September 2015 Issue 000

NEWSLETTER

For professionals working with text

OUT OF OFFICE

Welcome to the first edition of **OUT OF OFFICE**, a monthly newsletter for professional translators, editors, reviewers, authors and anyone working with texts for professional purposes.

I apologise in advance for this rather lengthy editorial, but as it is the first issue, there is a lot to say. Future editorials will be much shorter.

This newsletter deals exclusively with using Microsoft Office for work purposes. It is not intended to glorify or criticize this software suite, but rather to look at our main working tools, understand behaviours, and find solutions to specific daily issues. An important part of this newsletter will be a look at VBA (Visual Basic for Applications) and how to write macros to automate repetitive tasks that will accomplish routine tasks as quickly as possible – but from a linguist's viewpoint rather than the programmer's.

OUT OF OFFICE is composed of sections, each one focussing on a different component of MS Office:

- **OH MY WORD!** will provide tools, tricks and tips about Word®;
- **ALWAYS EXCEL** will do the same for Excel®;
- **POWER TO THE POINT** will discuss PowerPoint®;
- **THE OUTLOOK LOOKS GOOD** will deal with Outlook®;
- **VBA FOR EVER!** will suggest solutions and routines to automate repetitive tasks;
- **ANYTHING TO ADD-IN?** will investigate some useful tools available to help us in our daily tasks.

Each issue will contain at least two of those sections and will provide real solutions to ordinary issues. The solutions and advice given should apply to all post-2007 MS Office® versions in the Windows® environment. For instance, the shortcuts given are Windows® based (provided you have not redefined them yourself) and will not work in a Mac® environment. Users of other environments can try the suggested solutions and let me know whether they worked for them by sending an email to outofoffice.newsletter@gmail.com with "I tried this in a non-Windows® environment" in the subject line. Please include screenshots of any error messages you see, so that I can try to provide a solution to the issue.

If you would like to see a specific issue explained in this newsletter, please do not hesitate to send an email to outofoffice.newsletter@gmail.com with "How do I…?" in the subject line, a detailed explanation of what you would like to achieve, and if possible sample files.

This newsletter will not advertise products or services and will not send you promotional material. Once a month, subscribers will receive a zip file containing the PDF version of this newsletter accompanied by .xlsm, .bas, .frm or .frx files related to the content of the newsletter.

This newsletter can and should be counted as part of your **continuous professional development**. Nowadays, the Internet is a source of much free content. The only drawback is that you probably need a lot of time to find what you are looking for. Nothing will ever replace your personal know-how. You are the best person to understand the issues you need to resolve. Which is why, instead of creating video content or blogs, this newsletter aims to teach you additional skills that will enable you on your own to sort out some of the issues you encounter in your daily work. Should you still be unable to do so, you can send an email with the "How do I…?" in the subject line.

This first issue is free, but subsequent issues will be sent to paying subscribers only. For a modest fee of 20.00 GBP per year, you will receive 1 issue every month. Please register your interest in writing to the above address with "Subscribe" in the subject line, specifying your desired language (English, French, German and/or Spanish). You will receive an NDA (non-disclosure agreement) stating clearly that this newsletter is not to be made public and that its content may

not be placed into the public domain, in whole or in part, without our prior written consent. However, you can use the tips and tricks it contains for your professional use. Payment for the newsletter subscription is solely via Paypal. Any payment made indicates your clear acceptance of the NDA and of the consequences of breaching it.

Half of all profits will be given to charity organisations (Solidarités, Translators Without Borders Water Aid and Médecins Sans Frontières).

Enjoy this issue!

The OUT OF OFFICE team

Conventions

This blue font formatting will be used when we are talking about elements of the ribbon and other software options.

> Shortcuts are inserted within paragraphs using this format.

This red font formatting will be used when we are talking about key combinations. If the combination requires you to press two or more keys at the same time, there will be a hyphen between the keys in question, for instance **Ctrl-F** to open the basic find function as opposed to **Alt F** to access the "**File**" tab on the ribbon. The following abbreviations will be used to refer to keys on your keyboard:

- Alt — the left hand side Alt key
- AltGr — the right hand side Alt key
- Ctrl — any of the control keys
- Shift — any of the shift keys
- AppK — the menu key located on the right of the AltGr key with a little drawing on it
- Return — the carriage return key
- Tab — the tabulation key

For readability purposes, all letters on the keyboard are mentioned in uppercase, but if they MUST be in uppercase, this is indicated by a mention of the Shift key.

References to the number keys (not the keys in your numerical pad) where using the Shift key will give another character are made by specifying the character and the use of the Shift. For instance, using the "=" key to put the selection in superscript will be written **Ctrl-Shift-+**.

References to Unicode characters are made using **Alt-combination number**. The combination number refers to the numerical pad keys (which requires that the Num Lock be engaged).

`This font formatting` will be used when referring to anything that is associated to encoding, whether VB, XML or SQL.

```
All code will be contained in paragraphs using this format.
'However, comments in the coding will start with an apostrophe and will
follow this format (green font)
```

Screenshots are taken from our version of Office 2013 and are given merely to illustrate the location of items in the software. Your version of Office might be different and/or your ribbon might be customised. The little green circles or rectangles surrounding the items in question are not part of the Office suite; they are simply made to help you to quickly see the subject at hand.

Comments and notes are made in this format. They are not part of the text flow, but they contain important remarks concerning the subject at hand.

In this issue:

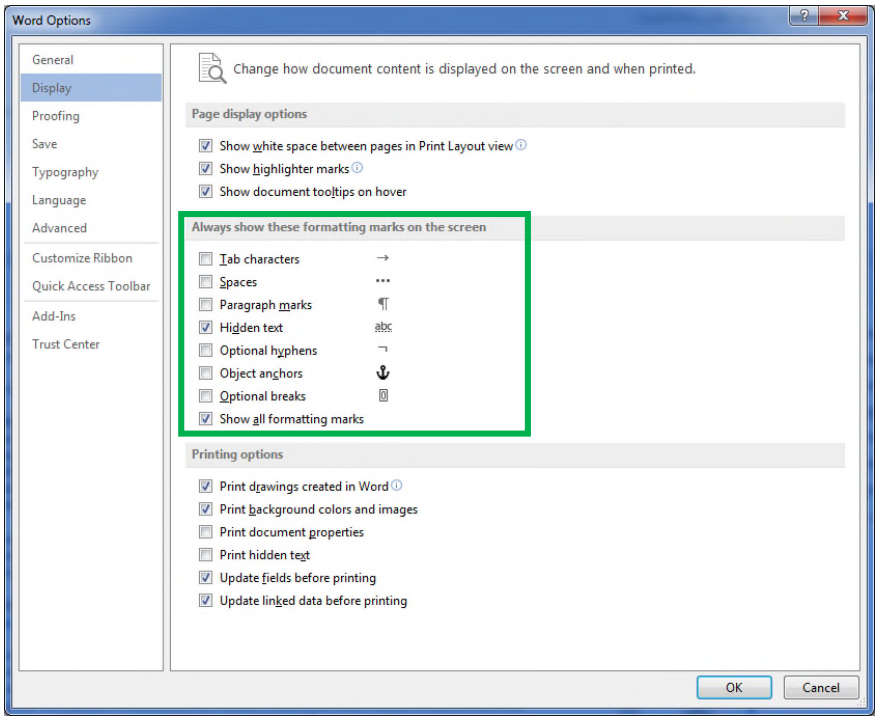
In the next issue

As this is the first issue, let's start with the basics.

Showing formatting marks

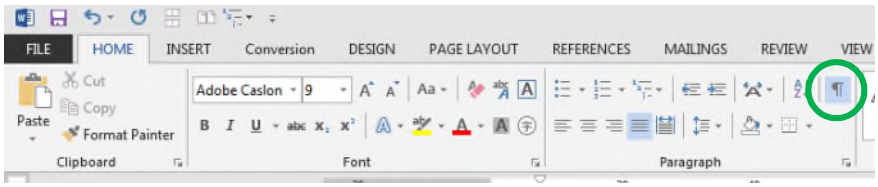
It is good practice to see what is really on your page, particularly what is not printed and not visible. Some people might find seeing spaces, tabulation marks and the rest quite of a challenge, but it is important for anyone working with text. Remember that you can always toggle this function on and off, depending on your needs.

In the ribbon, go to **File > Options** and in the section "**Display**", and select the elements you want to see.



In the central section, "**Always show these formatting marks on the screen**", you can decide which marks you want to be able to see and for each option, on the right hand side, you see how each mark is represented on your screen.

