Paper AD07

A Tool to Automate TFL Bundling

Mark Crangle, ICON Clinical Research, Marlow, United Kingdom

ABSTRACT

Finding a consistent method of delivering TFL packages to sponsors is a challenge for programmers working in a CRO environment due to the different requirements that different sponsors have. The most common delivery method requested is a bookmarked PDF file containing all of the TFLs to allow easy navigation through the package but this conflicts with publishing requirements that often prefer to work with individual output files. With ever increasing amounts of data resulting in larger and larger outputs it becomes difficult to find an efficient and repeatable solution that can produce the required output.

This paper explores the benefits and drawbacks of various different methods for creating combined TFL packages. The solution arrived upon is to create a HTML file that contains the table of contents with links to individual output files, and then read this into Adobe Acrobat® using the "PDF from Web Page" functionality. The paper will go on to explain the required structure of this HTML file, and demonstrates methods for automating the process VB scripts.

INTRODUCTION

Typically, the requirement for a TFL package, either a final version or for intermediate review, is a bookmarked PDF file with a table of contents (TOC). Often this necessitates combining individual TFL files, either already in PDF format, or in another format that can be opened in Microsoft Word and finding a consistent and robust way of doing this is a challenge. Additionally the process of creating the delivery package is by its nature, the very last thing to be done before the delivery is completed and so the process needs to be as quick and as automated as possible to save time.

The existing method at ICON used SAS® macros to read our TFL metadata from csv files and generate a Visual Basic Advanced (VBA) script that would then be copied into Microsoft Word to insert each TFL (in RTF format) into the document with a Word bookmark before each TFL. These Word bookmarks would then be used to generate a TOC and the whole document saved in Word before being converted into PDF with the TOC links being changed into PDF links to link through the document to the individual TFLs.

This method had worked well for a number of years but with increasing amounts of TFLs being required in delivery packages, and with the individual TFL files increasing in size in line with the increase in data, the process of adding all the files together in Word with a bookmark for each one became extremely time consuming and would often cause Word to crash. If the user did manage to get as far as creating the file in Word, then the conversion of that file to PDF would also take a long time. The final PDF file would have a TOC with links to each file but would not have PDF bookmarks which was often a requirement from reviewers but which we could not fulfil. Adding all this to the fact that the existing method of specifying TFL metadata was being required and their an opportunity and necessity to create a new process to combine TFL files into a single bundle in PDF format.

BUILDING A NEW TOOL

The requirements of the new tool were defined to be as follows:

- Able to generate a PDF file with a TOC and bookmarks that both had links through the document to the location
- Quicker and more reliable than existing solution
- Uses TFL metadata to ensure the TOC and bookmarks matches the content of TFL files
- Increased automation to reduce the amount of user input needed

The main issue for the existing solution was that joining the TFLs together into one file, saving that as a Word document, and then converting that to PDF was exponentially increasing the amount of work needed as file sizes became much bigger. What was required was to split the conversion to PDF into smaller chunks and then find a way to convert this to PDF and this was the focus of early development of the tool.

USING THE CREATE PDF FROM WEB PAGE FEATURE IN ACROBAT

Adobe Acrobat contains a function where by an entire web page can be read in as a HTML file and converted into a PDF file. The default output created by this process contains one PDF document with the links in the original HTML document preserved as links that will get opened in a web browser but there is also functionality to download the files referenced by these links and append them into the same PDF document. Running this over a full live web page takes a long time but if we run this over part of the 2016 Annual Conference section of the Phuse website we can see how it appears:

So this gives a top page that matches the input HTML file in appearance, with the webpages or files that this page has links to, appended to the end of the document and with automatically created PDF bookmarks to the place in the overall document where these extra files are appended. For a live webpage, these bookmarks to the appended files don't work that well but it is the principal that is important. If we can design a HTML file to represent the table of contents that we want at the start of our TFL package and add links to the individual TFL files then we can convert this to a PDF files with any HTML formatting that we want preserved, and Adobe will automatically append all of the individual files into this one PDF document for us.

HTML SYNTAX FOR A TABLE OF CONTENTS

For the purposes of creating the TOC we just need to use simple HTML syntax and use the table construct to create our TOC. There are numerous comprehensive guides to HTML available on various websites so this paper will not go into detail about the syntax required but essentially the HTML file follows the below structure:

```
<!–All HTML files start with a HTML tag-->
<html>
 <head>
  <!–Title tag is used to give the title of the HTML file-->
  <!–This will be how the TOC appears in the list of bookmarks-->
  <title>Table of Contents</title>
 </head>
 <body>
  <!–table tag starts a HTML table-->
  <table>
   <!–each row of the table starts with a tr tag-->
   <tr>
    <!–each column of the table is given by a td tag-->
    <td>
     <!–the a tag is used in HTML for a hyperlink-->
     <!–this will appear as a link to the TFL in the final PDF document-->
     <!–also used by Acrobat to append the TFL file and add PDF bookmark-->
     <a href='file.pdf'>Table 14.1.1</a>
    </td>
    <td>Title</td>
    <td>Page #</td>
   </tr>
   extra rows as needed
  </table>
 </body>
</html>
```

As the above example from the Phuse website shows, Acrobat can convert almost anything from the HTML file into the final PDF so we can also add any formatting that is required. Again, this paper won't go into any details of more advanced formatting of HTML files but tutorials for this are plentiful online. Images could also be added so it would be possible to add company logos if required. Using these basic HTML constructs, its relatively easy to develop a standard template that can be used as across multiple studies and deliverables so that there is consistency across different delivery packages.

