

IndexDOctoXML.dotm User Guide v1.1

IndexDOctoXML.dotm is a Word macro that converts a Word document to IXML format, which can then be imported into indexing software. It runs under PC Word, and works fine on a Mac running Windows under Parallels. This is an introduction to the macro, which at this point is version 1.1.

This macro is free and open source; I walk through the [code](#) below. If you improve it, please share your improved version with the community.

When indexing a new edition of a book, it can be a great help to have the previous edition index. This utility gets that previous edition index into your indexing software, saving typing.

A PDF of the previous edition index needs to be exported to a Word document.

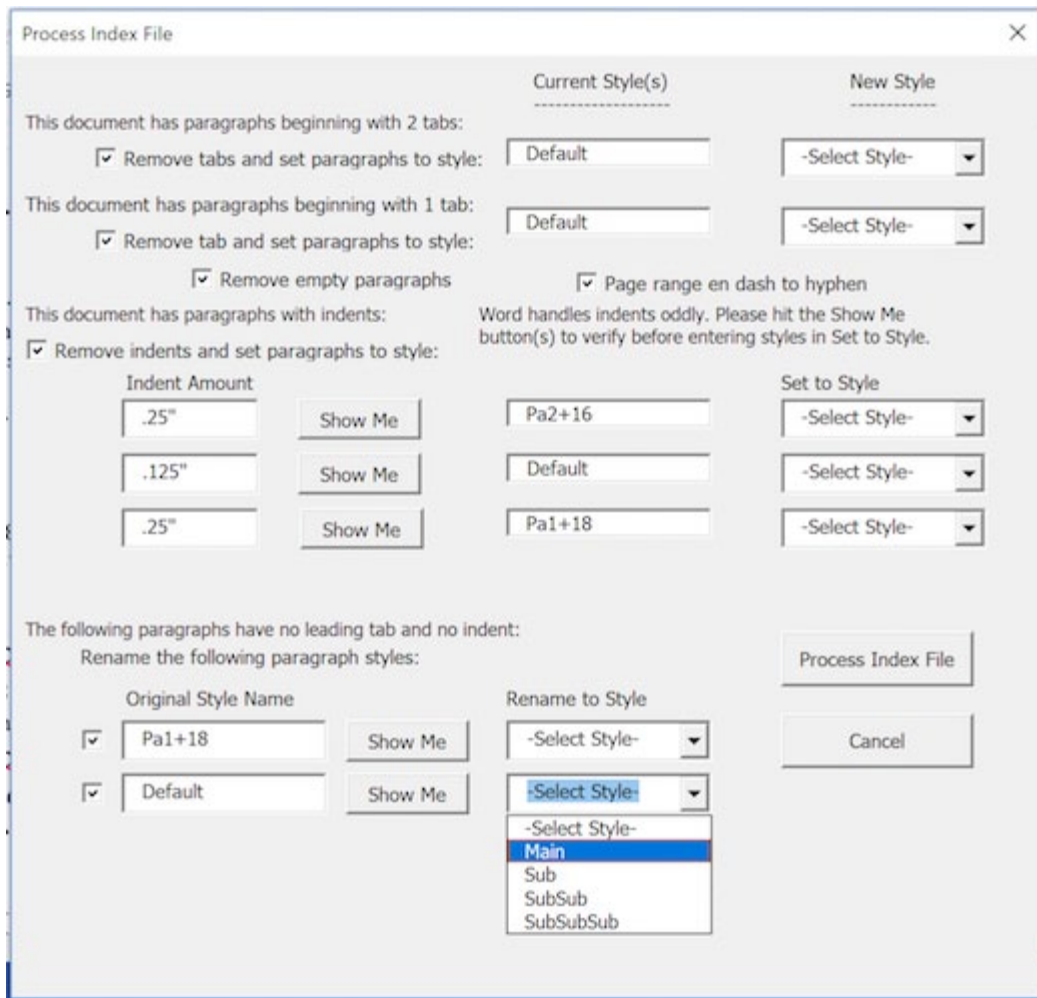
The index Word document has to be cleaned up: headers, footers, page numbers, the “Index” title, the headnote, and any continued lines need to be removed so that the text is only the index entries.

This user guide provides an overview of the macro, then shows how to [install](#) and [use](#) it. After that I walk through the [code](#) itself, loading the macro document as an ordinary Word document and opening the Visual Basic environment so that you can see the Visual Basic code. At this point there are no [version changes](#) to report.

Overview

IndexDOctoXML.dotm contains a public macro called IndexDOctoXML that shows up when you View Macros in Word. That’s how it has to be run at this point; there’s no menu yet.

