you have to do is click the cursor somewhere (anywhere) between the parentheses; the macro automatically finds the extent of the list.

And if you had, say "Andrews and Bloggs, 2007; Saheem, 2013; Darwish et al., 2010", so that only the final two had to be reversed (either because of the surname, or because of the date), you could select an area of text from somewhere (anywhere) in 'Saheem' to somewhere (anywhere) in '2010', e.g. the bit I've highlighted.

But one issue with this macro is the possible ambiguity in item (3)? [and (4), actually]

3) The winners are Simons T, Andrews P and Peters V.

How does the macro know if this is a list of two persons or three? *Answer: it doesn't!*

And I couldn't make it automatically split the list at 'and' because...

2) (Saheem 2013, Darwish et al. 2007, Andrews and Bloggs 2010)

would then be treated as **four** items!

So the macro will ask you, with item (2) [and similarly with (3) and (4)]:

[Andrews and Bloggs 2010] – Is this a single item?

and you click 'Yes' (or just press Enter).

So in cases (3) and (4), you would instead click 'No' because it's two items either side of the conjunction, not a single item.

But if you're only using the macro for citations, and this feature annoys you, look at the start of the macro and find:

```
allowSplitAtConjunction = True
```

and change it to `False`.

`Sub SortListInText()`

and I suspect this does the exact same thing…
`Sub CitationListSortByName()`

Sort in-line citations by year

(Video: youtu.be/Yx97w8XJ6iE)

It may be that your client wants groups of citations sorting by **date**, not by surname, so that

1) (Saheem, 2013; Darwish et al., 2007; Andrews and Bloggs, 2010)

needs to be sorted to:

1) (Darwish et al., **2007**; Andrews and Bloggs, **2010**; Saheem, **2013**)

So *CitationListSortByYear*, does that. Again, you can just click somewhere (anywhere) within the parentheses.

Similarly, if you had:

2) (Saheem 2013, Darwish et al. 2007, Andrews and Bloggs 2010)

it would be sorted to

2) (Darwish et al. 2007, Andrews and Bloggs 2010, Saheem 2013)

Now with reverse order:

To change the chronological order, change the first line to:

```
sortReversed = False
```

to get:

2) (Saheem 2013, Andrews and Bloggs 2010, Darwish et al. 2007)

Sub CitationListSortByYear()

Add item to existing list

(Video: youtu.be/8-nmzpAY5VA)

(Ha! This is basically a duplicate macro to AddWordToStyleList *that I wrote a few years ago! But it's probably better – more flexible.)*

STOP PRESS: CopyToListAlphabetic works in a similar way, but it looks through the target file, and places the word/phrase into the correct alphabetic position in the list.

If you're trying to create a list of something-or-other, then this macro copies the currently selected text across into a different file. The obvious applications for an editor are adding words to a word list, or adding notes to a style sheet.

If you're collecting items from one list to put into another list, just click on the item – if no text is selected, the macro assumes you mean copy the current paragraph (i.e. list item).

(However, if you'd rather it assumed you want to copy the current ***word*** rather than ***paragraph*** then change the line: `copyWholePara = True` to `False`.)

You can have several files open at the time because the macro looks for a file that contains the text 'list' (or 'List') somewhere in its filename (and you can, of course, create two copies of this macro, one for 'list' and one for, say 'sheet' if you're adding things to a word list ***and*** to a stylesheet).

You can change the keyword, using, say:

```
keyWord = "style"
```

And also if, like me, you have other "list"s that might be open, you can tell it ***not*** to use those. So I often have open a FReditList and/or a zzSwitchList, so I add:

```
wordsToAvoid = "FRedit,switch"
```

Hopefully, you can adjust to suit your way of working.

For some applications, you might want a blank line after the text in the list file; so, at the beginning of the macro, you can use:

```
addBlankLine = True
```

Another option is:

```
includeFormatting = True
```

So you can either bring across into the list just the text, or you can bring the text with any formatting – e.g. bold or italic – that is in the text you're copying.

One user wanted the word(s) to be added, not at the cursor, but always at the end of the list file. If so, use:

```
alwaysCopyAtEnd = True
```

If, after adding the word/phrase/paragraph, you want to ***stay*** in the list, there's the option:

```
goBackToSource = True
```

which can be changed to `False`.

If, after adding the word/phrase/paragraph, you want to highlight the item in the original (to remind you which things you've copied, and which not), then swap the apostrophe from:

```
myHighlightColour = wdColorBlack
' To add highlight, use:
' myHighlightColour = wdYellow
```

to:

```
' myHighlightColour = wdColorBlack
' To add highlight, use:
myHighlightColour = wdYellow
```

i.e. you are making the highlight colour Yellow, not Black.

Sub CopyToList()

Sub CopyToListAlphabetic()

Table stripper

If you have a file that contains tables, and you need to take them all out into a separate file, leaving behind a callout of the form '[Table 4.3 near here]' (or whatever), this macro will do it for you. The format of the callout is set in the first line of the macro.

Note: It would be worth running this macro *after* tidying up the file a bit because a rogue space can sometimes cause it not to find a table.

It *should* find the tables regardless of whether the caption is above or below the table.

Sub TableStripper()

Tables to tab-separated text

This macro changes the contents of all the tables into tab-separated text, so:

one	two	three
four	five	six

becomes

one two three
four five six

Sub TablesToTabText()

Callout inserter

The aim of this macro is to find the first reference to each figure (or table) and add, at the beginning of that paragraph, a suitable callout. However, the macro checks for three different versions of each reference (e.g. either 'Fig. 23' or 'Figure 23' or 'Figures 23') as set in the first three lines of the macro. The format of the callout text is set in the line: `Callout = "<Figure XXXX here>"`, where XXXX is the figure number.

If a given figure number is not cited, the macro stops and alerts you so that you can make a note, and then let it continue. Of course, when it gets beyond the range of the figure numbers, you just click on 'No' when asked: 'Continue?'

If the text says, say, 'Figures 4.3 and 4.4 show...' then it won't be able to find the 4.4, so it will tell you that it can't find 4.4, so make a pencil note and then, when it's finished, go back and add it in manually.

If the figure/table numbering has a chapter prefix (e.g. 'Fig. 4.23') then answer 'Yes' to the 'Existing chapter numbers?' question. However, if there are *not* any existing chapter numbers, and you want to *add* them, answer 'Yes' to 'Add chapter numbers?'

If it annoys you to have to answer three questions every time you want to use the macro (it does me!) then use the alternative lines in turquoise, and use vbYes or vbNo as appropriate.

A basically similar macro follows for use with table callouts.

Sub FigCallouts()

Sub TableCallouts()

Move all figures out into a separate file

If you have a file that contains figures, and you need to take them all out into a separate file, leaving behind a callout of the form '[Figure 4.3 here]' (or whatever), this macro will do it for you. The format of the callout is set in the first line.

N.B. It assumes that the caption, 'Figure 2.5 Diagram of an elephant', or whatever, is *below* the actual figure. If you need it to be the other way round, I'll have to do a special version. If it would be of help to you, do please ask.

It would be worth running this macro *after* tidying up the file a bit. For example, if the caption has a space before the word 'Fig(ure)', it won't find that figure.

The macro may miss the odd figure – I can't guarantee 100% success, but if it gets to be a problem, send me a sample file and I'll try to solve it.

(If your figures are captioned with 'FIGURE 3.1 etc...' then you need to change the line `myFig = "Fig"` into `myFig = "FIG"`.)

It occurred to me that you might want to have the captions either kept with the main text or placed with the figures (or both), so this can be set at the beginning of the macro. If you use:

```
captionWithText = False
captionWithFigs = True
```

then the caption will only be in the Figs file and not in the main text.

Sub FigStrip()

Edit the contents of table cells

(Video: youtu.be/P-6VdmT2BbE)

This macro started with the need to change all lone hyphens in table cells into em dashes, and then it extended into editing negative numbers in cells that used hyphens (e.g. -6.3) into proper Unicode minus signs (−6.3).

It can also strip off trailing full points and/or carriage returns and/or tabs (which can cause some real fun problems!)

I've added an optional highlight to show where changes have been made (but not where trailing characters have been stripped off.

If no text is selected, the macro works on all the tables in the document. If part (or all) of one table is selected, it works on just the selected bit. If several tables are selected, it works on just those tables.

You can customise the macro to some extent:

```
stripEnds = True
stripThese = "^p^t., "
```

This can be changed to `False`, if you don't want to strip off training characters, and you can add or subtract characters from the list.

```
addEmDash = True
```

If you don't want to add an em dash to all the empty cells, make this `False`.

```
doHighlight = True
myColour = wdGray25
```

I'm sure you can work out what these two do.

```
Sub TableEdit()
```

Add em dash to every table cell

In my experience, very few authors know that you're supposed to use an em dash if you're indicating empty cells in a table. Most use hyphens or, at best, en dashes. This macro goes through the whole of the text and changes cells with en dashes or hyphens into em dashes.

As an option, you can either allow empty cells to remain empty, or to put an em dash in them too:

```
dashIfEmpty = True
```

```
Sub TableEmDasher()
```

Add final character (full point) to every table cell

Someone asked if it was possible to add a full point to the end of every cell of a table. To be more specific, they wanted to add full points to a *selected area* of a table – say, a pair of columns, but not the other columns.

The following macro adds full points to just the selected area of the table. If there's already a full point there, it does not add a second one.

If you want to add a different character to every cell, you can change the following line accordingly:

```
myChar = "."
```

Note: Because of the strange way that Word handles the cells in a table, I've had to use a little trick. The first thing the macro does is to add the 'shadow' attribute to the selected cells (you could use a different attribute if you use shadow for something else). It then looks through the table cells, ignoring the non-shadowed cells, but adding full points to the shadowed cells. It then removes the shadowing from the whole table.

```
Sub CellsAddChar()
```

Add initial capital to every table cell selected

This macro ensures that every cell in the selected range has an initial capital letter.

```
Sub TableCellsInitialCaps()
```

Remove/restore borders and rules of table

This removes all the border lines and vertical and horizontal rule of the selected table – and then adds them back if you run the macro a second time.

```
Sub TableBordersToggle()
```

Textbox and frame removal

Authors may use Word frames or textboxes for tables, figures and their captions, or other pieces of text that can 'float'. These features of Word present traps for the unwary editor: they are invisible in Normal (Draft) view; they are not affected by global search/replace operations; and they will probably not be correctly imported into typesetting software. The following macro will convert their contents into normal text, tagged <TBX> for a textbox or <FRM> for a frame. If you know or suspect that a document contains frames or textboxes, you should run this macro during the clean-up process. It can save you hours of copying and pasting.

```
Sub TextBoxFrameCut()
```

List all styles used in a document

This macro generates an alphabetic list of all the styles that are used in the document, each with an indication of the page on which that style first occurs, plus the first few (six max) words of that paragraph:

Biblio Title	p.332	"Bibliography"
Footnote Text	p.123	"Brown 1990, "How the Word files"
Heading 1	p.12	"Introduction"
Heading 2	p.1	"Contents"
Heading 3	p.11	"Other notes:"
Table Grid	p.48	"C19 innovations"

The set-up options in the first few lines of the macro are:

```
' myStyles = "Normal,Default,"
myStyles = ""
```

Use the first line to list any styles that you **don't** want listed.

```
displayParaWords = True
numWords = 6
```

If the first is set to False, you don't get the third column, of the first words of the paragraph.
If it's True then the second line sets the max number of words to be displayed.

```
deleteTableBorders = True
```

The final option, if True, will **remove** the all the lines you can see in the sample illustrated above.