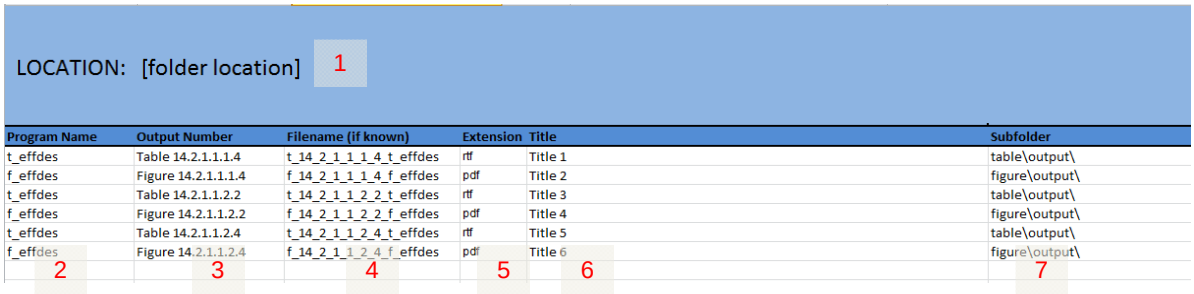
CREATING THE TOOL

Initially the tool was developed in Excel because some TFL metadata was already available in Excel format and so it was easy to copy the required information into the tool and then run VBA macros using this information. This would create a Word VBA macro that could be copied into Word and run to pull individual RTF files into one Word document, with a table of contents, and this would then be converted into a PDF document. This solution worked well for some deliverables but still required some level of user interaction and also still faced the issues of Word becoming unreliable when multiple large TFL files were combined together. Thus it was necessary to take the tool in a different direction.

We decided to use the functionality to create a PDF from a HTML file in Acrobat to create the final package and continued to use Excel as our input method to link in with our existing metadata which could be read in easily into an Excel document. An VBA macro written in Excel would then cycle through the list of TFLs provided, performing any conversion tasks before creating the HTML file which would be read into Acrobat to create the final PDF file.

Preliminary testing of this method showed that although the HTML file could specify links to the individual TFLs in any file format and Acrobat would be able to append these to the final package, it would could only be done with a default page setup decided by Adobe and often this wouldn't be what we required. Therefore, the user would need to either have already created PDF files when the TFLs were originally created, or the tool would need also handle the conversion of the individual TFL files to PDF before reading the HTML file into Acrobat. We determined that acceptable file types would be limited to only those that could be opened in MS Word and with most studies using RTF files this would not be a problem.

We designed the input tab of the document as follows:



(1) is used to set the location where the individual TFL files should be picked up from. This can either be one folder for all of the files, or can be used for a root folder with the specific subfolder for each file given by the subfolder column (7)

(2/3) program name and output number are copied from the TFL metadata files

(4) the tool automatically generates the filename using the convention for the way that the program name and output number are combined. If the filename does not follow this convention then the user has the option of inputting their own filename for each file