IndexDOctoXML reads through the current Word document paragraph by paragraph, looking at tabs, indentation, and styles. It then presents a box that displays the various types of paragraph formatting with Show Me buttons that highlight the paragraphs in the document with that formatting. You select from dropdown lists the types of index entries represented by the formatting: Is the highlighted entry a Main, Sub, Subsub, or Subsubsub entry?



Once the selections are made and the Process Index File button is clicked, the macro applies the appropriate paragraph style names to the appropriate paragraphs. It then writes the index file to another document in IXML format. The IXML document is saved in the same directory as the index document, with the index document name plus the IXML extension; a message box displays this information when the macro finishes. The re-styled index document is not saved.

Then you follow your indexing software's procedure for importing the IXML file.

The italics, boldface, underline, subscript, and superscript attributes are maintained.

Installing the Macro

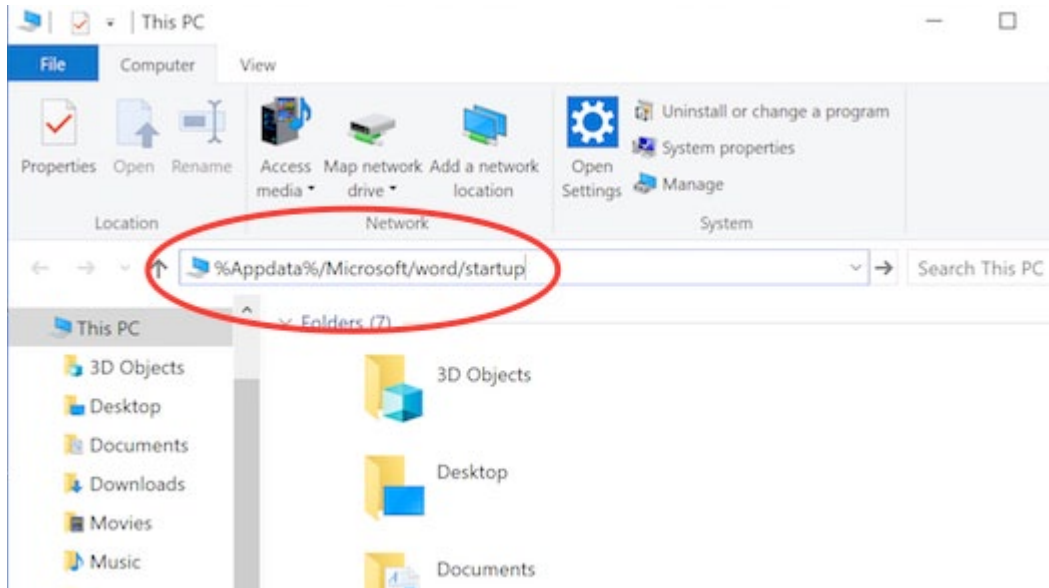
You do not need to install the macro to walk through its [code](#).

The macro needs to be placed in the Startup folder for Microsoft Word. Try looking on this path (taking away the square brackets and filling in the appropriate user name):

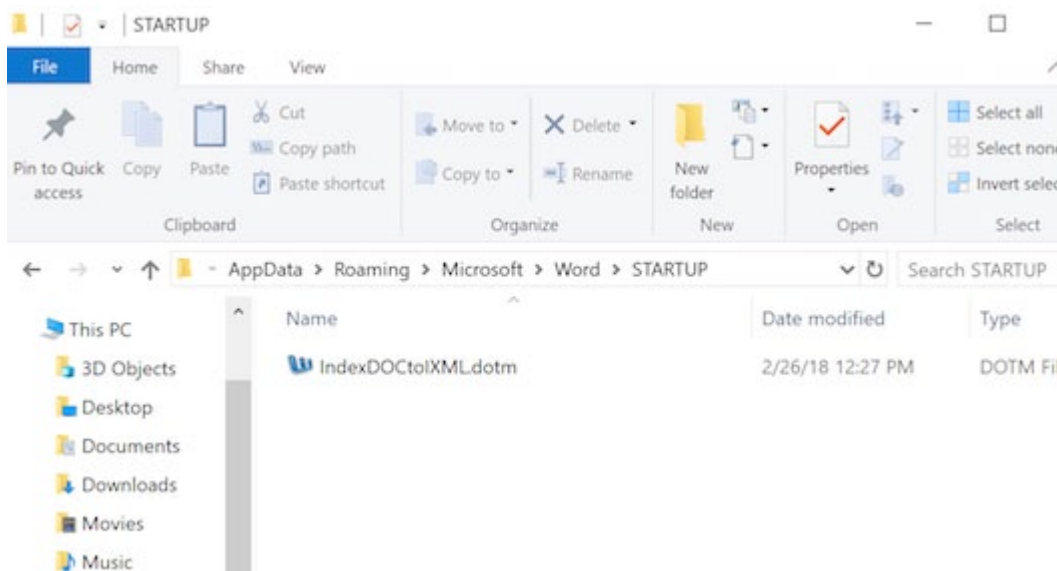
```
C:\Users\[User name]\AppData\Roaming\Microsoft\Word\STARTUP
```

Another way to find this folder is to open an Explorer window and paste the following text in the path box:

```
%Appdata%/Microsoft/word/startup
```



When you hit Enter, the Startup folder is presented. Put the IndexDOctoXML.dotm file into the Startup folder.