Most users would prefer to use the icon on the ribbon, located in the "**Home**" tab, in the "**Paragraph**" group, to toggle this option.



You will notice that this icon is shaded in blue when toggled on.

There are limitations to using only the ribbon icon to show or hide formatting marks.

In the first screenshot, two options are selected: "**Hidden text**" and "**Show all formatting marks**". The icon on the ribbon affects only the "**Show all formatting marks**" which means that any other options that you may have selected in

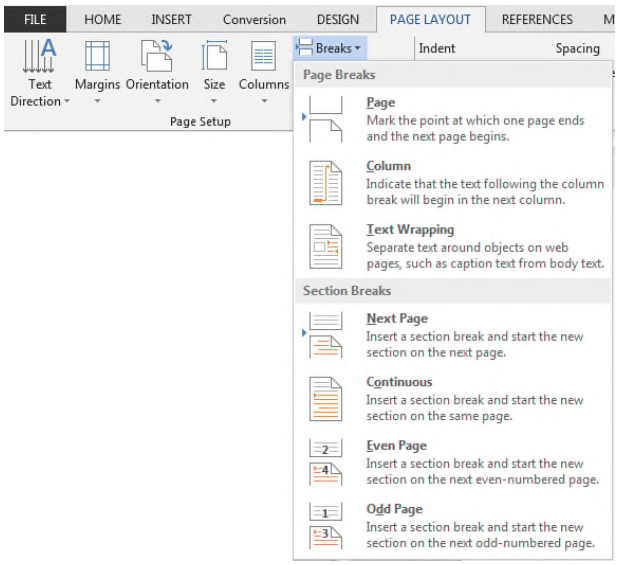
the option window will not be affected. In this case, the "**Hidden text**" will remain visible whether or not the icon is toggled on or off.

Why are formatting marks useful?

There are many reasons why it is useful to see the invisibles. Among other things, you can:

a) Distinguish between page breaks and section breaks.

To insert a page break, in the ribbon, tab "**Page Layout**", section "**Page Setup**", "**Breaks**", the first option "**Page**" will insert a page break.



To insert a page break, it is much faster to use the shortcut **Ctrl-Return**.

To insert a section break, select one of the lower options.

There are no shortcuts for section breaks, but you can customise one for your needs.

Alternatively, you can press the **Alt**-key to activate the keyboard control of the ribbon and then press **P** (to access the "**Page Layout**" tab), then **B** (to activate the "**Breaks**" option in the "**Page Setup**" section) and select **N**, **O**, **E** or **D** depending on your needs.

One advantage of a section break is that you can define, amongst other things, new headers and footers for a section of your document.

b) See the properties of section breaks.

As mentioned above, there are 4 types of section break and when the formatting marks are displayed, provided there is enough space after the paragraph mark, you should be able to see what type of section break was used.

¶ Section Break (Next Page)

¶ Section Break (Continuous)

c) Distinguish between line returns and forced line returns.

A forced line return is identified by this symbol ↵

To insert a forced line return, use **Shift-Return**.

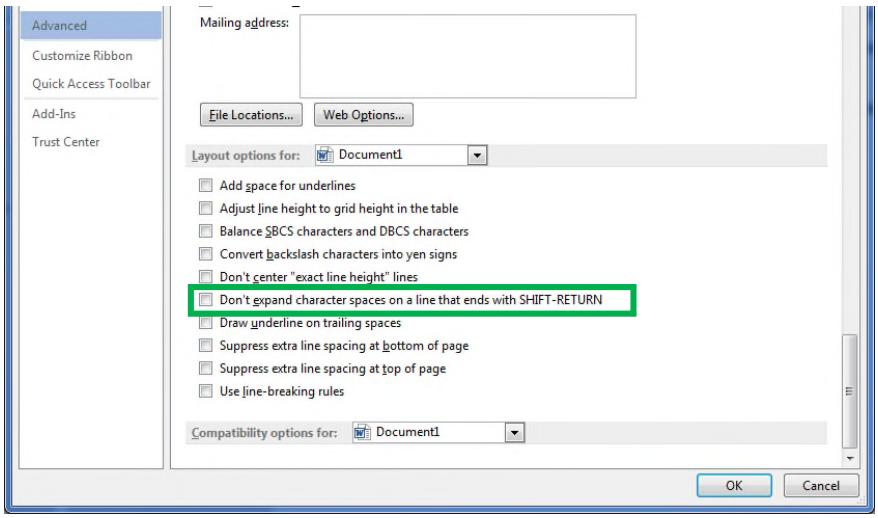
The normal end-of-paragraph mark or line return is ¶.

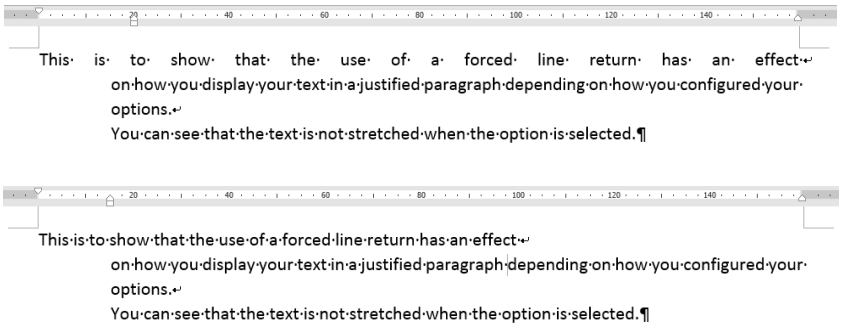
The difference between a forced line return and a normal line return is that a paragraph is defined, among other things where there is:

- a **First Line Indent** (the starting position of the first line compared to the margin).
- a **Hanging Indent** (the starting position of the other lines of the paragraph).
- an **Alignment** (left, right, centre, justified).
- a **Spacing** (Before, after, between lines).

A forced line return means it is still the same paragraph and, as such, your line will start wherever the **Hanging Indent** is defined (not the **First Line Indent**). Furthermore, you cannot change any properties of that part of the paragraph without affecting the entire paragraph.

If your paragraph is defined as "Justified", the text will stretch to the end of the line unless you have defined this default behaviour otherwise in **File > Options > Advanced** section "**Layout**", option "**Don't expand character spaces on a line that end with SHIFT-RETURN**".





d) View how many tabs or spaces are inserted.

Although it is possible to automate a search-and-replace process for rogue multiple tabs and spaces, it might be useful to see what the text contains and to make an informed decision based on the specific context.

e) Locate the anchor points of textboxes and shapes.

An anchor point shows the point of insertion of an object that is not wrapped in line with the text. This object is contained in a different layer than the original base layer. To make it visible, the given object must be selected. The anchor point is represented by a small anchor, at the edge of the margin, by the side of the paragraph where the insertion was made.

It does not mean that the object will be near the anchor (it may have been moved). However it means that deleting this paragraph will also delete the associated object(s).

The anchor point can be selected with the mouse and dragged to another location without moving the object.

If you select multiple objects, the anchor points will not be visible.

If you select an entire paragraph and a single object is associated with it, then the anchor point will become visible. However, if there are several anchor points collocated at the same point, they will not be visible.

It is therefore good practice to locate all the objects contained in your document and ensure their anchor points are not associated with any paragraph you might wish to delete. Here, the word "Object" refers to drawings, shapes, text boxes and any other element that can be inserted and defined differently from "**In Line with Text**".

f) Edit hidden text.

Hidden text is represented by a small dotted underline.