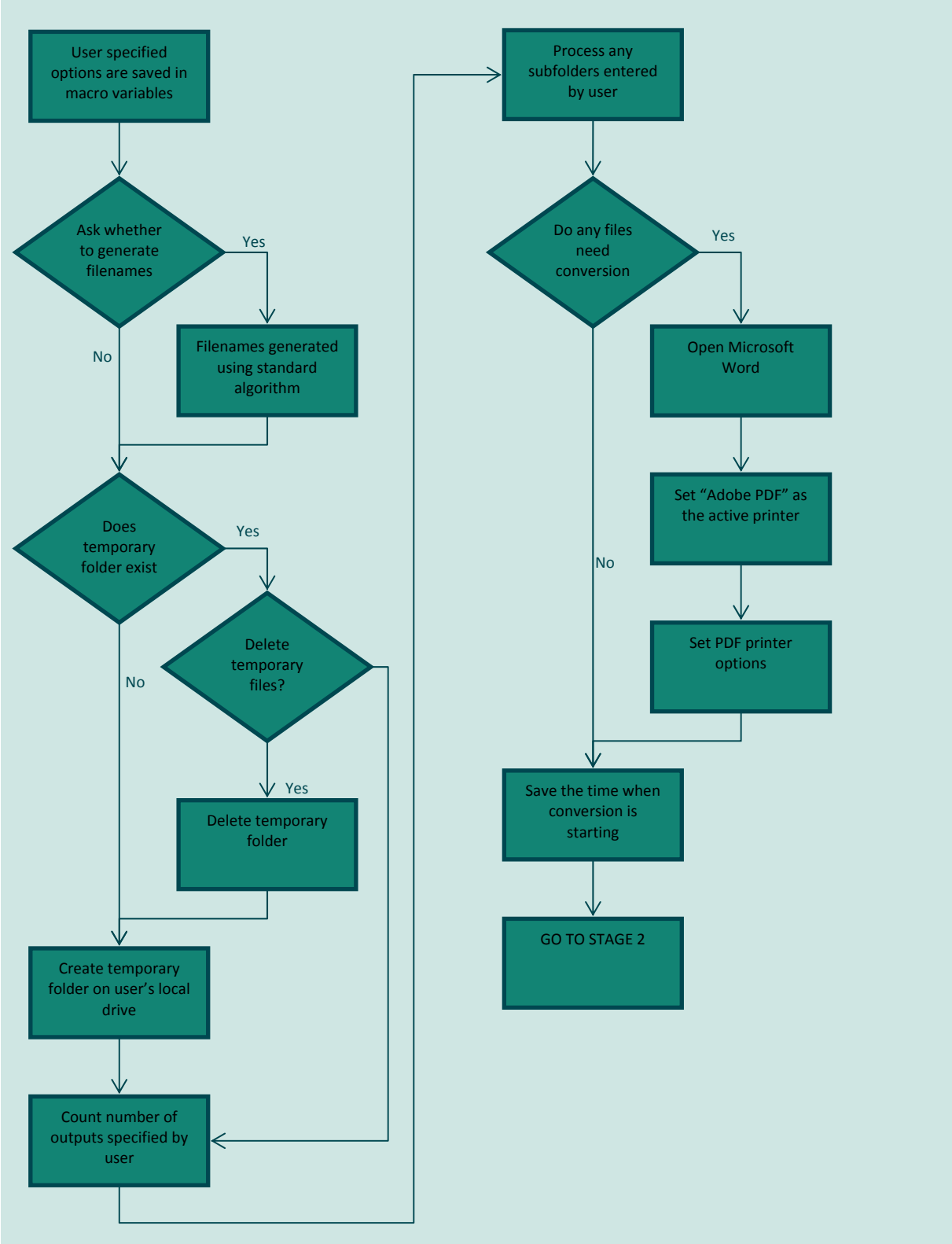
(5) file extension for the individual TFL file. Used to determine whether any intermediate conversions are needed before creating final TFL package

(6) TFL title which is used for the TOC in the final package. This is merged in directly from the TFL metadata to ensure that the TOC is correct

(7) subfolder can be used to pick up each TFL file from different folders

Using this information from the user, the process flow that we needed to follow was determined as:

Stage 1 – Processing User Options



From testing it was determined that if the individual output files were copied to a local folder before conversion and converted from there, the whole process ran much quicker. This also has the advantage that the user has access to the converted PDF files in their local directory and can retain them if required. In the process flow, the macros check if this temporary folder already exists and creates the folder if it doesn't, or asks the user whether to delete the files

that are currently in this folder. In this way, if the user wants to re-use one of the files converted in an earlier run, they can choose to keep this and save time by not having to re-convert.

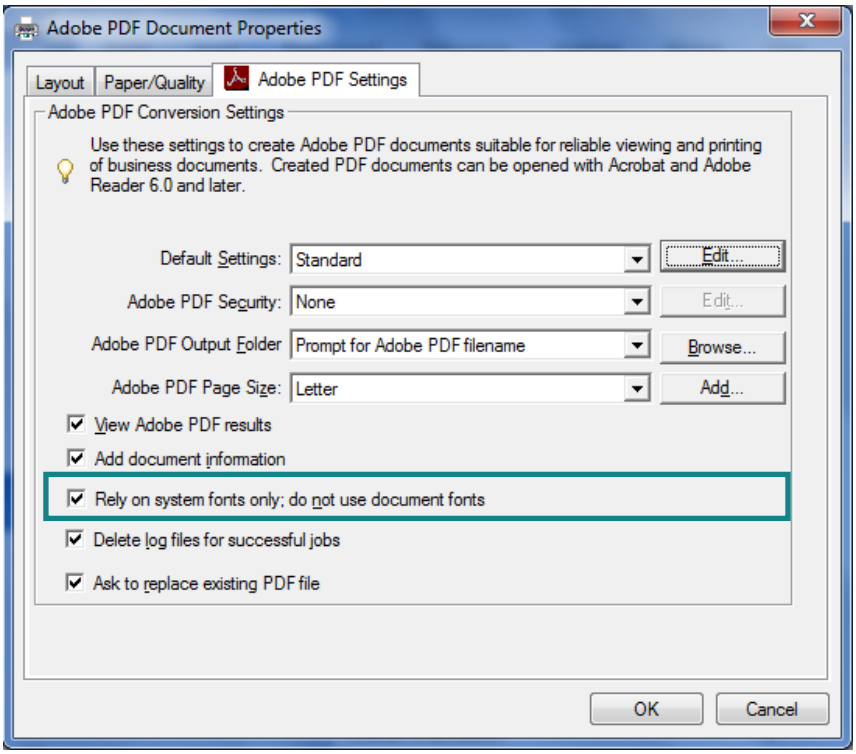
As we were using Excel for input and Excel VBA macros to control the tool, we were able to make use of the Word Application object to control Word directly from the Excel macros. The syntax required to start a new instance of MS Word with a blank document is:

```
Dim objword As Object
Set objword = CreateObject("Word.Application")

objword.Visible = True
objword.documents.Add
```