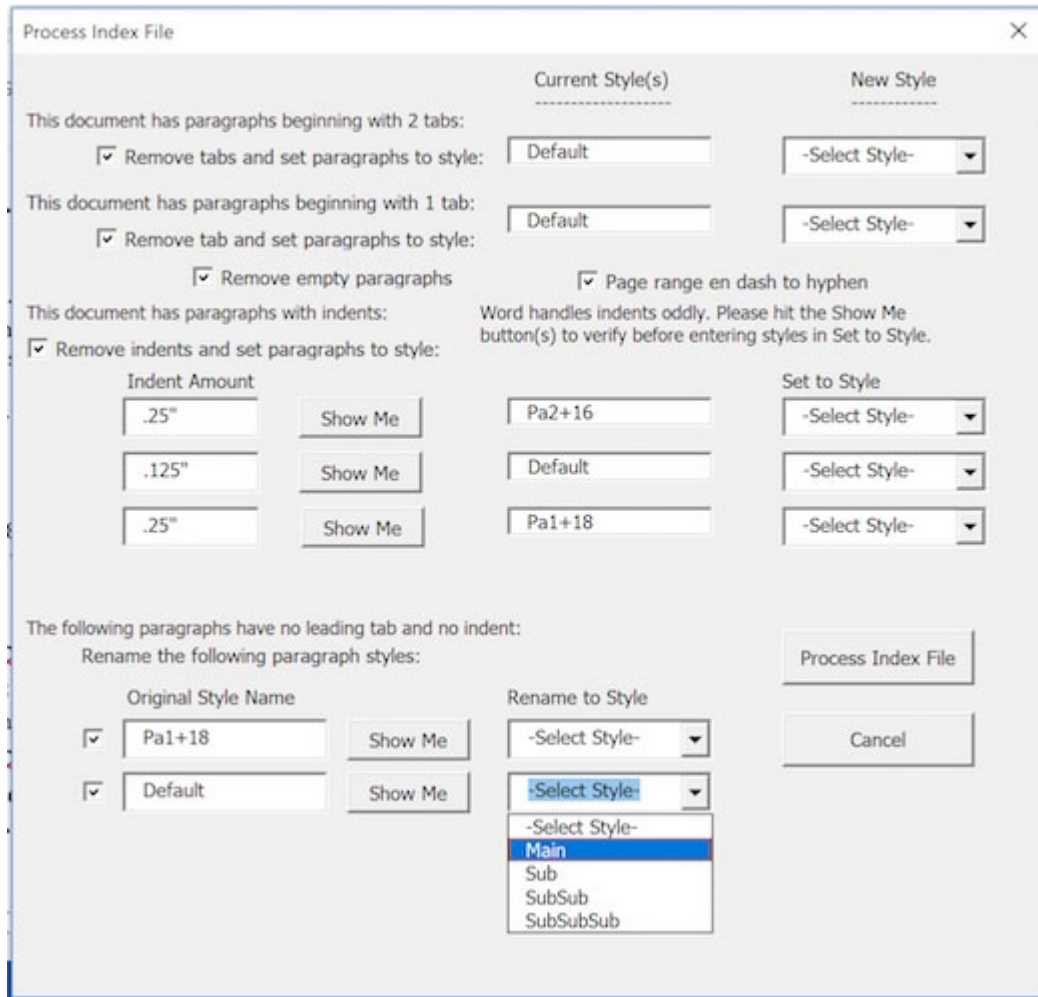
If Word is running, you'll need to quit and restart Word to load the IndexDOctoXML.dotm macro.

Running the Macro

With your index document open in Word, View Macros, select the IndexDOctoXML macro, and click Run.

A basic message box comes up that explains the macro and gives you a chance to cancel out or continue.