This·text·is·visible¶

But·not·this·text¶

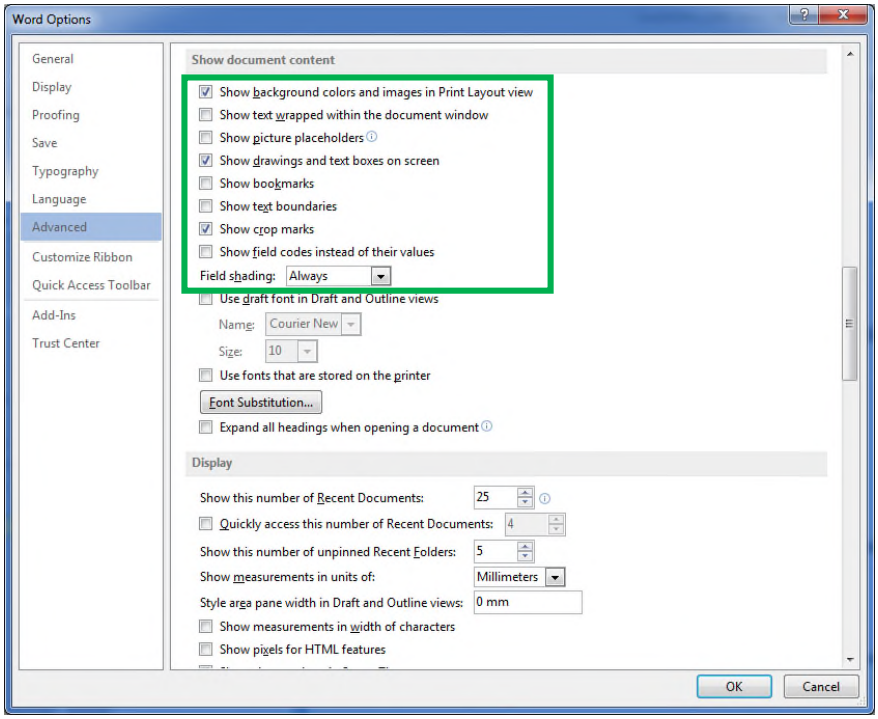
The only way to see the hidden text and edit it is to use the show/hide formatting marks. This can be very useful to check if the document contains hidden fields such as index entries or TC fields. We will be discussing fields in a later issue.

Fortunately, Word® has several options to underline text, with several types of dotted lines, but none are similar to the fine dotted lines of hidden text; hence, there is little chance of confusion.

Showing document content

There are marks that are not formatting marks, which, as people working with text, we might need to see. Some are essential for professionals - such as the ability to view text boxes, bookmarks, text wrapping, text boundaries and field content and shading.

To do so, you need to go to **File > Options > Advanced** in the section "**Show document content**".



Why is it useful to view the document content elements?

a) To get a proper rendering of the page content before printing

The option "**Show background colours and images in Print Layout view**" enables the user to have a true on-screen rendering of the document without having to ask for a "**Print preview**". However, as stated in the name of the option, this applies only to the "**Print Layout**" view.

b) To preserve the text wrapping when in outline view

The option "**Show text wrapped within the document window**" enables print layout wrapping to be preserved while working in outline view.

c) To ignore images

The option "**Show picture placeholders**" displays blank squares instead of the images. When working in "**Print Layout**" view with a document where either you do not need to look at the images, or where images/artworks are not ready for insertion, this option can be useful.

Microsoft® tool tip for this option indicates that this improves scrolling performance.

d) To view text contained in text boxes

Once again, this is an option that applies to the "**Print Layout**" view. The option "**Show drawings and text boxes on screen**" enables you to visualize objects (text boxes and shapes) that are on a different plane to the text (different layer). Other views such as "**Outline**" or "**Web Layout**" do not allow multiple layers and do not display them.

e) To view the location of bookmarks

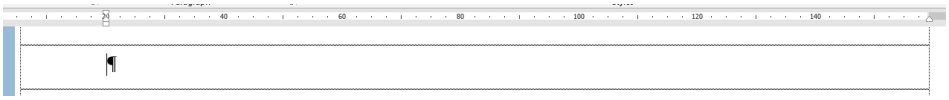
Bookmarks are extremely useful although few people use them to their full potential. The option "**Show bookmarks**" will show the location of the bookmark in your text not only in "**Print Layout**" view, but in all

views except "**Read Mode**". Making bookmarks visible ensures that they will not be deleted while reviewing or working on the document.

They are represented in the text by this symbol that cannot be selected:

f) To view text boundaries

The option "**Show text boundaries**", will display rectangles in dotted lines around each paragraph, from margin to margin and around shapes and text boxes. Once again, this only applies to the "**Print Layout**" view.



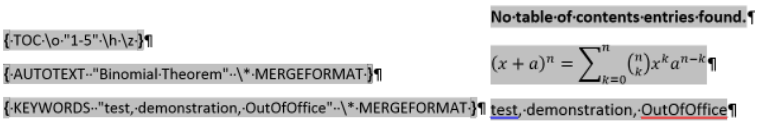
This will enable you to see the spacing before and after each paragraph.

g) To show fields

Fields are automated content that can come from the document itself - such as page number or the table of contents - or from outside data, such as other Word® documents or other MS Office® applications. In either case, it is always a good idea to make them stand out (provided they are not hidden in fields such as Index entries). The option "**Field shading**" allows you to decide the behaviour of the grey shading on fields.

h) To see the source of fields

Considering the great variety of fields, sometimes you might need to see what the field is and what it is associated with, regardless of its content.



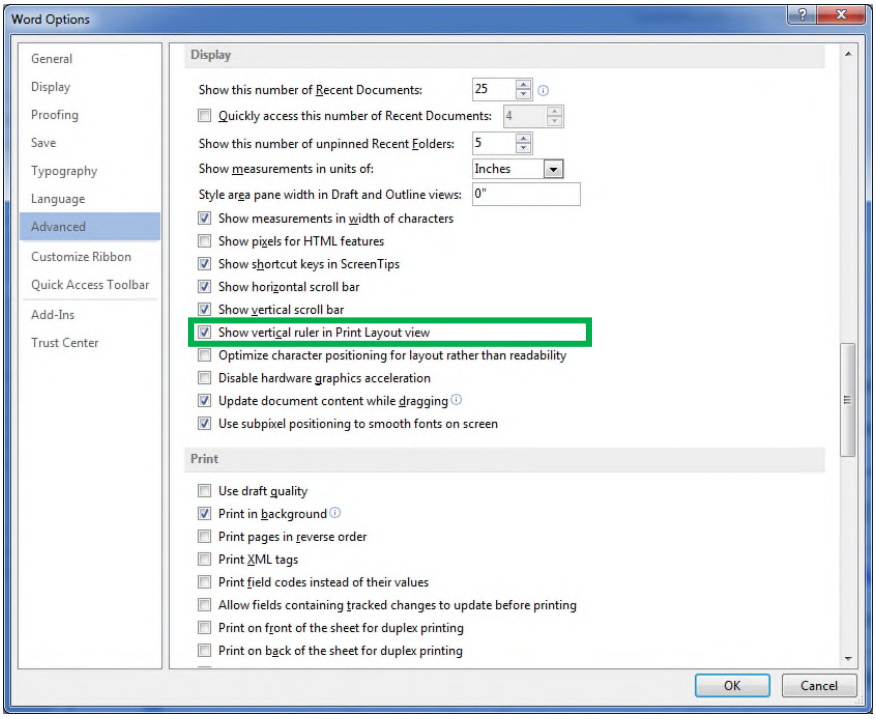
Depending on the type of work you need to do in any given document, you may need to change these options fairly regularly. In this issue, once you have learned the basics of VBA, you will learn how to write a macro that will enable you to set precisely the options you want for the work you need.

Show

This group located in the "**View**" tab of the ribbon contains three essential elements when working on documents.

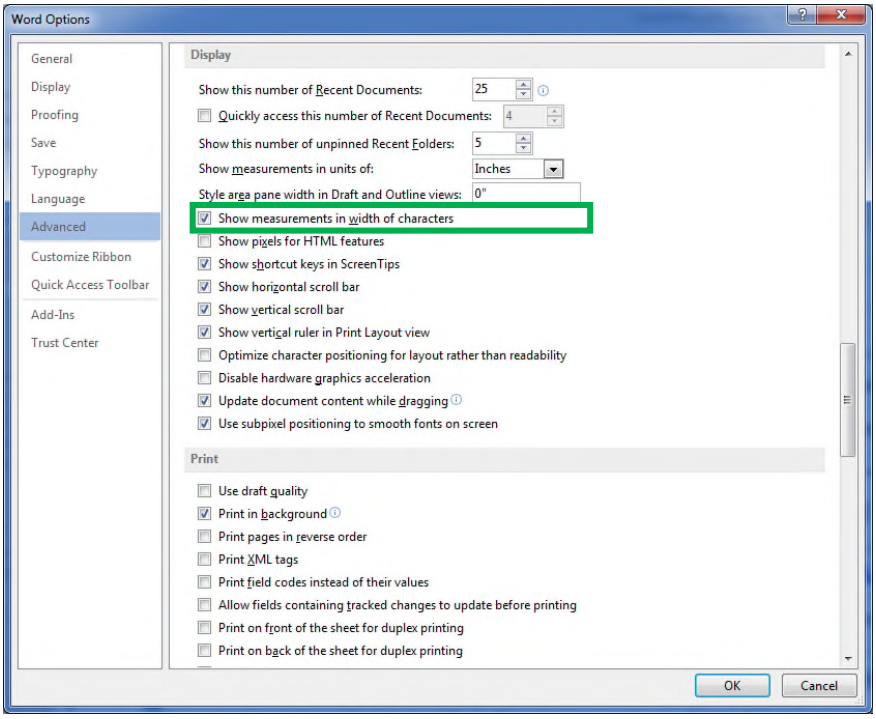
1. *Rulers*

The first option in **View > Show** is "**Ruler**". Despite being a singular word, it affects simultaneously two rulers, one horizontal and one vertical. However, the way the vertical ruler displays is not defined here, but is set in **File > Options > Advanced** in the "**Display**" section by ticking the option "**Show vertical ruler in Print Layout view**". This means that you do not have a means to display only the vertical ruler and that this ruler is only available in "**Print Layout**" view.