Apply styles to textboxes

This applies a style to all of the textboxes in a document.

The line `myStoryRange.Style = "List Bullet"` can of course be replaced by specifying any other style name, or indeed any other action(s) that you would normally apply to selected text, such as other formatting, or even find and replace etc.

Sub SetTextBoxStyle()

Copy text out of text boxes [textboxes] into main body of text

This copies the text out of text boxes and pastes it into the body of the text, hopefully in approximately the right place.

You have the option of inserting the text before or after the anchor paragraph:

```
placeTextAfterParagraph = True
```

Change the first line between True and False.

If you need to code/tag the text, it will add a code before and after the extracted text:

```
beforeText = "<Box>"
afterText = "</Box>"
```

(and obviously you can change the text used here) but if you don't want the added tags, change it to:

```
beforeText = ""
afterText = ""
```

There's also an option to highlight all the copied text in your chosen colour:

```
addHighlight = True
myColour = wdYellow
```

Sub BoxTextIntoBody()

Footnote (endnote) fiddling

This macro does corrects three 'funnies' that an author might put in footnotes or endnotes:

– it removes a space if there is one in front of any note number
– it deletes a trailing space (or two)
– it adds a full point to the end of every foot/endnote that doesn't already have one (or a !).

Sub FootnoteEndnoteFiddle()

Endnote (Footnote) Fiddling – 2

(This is similar to the previous macro, but checks different things. If either isn't quite what you want to do, please ask and I'll update one or other for you, time permitting.)

People do some funny things in their formatting of endnotes (and footnotes), and this macro was written because one editor had a file (well, a set of files) where someone had added a blank line between each endnote and the next. You should, in theory, be able to remove them by find and replace, but it doesn't always work properly, so I wrote this macro. But there's probably an easier way ...

Sub NoteDeleteDblSpace()

Footnote (Endnote) Fiddling – 3

This time the author had made all the footnote numbers italic. You can unitalicise (romanise) the callout numbers *in the text* with *FRedit* using:

^f|^&

but that doesn't affect the actual numbers of the actual footnotes themselves. We therefore need to use a separate footnote-fiddling macro:

Sub FootnoteNumberNotItalic()

And if you want these numbers not superscripted, the last line would be:

```
rng.Font.Superscript = False
```

and (being really silly), you could have:

```
rng.Font.Bold = True
rng.Font.Size = 18
rng.Font.Name = "Arial"
```

Footnote (Endnote) Fiddling – 4

Someone wanted to change the endnote numbers in the endnotes to superscript. This seems to do the trick.

Sub EndNoteFiddleSuperscript()

Footnote (Endnote) Fiddling – 5

Someone else wanted to remove any extra space the author had placed at the beginning of their footnotes. This seems to do the trick.

Sub FootnoteFiddleStartSpace()

Delete all endnotes

You can delete all the endnotes in a document at once:

Sub DeleteAllEndnotes()

Delete all footnotes

You can delete all the footnotes in a document at once:

Sub DeleteAllFootnotes()

Unembed footnotes or endnotes

The aim of this macro is to extract all the footnotes or endnotes, put them at the end of the file, and replace all the footnote numbers with ordinary superscripted numbers (but highlighted in turquoise). When I first wrote this macro, I said, 'This is clearly not something you want to do if there's any chance that someone is going to want to add or remove footnotes, because Word's automatic renumbering is, of course, lost once this macro has been run.' However,

someone had files in which the notes *had* been unembedded, and asked if I would write a macro to reverse the process – see 'Re-embed Notes' below!

[Later: This then means that if you have a file that has problems with the notes, you can unembed them, fiddle with them (maybe automatically – see below) and then re-embed them.]

```
Sub NotesUnembed()
```

Re-embed notes

If your notes are in the form of numbered paragraphs at the end of the document, *place the cursor on the first line of the first note* and run this macro, and it will insert them all as proper footnotes. These are now live and so you can add and delete notes, and Word will now automatically renumber them.

N.B. It might not work too well if the notes are 'untidy', e.g. ensure that there aren't unnecessary spaces and multiple newlines (^p) in the notes. And it also can't deal with notes of more than one paragraph, sorry. Maybe go to the notes and do a selective F&R of ^p into pqpq, run the macro and then change pqpq's back into ^p's.

```
Sub NotesEmbed()
```

Unembed endnotes by sections

This does the same sort of thing as *NotesUnembed*, but it is for the case where the notes are at the ends of each chapter, and the chapters are set up using Word's 'sections' feature. It can take quite a while to run with a big file, and even once it has run and beeped to show that it has finished, Word can then still take quite a while to 'release' the file. As far as I can tell, it's reformatting the file, and won't give you control back until it has finished.

```
Sub NotesUnembedBySections()
```

Renumber all superscripts

...and...

Renumber all note numbers

(These following two macros go together really, hence the weird double title!)

(N.B. The latest version renumbers either just the selected area of text or, if nothing is selected, it will ask if you want to renumber the whole of the text.)

The next macro goes right through a file looking for all superscripted numbers (which we are assuming are footnote/endnote markers) and replacing them with a consecutive series of numbers starting from 1.

Obviously, if your text has m^2, m/s^2 etc (i.e. other reasons for superscripting), then you've had it! You can't use this macro to renumber the footnotes. Well, you could use *FRedit* to change all those other superscripted numbers to something non-superscripted, run this macro, and then put them all back again into superscript.

Then following that is a 'Renumber all the notes' macro, the idea being that you have a set of notes, but the numbering is no longer consecutive, for some reason.

To use it, you just put the cursor somewhere on the first line of the notes (or, indeed, any list of numbered items), and it goes through and renumbers them all consecutively.

```
Sub RenumberSuperscript()
```

```
Sub RenumberNotes()
```

Sorting out messed-up footnote numbering

(This uses the previous four macros – it's just the recipe.)

So, you've got a file whose note numbers have got all messed up and are no longer consecutive. The idea of this rescue package is to change Word's automatic notes into ordinary editable text; then sort out the numbering, then re-embed the notes into the text so that, once again, they are using Word's note facility, and you could switch back from endnotes to footnotes and change to roman numerals, add and remove notes etc.

This recipe assumes that the notes have Arabic numbering, not roman numerals. What's more, it assumes that they are endnotes, not footnotes, so before you start, use Insert–Reference–Footnote… to change to endnotes with Arabic numerals.

So, the total process is:
0) Change footnotes to endnotes and roman numerals to Arabic numerals.
1) Run *NotesUnembed* which changes the automatic notes into just ordinary superscripted numbers with notes as a bunch of (formatted) text at the end of the file.
2) Run *RenumberSuperscript* to give the citations a set of consecutive numbers.
3) Move down to start of notes and run *RenumberNotes* to make them consecutive too.
4) Check that all is well, i.e. that you've got the same number of citations as notes!
5) Place the cursor on the first line of the first note and run *NotesEmbed*.

Endnotes/footnotes to inline bracketed text

This macro copies the text (with its formatting) from end/footnotes and places the text in square brackets following the related note citation.

```
Sub NotesCopyToInline()
```

Convert individual footnote to endnote or vice versa

This macro converts the current note from end to foot or vice versa. However, if the cursor is in the main text, it hunts for the first available note marker (foot or end) and converts that.

```
Sub NoteFootEndSwitch()
```

Bracketed notes to embedded footnotes

This macro first deletes any existing embedded notes, then inserts any text that appears in square brackets in new footnotes, with citations next to the source note (then deletes the text in square brackets).

The previous macro was a request from an editor in India, but when I'd done it, I thought I might as well reverse the process, as it's only a few lines of code, but then I realised that this pair might be useful as a way of sorting out messed up footnote numbering. Worth a shot, anyway!

```
Sub NotesInlineToEmbed()
```

Add a footnote (endnote) but in a different style

I prefer to use a keystroke to do most things, so I wanted a macro to add a footnote, to which I could attach a keystroke. However, someone asked me if it was possible to add a footnote, but using a different style. Answer: yes.

These two macros add a footnote or endnote, but with your specified style name.

Then someone asked if it was possible to add footnotes with [] around the number, as shown here.[3] Answer: yes.

```
addSquares = True
changeStyle = False
```

These two lines at the beginning of the macros specify whether or not each feature is used, i.e. as above, the foot/endnotes will be add with squares round the citation number, but with the default foot/endnote style.

Sub FootnoteAdd()

Sub EndnoteAdd()

Renumber any list

This macro renumbers any list within a file, starting from the number of the item at the cursor, i.e. you could start part way through a list). This contrasts with *RenumberNotes* (above), which assumes you are numbering from item 1, and that the list extends right to the end of the file.

So to give the macro chance to know when it has reached the end of the list, it asks for the maximum number of unnumbered paragraphs that can occur *between* the numbered paragraphs.

Sub ListRenumber()

Remove numbering from all headings

Santhosh Matthew Paul writes: "I needed to remove numbering from numbered headings in Word 2010 documents. The headings are styled Heading 1, Heading 2, etc., and the numbering is automatic. I couldn't see a way to accomplish this by adjusting a setting in Word, say, by modifying the heading styles. I was only able to change one heading at a time.

"So, I tried the macro approach. I recorded myself removing the numbering from one heading, and learned that the key command is: `Selection.Range.ListFormat.RemoveNumbers`"

Santhosh then wrote a macro; it sort of worked, but had a couple of issues. He shared it with me, and I was able to smooth out some of the problems. It's now available for other people to use.