The macro then analyzes the formatting of each paragraph in the current document. It then presents its results:



Current Style(s)		New Style	
<input checked="" type="checkbox"/> Remove tabs and set paragraphs to style:	Default	-Select Style-	
<input checked="" type="checkbox"/> Remove tab and set paragraphs to style:	Default	-Select Style-	
<input checked="" type="checkbox"/> Remove empty paragraphs			<input checked="" type="checkbox"/> Page range en dash to hyphen
Word handles indents oddly. Please hit the Show Me button(s) to verify before entering styles in Set to Style.			
<input checked="" type="checkbox"/> Remove indents and set paragraphs to style:			
Indent Amount	Set to Style		
.25" [Show Me]	Pa2+16	-Select Style-	
.125" [Show Me]	Default	-Select Style-	
.25" [Show Me]	Pa1+18	-Select Style-	

The following paragraphs have no leading tab and no indent:
Rename the following paragraph styles:

Original Style Name	Rename to Style
<input checked="" type="checkbox"/> Pa1+18 [Show Me]	-Select Style-
<input checked="" type="checkbox"/> Default [Show Me]	-Select Style- Main Sub SubSub SubSubSub

[Process Index File] [Cancel]

This box may not contain all the elements shown here, if they're not present in the index document.

Each type of formatting has a checkbox in front of it that you can uncheck if for some reason you don't want a new style name applied to paragraphs with that formatting.

Removing empty paragraphs can be turned off by removing the check mark from its checkbox. Replacing en dashes with hyphens can also be turned off by removing the check mark from its check box.

If paragraphs with two or one tabs in front are present in the index document, you can specify the types of index entries they mark.

Paragraphs that have indents applied are listed next. Word handles paragraph indents in an odd way, so each type presented has a Show Me button by it. Clicking on Show Me highlights a paragraph in the index document with that type of indentation so that you can see where it sits in the index.

Paragraphs that have styles (but no tabs or indents) are listed last, again with Show Me buttons so that you can see the paragraphs being referred to.

For each type of formatting presented, select the appropriate index entry level from the dropdown list: Main, Sub, Subsub, or Subsubsub.

When you've finished setting all the dropdown lists, click the Process Index File button.

If you want quit, click Cancel instead.

After the Process Index File button is clicked, the macro then applies to the appropriate paragraphs your selected index levels as style names. The index file appearance will change because of the new style names applied.

Then the macro will open a new document, write out the index file in IXML format to that new document, then save that IXML document.

The changed index file is not saved.

Once the IXML is written, the macro puts up a message box stating where the IXML was saved. The macro ends when OK is clicked in the message box.

Go to your indexing software and import the IXML file.

Walking Through the Code

You do not have to install the macro to look at its code.

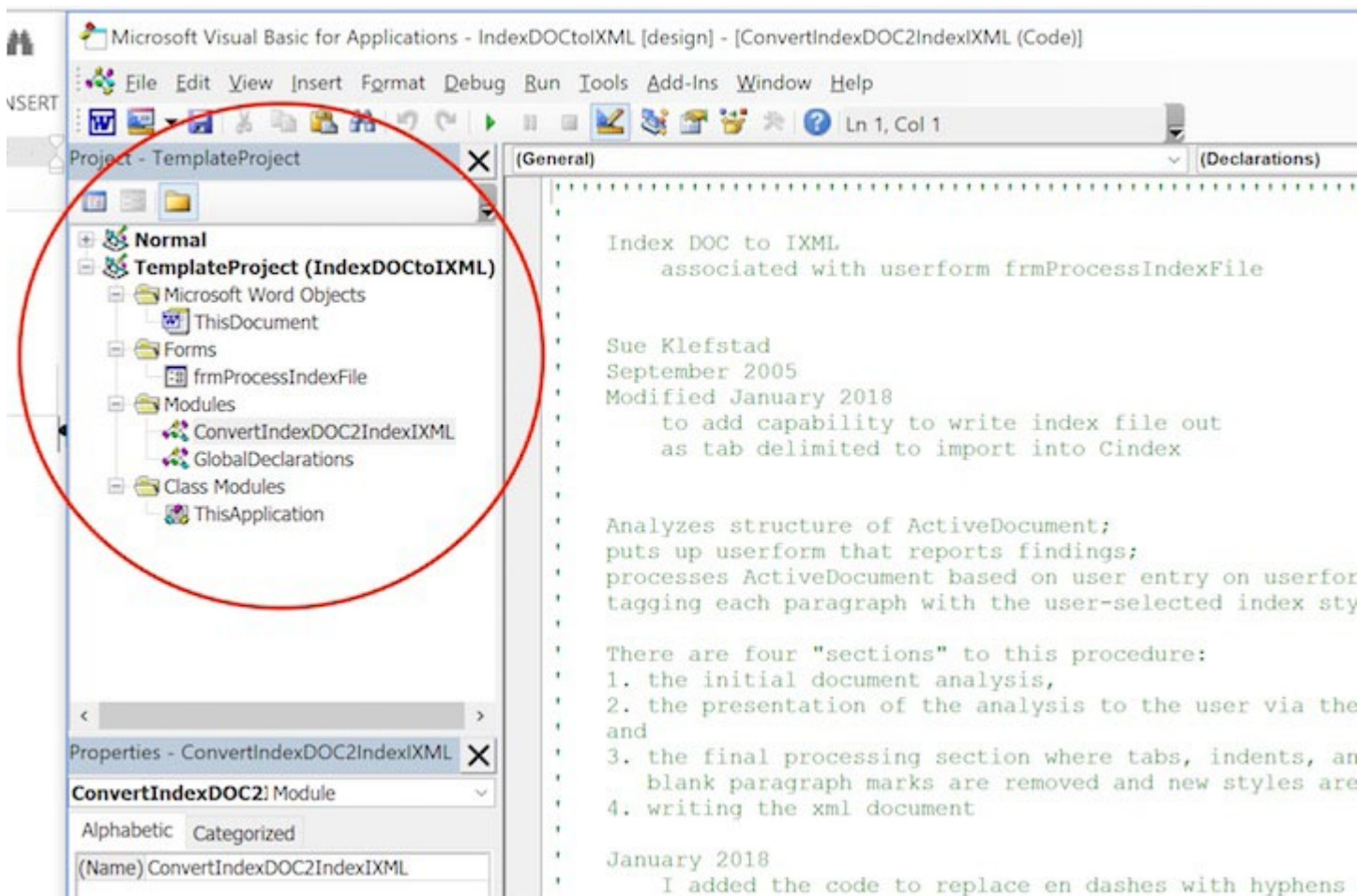
In Word, open IndexDOctoIXML.dotm as a regular Word document. When the document opens, you will likely get a banner across the top that says