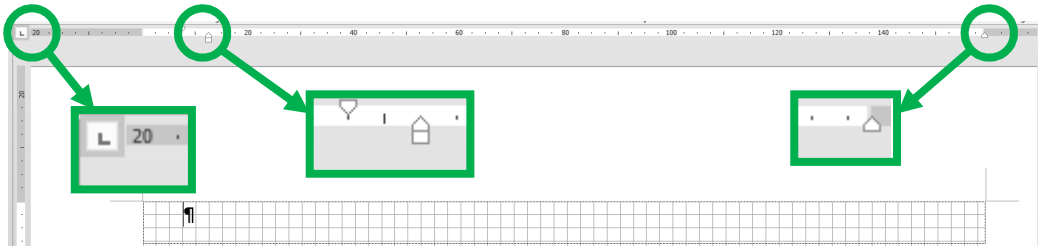
The numbers the ruler(s) will display use the measurement unit you have defined in your options. The numbering frequency also depends on your unit. For instance, if you decided to use inches, your rulers will show each unit with markers for every one-eighth of an inch, whereas if you opted for centimetres, your markers will be for every one-fourth of your unit.

There is one exception to this. If in **File > Options > Advanced** in the section "**Display**" you selected the option "**Show measurements in width of characters**", then, the number will display the number of character. Most fonts use different character widths, but some, such as courier, are monospaced.



The option "**Show measurements in width of characters**" is only available if you are working with Asian languages such as Japanese which use standard width characters.

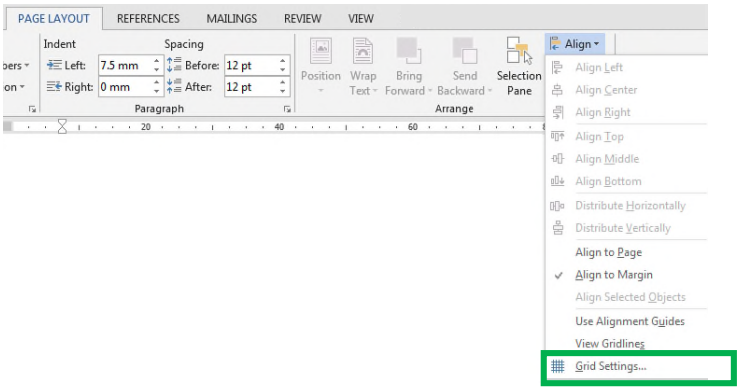
Besides numbering the distance between left and right margins and top and bottom margins, rulers also provide other extremely useful information. When your cursor is positioned within any given paragraph, the ruler will also show you the location of the First Line Indent and the Hanging Indent (both forming the Left Indent), the Right Indent and any defined tabulation mark. The extreme left of the ruler also contains a very useful tool that enables you to define the type of tab you want to use.



2. *Gridlines*

The second option in the "**Show**" group on the "**View**" tab enables you to see the "**Gridlines**". This is a useful tool to check that everything is properly aligned in your document. However, it can be tiresome for the eyes and you need to properly configure it.

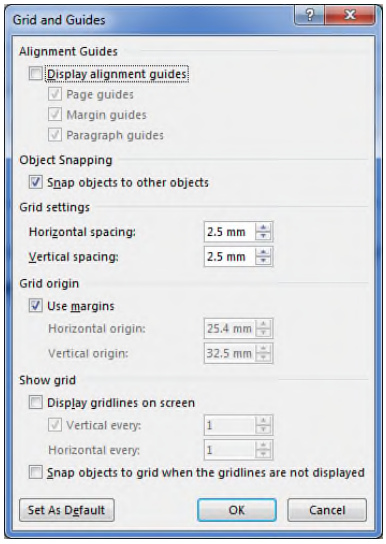
To configure it, you need to go to the "**Page Layout**" tab in the ribbon, in the group "**Arrange**", click on the "**Align**" option and select "**Grid Settings**" at the far end of the option list.



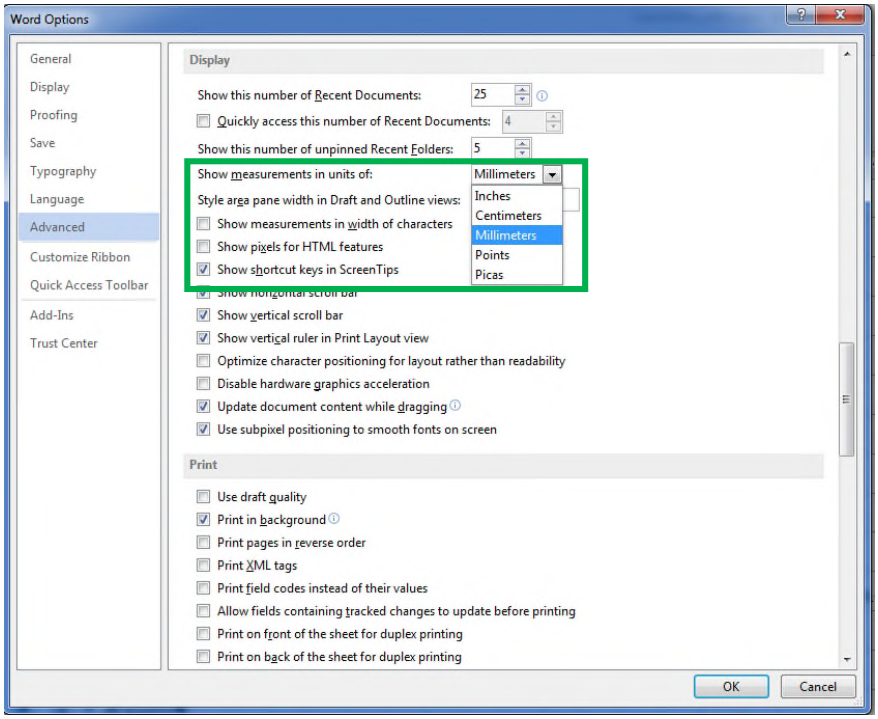
One option is to use the Alt key and to do the following key combination to access the grid settings without using the mouse: **Alt P AA G** (the absence of a hyphen between the key combinations means you do not need to keep the Alt key pressed)

This will open the following dialog box enabling you to define precisely your grid (spacing between vertical lines, spacing between horizontal lines, whether to display the grid and which lines to display, where the grid starts, etc.). If you make the grid visible, your working page will look like a page using squared paper.

If you prefer to work with a lined background, simply untick the option "**Vertical every**" in the "**Show Grid**" section under "**Display gridlines on screen**" (which must be ticked).



In the above screenshot, you can see that the measurements are in millimetres. You can define the type of unit you want to use for all your needs in **File > Options > Advanced** in the section "**Display**".



Whichever unit you decide to use, even for default options that use other measurements, you can always specify any unit you want and Word® will automatically change it. For instance, if you have no idea what a point represents, you can define paragraph spacing as "0.5 cm". You must include the abbreviated measurement unit Word® uses, but Word® will convert it to "14.2 pt" as soon as you click on the "OK" button.

3. *Navigation Pane*

The "**Navigation pane**" allows you to see the structure of the document. To activate it, in the ribbon, go to tab "**View**", group "**Show**" and tick "**Navigation Pane**".

You can access the navigation pane with **Ctrl-F**, which launches the basic search function. This will open the navigation pane in the tab "**Results**". Clicking on the tab "**Headings**" (default tab opening from the ribbon) or "**Pages**".

The "**Headings**" tab will show the document structure (outline) based on the levels of each paragraph (from 1 to 9). Anything at "Body level" will not appear. This is a practical tool enabling you to navigate quickly to the relevant paragraph level without having to scroll.