Sub RemoveNumbersFromHeadings()

Delete all bookmarks

Delete all bookmarks in a Word document at once:

Sub DeleteAllBookmarks()

Delete all comments

You can delete all the comments in a document at once:

Sub DeleteComments()

In Word 2002 or later, you can reduce it to a single-line macro:

Sub DeleteComments2()

```
' Version 18.06.10
' Delete all comments
ActiveDocument.DeleteAllComments
End Sub
```

And in fact, you don't even need a macro. Simply assign a keystroke to the Word command DeleteAllCommentsInDoc. (Tools –> Customize; click Keyboard; select AllCommands in LH window; select DeleteAllCommentsInDoc in RH window, pick a keystroke and click Assign.)
On the other hand, you might like to use a macro because you can give yourself feedback:

```
Sub DeleteComments3()
' Version 18.06.10
' Delete all comments
numberCmnts = ActiveDocument.Comments.Count
ActiveDocument.DeleteAllComments
MsgBox("Comments deleted: " & str(numberCmnts))
End Sub
```

Delete all comments by a specific author/editor

If you have a document with comments by different people, maybe an author and an editor, you might want to delete the comments of one, but keep the others.

```
Sub DeleteCommentsSelectively()
```

Transfer comments from square brackets to bubbles

(Video: youtu.be/2PG7n5MCMCo)

Within the text, the client had placed a load of comments within square brackets. The task was to copy each comment and insert a comment bubble at that point, and finally place the text of the comment in the bubble.

```
Sub CommentBracketsToBubbles()
```

Transfer comments from square bubbles to brackets

(Video: youtu.be/2PG7n5MCMCo)

Having done that, why not write a macro to reverse the process?!

```
Sub CommentBubblesToBrackets()
```

Add (and remove) serial numbers to (from) initials in all comments

The idea here is that within the comments, you insert something like "AQ:" as an indicator that this is a comment or query for the author (or "TS:" for the typesetter).

This macro then goes through all the comments and adds serial numbers to any comment containing an "AQ:" tag.

However, if some of the "AQ:"s already have serial numbers, it assumes, rather, that you want to **remove** the serial numbers from all the tags. They will all be restored to just "AQ:", without numbers, so you can then run the macro a second time, and it will serialise them again.

```
Sub CommentsAddIndexOnInitials()
```

Delete all comments that don't have tags

The editor then wanted to delete all of the comments (presumably their own temporary comments) that did NOT contain these AQ tags (numbered or not), leaving just the comments for the author.

```
Sub CommentsDeleteAllNotTagged()
```

Delete all comments that DO have a specific tag

(Video: https://youtu.be/gOBpOMbIogU)

The reverse of the above. You can put any tag you like into comments then run this macro and it will delete all those comments. But obviously you have to choose you tag wisely because comments that are deleted stay deleted! The tag is set in the firat line of the macro:

```
myDeletionMarker = "***"
```

```
Sub CommentsDeleteSelectively()
```

Delete all hyperlinks

You can delete all the hyperlinks in a document at once.

Health warning: If your text has equations don't use this macro; instead use FieldsUnlink, below.

```
Sub DeleteAllLinks()
```

Unlink all fields except equations

This macro unlinks all fields except equations.

```
Sub FieldsUnlink()
```

Fields codes visible (or hidden)

If you need to, say, edit the URL code behind a link, you can assign a keystroke to the Word command ViewFieldsCodes – no need for a macro. You just go to All Commands in the Customize Keyboard window, and find ViewFieldsCodes and assign a keystroke. But...

If you have a large file with lots of links (like this one!!), you may find that, when you make the codes visible, because there's now more visible text, the cursor position will have have disappeared way off the screen. OK, if you move the cursor left or right, the screen will jump back to the right place, and you can see your cursor again. However, when you switch them off again, you'll have to the same again!!

So you can use this macro, which will switch the codes on or off, and you'll still be able to see the current cursor position – at the bottom of the screen – so you can immediately make your edits.

```
Sub FieldCodesVisible()
```

Delete selected hyperlinks

This macro, as written, looks through all the hyperlinks in the text and, if they are URLs, i.e. contain 'www' or 'http' it does *not* delete them, but deletes all the rest. This was needed by a client who wanted the hyperlinks to authors' names to be deleted, but the URL links to be preserved.

If you have a different selective hyperlink deletion criterion, I'm sure we could edit this macro accordingly.

Someone suggested it would be good to see where links had been removed. If you take the apostrophe out of the line

```
' myColour = wdGray25
```

then the text of the deleted links will be highlighted in light grey (or change the colour to wdYellow, or whatever you fancy).

Sub DeleteSomeLinks()

Check each of the URLs

The aim of this macro (I think!) is to check whether each of the URLs in the text appears in the references list. It then highlights them red or green accordingly. If this is something you might want to do, please send me a sample file, as the macro will probably need adjusting.

Sub ReferenceCheckWeb()

Citation and bibliography (references list) field conversion

If you have citations automatically linked to a bibliography (references list), and want to turn both into unlinked, editable text, this macro does the trick. It also has the option ('cos the guy who asked for it wanted it) to turn the citation into italic. If you don't, then change the first line to:
`makeCitationItalic = False`.

Sub UnlinkCitationsAndRefs()

Mendeley citations and punctuation correction

If authors have used the Mendeley reference system, but has placed the citation *inside* the punctuation, then this macro corrects it.

Sub MendeleyPunctuationCorrection()

Delete all figures from a file

This macro looks through all the inline images, and looks for a caption that might indicate that it's a figure with a caption. If so, it deletes the image.

Sub DeleteAllFigures()

Delete all inline images from a file

This macro looks deletes all the inline images from a file, regardless of what they might be (so don't say I didn't warn you!).

Sub DeleteAllInlineImages()

Delete all inline images from a file and close the space

This macro looks deletes all the inline images from a file, and closes up the space where they were.

Sub DeleteAllImagesAndCloseUp()

Delete all inline images from a file and add a call-out

This macro looks deletes all the inline images (figures, hopefully) from a file, and replaces them with a message, such as “<Figure 19.3 about here>”.

You can either add the relevant chapter number at the beginning of macro each time, or get the macro to dig the chapter out of the filename, e.g. “19_Further Explanations”.

Sub DeleteAllImagesAddCallout()

Delete all paragraphs that are mainly italic

This finds all paragraphs that are mainly in italic and deletes them. The line

```
deletionFactor = 6
```

sets the decision level for deletion, i.e. delete the paragraph if there are (6) times as many italic words as roman words.

Sub ItalicParaDelete()

Convert combo boxes to text

To replace the combo boxes with the text that each is currently displaying you run a very simple macro, provided by Howard Silcock of New Zealand.

Sub ComboBoxAccept()

Unbold every colon followed by roman text

This is where you’ve got headwords in bold, each followed by roman text, but you don’t want the colon to be bold. However, you can’t just use global F&R to unbold the colons because there might be colons, say, in bold headings, where the whole line, including the colon, must remain bold.

So this macro looks for bold colons that are followed by roman text.

Sub ColonUnbold()

Auto-lists to text

This macro converts all Word automatically numbered and bulleted lists to proper numbers and bullets. It makes the file more suitable for sending to a typesetter: if you don’t do this, the bullets and numbers can sometimes get lost when the file is imported into the typesetting system.

Sub AutoListOffSimple()

If you want ‘proper’ bullets and not the ones in Symbol font that Word tries to make you use, there’s a more complex macro that uses F&Rs. It converts the bullets, sub-bullets and sub-sub-bullets to Unicode characters.

Technical details: The code &HF0B7 in the first F&R is for the Symbol bullets. (If you use the *WhatChar* macro, that will confirm that the hex is F0B7.) The second F&R is for the tick symbol in the Wingdings font. Again, I got the F0FC code by using *WhatChar*.

In both F&Rs, I replace with an ordinary bullet, but you could use a different symbol, e.g.

```
newCharacter = "*": ' asterisk
```

If you want to use the same macro (i.e. still use the same keystroke) but without changing the bullets, you can change the first line to `changeBullets = False`.

Sub AutoListOff()

Full-out paragraph under all headings

This macro ensures that the first paragraph under each heading is full out by applying a style with no first-line indent.

It uses Word's built-in styles Heading 1, Heading 2, Heading 3, Body Text and Body Text First Indent as examples so, if your document uses different styles, you will need to edit the style names accordingly.

First, make a list of all the names of the styles after which you *don't* want an indent. This is in the line:

```
StyleList = "Heading 1, Heading 2, Heading 3, and any more you want"
```

Then specify the names of styles for with- and without-indent text:

```
NoIndentStyle = "Body Text"
IndentStyle = "Body Text First Indent"
```

Here's the complete macro.

Sub FirstNotIndent()

Selective format changing

Suppose you have a text where the author has not used styles, but simply applied effects (bold, italic, font size, alignment etc) on a piecemeal basis. Now, I prefer to edit in a left-justified style rather than fully justified, because it's easier to see if there's any odd spacing – each space is the same size. Now, if the author has centred some paragraphs and maybe right justified others, you probably want to keep those as they are and only change the fully justified paragraphs to be left justified. Here's a macro to do it:

Sub JustifyOFF()

But: you don't need to use a macro, do you?! You can just use the F&R dialogue box. Leave the Find and Replace boxes empty, and in the Find, set Format – Paragraph – General – Alignment – Justify, and then in the Replace, set Format – Paragraph – General – Alignment – Left. The 'Replace All' does the rest.

Still, I've left the macro in the book, just in case you can use it as a pattern for doing something else on a paragraph by paragraph basis. I can't think why, but suppose you want to change all the paragraphs in 14pt bold justified into 12pt italic left aligned – you couldn't do *that* with ordinary F&R! Here's the macro version:

Sub FunnyChange()

Raised/lowered text to super/subscript

If the author has used the font attribute 'Raise by 3pt', or whatever, to indicate superscript and 'Lower by 3pt' for subscript, this macro converts them to proper super/subscript, regardless of how much they have raised/lowered the text by. It also highlights any changes it makes, in your chosen colour, so that you can keep a track of what it has done.

Sub SuperSubConvert()

Greek symbol font checker

(This may be redundant in the light of the next macro.)
(N.B. This works for other Symbol fonts too – not just Greek ones. And it also now checks for Wingding fonts, which it highlights in red.)

There's a *FRedit* list for this in the *FRedit* library (search for 'greek'), but there may be characters in your text that it doesn't yet cover. The best thing to do is run the *FRedit* list, and then run this macro *on a copy of your text (don't say I didn't warn you)*. It will point up any funny codes that it finds, and offer you the codes ready to extend the *FRedit* list. All you then have to do is find the proper Unicode number for the character and add that to the list.

The macro sets up the Find box so that you can look through for any hex codes that it has found for you, copy each one and add it to your *FRedit* list.

```
Sub SymbolFontCheck()
```

Symbol font to unicode converter

MS Word's Symbol font can be a pain in the proverbial, so the idea of this macro is to completely change every character that's in Symbol font into the Unicode character equivalent.

The macro has two modes: Test, where it types the Unicode character equivalent alongside the original, and Normal where it simply ***replaces*** the character with the Unicode character. If you're nervous that this macro might get some things wrong, run it first in Test mode on a copy of the text, and check that they are all OK.