SECURITY WARNING Macros have been disabled

with a button that says Enable Content.

You don't have to enable the content, so don't click the button. Click the x on the far right to dismiss the banner.

Click in the blank document then hit Alt-F11 to open the Visual Basic for Applications development environment.

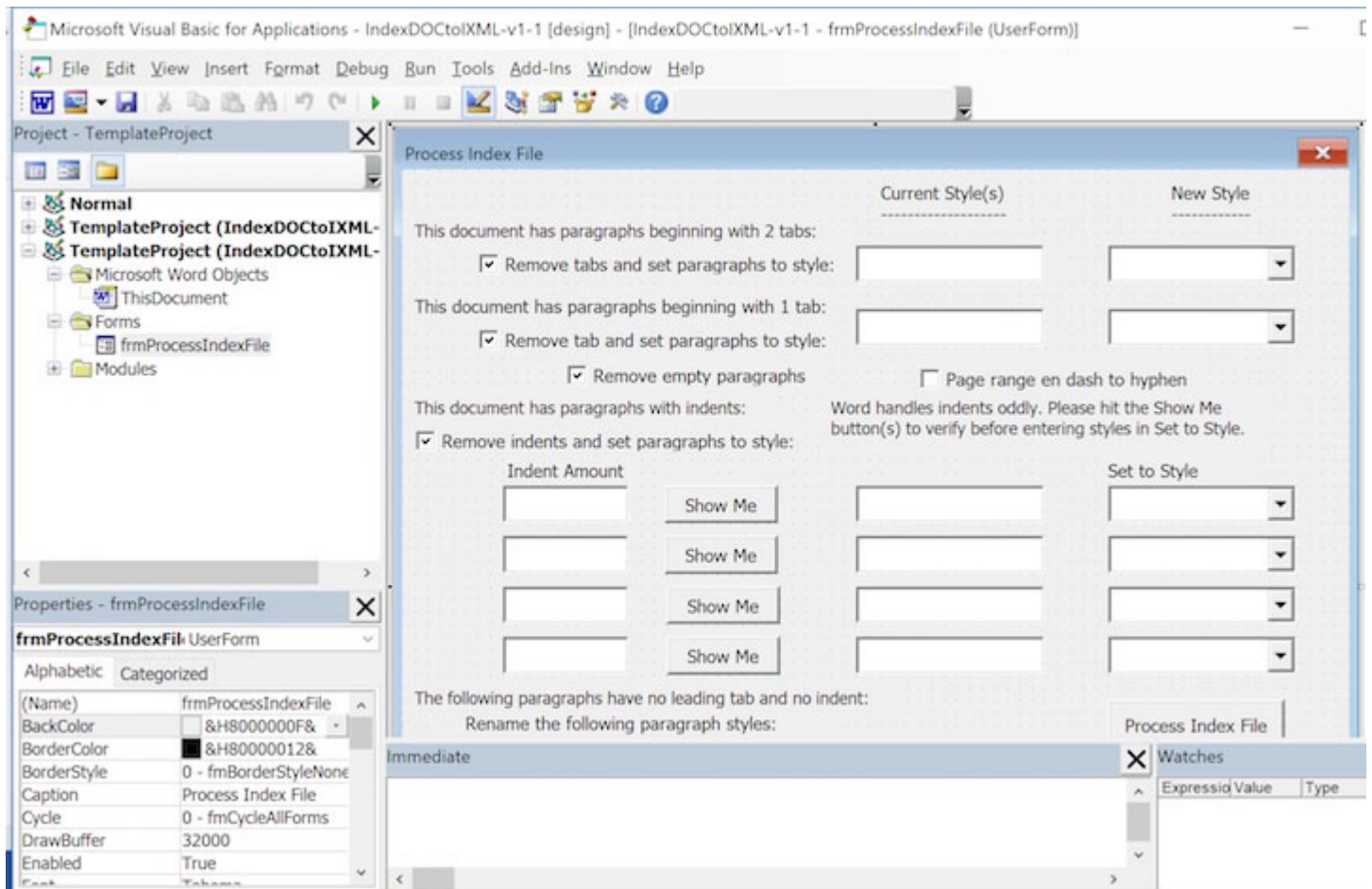


If you have the macro installed, you will see two IndexDOctoIXML projects listed in the Explorer window of the VBA IDE (integrated development environment).

One won't let you expand it when you click on it; you'll get a message that the project is locked. This is the installed macro, which you can't modify.

The other (or only) IndexDOCtoXML project is the document you have open in Word. To look at its code, click the pluses next to each level to fully expand it.