Our preferred method of creating the PDF file, as it can be done completely in the background and proved to work quicker, was to print the original file to a postscript file, and then use the PDF Distiller object that is available in VBA to convert this to PDF format. In order to allow the initial creation of the postscript file we had to set the active printer to Adobe PDF using the following syntax:

```
objword.ActivePrinter = "Adobe PDF"
```

As well as setting the printer, we also found that the default option “Rely on system fonts only; do not use document fonts” meant that the step of converting the postscript file caused an error every time it was run and that we needed to “switch off” this option. Initially, the excel macros would run up to the point of opening MS Word but would pause until the user confirmed that they had manually adjusted the print options.

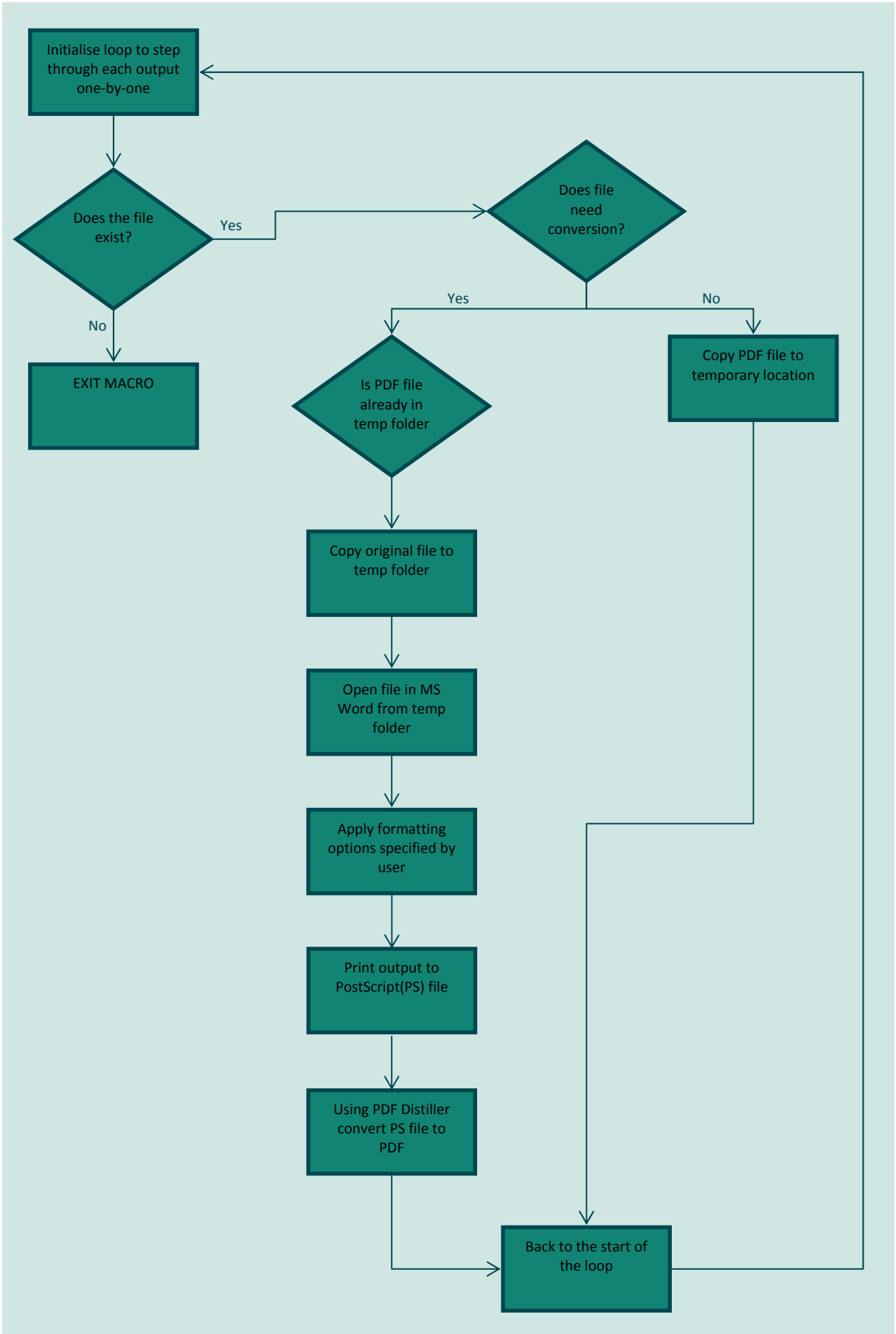


As this was open to a user error, and did not accomplish the goal of automating the whole process, we looked for a new way of setting this print options, and found the SendKeys method which is available in VBA to send keystrokes to the active application. Using this, once we had set the new instance of Word that the macro opened as the active application, we could use the key shortcuts available in Word to open the print options screen, change the option we needed, and then close the print options screen, all without any input from the user. This process was documented in the user guide that we produced so that the user was aware that while this was happenning, then needed to not touch the keyboard or activate any other applications, as this would cause incorrect keypresses or keypresses to be sent to the wrong application.

The syntax that we defined for this SendKeys section which only works with Office 2010 is:

```
'Sends a Alt keypress
SendKeys "%", True
Application.Wait (Now() + TimeValue("00:00:01"))

'Sends F to open the file tab on the ribbon
SendKeys "F", True
Application.Wait (Now() + TimeValue("00:00:01"))

'Sends P to view the print screen
SendKeys "P", True
Application.Wait (Now() + TimeValue("00:00:01"))

'Sends R to open the print properties window (shown above)
SendKeys "R", True
Application.Wait (Now() + TimeValue("00:00:01"))

'Sends TAB to move out of the first drop down box
SendKeys "{TAB}", True
Application.Wait (Now() + TimeValue("00:00:01"))

'Sends N to toggle the setting of the Rely on system fonts option
SendKeys "N", True
Application.Wait (Now() + TimeValue("00:00:01"))