This has by no means been an easy macro to write – the shortness of the resulting macro is deceptive! This is because there are myriad ways in which different versions of Word can 'mash' Symbol fonts. Now, there is one feature that I was nervous of (though I can't remember what the feature was now!), and so any characters that have this feature are coloured in red. So just double-check that they have come out as intended.

At the beginning of the macro there is a conversion table, listing the character number of the old Symbol font, followed by the Unicode number to which it is converted. This list is not exhaustive – it's just the ones I've come across in my work. So if the macro finds a Symbol font it doesn't know, it beeps at you and highlights it in turquoise. If you then find the necessary codes (use the *WhatChar* macro to find the Symbol font number and search the web for the required Unicode number), you can add them to the list. But if you're unhappy to mess with the macro, send me a sample of the different character – it's only a couple of minutes' work to find it.

```
Sub SymbolToUnicode()
```

Highlight all 'funny' fonts

The idea of this macro is to bring to your attention any characters/words/paragraphs that are *not* in the main font(s) used in the file. So in the first few lines of the macro, you name all the fonts you *don't* want highlighting.

N.B. This doesn't seem to work 100%. It works on some fonts but not others. It seems that Word's Find facility (which is what the macro uses) doesn't respond to all font names. That said, because the macro is defining which areas of text *not* to highlight, it might highlight more than it should, but it won't fail to bring to your attention things you're interested in.

The macro allows you to specify up to five fonts you *don't* want it to highlight.

```
Sub FontHighlight()
```

Get rid of 'rogue' fonts

Suppose you have a file that's in, say, Times New Roman (TNR), but there are various places where Arial have been used, or Garamond, or some such, and you want to restore them to the font used in the Normal style.

This macro reads the name of the font at the cursor (say Arial), reads what the Normal font is (say TNR) and changes any text in the whole document that's in Arial into TNR.

```
Sub FontEliminate()
```

Get rid of 'locked' fonts

This is an obscure problem, but if you've experienced it, you'll know what I mean!

You notice that in a paragraph of, say, Times New Roman a character(s) is in the wrong font (I've had them in Calibri, Cambria, Sim Sun and MS Gothics) but when you select the text around the characters it refuses to change into TNR.

I have tried several ways to fix this, both manually and using macros, and always failed. However, one day I shut myself in a darkened room, and two hours later, after trying various really complicated ideas, I found a simple one that worked! Hurrah!

So here are two macros. The first searches from the cursor downwards, to find any paragraph that contains mixed fonts (though that might be for valid reasons). With the second macro, if you select some text either side of the rogue character(s), it should make all the characters the same font as the first character. Enjoy!

N.B. FunnyFontClear doesn't work with track changes switched on, sorry!

```
Sub FunnyFontFind()
```

```
Sub FunnyFontClear()
```

Funny font full facilities

These 'funny' fonts are 'interesting' to say the least, so I've created a group of macros to try to analyse and fix (if 'twere possible) these funnies.

First we want to know what fonts are being used in a given file, so the first macro finds all the different fonts used at paragraph, word or character level. Each time it finds a new, different font name, it stops and shows you what it's found and where (this will become important later, trust me!).

It types out the list of font names it's found at the head of the file,

```
Sub FontLister()
```

Now we want to strat digging around the file, looking at the use of these different fonts, some of which will be perfectly valid, of course. So the next macro uses Word's ordinary Find facility to find "some text in such-and-such font".

If you click in one of the font names at the head of the file and run the macro, it will search for that font name.

If you select a bit of text in a given font, it will search for that instead.

```
Sub FontFind()
```

Once you've got some selected text in one of the funny fonts, and you want to restore it to the base font – Times New Roman, or whatever – you can use this third macro.

After it (thinks it) has restored the text to a sensible font, you perhaps want to go through the text and find the next bit of text in the funny font, so as Find is set up, you can just use my *FindFwd*, or however you move to the next find.

However, if instead you just run this same macro a second time, it detects that you've already cleared this bit of text and so it thinks "Oh, I suppose you mean that I should just move to the next bit of funny font text." So you click this macro once to correct the text and a second time to jump to the next bit of funny font text.

```
Sub FontFunniesClearThisOne()
```

(This macro sort of half duplicates the macro in the previous section, FunnyFontClear.)

But this is where the fun starts! In my experience (well, in the file I'm working on at the moment), FontLister gives me:

Arial
Calibri
Cambria Math
MS Gothic
NSimSun
SimSun
Times New Roman
Times-Roman

(Some interesting 'fonts', eh?!)

However, when I ask FontFind to find either Arial or Calibri, it denies all knowledge of them. But I know they are there; I saw them with my own eyes, when I ran FontLister!

If I now try to use Word's Find window, it says it's looking for "Font: (Default) Arial" and "Font: (Default) Calibri", whereas for the others it says, e.g. "Font: Cambria Math".
Go figure!

Anyway, one more macro in the set. If you have some characters in, say, MS Gothic, if you select one of them, then next macro goes through the whole file and reverts all MS Gothic characters to the default font.

```
Sub FontFunniesClearAll()
```

Add thin (or other) space to units

If you want to have a thin space in between every number and its unit, e.g. 16 kg, then this macro finds all the numbers-with-a-unit (whether already spaced or not) and inserts your chosen kind of space – thin, non-breaking, '<spc>' or whatever. It also recognises degree symbols (whether Unicode character or Symbol font) and removes their space.

The macro needs to know what is or is not a 'unit'. For example, you don't want it to use a thin space for "16 men on a dead man's chest". So I've used a set of criteria but then there are two sets of exceptions. It sounds complicated, but once you have it working, all you need to do is add or subtract items from the exceptions list to suit your own specialist texts.

The criteria are different for different lengths of the word that the macro thinks might be a unit. First, it will assume that all one- and two-letter 'units' need spacing. It will also space all three-letter 'units', ***unless*** the three letters form a word that the spelling checker recognises, e.g. '16 men'. No 'units' of four or more letters will be spaced unless you specifically list them at the start of the macro.

Here are the suggested exceptions lists as they appear in the macro:

```
' one- and two-letter words to ignore
ignoreThese = ",a,b,c,d,e,f,g,h,i,j,k,n,o,p,q,r,t,u,v,w,x,y,z," & _
              ",D,E,I,O,X,Y,Z" & _
              ",An,an,as,As,at,be,by,do,en,gh,ie,if,in,In,is,Is,nd," & _
              ",of,no,No,on,On,or,pp,rd,Re,so,So,st,th,to,To,UK,US,vs,we,"
```

```
' three-or-more letter words to include
includeThese = "kWh,MPa"

' three-or-more letter words to ignore
excludeThese = "exp,"

' Avoid things like "Fig. 2.3 A view..."
notAfterThese = "Fig,Figure,Table,Box,Section"
```

So the macro is in a perfectly usable state. If you run it, you might find it misses some units and/or makes some false positives. In which case, you can refine one or other of the exception lists in the macro.

(Aside: You might think that having 'kWh' and 'MPa' in the 'include' list is unnecessary, since they aren't proper words. However, Word's spellchecker recognises them as correctly spelt words, so they would be ignored if they weren't included in the list.)

Other things set at the beginning of the macro are (1) the colour in which to highlight the spaces, so use wdNoHighlight if you don't want the added spaces to be highlighted. (2) the type of space to be used: thin, non-breaking or textual.

I've also added the facility to 'hide' text such as reference lists by using the single strikethrough attribute (as I do on many of my macros). Also, I've added a facility where it checks the word immediately before the 'unit', and ignores it for certain words. This is to avoid things like, 'Fig. 2.3 A view showing...' being interpreted as meaning '2.3 amps'.

```
Sub UnitSpacer()
```

Language selection

When changing language, it's worth doing it using a macro, rather than just clicking on the language change icon.

Why? Well, it's possible that the author has specifically set the language of some of the various elements of the document (main text, notes, comments etc.), and the language icon ***only*** sets the language of the element where the cursor is currently placed.

Over the years, I've discovered a number of spelling-related problems that people have experienced, and added their solutions into the macro(s).

Specifically, it sets the language of the text inside textboxes, in footnotes and endnotes and in comment boxes (which can be a problem in files originating in the Far East) and even in the Normal Style – yes, if someone has specifically set the language of the Normal style to, say, French, you can click the language change icon to UK English, but the style for the Normal style stays French (well, it does on my system, anyway: Word 2010).

This might be a belt and braces approach, but it takes no longer to run a macro than to click an icon, so you might as well be safe and sure.

Plus, as it's a macro, if you're using *FRedit*, you can add a line:

```
DoMacro|LanguageSetUK
```

so you don't forget to change the language.

You can, of course, create a second copy of the macro that sets US English – or any other language.

German versions now added.

```
Sub LanguageSetUK()
```

```
Sub LanguageSetUS()
```

```
Sub SpracheDE()
```

`Sub SpracheCH()`

(The next four all do more or less the same! Oops!)
`Sub LanguageToggle` [29.10.21] – Toggles the language setting of (part selected) text
`Sub LanguageSetMulti` [07.06.23] – Toggles between different language country settings
`Sub LanguageSwitch` [16.08.23] – Switches the language between two or three alternates
`Sub LanguageUSUKswitch` [06.10.23] – Switches language between UK and US English

(The last one is for me, as I'm using the [cheap, non-subscription] Word 2021, and the status bar refuses to display the current language, so this macro displays the language name on the bar as a confirmation.)

Highlight text not in main language

If you have a file that, you suspect, has bits of text where the language has been changed, you can obviously change the whole file to a given language using the macros above, but you might want to know exactly which bits of text have, for some reason, had their language changed. This macro highlights all words not in the language that prevails at the current cursor position.

For speed, it checks the text one paragraph at a time and if it finds a 'mixed' paragraph, it checks that paragraph a word at a time and highlights the 'funny' ones.

`Sub LanguageHighlight()`

Mark long sentences

If I am asked to shorten over-long sentences, I find it helpful to be alerted to the fact that any given sentence *might* be on the long side – it's one thing less to think about as I'm reading through the text. This macro therefore finds any sentences longer than a certain number of words and changes the font colour to one of two colours, so that they are drawn to my attention.

I decided to use two different colours in case two consecutive sentences are over long – it makes it obvious that it's two long sentences, not one really, really long sentence.

The choice of colours and the critical number of words are set in the first three lines of the macro.

`Sub LongSentenceCheck()`

Ensure all sentences have two spaces following

This macro ensures that every sentence has two spaces after it.

`Sub DoubleSpaceAfterSentence()`

Highlight all questions

One editor wanted all questions to be drawn to their attention while reading through the text. This macro highlights in green (change it if you like) all sentences that end with a question mark.