> This view is particularly useful to see what your table of contents could look like. However, as this view displays all paragraphs up to level 9, more often than not, you only want your table of contents to show the first three or four levels. There is no method of reducing the number of paragraph level displayed in this tab.
>
> You can also use this view very efficiently to work out your "**Multilevel list**". We will discuss multilevel lists with numbering in a future issue.

The "**Pages**" tabs will give a preview of the pages and provide you with the opportunity to navigate quickly through the pages. Unfortunately, you can only adjust the width of the pane and not the size of the page preview, which means it is difficult to read the text.

Find and Replace

As previously mentioned, shortcut **Ctrl-F** will open the navigation pane in the "**Results**" tab. This is the basic search function where you simply search for a specific chain within the text.

If you want to access more advanced search options, you need to launch the "**Find and replace**" dialog box.

Ctrl-H will open the "**Find and replace**" dialog box in the "**Replace**" tab.

Ctrl-G will open the same dialog box, but this time it is in the "**Go To**" tab.

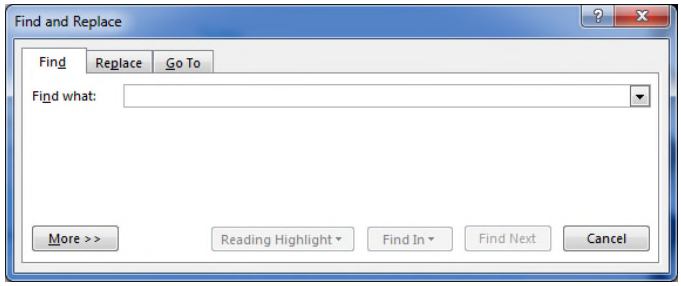
Alternatively, in the navigation pane, in your "**Results**" tab, on the right hand side of your search field, after the magnifying glass, you can click on the little arrow to see the menu and select any of the first four options available: "**Options…**", "**Advanced Find…**", **Replace…**" and "**Go To…**".

Clicking on "**Options…**" opens a different window that enables you to define some search options. The remaining three options will open the "**Find and Replace**" dialog box directly in the tab of your choice.

If you have used either of the shortcuts previously given, you can click on the "**Find**" tab.

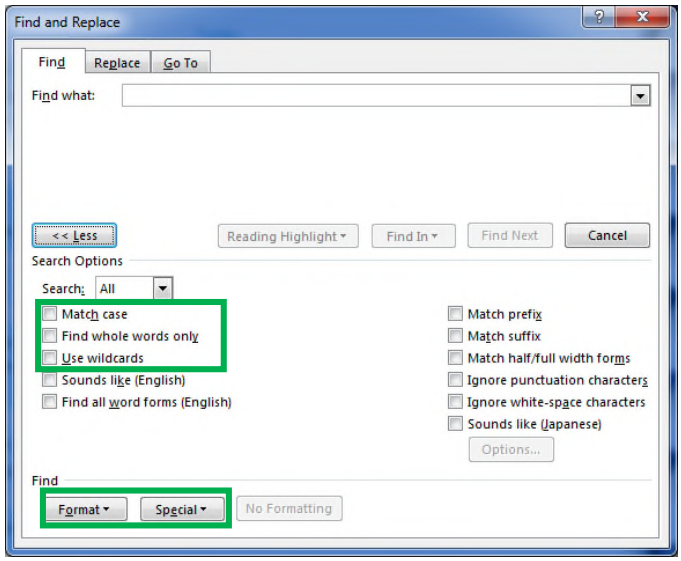
If you do not like using the mouse, **Alt-D** will get you to the tab you want.

In most dialog boxes, the various components have captions with one letter underlined. To access the desired function, you just need to use the **Alt + the underlined letter** to activate that function.



This basic "**Find**" tab will fulfil the same function as the search function in the navigation pane. However, clicking on "**More > >**" will enable you to access more functionalities.

> Here again, if you don't like using the mouse, you can expand the window to see the find options by pressing the **Alt-M** key combination.



The options available will depend on your languages, but the basic ones are available for all languages:

- Search
- Match case
- Find whole words only
- Use wildcards

While the first three basic options are obvious, wildcards are a different matter. We will speak about those in a later issue. At the moment let's just say that they are the only way you can find, for instance, all occurrences of "form" and "from" in a text to check whether you have inadvertently inverted the core letters.

Under the options, you have two buttons that allow you to search for specific "**Format**" and "**Special**" characters. The content of the "**Special**" will depend on whether you ticked wildcards.

Many linguists consider that programming is for people who like maths. Another way of looking at it is that any programming LANGUAGE is just that, a language with grammatical rules, made solely for the purpose of speaking to the machine and telling it what to do. In this edition, I will introduce the basic grammatical rules from a linguistic point of view. As such, the terminology used will not be the standard geeky terminology of object oriented languages (which will be put in brackets behind the term), but that of a linguist.

The MS Office suite has a very powerful tool hidden in the background that enables users to create specific sets of actions within a program of the Office suite or between various programs. And you can even use this language to create standalone home-made programs if instead of using the Visual Basic Editor from within one of MS Office software, you are using Visual Basic 2010 or any version available.

The VBE

Anything you want to order your computer to do, you will do from the VBE (Visual Basic Editor).

> To access the VBE from any MS Office software, just press **Alt-F11**.

This interface is where you will write your sequences of instructions.

On the top left-hand side, you will see the project window, which enables you to select which file will be affected by your instructions. For instance, if you tried the above shortcut in Word®, you should see at least "Normal" and whatever document you have opened in this section.

Writing instructions in "Normal" will make these instructions (or macro-instructions) available in all documents based on your normal template. If you decide to write your instructions in "Project (OutOfOffice_0000)", they will only be accessible when that specific file is opened.

VBA

1. *Components*

VBA stands for Visual Basics for Applications. Each individual software is called an application (we live in a world of apps!). Within each application, you have specific components called objects.

> To bring it back to our phenomenal world, let's take the example of the application "Tree". A "Tree" is composed of elements: leaves, trunk, branches, buds, etc. Those would be the "objects" of the application "tree".
>
> In the VBA for Word®, for instance, some objects will be "`document`", "`paragraph`", "`table`", etc.

2. *Collections and Items*

These components can be generic (collection) or specific (individual items).

> To come back to our "Tree" application, we can have "leaves" or "leaf", "branches" or "branch" depending on whether we are referring to all the elements of that group or one specific one.
>
> In VBA for Word®, for instance, you will have "`documents`", "`paragraphs`", "`tables`", etc.

3. *Hierarchy*

The components are in a hierarchy, which means you cannot have one without having the other.

> In our "Tree" application, each tree will have a trunk, each trunk will have branches and each branch will have leaves.
>
> In VBA for Word®, for instance, you cannot have cells without a table and you cannot have a table without a document.

4. *Adjectives and Verbs*

Items have adjectives (properties):

> In our "Tree" application, leaves can have a shape and a colour.
>
> In VBA for Word®, table rows can have colours.

and verbs (methods):

> In our "Tree" application, you can count or add the number of leaves on any given branch.
>
> In VBA for Word®, you can count, delete or add a paragraph or a section to a document.

5. *Syntax*

VBA syntax groups together objects and its properties and methods using a dot to separate elements.

> In our "Tree" application, to specify that the 25th leaf of the 1st branch is green, we would write:
>
> ```
> trunk.branches(1).leaves(25).color = Green
> ```
>
> In VBA for Word®, to close a document and save it before closing, we would write:
>
> ```
> Document.Close Save:=True
> ```
>
> In the previous example, the "close" method has options. The options for a method are added after the method. In our example, the "Save" option. We could have written:
>
> ```
> Document.Save
> Document.Close
> ```
>
> However, the more instruction lines you write, the harder and the slower it is for the computer to execute them.

6. *Lexicon*

Each application has its own lexicon, but you can also define your own. VBA bases its lexicon on English, regardless of your working language. These new words (Variables) can change values. Before being able to use these new words, it is good practice to inform the computer of their register. This is generally done by writing a sentence (statement) starting with keyword "`Dim`", followed by the name of your own word (it must not contain any spaces and must not be a word VBA uses for other purposes), followed by "`As`" and its register (the type of variable it can contain):

```
Dim This_Is_A_Number As Integer
Dim This_Is_A_Chain_Of_Alphanumeric_Characters As String
```

Informing the computer of the words you want to use and their registers saves memory and also will help you to write more readable instructions.

The words you define can be assigned values:

```
This_Is_A_Number = 25
This_Is_A_Chain_Of_Alphanumeric_Characters = "I hope my explanation is
clear."
```

Here, the = sign is equivalent to "is".

You can also use your own words to clarify what you are saying and refer to some objects of the application.