'Sends 3 presses of TAB to move to the OK button
SendKeys "{TAB 3}", True
Application.Wait (Now() + TimeValue("00:00:01"))

'Sends ENTER to press the OK button
SendKeys "{ENTER}", True
Application.Wait (Now() + TimeValue("00:00:01"))
```

Stage 2 – Copying and converting individual files



If the user has specified a file type other than PDF then stage 2 handles the conversion of that file to PDF. As mentioned above, the conversions are done by copying the original file to a local folder as this proved to be quicker, and for each file that needs to be converted a check is done to see if a converted file already exists in the temporary folder. This gives the user the option of skipping the conversion if the original file has not changed, as it is the conversion of individual files that is the most time-consuming part of the whole process.

A further enhancement of this is to compare the timestamps of the converted file that already exists and the original file and only present the user the option of skipping the conversion if the original file has not changed since the converted file was created. The check is accomplished by using the Scripting.FileSystemObject object available in VBA to directly access the file and its attributes to access the date/time when it was last modified. This can be setup as a user-defined VBA function as:

```
Function FileLastModified(strFullFileName As String)

    Dim fs As Object, f As Object, s As String

    Set fs = CreateObject("Scripting.FileSystemObject")
    Set f = fs.GetFile(strFullFileName)

    FileLastModified = f.DateLastModified

    Set fs = Nothing: Set f = Nothing

End Function
```

This can then be used to compare the dates of the converted file and original file as follows:

```
If FileLastModified(convfile) < FileLastModified(origFile) Then


End If
```

As discussed above, the actual conversion to PDF is done in two steps. The first is to open the file using the Word object created in stage 1 and then print this out to a postscript (PS) file, and then the second is to use the PDF Distiller, which is also available as an object in VBA, to convert the PS file to PDF.

The syntax to open a file in word, using the `objword` object created above is:

```
Set objdoc = objword.documents.Open(NewFile)
```

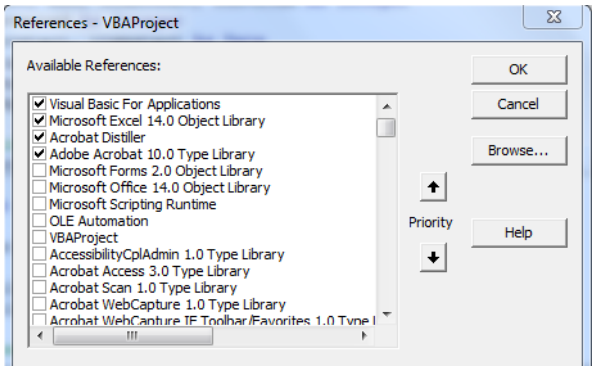
Once the objdoc object holds the word document, we can use this object to print this out to the PS file using the following syntax:

```
objdoc.PrintOut PrintToFile:=True, OutputFilename:=tempPSFileName
```