`Sub HighlightAllQuestions()`

Highlight all long quotations

(N.B. These two macros are effectively superseded by *QuotationMarker* above.)

I was asked if I could write a macro to look for over-long quotes and put them into displayed quote style. It's a bit of a big ask if you think about all you do when you find a long quote. I decided as a starter to at least highlight all over-long quotes – 'that should be easy', I thought. Ha! I hadn't reckoned on apostrophes.

The `wordLimit = 50` will set it so that if a quote is 50 or more words, it will highlight it.

The first macro is easy. If your text uses double quotes, there's no problem (the word limit for highlighting the quotes is set at the beginning of the macro):

`Sub HighlightLongQuotesDouble()`

However, if the text uses single quotes, you can find the beginning of each quote easily enough, but how do you distinguish between an apostrophe and a close single quote – difficult! Of course, *haven't* and *we've* etc are easy to identify, and so is the apostrophe-s: *the boy's book*, but it's the s-apostrophe that is the real problem. If you have *the poets'*, is that something belonging to them, or is it the end of the quotation?

What I've done, therefore, is simply err on the side of assuming that, say, *the poets' blah blah* is an apostrophe, and carry on searching for another close quote/apostrophe. So if, in fact it was a close quote, the result is to generate a 'quotation' which is actually two quotations plus the intervening text. Still, at least it highlights it, so it's easy for you to check it out later. And to make it stand out better, I have unhighlighted any apostrophe/close quote mark that is highlighted in your chosen colour.

(It's difficult to explain, but it'll make sense when you see it in action ... I hope.)

`Sub HighlightLongQuotesSingle()`

Displaying long quotes

If you've got lots of long quotes, it's useful to be able to remove the quotation marks, make it into a displayed quote, and change the style of the quote, e.g. use a style with left and right margins, and maybe smaller typeface:

> This is a meaningless long quote that I've made up on the spur of the moment to illustrate what I mean in the previous paragraph, and it is what I wrote and that's that, blah, blah, blah.

If some text is already selected when you run this macro, it assumes that this is the text to be displayed. However, if no text is selected, it searches for the next long quote.

For the displayed text, it uses the style whose name is set at the beginning of the macro, and then if the quote is currently in the middle of a paragraph, is sets the following paragraph to a style of your choice. This is useful where most paragraphs have a first line indent, and then you have a zero indent on the following line to show that the paragraph is still continuing.

```
displayedQuoteStyle = "DisplayQuote"    [Style for displayed paragraph]

nextParaStyle = "Body No Indent"        [Style for next paragraph]

' Or if you don't want to change next paragraph style
' nextParaStyle = ""

removeQuotes = True          [Remove the quote marks or not]
minWords = 40                [min length of quote for displaying]

singleQuotes = False         [single or double quotes]

addTags = True/False         [do/don't add tags]
startTag = "<DQ>"            [start and (below) end tags]
endTag = "<\DQ>"
```

`tagOnNewLine = False` [Tag on end of displayed para or on next line]

Sub DisplayQuote()

Highlight (and/or style) all indented paragraphs

Another related macro is to highlight those paragraphs that are indented, the idea being to identify the displayed quotes. The trouble is that you can't easily use F&R because, often, the author will 'hand indent' the quotes. In other words you don't know exactly what the magnitude of the indent will be, so F&R is difficult, to say the least.

The alternative offered here is simply to highlight (and/or style) all paragraphs that have any indent at all. OK, you may get some false positives, but it'll probably be quicker than doing in all manually.

The macro was written to just highlight the paragraphs, but if you use the two turquoise lines it will actually apply the style to them.

Sub HighlightIndentedParas()

Change the indent of specific indented paragraphs (1)

One reader wanted to find all paragraphs indented by 1.25 cm and change the indent to zero. This macro does so.

Sub IndentChanger()

Change the indent of specific indented paragraphs (2)

Another reader's client wanted to find all paragraphs with a first line indent and (a) remove the indent and (b) add a tab stop at 0.25″. This macro does so.

Unfortunately, some paragraphs already had a 0.1″ FLI, so this had first to be remove, or the tab would only take the text to 0.1″. This is done with:

```
.ParagraphFormat.TabStops(InchesToPoints(0.1)).Clear
```

Adjust to taste. :-)

It also now has the option to use centimetres instead of inch measurements. For this, use:

```
useInches = False
```

Sub FirstLineIndentToTab()

Apply styles to all paragraphs except headings

One reader wanted to apply 'BodyStyle' to all paragraphs other than those such as 'Heading 1', 'Heading 2', etc. But the paragraph immediately after each heading should be 'NoIndent' style. This macro does so.

Sub StyleBodyIndent()

Apply character style to headwords

One reader wanted to apply 'EntryBib' style to all paragraphs that have a specific style ('Normal,Normal full left'). This macro does so.

```
Sub FormatHeadwords()
```

Count the highlighted areas

The purpose of this macro was, originally, to give a way of quickly counting how many serial comma or not serial comma occurrences there were in a text (see under *DocAlyse*). I used a *FRedit* list to highlight one in light grey and the other in dark grey, and then counted them.

(But *DocAlyse* has been greatly improved, so this is probably redundant.)

```
Sub CountHighlightColour()
```

Every 'Normal' paragraph to 'Body Text'

(This macro is, of course, redundant because, using the Styles and Formatting pane, you can select all occurrences of Normal, and then apply the style Body Text. Doh! But I've left it in because if you have to change several styles, you could use this macro as a way of automating it.)

This macro applies the style 'Body Text' to every paragraph that is currently in Normal style. Every other paragraph that is in 'Heading 1' or 'Table Text' or whatever style remains unchanged.

Using Body Text rather than just Normal is, apparently, more helpful when importing into InDesign.

The same macro could, of course, be used for changing paragraph between any two named styles.

```
Sub BodyTexter()
```

Coding (tagging) every bold heading

If the author has simply made every heading bold, this macro will add a code (myCode1) to each such heading. Optionally, if you have some of the headings that start with a number and you want those to be a different code (myCode2), this macro will do the necessary wildcard F&R for you (or you could just put it in your *FRedit* list).

If you don't want this optional feature, then set `myCode2 = ""`.

```
Sub CodeBoldParas()
```

Adding coding (tagging) automatically

If you have to code the various styles in a document, this macro will do it automatically. Basically, you add whatever styles you want to the document and then run the macro. It goes through the whole document, paragraph by paragraph, checks the styles and adds the appropriate coding tags.

If you look at the macro, you'll see that coding tags to be used are specified by the items in the 'Case' list, such as:

```
Case "Heading 1"
  startText = "<A>": endText = "</A>"
```

So you can add whatever style names and tag texts you want. Any style that is not included in your list will not be tagged.

If you only want a tag at the beginning of the paragraph and not at the end, just use, for example:

```
startText = "<A>": endText = ""
```

```
Sub AutoTagger()
```

If you run the macro and then decide that another style type needs coding, you can just add it to the list and rerun the macro. Any line that is already tagged doesn't get retagged; it is simply ignored.

If you want the tags to stand out in some way, you can make them bold or larger font size or whatever by extending the 'If' statement at the end of the macro as follows (adjust to taste):

```
  If rng.Characters(1) <> "<" And startText > "" Then
    rng.InsertBefore startText
    rng.End = rng.End - 1
    rng.InsertAfter endText

    Set bit = ActiveDocument.Range(rng.Start, rng.Start + Len(startText))

    bit.Font.Size = 24
    bit.Font.Color = wdColorRed
    bit.Font.Italic = False
    bit.Font.Bold = True
    bit.Font.Name = "Arial"

    Set bit = ActiveDocument.Range(rng.End - Len(endText), rng.End)

    bit.Font.Size = 24
    bit.Font.Color = wdColorGreen
    bit.Font.Italic = True
    bit.Font.Bold = False
    bit.Font.Name = "Arial"
  End If
```

Add line space after all tables before headings

Sounds a bit odd, but this is for when I'm tagging all hierarchically numbered headings using wildcard find and replace, for example:

| anything such as 3.4 followed by <tab> or <space> = A head
| and such as 3.4.5 is a B head, etc
~^13([0-9]@).([0-9]@)[^t^32]|^p<A>\1.\2^t
~^13([0-9]@).([0-9]@).([0-9]@)[^t^32]|^p<B>\1.\2.\3^t
~^13([0-9]@).([0-9]@).([0-9]@).([0-9]@)[^t^32]|^p<C>\1.\2.\3.\4^t

So, the trouble is that these F&Rs don't work if a table immediately precedes the heading, because the table doesn't end in a conventional newline (^p or ^13) character. The idea then is to look through all the tables and add a newline after any table that is immediately followed by a heading.

```
Sub TableSpaceBeforeHeading()
```

Coding (tagging) displayed quotes

If the author has helpfully indented the displayed quoted, then even if they've not used a style, you can code (tag) them automatically. This macro looks for any paragraph(s) that have a left indent and adds <disp> in front and </disp> at the end (you can change it to whatever text you want).

<disp>The codes are added in the positions as shown in this paragraph, but if you want the </disp> tag to be on the beginning of the next paragraph, just let me knw; it's a simple enough tweak to make to the macro.</disp>

And if the quote consists of multiple paragraphs, the macro removes the </disp> and <disp> that occur in between two indented paragraphs, so you end up with one tag at the beginning of the quote and one at the end.

```
Sub CodeIndentedParas()
```

Coding (tagging) bulleted list

If bulleted lists are in Normal style (and not, say, ListBullet), you can tag/code them by the fact that they have a negative first indent. Run the macro, and all the bulleted lists will come out as:
<BL>• This is the first line.
• Second line
• Third line
• Fourth line
• Fifth line
</BL>

```
Sub TagBulletLists()
```

Formatting numbered list item

This is one I use when I'm working for a publisher that likes lists numbered as:

1. This is the first line.
2. Second line
3. Third line
4. Fourth line
5. Fifth line

But the authors have used:

1) This is the first line.
2) Second line
or
(1) This is the first line.
(2) Second line

So the macro corrects it line by line.

```
Sub ListItemNumberFormatter()
```

Adding bold to first word of list (glossary)

If you have, say a list of words with their definitions, and you want each of these headwords to be bold, this macro does it for you.

```
Sub BoldFirstOccurrence()
```

Using manual line breaks in and poetry (verses) and/or lists

If you have a poem that is formatted using double returns between verses, the formatting can be problematic. This can by changed using manual line breaks after each line within each verse, and a single return (end-of-paragraph, ^p) between verses.

So, if you click in the first line of the first verse, and run this macro, it will change the returns to line breaks (^p to ^11) for each line, except the last, and will remove the double return at the end.

If the macro is assigned a keystroke, then you can click, click, click your way through the poem. However, if you select the whole poem, from somewhere in the first line to somewhere at the end (you can be quite approximate in your selection) and run the macro, it formats the whole of the selection in one go.