Let's start with the form. Double-click on the name of the form, frmProcessIndexFile. The form shows up in the big code viewing space.



Click on the View menu. The top two items, Code and Object, alternate between showing this form (the object) or showing the code behind it (the code).

A form has code associated with it that handles initialization of variables when the form is displayed: Private Sub UserForm_Activate().

The "private" here means that this code is only visible to this project; you won't see this name when you View Macros. UserForm_Activate() is an event that is triggered when the

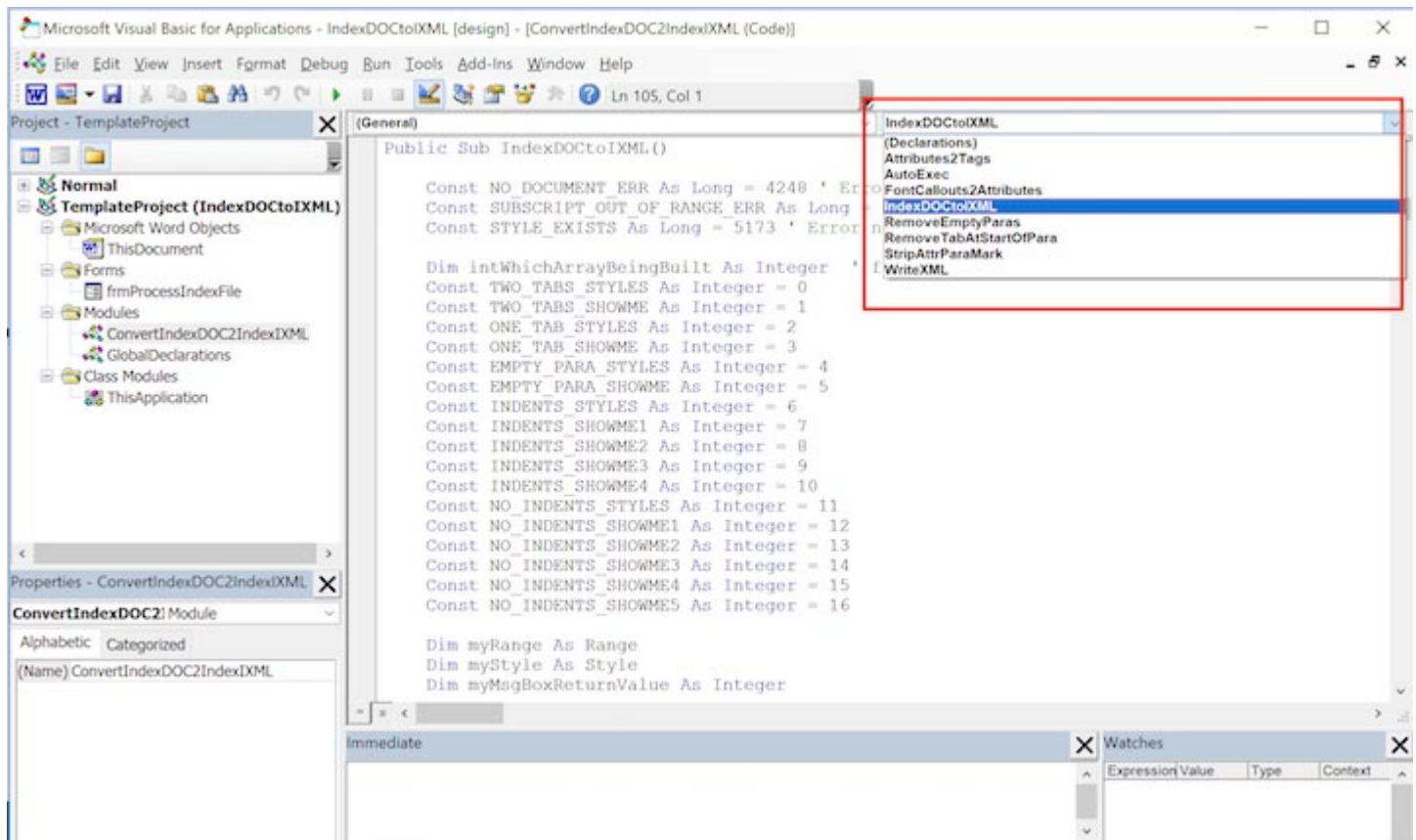
form is displayed, which happens in the midst of the main procedure, as we'll see soon.