Once Excel sends this command through to Word, it immediately moves on to the next step in the macro which is to convert the PS file to PDF. However, this will not work if the objdoc object is still creating the PS file which can take several minutes for very large TFL files. In order to only allow the Excel macro to move to the next step a loop is setup to check the PS file size, and the loop is only exited when the file size has stopped changing.

```
'get initial file size
lngFSize = FileLen(tempPSFileName)
'initialise flag to 0 to enter the loop at least once
flag = 0
i = 1

Do While (flag = 0)
  'Get the new file size
  newsize = FileLen(tempPSFileName)
  'If the file size is the same as the previous loop then set flag to exit the loop
  If newsize = lngFSize Then
    flag = 1
  Else
    'If file size is still changing then save the current file size
    lngFSize = newsize
    'For the first 10 loops, wait 2 seconds before going round the loop again
    'After 10 loops wait 10 seconds before going round loop again
    If i > 10 Then
      Application.Wait (Now() + TimeValue("00:00:10"))
    Else
      Application.Wait (Now() + TimeValue("00:00:02"))
    End If
    i = i + 1

  End If
Loop
```

Once the loop exits, the macro moves onto the next step which is to convert the PS file to PDF. This is done using the PDF Distiller object which is installed when Adobe Acrobat Pro is installed in the environment where the tool is being used and needs to be included by adding it in the references window in Visual Basic:



These references are saved when the document is saved so assuming that the libraries are available in the environment where the file is opened (they are installed when a version of Adobe Acrobat Pro is installed) then the references will work.

A PDFDistiller object is created using the following syntax:

```
Dim mypdfDist As New PdfDistiller
```

And then the method to convert the PS file to PDF is called using:

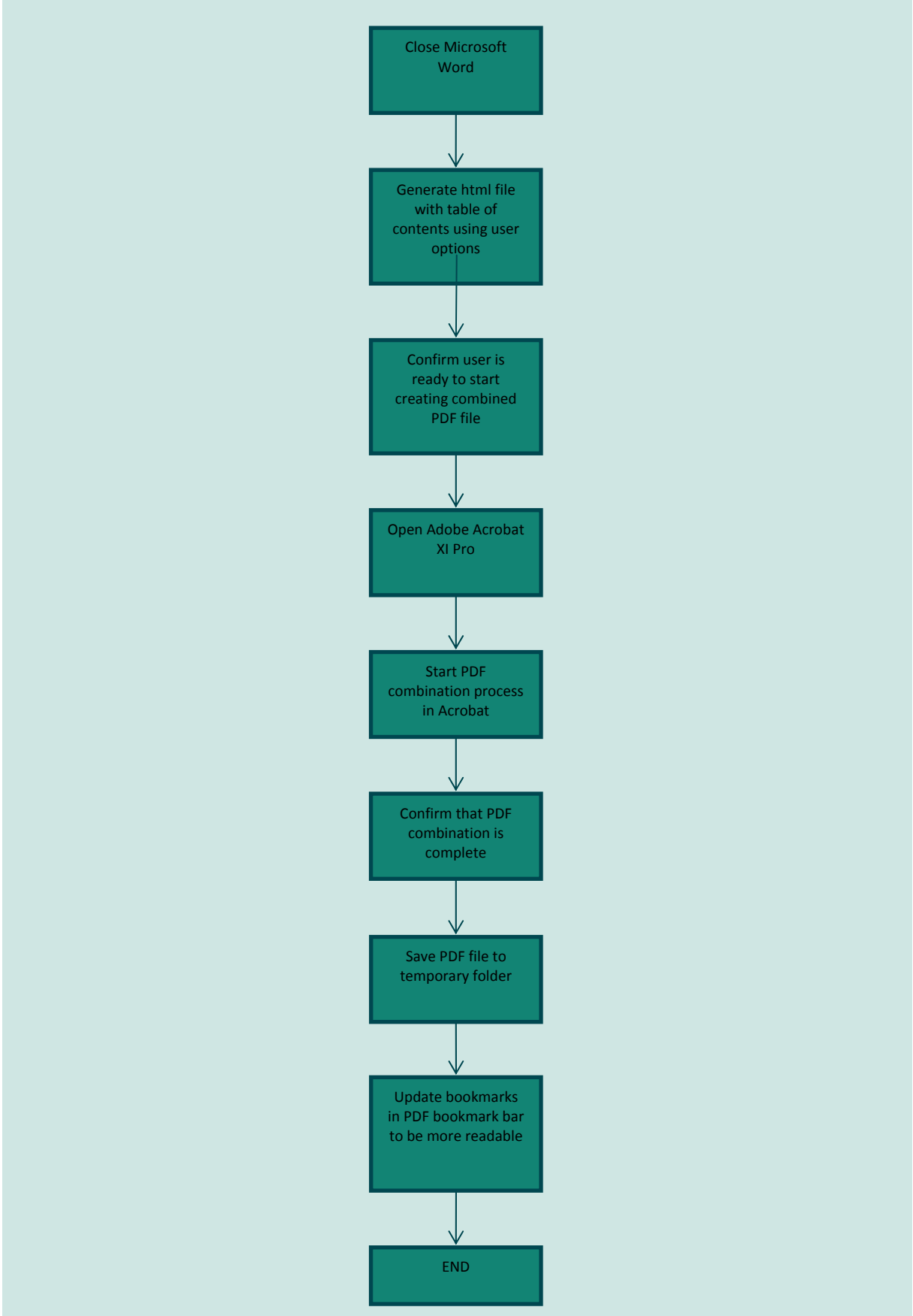
```
mypdfDist.FileToPDF tempPSFileName, tempPDFFileName, ""
```