The same macro can also be used for a list or lists.

Sub VerseListFormat()

Adding (coloured) tags to all italic/bold text

Some publishing clients want all italic text to have tags before and after in colour, as *shown* here. So this macro goes through the whole file and adds such tags. (And ditto for bold.) You can do italic and/or bold by setting the two variables at the beginning, e.g. for italic but not bold:

```
doItalic = True
doBold = False
```

Sub ItalicBoldTagger()

And a more comprehensive version that covers bold, italic, sub- and superscript, underline and strikethrough:

Sub TagVariousAttributes()

Adding (coloured) tags to selected text or to next bold text

If the previous macro – doing the tagging globally – is not suitable, this macro speeds up local, individual tagging. It has two mechanisms.

If no text is selected, it zooms along the line to find the next bit of text in bold, and adds coloured tags to it.

If some text is selected, it tags it.

(The macro could easily be changed to tag the next italic bit, instead of bold. If you're not sure how, do just ask me.)

Sub TagSelectedOrBold()

Adding an <ni> tag to the first line after each heading

If you want a no-indent tag (e.g. <ni>) after each heading, i.e. before the next paragraph, then this macro does it for you.

Sub TagNI()

Listing all tagged section headings

(Video: youtu.be/DnG1XCuOUlk)

If you've applied tags <A>, <B>, etc to your section heads (using FRedit?), then one way to check that they are correctly numbered is to create a list of all the paragraphs that start with <A>, <B>, etc.

The list might also be used for a contents list, but the ContentsListByTag macro is better for that as it deletes the tags and formats the list. If you just want to check the numbering then this is a lot quicker than ContentsListByTag.

If it would be helpful, you can create a copy of the macro, call it ListOfTaggedHeadingsTextOnly, then run them both, at the end of your FRedit list:

DoMacro|ListOfTaggedHeadingsTextOnly
DoMacro|ListOfTaggedHeadings

Then you end up with a pure text version, and one with formatting – but I guess you could just do it formatted and then take the formatting off with *NormaliseText*.

If you then want to check that the numbering is properly hierarchical and contiguous, you can use NumberSequenceCheckerHierarchical, although you'll first have to strip off the tags:

~\<[ABCDE]\>|

```
Sub ListOfTaggedHeadings()
```

Adding full point to ends of captions

One publisher I work for insists that all figure and table captions should have a full point, whether the text forms a sentence or not. This macro looks for a specific tag (<Cap>, but you can change it) and, if necessary, adds a full point.

```
Sub FullPointOnCaptions()
```

Adding styles to numbered headings

If your text uses various levels of numbered headings, subheadings and subsubheadings – 1.2, 1.4.1, 1.2.1.6 etc then this macro will look at each heading, decided what level it is from the number of full stops in the heading number, and style the heading accordingly: Heading 1, Heading 2 etc.

N.B. It assumes that there is a tab character after the heading number. If the text has a space, then change the 'delimiter' at the start of the macro.

It works so quickly that I've added a beep at the end so that you know it has finished. If you are changing heading levels, by adding and subtracting numbers then you can, of course, change the heading style manually, but you can just as easily run the macro to 'refresh' the heading styles.

```
Sub HeadingStyler()
```

Coding the first lines of a chapter

This is just a macro I ran up for a specific job, but it's maybe something you could adapt and use in your own situation. Each chapter started with, something like:

6
Visual Attention in Coding

Joe Bloggs and Fred Brown

The Concept of Visual Attention

Visual attention is, blah, blah, blah

So I was coding it, by adding the same codes in each chapter. (Doing something over and over again => use a macro.)

In this case, I'll break my normal rule, and ***not*** put the macro separately at the end of the book, but rather put it here so you can see how it works and therefore, hopefully, be able to modify it for your own use.

```
Sub CodeFirstLines()
' Version 05.12.12
' Prepare the file by adding codes etc

code1 = "<CN>"
code2 = "<CH>"
code3 = "<CA>"
code4 = "<A>"
```

```
myLine = 1
For Each myPara In ActiveDocument.Paragraphs
  If Len(myPara.Range) > 1 Then
    If myLine = 4 Then
      myPara.Range.InsertBefore Text:=code4
      Exit For
    End If
    If myLine = 3 Then
      myPara.Range.InsertBefore Text:=code3
      myLine = 4
    End If
    If myLine = 2 Then
      myPara.Range.InsertBefore Text:=code2
      myLine = 3
    End If
    If myLine = 1 Then
      myPara.Range.InsertBefore Text:=code1
      myLine = 2
    End If
  End If
Next myPara

' Set 1.15 spacing
Set rng = ActiveDocument.Content
rng.ParagraphFormat.LineSpacing = ActiveDocument.Styles("Normal").Font.Size *
1.15

' Set language UK

ActiveDocument.TrackRevisions = False
Set rng = ActiveDocument.Content
rng.LanguageID = wdEnglishUK
rng.NoProofing = False
ActiveDocument.Styles("Normal").LanguageID = wdEnglishUK

' Switch on track changes
ActiveDocument.TrackRevisions = True
End Sub
```

So the macro first adds codes to each of the first four paragraphs. Then it sets the line spacing of the whole file to 1.15 lines (obviously you can use a different spacing, or delete it altogether.

Then it sets the language for the whole file to UK English. (It switches off track changes before doing so; otherwise it will generate unwanted ‘Formatted: English (U.K.)’ track changes.)

Then finally, it switches track changes on, because that’s what I needed for that job.

Showing style names within text (= adding style codes)

I wrote this macro because I was frustrated that the only way to make style names visible on screen is to go into Draft mode, and I hate working in Draft mode! So what this macro does is to add the style names, as text, at the beginning of every (non-Normal style) paragraph:

<|Heading 2|>This is a sample heading

As Fred Bloggs, that famous expert, once said:
<|Displayed quote|>This is a dummy quotation saying nothing but illustrate the principle of what I’m trying to do here.

I've used the hopefully unique combination of angle brackets and a vertical bars so that if you run the macro a second time, it recognises that the file contains these visible style names and deletes them all. So the macro is like an on-off switch for these visible style names.

Obviously, you don't want Normal style showing, but if there are others you don't need to see, just add them to the noShow list:

```
noShow = ",Normal,"
noShow = noShow & "TOC 1,TOC 2,TOC 3,Table of Figures,"
```

You can add names to the list, but just be sure to keep the commas as separators.

Then I thought it would be good to allow abbreviations – to make the names less intrusive. So instead the heading above could be:

```
<|H2|>This is a sample heading
```

But hang on! If you set Heading 1 as A, Heading 2 as B, Displayed Quote as DQ etc, then take away the vertical bars, that's what you want for typesetting codes:

```
<B>This is a sample heading
```

So that's the other job you can use this macro for: coding your text. If you have

```
removePads = True
```

at the beginning of the macro then, when it has added the codes, it will ask if you want it to also remove the pads.

The abbreviations/codes are set at the beginning of the macro:

```
abbrvs = ",MTDisplayEquation,Disp,Heading 1,A,Heading 2,B,"
abbrvs = abbrvs & ",Heading 3,C,Heading 4,D,"
```

If you add/subtract abbreviations, just make sure that you have a comma before and after each name or code.

```
Sub ShowStyles()
```

Simple number sequence checker

(Video: youtu.be/2hrfWRyDx18)

This is a very simple macro that looks to see if a series of numbers, whether as a numbered list, or numbers within a paragraph are consecutive. So it will spot errors such as this:

> There are ten reasons for this: (1) ksjhgkjdhf (2) skjdf ghdsfgdkjsf gkjd (3) hadk hg kajdhsfkj asd (4) fg kjdf (5) mngfngndng (6) ahsd fakj fkj (7) akjhsd g gmj dfg (9) ghskdfhgkhsfkjdgh (10) jahsdgfhfj akj

or this:

> There are ten reasons for this:
> 1. ksjhgkjdhf
> 2. skjdf ghdsfgdkjsf gkjd
> 3. hadk hg kajdhsfkj asd (Smith 2016)
> 4. fg kjdf
> 5. mngfngndng
> 6. ahsd fakj fkj (Brown 1989)
> 7. akjhsd g gmj dfg
> 9. ghskdfhgkhsfkjdgh
> 10. jahsdgfhfj akj

It will ignore years, so in the list above, it won't be distracted by the citations. This is set by the line at the beginning of the macro:

```
dateStart = 1800
```

So if it's dates that have to be consecutive, you can change it to, say, `dateStart = 100000`.

Sub NumberSequenceCheckerSimple()

Decimal number sequence checker

(Video: youtu.be/2hrfWRyDx18)

This macro check that a series of decimal numbers is 'consecutive', so it will spot the error in this:

> There are ten reasons for this: (2.1) ksjhgkjdhf (2.2) skjdf ghdsfgdkjsf gkjd (2.3) hadk hg kajdhsfkj asd (2.4) fg kjdf (2.5) mngfngndng (3.1) ksjhgkjdhf (3.2) skjdf ghdsfgdkjsf gkjd (3.3) hadk hg kajdhsfkj asd (3.4) fg kjdf (3.5) mngfngndng (3.6) ahsd fakj fkj (3.7) akjhsd g gmj dfg (3.9) ghskdfhgkhsfkjdgh (2.10) jahsdgfhfj akj

or this

> There things in different chapters:
> 2.1 ksjhgkjdhf
> 2.2 skjdf ghdsfgdkjsf gkjd
> 2.3 hadk hg kajdhsfkj asd
> 2.4 fg kjdf
> 2.5 mngfngndng
>
> 3.1 ksjhgkjdhf
> 3.2 skjdf ghdsfgdkjsf gkjd
> 3.3 hadk hg kajdhsfkj asd
> 3.4 fg kjdf
> 3.5 mngfngndng
> 3.6 ahsd fakj fkj
> 3.7 akjhsd g gmj dfg
> 3.9 ghskdfhgkhsfkjdgh
> 3.10 jahsdgfhfj akj

I've set it up so that when it gets to the end of a list, it doesn't search too far through the text before it finds something that's 'not consecutive'. The distance it travels before it gives up is set by:

```
tooFar = 3000
```

That's 3000 characters, so about 500 words.

Sub NumberSequenceCheckerDecimal()

Number sequence checker hierarchical

(Video: youtu.be/DnG1XCuOUlk)

If you have a long, numbered list and you want to check that there aren't any missing (or extra) numbers, simply place the cursor on the first item (or indeed any item further down the list) and run the macro. It will do its best to see what the pattern is (e.g. a number and a space, or a number and a tab), and then go down the list number by number.

If it finds an error, it stops – and optionally highlights the numbers that it thinks are not in sequence. You can sort it out and then start again from whichever item you like.