For instance, if you want to use a document located on **c:\my computer\my folder\my sub folder\my sub sub folder\the file for my example.docx**, you do not want to have to write the entire path each time you are going to speak about it. You can therefore write:

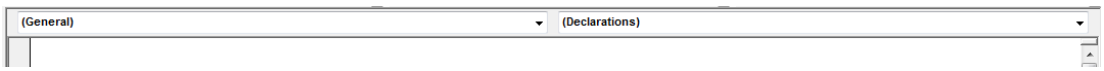
```
Dim MyDoc As Document
Set MyDoc = Documents.Open ("c:\my computer\my folder\my sub folder\my sub
sub folder\the file for my example.docx")
```

After this, you will only need to mention "`MyDoc`" and the computer will know what you are referring to. You will notice that this time, it was necessary to start the sentence with the keyword "`Set`". "`Set`" must be used when referring to application objects.

7. *Where do you write your instructions?*

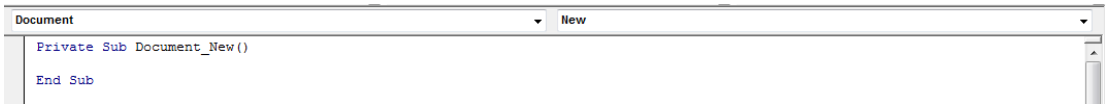
In the VBE, depending on the type of instructions, you will write either in the document code page or in a separate folder.

If you want to write in the document page, in the **project** window, select the document and double-click on it or press **F7**. The large grey window on the right where you will write becomes white and the top of this window will look like this:

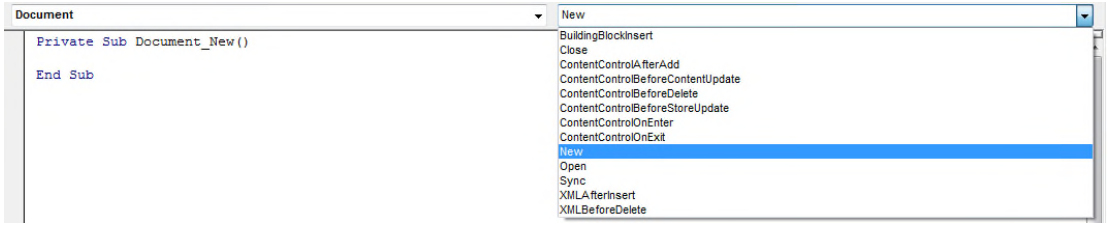


Writing in this section is generally so that you associate your instructions with actions (events) taking place in your document.

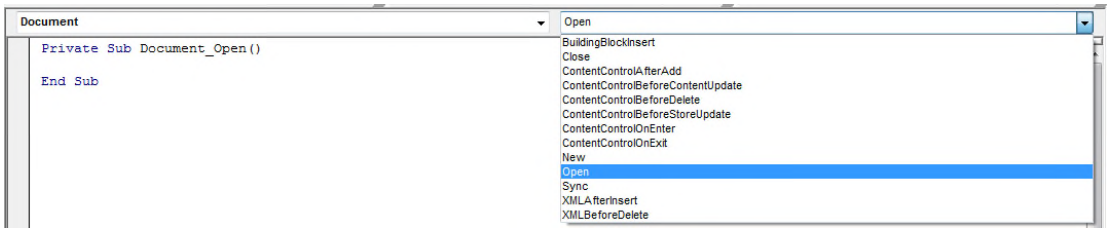
For instance, if you use the drop down button on the right of "**(General)**", you will see "**Document**". Selecting "**Document**" will immediately change the right hand side from "**(Declarations)**" to "**New**". And you will also notice that some text just appeared under.



If you now look at the drop down menu, you have a list of actions/events that you can use to execute your instructions.



For instance, you might want an email to be sent to you as soon as your document is open, so you could select:



and write your instructions between the two automatically generated lines.

Most of the time, you would write your instructions in a separate folder called a "**Module**" or in a "**UserForm**" or in a "**Class Module**".

- A **Module** is simply a folder containing code that you can call from your application.

> From your application (and not the VBE), you can call the list of available macros pressing **Alt-F8**.

- A **UserForm** is simply a visual interface on which you will locate buttons and objects and specify the code for the behaviour of each object. Most User Interfaces in software are just userforms.

For instance in Word®, calling the Font or Paragraph format dialog boxes means you are calling a userform.

- A **Class Module** is when you want to create your own class, that is to say a defined word with specific properties and methods.

For instance you might want to create your own application where you have a class called employees which has its own methods such as count, add, delete, and its own properties such as name, address, etc.

8. *Grammar rules*

As with all languages, there are strict grammar rules that must be followed.

The first rule is that wherever you write your instructions, it must be contained between two lines, one marking the beginning, the other signalling the end. In our previous screen shot, the computer automatically wrote those lines:

```
Private Sub Document_Open()

End Sub
```

Any instructions you give must be contained between such two lines.

Anything contained between these two statements is a routine. Routines can be **Sub** routines or **Functions**.

A **Sub** can be **Private** or **Public**. **Private** means that it can only be called from within the module in which it is located, whereas **Public** means that this routine can be called from another module and that it will be in the list of available macros of your application.

The name of a **Sub** or a **Function** must be unique and cannot contain any space.

A **Function** is made to return a single value.

9. *A bit of fun*

Now that you have the basics, let's have a bit of fun and write our first macro together in Word®.

We are going to change the background colour of an opened document and change the colour of every word. The first word will have the same font size and will be blue, the second word will increase the font size by 2 points and will be white, and the third word will have its font size increase by 1 point and will be red.

So how do we go about it?

First we will open the VBE (**Alt-F11**)

We will select "**Normal**"

Then we will insert a module: **Insert>Module**

In the blank section on the right, we will write our code starting with giving a name to our action (naming the routine).

Let's call it "`ColourMeSilly`"

```
Public Sub ColourMeSilly()
```

You can see that by pressing "enter", the line

```
End Sub
```

is automatically added and the cursor is located just under the first line.

> You don't need to include the brackets after the name of the routine. The VBE will do that automatically. We will explain the function of those brackets in a future issue.

Before we write anything, let do some planning and define exactly what we want our macro to do.

- Change the colour of the background to light purple.
- Go through every single word contained in the document
    - If it is a 1, then colour is blue
    - If it is a 2, then colour is white and font size is +2
    - If it is a 3 then colour is red and font size is +1
- Let's ignore any font superior to 25 points.

Now what variables will we need?

- `MyDoc` for the document it will apply to
- `WordCounter` for going through all the words contained in the document
- `MiniCounter` to count from 1 to 3

So the first lines of our code should be:

```
Dim MyDoc As Document
Dim WordCounter As Long
Dim MiniCounter As Integer
```

`Long` is a numerical value of a non-decimal number that can be extremely high, whereas `Integer`, which is also a category of non-decimal number, is limited. We do not know the number of words in the document, but if it is too high and we are using `Integer` for our `WordCounter`, this will generate an error if the value goes above the integer upper limit.

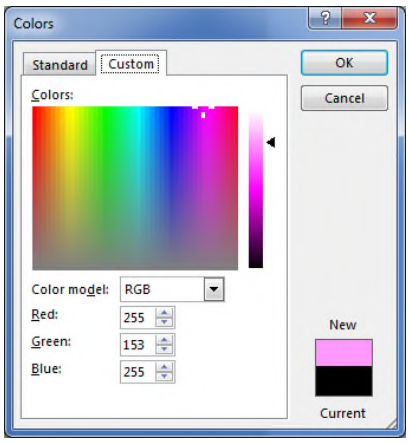
We then need to define some of our values

```
Set MyDoc = ActiveDocument
MiniCounter = 1
```

> Referencing in VBA is essential. We could have defined `MyDoc` as either `ActiveDocument` or `ThisDocument`. As we are writing in a module contained in the normal.dotm template, `ThisDocument` would apply to normal.dotm, whereas `ActiveDocument` refers to the currently active Word® document used to call the procedure. This means that with our reference, our changes ONLY affect the document we are working on. Had we selected `ThisDocument`, our changes would not be seen in our working document, but on our normal.dotm template.

We then need to change the colour of our background

- Fill the background forecolor of `MyDoc` with a purple




```
MyDoc.Background.Fill.ForeColor = RGB(255, 153, 255)
```

> Please note that when typing, the VBE gives you the list of acceptable "requests". You can use the up and down arrows to highlight your selection and use the "**Tab**" key to insert the selected element. If in our example, "`MyDoc`" was not defined, the interface would not be able to suggest anything. However, because it knows that "`MyDoc`" is a "`Document`", then it can suggest any of the potential properties and methods associated with a "`Document`".