Similarly to the way the PS file is created, once the macro invokes the FileToPDF method, it immediately moves onto the next line, even if the PDF Distiller has not finished creating the PDF file. Therefore another loop is setup, this time to loop around until the PDF file exists:

```
'the dir function returns the filename if it exists, or returns an empty string if it does not exist
Do Until Dir(tempPDFFileName, vbNormal) <> ""
  Application.Wait (Now() + TimeValue("00:00:01"))
Loop
```

Once this loop has finished, and the PDF file exists, then the macro continues in its larger loop until all of the PDF files have been created. Once this larger loop has finished, it moves to stage 3 to create the HTML file with the TOC and read this into Adobe.

Stage 3 – Creating Combined PDF File with TOC



After all the individual PDF files are created, the HTML file containing the TOC and links to the PDF files is created by the macro code. This works by opening a HTML for output using the Open command in VBA, adding the necessary header information at the top, and then looping through each row on the input sheet. For each iteration of the loop it adds a cell with the table number and a cell with the table title, with the number also acting as a link to the individual PDF file for the TFL:

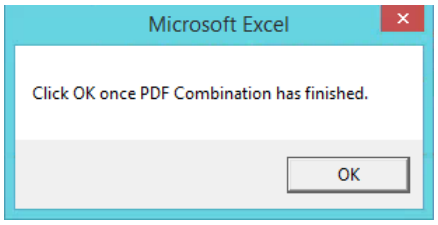
```
Private Sub AddRowXL(f As Integer, txt As String)
   If f = 1 Then
    Print #1, txt
   End If
End Sub

…

Open "U:\STFCToolTemp\pdf.html" For Output As 1

AddRowXL 1, "<html>"
AddRowXL 1, " <head>"
AddRowXL 1, "  <title>"
AddRowXL 1, "   Table of Contents"
AddRowXL 1, "  </title>"
AddRowXL 1, "  <style>"
AddRowXL 1, "   th {font-size: 8; font-family : Arial; }"
AddRowXL 1, "   td {font-size: 8; font-family : Courier New; }"
AddRowXL 1, "   a {color: blue;"
AddRowXL 1, "      text-decoration: none}"
AddRowXL 1, "  </style>"
AddRowXL 1, " </head>"
AddRowXL 1, " <body>"
AddRowXL 1, " <table cellspacing=0 cellpadding=2 width='100%'>"
AddRowXL 1, "  <tr><th>" & StudyTitle & "</th></tr>"
AddRowXL 1, "  <tr><th>" & DeliveryTitle & "</th></tr>"
AddRowXL 1, "   <tr><th>Date of Delivery: " & Format(DeliveryDate, "ddmmmyyyy") &
"</th></tr>"
AddRowXL 1, " </table>"
AddRowXL 1, " <br>"
AddRowXL 1, " <table cellspacing=0 cellpadding=2 width='100%'>"

'Loop through entries in the input sheet
For outn = 1 To count
  tabnum = Sheets("Input").Cells(outn + 3, 3).Value
  fname = Sheets("Input").Cells(outn + 3, 5).Value
  ext = Sheets("Input").Cells(outn + 3, 6).Value
  ttl = Sheets("Input").Cells(outn + 3, 7).Value

  Row = Row + 1

  AddRowXL 1, "   <tr>"
  AddRowXL 1, "       <td width='20%'><a href='" & fname & ".pdf'>" & tabnum &
"</a></td>"
  AddRowXL 1, "     <td><a href='" & fname & ".pdf'>" & ttl & "</a></td>"
  AddRowXL 1, "   </tr>"
Next outn

AddRowXL 1, " </table>"
AddRowXL 1, " </body>"
AddRowXL 1, "</html>"

Close 1
```

After this HTML file is generated, the user is shown a dialog box asking them to confirm whether they are ready to move to the next stage of the process. The reason for this is in order to read in the HTML file, the tool needs to open Adobe Acrobat Pro and use the SendKeys method to send keypresses to the application and if this just popped up while the user was working in another applicatio this could cause the keypreeses to be sent to the wrong application. Once the user has confirmed that they are ready for the macro to procede, the macro uses the AcroExch object to create an Adobe Acrobat object and set it as the active application so that the keypresses can be sent:

```
Set Acroapp = CreateObject("AcroExch.App", "")
Acroapp.Show
```

The AcroExch app is part of the Inter Application Communication (IAC) library that is made available by Adobe Acrobat and uses OLE through Windows. This provides a library of objects that we can reference directly in the VBA code to open documents Adobe Acrobat and modify them using the information that we are supplying in our input. These libraries allow us to automate steps of the process that we would otherwise have to ask the user to perform manually so counts towards our goal of making the new tool more automated. This library is called “Adobe Acrobat 10.0 Type Library” and is made available in the same way that the PDF Distiller library is made available, as described above. Unfortunately these libraries do not provide any way of creating a PDF file from HTML which is why we have to use SendKeys to send the keypresses to perform these tasks.