It will even work with, say, Figure 1, Figure 2 etc, or Table 1, Table 2 etc.

What's more, if your document has 'dotted' section numbers (or figure or table numbers) at various levels (e.g. 2.3.6.1), then the macro checks the numbering, in and out of all of the different levels of heading.

In use

If you start from a line that says, for example

Figure 2.4 This is a picture of an elephant

It looks to see what comes before the number ("Figure<space>") and what the character immediately following is (here a tab), and it looks to see that the next "Figure " (or whatever word is used) is either 2.5, or 2.4.1 (more likely with section numbering than with figures).

So if we were to start at, say, section 3.4.5.2, it would check whether the next one was 3.4.5.2.1 or 3.4.5.3, or 3.4.6 or 3.5 or 4.1.

And there's even an option (`allowSingleNumbers = True`) to go down to the next single numbers – in this example it would be 4. The only trouble with that is that it's more likely to get confused and find an 'error' in the consecutivity that is just some other text within the flow of the document.

And it also tries to cope with the fairly frequent case where a paragraph starts with, say, "Figure 3.4 shows that...". This wouldn't be a problem if the captions used a tab after the fig number – as above – but if the caption uses a space, the paragraph might then look like a caption. To try to give some discrimination, there's a parameter (`captionWordsMax = 30`) that sets the maximum number of words which it considers to be a *caption* as opposed to a *paragraph*. Clearly, this might need to be increased if you were checking, say, a references list.

Errors to watch for

If the program stops, and it has not reached the end, it highlights the last 'acceptable' number (turquoise) and the number it thinks is erroneous (red). Now, if, for example, a heading has a space instead of a tab, it will be ignored, so the macro will jump to the *following* heading and say that *that* heading is out of sequence, so always check the previous heading, as well as the one on which the macro stops.

When the macro stops, the Word's Find function is set up for you with the wildcard find needed to jump from heading to heading, so you can jump back and forth between the 'headings'.

It might actually be worth having formatting showing (Ctrl-Shift-8) because spaces and tabs are crucial. For example, a redundant space after the number and before the tab would be invisible but would cause an error. (But then, as part of your tidy-up of the file, you will presumably have changed '^32^t' into '^t', won't you?!)

Phantom errors

False positives can sometimes be caused by track changes in the heading, so make sure that track changes are visible, and then just try accepting the track change on the heading, then go back to the previous heading and run the macro again.

The macro can sometimes have problems when it gets into tables, and it may well not be able to cope with textboxes. It *should* navigate its way out of each table, but if it gets stuck in a loop, stop the macro (Ctrl-Break[†]), and click on 'End'. Then manually check the heading number above the table, move past the table to the next heading and carry on from there.

([†]If your keyboard doesn't have a Break key, you can still stop a macro mid-program. If you run the macro with the VBA window open and visible on screen, then you can use the stop '■' icon to stop the macro running. **STOP PRESS!** I've just discovered that, while a macro is running, yes, don't move the mouse, but you can use the keyboard – press Alt-F11, VBA will then open, and you can press pause '‖' or stop '■'. Yay! Result!)

If you have a file on which the macro founders, and you'd like to use the macro for similar jobs, please send me a sample file (confidentiality permitting, of course), and I'll try to find a way of enabling the macro to cope with the extra complications.

Sub NumberSequenceCheckerHierarchical()

Contents list creator by style or number or tags

(Video: youtu.be/DnG1XCuOUlk)

Continuing on the theme of numbering, it can be useful sometimes to create a contents list of a whole document. Here are two macros that do that. The first tries to create the contents list on the basis of the style, specifically paragraphs that are in styles such as 'Heading 1', 'Heading 2' etc.

The second works on the basis of hierarchical heading numbering.

If the heading levels are tagged <A>, <B> etc the third creates a contents list based on those tags.

I wrote the macros some long while ago (and forgot to put them in my book) in order to check and compare ***all*** the headings, rather than to actually generate a real-live contents list, so neither macro currently has the facility to exclude headings at lower levels within the hierarchy. So, if your client wants a contents list of only, say, level 1 and level 2 headings, rather than sit there deleting the lower level headings, let me know, and I'll add in an exclusion option.

Sub ContentsListerByStyle()

Sub ContentsListerByNumber()

Sub ContentsListerByTag()

Find mismatched parentheses

This macro EITHER goes through the text (starting at the cursor) a paragraph at a time and checks to see if any of them has unpaired brackets, i.e. a different number of open and close brackets (parentheses, I mean). It reports how many such paragraphs it has found, and these paragraphs have the underline attribute added.

It also sets up the F&R so that is ready to find any underlined text, so you can skip through them one by one.

OR it starts from the cursor and searches out the next possible mismatch, allowing you to ignore it and continue, or stop and correct it. For this mode, set at the start of the macro:

```
stopEachTime = True
```

Sub MatchParentheses()

And a square bracket version.

Sub MatchSquareBrackets()

Find mismatched double quotes

This macro goes through the text a paragraph at a time and checks to see if any of them has unpaired double quote marks, i.e. either (a) a different number of open and close curly doubles, or (b) an odd number of non-curly double quotes. It reports how many such paragraphs it has found, and these paragraphs have the underline attribute added.

Sub MatchDoubleQuotes()

Find mismatched single quotes

(Latest version allows it to be used in Dutch, which uses single quotes in far more places than English.)

This macro goes through the text a paragraph at a time and checks to see if any of them has unpaired single quote marks, i.e. either (a) a different number of open and close curly singles, or (b) an odd number of non-curly single quotes. It reports how many such paragraphs it has found, and these paragraphs have the underline attribute added.

BUT, as you've probably just realised, this isn't as easy to check the single quotes because of apostrophes, as demonstrated by this paragraph! But I've tried to cover as many eventualities as possible.

It underlines all "suspect" paragraphs, but if it's suspect because there's an s-apostrophe, it highlight just the s-apostrophe, whereas if it's more sure there's a problem, it highlights the whole paragraph.

If you want to use it for other languages that use single quotes in different places, you can adjust the line at the beginning of the macro:

```
myList = "'s,s','t,'v,'r,'l,'m,'d,'y,'c": UK list
' myList = "'i,'k,'m,'n,'s,'t,'r,'n": Dutch List
```

```
Sub MatchSingleQuotes()
```

Correct double quotes inside double quotes to singles

If the author has used double quote marks inside double quote marks then this macro will go through the whole document and replace the ***inner*** quote marks with singles.

```
Sub QuoteMarkEmbedder()
```

Section number adding

Another macro linked to the ones above is where you've got to add consecutive section numbers to the subsections following a section heading. Here's an example text:

> **1.9.6 This Title has a Number Already**
> **This is a subheading**
> Here is the text in this subsection etc, etc...
>
> **Here is another subheading**
> And some more text etc, etc...
>
> **Then yet another**
> Here's yet more text etc, etc...

So, to use this macro, you select '1.9.6', copy it with Ctrl-C and run the macro. Because some text is selected, it knows you're in setup mode, so it suggests a start number for the series of headings. You can press Return to accept the offered number, or enter a different number. In this case you'd want it to be '1', but you might want to add some heading numbers to an existing run.

You now place the cursor somewhere on the first subheading and run the macro. It will add '1.9.6.1<tab>' to the start of the line. Then place the cursor somewhere on the second subheading and you automatically get '1.9.6.2<tab>', and then on the next '1.9.6.3<tab>'.

Now suppose that you have a run that ends with, say, 2.2.5, and you want to continue it, select the '2.2' and copy it – the base number – then select the '5' and run the macro. It reads the '5' and offers you '6' as a possible next number. So, if you accept that and then click on the next heading, it will, indeed, be numbered as 2.2.6.

```
Sub AddSectionNumber()
```

Automatic section numbering

If the text has headings styled as Heading 1, Heading 2, Heading 3 etc then this macro adds the hierarchical section numbers, starting from the first 'Heading 1' style paragraph.

However, if you have prelims that use Heading 1, you can avoid them getting numbered by putting `chapWord = "Chap"` at the beginning of the macro. (This is assuming that each chapter starts with 'Chapter 1' etc, but if you don't have a standard word for each chapter, I'll have to make a small rearrangement – let me know.)

If the file starts from, say, chapter 4 then change to `chapNum = 4` at the beginning of the macro.

If you don't want the Heading 1 heading to be numbered, but just to have the Heading 2's as 1.1 etc then put `addChapterNum = False` at the beginning of the macro.

Sub NumberParasAuto()

Automatic section numbering (2)

In this case, the numbering is for tagged headings, and for first level headings only ('cos that's what the person wanted, who asked for it). It reads the chapter number from the first line of the file, which was, e.g. "<cn>3". The tag text is set in the line: `myTag = "<a>"`.

Sub NumberParasTagged()

Semi-automatic section numbering

(Video: youtu.be/1O8Q-3ys1uo)

I had a job where the headings were not numbered, and they weren't in the correct case (the client wanted title case, and the authors had used sentence case). Also some of the headings had full points and other punctuation at the end of the line. In some chapters the only indicator of heading level was the font size so I created two new macros, the first of which jumped down to the next heading (i.e. paragraph) that was in a font size larger than the Normal font.

The main macro here is one that types in a section number, plus a tab, and formats the heading, removing any stray punctuation.

So I jump to the next heading then run this numbering macro. It tries to give me the section number that it thinks is right. If it's not, I can type into the input box the number I want, e.g. 3.1 and press Enter.

Then I jump down to the next heading (or just click in it if it's visible on screen) and run the macro. It offers me 3.2, so if it's the same level heading then I just press Enter and carry on to the next heading, for which it offers me 3.3.

But if the next is a lower level heading, I type "-" and press Enter, and it gives me 3.2.1.

Maybe the next is the same level, so I accept 3.2.2. But then if the next heading is a higher level, I type "+" and Enter, and it types in 3.3.

If I get a section number wrong, I can just delete the number, rerun the macro and type in the correct number.

However, I don't trust myself to get it right! So instead I can go back up to the previous heading and run the macro; now, because the heading is already numbered, the macro knows to ***read*** the section number, and not add another one.

Then I can move back down to the incorrectly numbered heading, delete the number and run the macro; it will then increment the section number it has just read, and I can carry on as before.

If I need to move up ***two*** levels of the heading hierarchy, I can just type "++".

As it stands, the macro asserts title case, but if you want sentence case (or you don't want any changes of case), you can alter the settings at the beginning of the macro:

```
doSentenceCase = False
doTitleCase = True
```

Sub TypeSectionNumber()

Sub FindNextBigText()

Do 'such and such' to every 'so and so'

Sorry, that sounds a bit weird, but this macro is a 'shell' for the more adventurous. You want to go through a text finding something specific using F&R, and then you need to do something to each one – i.e. something more than just replacing it with something else.