- Make it visible

```
MyDoc.Background.Fill.Visible = msoTrue
```

For those last 2 lines, we can see that both start with

```
MyDoc.Background.Fill
```

As linguists, we do not like repetitions, so, we can clarify those 2 lines as follows:

```
With MyDoc.Background.Fill
    .ForeColor = RGB(255, 153, 255)
    .Visible = msoTrue
End With
```

> Please note that all elements referring to the "`With`" element (here "`MyDoc.Background.Fill`) start with a dot, and that when typing, the VBE gives you the list of acceptable "requests". You can use the up and down arrows to highlight your selection and use the "**Tab**" key to insert the selected element.

- Loop through all the words contained in the document

```
For WordCounter = 1 To MyDoc.Words.Count

Next
```

It is in this `For-Next` loop that we will decide what colour and font size to apply.

A `For-Next` loop starts at a number (here 1) and increments each time it reaches the next (here increment by 1 as no "`Step`" is defined after the end number) until it reaches the last number (here the number of words counted in the document).

For our purpose, we want the `MiniCounter` to have a value from 1 to 3, so before the instruction `Next`, we must tell the computer to:

- Add 1 to the `MiniCounter`
- Check if the value of the `MiniCounter` is more than 3 and if this is the case, change to the value of the `MiniCounter` to 1

```
MiniCounter = MiniCounter + 1
If MiniCounter > 3 Then MiniCounter = 1
```

Before this (because our `MiniCounter` starts at 1), we need to select actions depending on the value of our `MiniCounter`. We are going to use the `Select Case` function

```
Select Case MiniCounter
    Case 1

    Case 2

    Case 3

End Select
```

For each `Case`, we will look at the word number, the number given by our `WordCounter` and we will look for the colour of font and its size.

```
MyDoc.Words(WordCounter).Font.ColorIndex
```

And

```
MyDoc.Words(WordCounter).Font.Size
```

We also need to put conditions stating whether to increase the font size or not. For this we will use an `If-Then` structure.

```
If MyDoc.Words(WordCounter).Font.Size<25 then

End if
```

Our final code looks like this:

```
Public Sub ColourMeSilly()

    Dim MyDoc As Document
```

```
    Dim WordCounter As Long
    Dim MiniCounter As Integer

    Set MyDoc = ActiveDocument

    MiniCounter = 1

    With MyDoc.Background.Fill
        .ForeColor = RGB(255, 153, 255)
        .Visible = msoTrue
    End With

    For WordCounter = 1 To MyDoc.Words.Count
        If MyDoc.Words(WordCounter).Font.Size < 25 Then
            Select Case MiniCounter
                Case 1
                    MyDoc.Words(WordCounter).Font.ColorIndex = wdDarkBlue
                Case 2
                    MyDoc.Words(WordCounter).Font.ColorIndex = wdWhite
                    MyDoc.Words(WordCounter).Font.Size =
MyDoc.Words(WordCounter).Font.Size + 2
                Case 3
                    MyDoc.Words(WordCounter).Font.ColorIndex = wdRed
                    MyDoc.Words(WordCounter).Font.Size =
MyDoc.Words(WordCounter).Font.Size + 1
            End Select
            MiniCounter = MiniCounter + 1
            If MiniCounter > 3 Then MiniCounter = 1
        End If
    Next

End Sub
```

Note that all lines that seem to be on several lines such as:

`MyDoc.Words(WordCounter).Font.Size = MyDoc.Words(WordCounter).Font.Size + 2`

are, in fact, not.

The standard method is normally to add an underscore (_) at the end of the line to show that the line continues. In former times, lines used to be numbered. In our case, if in doubt, copy and paste the lines in the VBE and if the line does not contain a line return, it will be displayed on a single line.

You can easily notice those lines because the second and following lines of such paragraphs are not indented.

Furthermore, the indentation used has no purpose. It is simply to make the code easier to read.

You can see that there are many instances of `MyDoc.Words(WordCounter).Font` in our code, so we could simplify it this way:

```
Public Sub ColourMeSilly()

    Dim MyDoc As Document
    Dim WordCounter As Long
    Dim MiniCounter As Integer

    Set MyDoc = ActiveDocument

    MiniCounter = 1

    With MyDoc.Background.Fill
        .ForeColor = RGB(255, 153, 255)
        .Visible = msoTrue
    End With

    For WordCounter = 1 To MyDoc.Words.Count
        With MyDoc.Words(WordCounter).Font
            If .Size < 25 Then
```

```
                Select Case MiniCounter
                    Case 1
                        .ColorIndex = wdDarkBlue
                    Case 2
                        .ColorIndex = wdWhite
                        .Size = .Size + 2
                    Case 3
                        .ColorIndex = wdRed
                        .Size = .Size + 1
                End Select
                MiniCounter = MiniCounter + 1
                If MiniCounter > 3 Then MiniCounter = 1
            End If
        End With
    Next

End Sub
```

What we wrote might be clear to us right now, but probably not to anyone else or even to us in a few weeks or months. Therefore, it is good practice to add comments in your code to help you remember what you intended to do. To add a comment, you can write on any line, whether empty or at the end, starting with an apostrophe, what you want to say. Here is what our finished routine could look like:

```
'************************************************
'Routine to colour a text in 3 different colours *
'************************************************
Public Sub ColourMeSilly()

    'This defines MyDoc as a Word® document
    Dim MyDoc As Document
    'This defines WordCounter as a large natural number
    Dim WordCounter As Long
    'MiniCounter is a small number used to count up to 3
    Dim MiniCounter As Integer

    'MyDoc is defined as the document used to call the macro
    Set MyDoc = ActiveDocument

    'This initializes the MiniCounter
    MiniCounter = 1

    'Change the background colour of the document
    With MyDoc.Background.Fill
        .ForeColor = RGB(255, 153, 255) 'light purple
        .Visible = msoTrue 'make it visible
    End With

    'This loops through all the words contained in the document
    For WordCounter = 1 To MyDoc.Words.Count
        With MyDoc.Words(WordCounter).Font
            'This happens if the font size is less than 25 point
            If .Size < 25 Then
                Select Case MiniCounter
                    Case 1 'if MiniCounter equals 1
                        'changes the colour to dark blue
                        .ColorIndex = wdDarkBlue
                    Case 2 'if MiniCounter equals 2
                        'changes the colour to white
                        .ColorIndex = wdWhite
                        'increases the font size by 2 points
                        .Size = .Size + 2
                    Case 3 'if MiniCounter equals 3
                        'changes the colour to red
                        .ColorIndex = wdRed
                        'increases the font size by 1 point
                        .Size = .Size + 1
                End Select
                'Increments the MiniCounter
                MiniCounter = MiniCounter + 1
                'Checks if the MiniCounter is superior to 3
                'If that is the case, resets the MiniCounter to 1
                If MiniCounter > 3 Then MiniCounter = 1
            End If
```

```
        End With
    Next

End Sub
```

Now, you can open one of your documents in Word® and from that document, press **Alt-F8**, select **ColourMeSilly** and see the result as it applies to that specific document.

10. Not what I expected!

You will notice that it does not exactly what you would expect it to do. The reason is simple. The meaning of "`Words`" in VBA is not exactly the same as for us linguists. In VBA for Word®, a WORD is a word as linguists understand it followed by a space if there is no punctuation or formatting mark, but every single punctuation mark is also considered as a word. So are hyphens, paragraph marks, etc. You might also notice that automated numbering takes the colour of the previous paragraph mark. The best way to see this is to show the formatting marks in your text.

If the document on which you run a test has headers and footers or text boxes, you will notice that the macro did not affect those. Are these not also words? Well, they are, but those elements are not within the same "story" line. In OH MY WORD!, there was a mention of layers. Each layer contains a different story. As we did not instruct our macro to run through all the stories, it simply considered the main default one (the main text).

Also, why did we limit the font size to 25? The intention was to prevent an error message for users who might test the macro on files containing tables. Each table cell in Word® contains an "end of cell" character, which does not have a font size (not exactly true, the font size is 9999 and cannot be increased and it is just a way to not truly allocate a font size to elements) and that the VBA for Word® considers as a word. Without this (completely arbitrary as we could have decided to define the upper font limit up to 9997) limitation, our macro might have encountered an impossible hurdle: increase the font size above the maximum defined limit. In layman's terms, an error with the dreadful error window and generally useless error number (that's generally the point at which anybody of sane mind would simply click on the "end" button and run away from the thing!) asking you what to do.