The process of appending the PDF files together can take a several minutes so the Excel macro needs to pause while this is happening. To accomplish this, the user is shown the following dialog box and asked to confirm when Acrobat has finished creating the combined PDF file:



Once Acrobat has finished making the combined PDF file there is one final step to save the file and to re-format the bookmarks. Unless bookmarks have been added to the original PDF file, which is typically not the case, the default bookmarks are used which uses the filename of the original TFL file.

To save the file, again the SendKeys method is used to send the keypresses needed to the Acrobat application:

```
‘After reading in the HTML file, the application name will change
‘We know that this will be the title because of the way that the HTML file has been
created
AppActivate "Table of Contents - Adobe Acrobat Pro"
Application.Wait (Now() + TimeValue("00:00:01"))

‘Send ALT
SendKeys "%", True
Application.Wait (Now() + TimeValue("00:00:01"))

‘Send F to open file menu
SendKeys "F", True
Application.Wait (Now() + TimeValue("00:00:01"))

‘Send A to invoke the Save As command
SendKeys "A", True
Application.Wait (Now() + TimeValue("00:00:05"))

‘Save PDF file temporarily
SendKeys filename, True
Application.Wait (Now() + TimeValue("00:00:01"))

‘Confirm saving the file
SendKeys "{ENTER}", True
Application.Wait (Now() + TimeValue("00:00:10"))
```

Once the file is saved, we can use two other OLE objects to access the PDF directly from the macro code and modify the bookmarks and some of the file properties. These objects are defined in two layers, the application (AV) layer and the portable document (PD) layer. The AV layer controls the user interface and can open documents in a visible application while the PD layer provides access to the information within the document and can perform basic manipulations of the document. The syntax to create instances of these objects is:

```
Set PDYourDoc = CreateObject("AcroExch.PDDoc", "")
Set AVYourDoc = CreateObject("AcroExch.AVDoc", "")

'Open the file that was saved in the step above
bFileOpen4 = AVYourDoc.Open("C:\pdf.pdf", "Package")

'Load the PD layer for the file that we just opened
Set PDYourDoc = AVYourDoc.GetPDDoc
```

Once the file is open and we have access to the PD layer we can update the default bookmarks using another OLE object called the PDBookmark object. This has two methods that we will use to change the bookmark text, GetByTitle and SetTitle. First we use the method GetByTitle to load the bookmark with the title “Local Disk”, which is the default name for the first bookmark:

```
Set PDBookmark = CreateObject("AcroExch.PDBookmark")

PDBookmark.GetByTitle PDYourDoc, "Local Disk"
```

Then, once this bookmark is loaded, we set the title of this bookmark to “Bookmarks”:

```
PDBookmark.SetTitle "Bookmarks"
```

Because the PDBookmark, PDDoc and AVDoc are all linked together, this update is displayed in the visible Acrobat window as well as being updated in the PD layer.

The macro then enters a loop to cycle through each row of the input tab and update the bookmark from the default text (which is based on the filename and folder where the converted TFL files are saved) to display the table number as the bookmark text.

Finally, the settings for default page view are changed so that when the document is opened the bookmarks pane is displayed and the title and author are updated:

```
PDYourDoc.SetPageMode 3
PDYourDoc.SetInfo "Title", title
PDYourDoc.SetInfo "Author", author
```

CONCLUSION

This paper has shown how using a Microsoft Excel input sheet and a mixture of different objects within Excel VBA macro code to interact directly with Adobe Acrobat and Microsoft Word can create a robust system to combine individual TFL files into one PDF package with a table of contents and bookmarks in a standard format. The use of the SendKeys method enables greater automation by sending key presses direct to the necessary applications instead of requiring the user to perform them manually.

Future enhancements may include:

- Better integration with TFL metadata so that less input information is required
- Adding page numbers to the TOC by analysing the TFL files as they are converted
- Estimates of run times using file sizing

REFERENCES

[1] Adobe. (2006, November). Developing Applications Using Interapplication Communication. Retrieved 2015, from Adobe website: http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/iac_developer_guide.pdf

CONTACT INFORMATION

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

Mark Crangle
2 Globeside Business Park
Marlow
Buckinghamshire
United Kingdom
SL7 1HZ
Email: Mark.Crangle@iconplc.com