The code behind this form handles the events triggered by clicking on the Cancel button [Private Sub cmdCancel_Click()] or the Process Index File button [Private Sub cmdProcessIndexFile_Click()].

This code also handles the events of clicking on any of the Show Me buttons. These procedures have ShowMe in the name, such as Private Sub cmdShowMeIndent1_Click().

Now let's get to the meat of the matter.

Double click on ConvertIndexDOC2IndexXML in the Modules folder. Its code appears in the large code viewing area.



Notice the dropdown in the upper right of the window.

This module contains not only the primary procedure IndexDOctoIXML but also all the supporting subroutines it calls. This dropdown lists all the procedures in this module and provides a simple way to get to that code in the file.

Select IndexDOCtoIXML from the dropdown to see the main code of this macro.

Scroll up a few lines to see a line of code that appears before this procedure:

```
Option Explicit
```

If you write VBA, this line should be the first in your code, as it is here. What this means is that you must declare every variable you use, rather than simply allowing them to be created on the fly. That way, if you mistype a variable name, it will be flagged rather than created anew, saving a lot of debugging time.

Above this line, from the top of the file down, are a bunch of comments about the procedure that outline its structure and approaches.

Okay, Public Sub IndexDOCtoIXML():

The Public part of that opening is what makes this procedure visible when you View Macros. All the other procedures in this module are Private, making them visible only to each other within this module.

With the Public/Private mind-set, let's take a sidestep over to the other file in the Modules folder: GlobalDeclarations. This file declares a bunch of global variables, which are variables that hold their values the entire time this macro runs, as opposed to variables that are private to the procedures they're used in. Private variables go away once their procedure finishes running; public variables hold their values until the macro IndexDOCtoIXML quits running.

This IndexDOCtoIXML macro repurposes old production code that was part of a larger suite of utilities. This separate GlobalDeclarations file is a remnant of that old structure. At some point these variables could be brought into the main ConvertIndexDOC2IndexIXML file. I haven't bothered to do that yet. And there could be a bunch of variables defined here that are never used—another remnant.

Okay, back to Public Sub IndexDOCtoIXML():

This procedure essentially consists of two pieces: the first analyzes and styles the index document and the other piece writes the index out to IXML format. As I mentioned above, this macro repurposes some old production code from 2005; this is the first part that analyzes and styles the index file.

The part that writes out the IXML format (look for WriteIXML in the procedure dropdown) is new and therefore the more likely place for any bugs. However, the analysis part does occasionally trip, not detecting formatting or misinterpreting it or something. I have one trouble-child index document for which my code detects no difference between the no-indent and indented types of paragraphs and tags them all as Main. Still working on this one.

So, got your boots on? We're starting with the first part, the analysis and style part. It's 2005 code and I'm writing this in 2018, having essentially dropped in the code and started using it. Sort of the blind leading the blind here. Fortunately I comment a lot.

The essence of this code is that you've been given an index in a Word doc and have to figure out its index structure so that the paragraphs can get tagged (styled) with the right heading style: Main, Sub, Subsub, or Subsubsub.

The file opens with its variable declarations, then the error handling. A message box is displayed that summarizes the macro and provides an exit point. Variables are initialized, then the macro removes any spaces that appear before or after tab characters so that tabs used as indents can be more easily found. And it removes whitespace from otherwise empty paragraphs.