You'll have to have enough macro programming to know how to do to the text whatever you want to do to it, but at least this gives you the skeleton. The format is:

```
Find 'something'
Do (if you've found one)
    Make some changes to it
    Find the next 'something'
Loop
```

I hope that makes sense.

As it stands, it looks for each 'e' (or 'E') and if it finds it, it uppercases (or lowercases) it and adds a yellow (or red) highlight. (A silly example, I know, but it illustrates the If...Then possibilities.)

Sub FindAndDo()

Sample practical FindAndDo application

One reader discovered that some global F&Rs trip over the track changes. 'I want to change the hyphen in number ranges to a dash. So that's Find: ([0-9])-([0-9]) and Replace with: \1^=\2. The macro works until I turn on track changes on and then "4-9" changes to "49–"! How can I avoid that?'

The answer is that you have to use *FindAndDo* to find each occurrence and the select the hyphen and type a dash instead.

Sub SampleFindAndDo()

Multiple choice answer tidier global

If you have a file full of answers looking like this:

8. Electron transport occurs in the cells:
A. Nucleus.
B. Mitochondria.
C. Cytoplasm.
D. Golgi apparatus.
E. Plasma membrane.
Answer: B
Type: MC
Points: 1

And you want all the answer items to look like this:

8. Electron transport occurs in the cells:
A. nucleus
B. mitochondria
C. cytoplasm
D. golgi apparatus
E. plasma membrane
Answer: B
Type: MC
Points: 1

i.e. to strip off all the rogue spaces/full points etc off the ends and to lowercase the initial letter, then the following macro will go through the whole file and do so.

(OK, yes, you'll have to go back and uppercase words like Golgi, but that should take less time than doing this item-clean-up manually.)

It assumes each item is A B C D or E followed by a full point followed by a space or tab, and it has a list of erroneous characters that it will strip off the end. These are set in the macro. Please adjust to taste.

As standard, it also highlights the items it has changed, but you can switch that off with: `myCol = 0`.

`Sub MultiChoiceTidierGlobal()`

If you prefer to do this one question at a time, there's a second version.

You run it once and it jumps to the first set of answers, conditions them and stops. If you're happy with the result (i.e. no need to uppercase, say, Gogli), then you just run the macro again, and keep running it until you get a question where you need to change something.

`Sub MultiChoiceTidierSingle()`

Highlight all serial commas (or not serial commas)

(This macro could well be superseded by *SerialCommaAlyse*.)

This assumes that you want to have drawn to your attention, as you read, any text that might be using a serial (Oxford) comma when the brief says 'no serial comma'. Or contrariwise highlight the not-serial-comma text if the brief tells you to use serial commas.

It's not obvious to the computer exactly what is and is not a serial comma; consider for example:

1) "I like fish, chips, and peas"

2) "The job entails drilling a hole, countersinking it to the correct depth, and inserting the screw."

3) "The job is difficult, so you will have to be very careful how you approach it, and it takes a long time."

So what the macro does is to look for the pattern of commas and words plus the word 'and', and it then checks how many words there are in the section of text it has found. If there are too many words, it ignores it.

```
maxWords = 10
```

Increase the value of `maxWords` and you get more false positives, but miss fewer actual (not)-serial commas, and vice versa.

The macro does the test for both 'and' and 'or'.

You can do the 'highlighting' either with underlining and/or actual highlights. This is set at the beginning of the macro, where it says, for example:

doUnderline = True

doHighlight = False

myColour = wdYellow

So you change it to, say:

doUnderline = False

doHighlight = True

myColour = wdYellow

and change the colour if you prefer:

myColour = wdBrightGreen

```
Sub SerialCommaHighlight()
```

```
Sub SerialNotCommaHighlight()
```

Count the serial commas (or not serial commas)

Running DocAlyse will give you an estimate of the number of lists that have a serial comma and the number that have no serial comma. However, it is only an estimate because what actually constitutes a list is difficult for a computer to accurately judge, so suppose you end up with (as I did on my last job):

serial comma	**7**
no serial comma	**9**

I needed a more accurate count, so I wrote this macro. To use it, first run *CopyTextSimple* to create a nice clean version of the text – after all, it's only the words that you are assessing, not the formatting (better still, especially for large files, use *CopyTextVerySimple*).

Then, when you run the macro, it identifies strings of words and commas that ***might*** be lists. While it is doing so, it reports how many look serial-like or and how many do not, but note that this may well be a ***less*** accurate count than DocAlyse produces.

The macro then gives you the option to decide which of these really are lists by checking the context. (To do this more easily, open the window quite wide, and place the top LH corner of the window up in the top corner of the screen. This makes sure that the prompt windows that the macro throws up do not obscure the text you're trying to assess.)

As you go through, saying yes or no to the question "Is this really a list?", it will tot up the numbers of serial/not serial lists, so you can carry on doing this until you feel you've got a good enough assessment.

You click Cancel instead of saying yes or no, and this will drop you out of the macro. But then if you restart the macro it will take up from where you left off, giving you the current count, up to that point, and you can check some more potential lists. You will see that it has coloured the lists green or yellow for serial or not serial, respectively

In the example of the file that prompted me to write this macro, the initial count by the macro gave:

serial comma	**9**
no serial comma	**25**

and then when I had gone through and checked all the potential lists I got:

serial comma	**9**

So, compared with the 7/9 estimate that DocAlyse gave me, I really had got a clearer answer to my question.

(In fact, I didn't write this macro until ***after*** I had sent the file back to the author, because it was urgent. Fortunately, based on DocAlyse's 7/9 count, I chose no serial comma!)

Sub SerialCommaCounter()

Highlight duplicate sentences

This macro looks at every sentence in the current document and checks that this same sentence doesn't occur anywhere else in the document. If it finds duplicates, it highlights them both, the first in grey, the second in yellow.

It ignores sentences shorter than 10 words, so as not to pick up duplicate headings – but you can adjust that to taste.

Sub HighlightDuplicateSentences()

It's pretty slow, so I've tried another technique that's faster, but it has some limitations. Do give the two of them a go, and see how you get on and report back to me if you think one or other is better.

Sub HighlightDuplicateSentences2()

Check and correct the hierarchy of brackets/braces/parentheses

The aim of this macro is to check and, if necessary correct, hierarchical enclosures – {, [and (– using whatever order your client wants.

The macro starts from the current cursor position and works its way down through the text, correcting the order of the enclosures used. However, if there is a section of text where the number of open enclosures is more than three, or if the number of close enclosures is too few or too many, it stops, highlighting the region in which it thinks there's an error.

You can then correct the mistake and start up the macro again from that point onwards.

If there are areas of text (such as quotations or references lists) that you ***don't*** want changing then, as with many of my macros, you can apply the strikethrough attribute, in which case the macro ignores that section of text.

The actual hierarchy used is set at the beginning of the macro with lines such as:

```
'myOrder = "{([])}"
'myOrder = "([{}])"
myOrder = "{([])}"
```

so just add or delete the 'comment' apostrophes, according to taste.

The macro will work with track changes switched ON, but be very careful because track changes can easily cause problems.

The most important principle for using it with track changes is: only run the macro ***once*** on any given section of text. The reason is that once some of the enclosures have been edited, the F&Rs that the macro uses will 'see' all the enclosures – the originals, and the replacement ones – so it can't work out what's what.

So this is my suggested procedure with track changes.

1) Move the cursor to the top of the document and run the macro. When it finds a set of enclosures that don't match, it stops and highlights the area within which the error lies.

2) Correct the number and hierarchy of enclosures yourself (with track changes still ON).

3) Move the cursor immediately past that set of enclosures and run the macro again.

Sub EnclosureFixer()

Remove all formatting except URLs

Someone wanted a clean, text-only version of a Word file, but with the URLs still blue and underlined.

Sub FormatRemoveNotURLs()

Apply global highlighting with track changes

If you want to apply a highlight to a specific word/phrase in a document, but ***want the highlighting tracked***, you can't do it with global F&R. Even if track change is on, global F&R will ***not*** track the highlighting. So you have to use a trick.

Use a dummy attribute that is not used anywhere else in the document, and apply ***that*** to the word/phrase(s) using FRedit. Then run this macro, which uses the find-and-do technique to find each of these attributes, switch track changes off and remove the attribute.

I've set it up so that you can choose to use any (or all) of three different attributes to give three different colours of highlighting. So at the start of the macro you have, for example:

```
myAllCaps = wdYellow
mySmallCaps = wdBrightGreen
myUnderline = wdNoHighlight
```

which means that any text in allcaps will go back to not-allcaps and be highlighted in yellow, any text in smallcaps will go back to not-smallcaps and be highlighted in bright green; however, any text that is underlined will be left untouched. (The 'wdNoHighlight' just gives the value zero, so you could equally well use: `myUnderline = 0`.)

So a sample FRedit list for this would be:

| 'and', using allcaps
AND|^&

| 'you', using smallcaps
YOU|^&

DoMacro|HighlightWithTrackChange

I realise that allcaps and smallcaps can be a bit confusing in the FRedit list, and so I've provided the option to use underlining, though I realise it's often us for URLs such as: www.archivepub.co.uk.

Sub HighlightWithTrackChange()

Apply highlights and/or colours to 'confusables'

(Video: https://youtu.be/GPa1ItHFfCc)

This macro was generated to assist someone who wanted to draw attention to homophones and homonyms within a text, to provide a prompt to check that the word was indeed the right one. However, the macro is 'content free', i.e. you can use it for whatever words/phrases you want.

Simply create a list of the words/phrases and colour or highlight them according to taste. For example:

assent
ascent

horde
hoard

premier
premiere

N.B. If any of the words in the list are both uncoloured and have no highlight, the macro will do what you ask – apply **no** colour and **no** highlight to those words! Just saying. :-)

Now save this file with the name 'Confusables' in a folder of your choice, and then add the full address of that file into the macro:

```
myFile = "C:\Documents and Settings\Paul\My Documents\Confusables.docx"
```

To avoid highlighting words within words, at the beginning of the macro it says:

```
onlyColourWholeWords = True
```

However, if you have some reason to want to use this for part-words, simply change it to `False`.

Sub Confusables()

Correct accidental double capitals

HEre's a macro to correct a very common typing error. It looks for occurrences of a double initial capital and corrects them to a single initial capital.

You can highlight and/or font-colour the changes according to the settings at the beginning of the macro.

Sub CapitaliseUndoubler()

Correct spaces and punctuation on superscripted numbers

Suppose you have references or footnotes callouts that aren't linked fields but rather are just pure text, superscripted numbers, like this [23]. So if, as in this example, there is a rogue space in front of the number and/or the punctuation comes after the number, rather than before, then this macro corrects them, right through the document.

Sub SupercriptNumberFormatter()