This type of useless macro is in fact very useful to grasp the exact meaning of the terms used in VBA and have your macros doing precisely what you want them to do.

In our example, we could now rewrite it to add exceptions, for instance:

- If what follows the word is a punctuation mark, then apply the same changes and increase accordingly the value of our `WordCounter`.
- If the word is an opening bracket, then include the following word so that it has the same changes and check if it is followed by a closing bracket.
- Etc.

11. Conclusion

VBA is a powerful language that we can use to tell the computer to execute sets of various instructions so that we do not have to execute those manually.

As VBA is a language, we need to respect its grammatical rules and have a good understanding of the precise definition of every single word. This should be fairly easy for us linguists.

As VBA is a computer language, we must be extremely precise in the way we write our instructions to avoid errors, pitfalls and unexpected behaviours. We must ensure the logic of our instruction is sound. As linguists, this can be slightly harder.

In our sample code, we decided to write it one way, but we could have used many different ways. For instance instead of using a `Select Case` structure for our `MiniCounter`, we could have used `If-Then` constructs. We could have created separate routines for each action of font size and colour change and call them from our main routine. VBA is a language and you can have some degree of liberty within its framework. Of course, some coding methods are better than others (faster execution, less memory usage). From my standpoint, as long as the set of instructions is error-free, execution is much faster than having to do the same thing manually and that the time spent in writing the macro is cost effective (in other words, if the time spent developing the instructions will be offset by the time the macro will save), then it is worth trying to automate that task.

12. Something more useful

As we saw in the first part of this newsletter, our work often involves changing the options depending on the work in hand. This can be finicky, repetitive and you might sometimes forget to activate or deactivate an option. It is good practice to write a few profiles where you define exactly the status of your options so that you can call them in a couple of clicks.

Here is the list of the options covered in this issue and how they are called upon in VBA:

File>Options	VBA full name	Status
Tab **Display** section **Always show these formatting marks on the screen**		
Tab characters	`ActiveWindow.View.ShowTabs`	`True` or `False`
Spaces	`ActiveWindow.View.ShowSpaces`	
Paragraph marks	`ActiveWindow.View.ShowParagraphs`	
Hidden text	`ActiveWindow.View.ShowHiddenText`	
Optional hyphens	`ActiveWindow.View.ShowHyphens`	
Object anchors	`ActiveWindow.View.ShowObjectAnchors`	
Optional breaks	`ActiveWindow.View.ShowOptionalBreaks`	
Show all formatting marks	`ActiveWindow.View.ShowAll`	
Tab **Advanced** section **Layout**		
Don't expand character spaces on a line that end with SHIFT-RETURN	`ActiveDocument.Compatibility(wdExpandShiftReturn)`	`True` or `False`
Tab **Advanced** section **Show document content**		
Show background colours and images in Print Layout view	`ActiveWindow.View.DisplayBackgrounds`	`True` or `False`
Show text wrapped within the document window	`ActiveWindow.View.WrapToWindow`	
Show picture placeholders	`ActiveWindow.View.ShowPicturePlaceHolders`	
Show drawings and text boxes on screen	`ActiveWindow.View.ShowDrawings`	
Show bookmarks	`ActiveWindow.View.ShowBookmarks`	
Show text boundaries	`ActiveWindow.View.ShowTextBoundaries`	
Field shading	`ActiveWindow.View.FieldShading`	`wdFieldShadingNever` `wdFieldShadingWhenSelected` or `wdFieldShadingAlways`
Show field code instead of their values	`ActiveWindow.View.ShowFieldCodes`	`True` or `False`
Tab **Advanced** section **Display**		
Show vertical ruler in Print Layout view	`ActiveWindow.DisplayVerticalRuler`	`True` or `False`
Show measurements in units of	`Options.MeasurementUnit`	`wdInches` `wdCentimeters` `wdMillimeters` `wdPicas` or `wdPoints`
Show measurements in width of characters	`Options.UseCharacterUnit`	`True` or `False`

Ribbon tab **View** group **Show**	VBA full name	Status
Ruler	`ActiveWindow.ActivePane.DisplayRulers`	`True` or `False`
Gridlines	`Options.DisplayGridLines`	
Navigation Pane	`ActiveWindow.DocumentMap`	

Ribbon tab **Page Layout** group **Arrange** Option **Align** selection **Grid Settings…**	VBA full name	Status
Display alignment guides	`Options.DisplayAlignmentGuides`	
Page guides	`Options.PageAlignmentGuides`	
Margin guides	`Options.MarginAlignmentGuides`	`True` or `False`
Paragraph guides	`Options.ParagraphAlignmentGuides`	
Object Snapping		
Snap object to other objects	`ActiveDocument.SnapToShapes`	
Grid settings		
Horizontal spacing	`ActiveDocument.GridDistanceHorizontal`	`MillimetersToPoints(VALUE)` `CentimetersToPoints(VALUE)` `InchesToPoints(VALUE)` `PicasToPoints(VALUE)` or VALUE [the default unit is points]
Vertical spacing	`ActiveDocument.GridDistanceVertical`	
Grid origin		
Use margins	`ActiveDocument.GridOriginFromMargin`	`True` or `False`
Horizontal origin	`ActiveDocument.GridOriginHorizontal`	`MillimetersToPoints(VALUE)` `CentimetersToPoints(VALUE)` `InchesToPoints(VALUE)` `PicasToPoints(VALUE)` or VALUE [the default unit is points]
Vertical origin	`ActiveDocument.GridOriginVertical`	
Show grid		
Display gridlines on screen		
Vertical every (tick option)	`ActiveDocument.GridSpaceBetweenHorizontalLines`	0
Vertical every	`ActiveDocument.GridSpaceBetweenHorizontalLines`	VALUE [integer]
Horizontal every	`ActiveDocument.GridSpaceBetweenVerticalLines`	
Snap objects to grid when the gridlines are not displayed	`ActiveDocument.SnapToGrid`	`True` or `False`

Now, you can easily create a macro to define specific working profiles. Here is an example.

What I want	How to ask the computer to do it with VBA
	`Public Sub Profile_TextEditing()`
I want to see	`    With ActiveWindow.View`
Tabulation marks	`        .ShowTabs = True`
Spaces	`        .ShowSpaces = True`
Paragraph marks	`        .ShowParagraphs = True`
Hidden text	`        .ShowHiddenText = True`
Object anchors	`        .ShowObjectAnchors = True`
Bookmarks	`        .ShowBookmarks = True`
Field content (and not codes)	`        .ShowFieldCodes = False`
But not	
Optional breaks	`        .ShowOptionalBreaks = False`
	`    End with`
I also want to see	`    With ActiveWindow`
The vertical ruler	`        .DisplayVerticalRuler = True`
The ruler(s)	`        .ActivePane.DisplayRulers = True`
The document map	`        .DocumentMap = True`
	`    End with`
And I want my measurements in inches	`    Options.MeasurementUnit = wdInches`
	`End sub`

Now writing this code into a module (a new one or after the last routine you wrote), pressing Alt-F8 from any document and selecting `Profile_TextEditing` will immediately activate all of these functions.

We hope you enjoyed this issue and learned a few useful things along the way. We also hope to see you again for our next issue where we will look at Outlook® and how to write a macro that lists all the files you may have attached to your email.

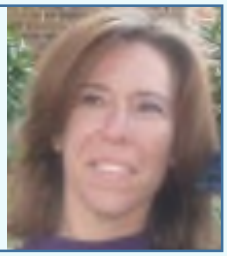
The Out Of Office Team

Hacène Dramchini
MCIL, MNWTN, MIAPTI
The main author
A technical English to French translator passionate about IT, Hacène is responsible for the content of this newsletter.

Lucy Brooks, CL, FCIL, MITI
The English Editor
A successful French, German and Spanish to English translator and director/founder of eCPD webinars, Lucy has a passion for sharing knowledge.

Angelika Körber, MATICOM, MCBTIP, MCioLGS and Sworn translator
The German localiser
An English and French to German technical translator and a reporter, Geli has an inquisitive mind and always strives for perfection.

Muriel Louchart, Sworn translator
The French localiser
A successful English to French legal translation and software localisation, Muriel deals with the localisation of this monthly periodical for European French.



Patricia Torres, MCONALTI, MATA and Sworn translator
The Spanish Localiser
An English and Italian to Spanish translator, as well as a university translation studies teacher, Patricia loves technology.

CONTACT US
outofoffice.newsletter@gmail.com