I'd like to point out a small piece of code that appears here and many other places throughout this macro:

```
' Flush bad karma from failed find with wildcards
ActiveDocument.Range.Find.Execute FindText:="^p",
MatchWildcards:=False
```

When this original code was written back in 2005, Word had a long-standing bug with its Find operation when used with wildcards: If a Find with wildcards didn't find its search text then the next Find operation, with or without wildcards, might fail even if its search text existed. I don't know if this bug is still present in Word. The easiest workaround, after doing a Find with wildcards, is to just find something like a paragraph mark; it doesn't matter if it fails. Then do the next search that matters. So sprinkled throughout this code is that above pair of comment and code to work around that bug.

Okay, we're done with the prologue sort of stuff and have reached the first main section of this code: analysis of the

index doc formatting, checking for empty paragraphs, then initial tab(s), then indents. This section leads into the second main section: displaying the results of the analysis via the form frmProcessIndexFile.

For each type of formatting, paragraph indexes are saved in arrays so that when a Show Me button on the form is clicked, the appropriate paragraph is selected.

Then we have Section II: the form is displayed, reporting the results of the analysis and presenting Show Me buttons for highlighting paragraphs in the index document and dropdowns for assigning the appropriate index heading style.

A new form object is created, and with the line

```
frmobjProcessIndexFile.Show
```

control is given to the code associated with the form frmProcessIndexFile, discussed [above](#).

Sections of the form are displayed or not, depending on whether that formatting was found in the document.

The code associated with that form sets the values of many global variables. When either the Process Index File or Cancel button is clicked on the form, control returns to this IndexDOctoIXML procedure.

If the Cancel button is clicked, the global variable gProcessIndexFile is set to False, but it's set to True if the Process Index File button is clicked. So Section III of this macro, styling the paragraphs with the appropriate index heading style, only happens when gProcessIndexFile is True.

If the Process Index File button is clicked, setting gProcessIndexFile to True, then we step through a series of Find and Replaces, starting with en dashes replaced with hyphens.

Then we add the four index heading paragraph styles to the index document: Main, Sub, Subsub, and Subsubsub. If any of those styles happen to already exist in the document, the error triggered is ignored.

Then we call the subroutine Attributes2Tags().

This subroutine searches the index document for italics, boldface, underline, subscript, and superscript attributes and surrounds them with IXML tags for those attributes.

Back at the main IndexDOctoXML procedure, we next step through each of the formatting types, applying the appropriate paragraph styles to the appropriate paragraphs.

After all that paragraph style changing is done, a couple more Find and Replaces are done—removing tabs at the starts of paragraphs and possibly removing empty paragraphs, depending on a check box value.

And we're done with the big first chunk of code. We've analyzed the index document, presented the results, and applied the paragraph styles.

So now that we know the index structure via the paragraph style names, we can write the index out to IXML. This is done in the WriteIXML() routine.

In WriteIXML(), we open by saving the current value of the Smart Quotes setting. We're going to be writing computer code, which needs straight quotes, not curly, so we turn off Smart Quotes.

After saving the path and name of the index document, we create a new document and write the IXML header tags to it. Then we step through each paragraph of the index document, grabbing the text of the paragraph. The ending paragraph return is stripped off, curly quotes are replaced with straight, and the comma-closing quote mark pair is reversed to closing quote mark-comma.

Next I check the paragraph text to see if it contains a cross-reference.

I Split() the paragraph text at commas. This puts into an array each chunk of the paragraph text separated by commas. For example, the heading

Klefstad, Sue, 25, 60

would split into an array of four elements: Klefstad, Sue, 25, and 60.

So headings containing commas need to be reunited into one chunk of text: Klefstad, Sue.

Also, a paragraph that is a subsub heading needs to go into the IXML file with its main and sub heading texts. So the string variables sMainStr, sSubStr, and sSubSubStr hold their values until replaced by new.

The string variable sCurrentLine strings those sMainStr, sSubStr, sSubSubStr, and sSubSubSubStr values together, separated by tabs. sCurrentLine is Spit() at the tabs into an array of four elements (Main, Sub, Subsub, and Subsubsub) that are written to the IXML document. Then the page numbers are written to the IXML file as locators.

If a cross-reference was found in this paragraph, an IXML record is added as a locator.

When all the paragraphs in the index document have been looped through, then the closing IXML tags are written to the IXML document.

The original Curly Quotes setting is restored.

The IXML document is saved as a text file in the same path as the index document with the index document file name plus the IXML extension. A message box is displayed showing this information.

When the OK button of the message box is clicked, the macro ends.

Versions and Changes

I have nothing to report at this point.

Contacting Me

Feel free to contact me: I'm Sue@SuetheIndexer.com.

There's a good chance that I will not get back to you promptly, not because I'm ignoring you but because I'm busy with work.

Check out my [website](#) and my [articles](#) on indexes and indexing.