Merging RTF files using SAS®, MSWord®, and Acrobat Distiller®

Oliver Wirtz

Global Statistical Programming, Global Data Operations, Monheim, Germany

Producing tables, listings, and figures (TLF) is one of the key responsibilities for programmers in the pharmaceutical industry. While SAS® Software offers a wide range of procedures to produce outputs in nearly all possible formats, collating multiple documents into an overall PDF file is not a trivial task. This paper will combine main aspects and show how you can implement this procedure within your organization. All you need is SAS, MSWord® and Acrobat Distiller®.

Keywords: RTF, PDF, PS, VBScript, Acrobat Distiller

Introduction

Programmers are often faced with tough timelines to finalise tables, listings, and figures (TLF). Once all TLF have been produced you end up with hundreds of single files, most of them showing just one table, listing or graph. And now the real challenge begins, because all outputs need to be reviewed by the study team and will later be archived in a document management system. Having a single merged file in Portable Document Format (PDF) is a big help for everyone: all files are in one place, an overall table of contents (TOC) links to each document in the file, bookmarks can be used to categorise the document by sublevels of interest e.g. by parameter or subject. Producing such a document is a lot of work if it is performed manually using, e.g. Acrobat Professional. And not all nifty details we want to have in the final PDF are easy to implement, if possible at all. Moreover, all steps need to be repeated once a new version of TLF is produced. There are already publications available which deal with this issue.[1,2]

In the following sections I will show how to merge existing SAS® Rich Text File (RTF) output to an overall PDF. The main idea is to use SAS to produce the output as RTF, convert it to PostScript (PS) with MSWord® and to create a TOC using metadata of the RTF outputs. A prologue.ps file with the interactive parts like bookmarks and links is also generated based on metadata. Finally, we use Acrobat Distiller® to merge all PS files to one big PDF. Figure 1 shows a scheme of the general workflow.

A SAS macro (%printAll) will be used to control MSWord via a Visual Basic Script (VBS) to produce PS files from RTF. Metadata about the currently processed file is then saved in the metadata dataset. After this process has been repeated for all RTF files, a second SAS macro (%TOC) uses the metadata dataset to produce a TOC, a prologue file and a batch file. The prologue file consists of PostScript code for links, bookmarks and watermarks. The batch file simply points to all PS files and is the only input file for Acrobat Distiller. Finally, Acrobat Distiller merges the PS files and the prologue information to the overall PDF document.

Correspondence to: Oliver Wirtz, UCB Biosciences GmbH, Alfred-Nobel-Str. 10, 40789 Monheim, Germany. Email: oliver.wirtz@web.de

Environment Settings

Windows Server 2003
- SAS 9.2 via Citrix
- Microsoft Office Word 2007
- Acrobat Distiller 9.0

Setup of the Metadata File

Which information do we need to set up a merged PDF file? Well, surely the name of the RTF source files to form the PDF in the end. Additionally, we want a TOC in the final file to show titles and page numbers such as shown in Fig. 2.

The very basic information like table number and title can be obtained very easy as most programmers use a static dataset to produce headers and footnotes of TLF. After filling this dataset with the filename of the output file we already have 90% of our TOC available without too much effort. What we also need to know is the page orientation and the number of pages of each document. Knowing about what we have and what we still need to collect along the way we can set up the metadata file (Table 1).

The missing information in the metadata file will be obtained during next steps and updated. Later, we will use this file to output a TOC using table number (tabid), title (tabtitle) and number of pages (pdfpage).

Published by Maney
DOI 10.1179/1757092112Z.0000000008

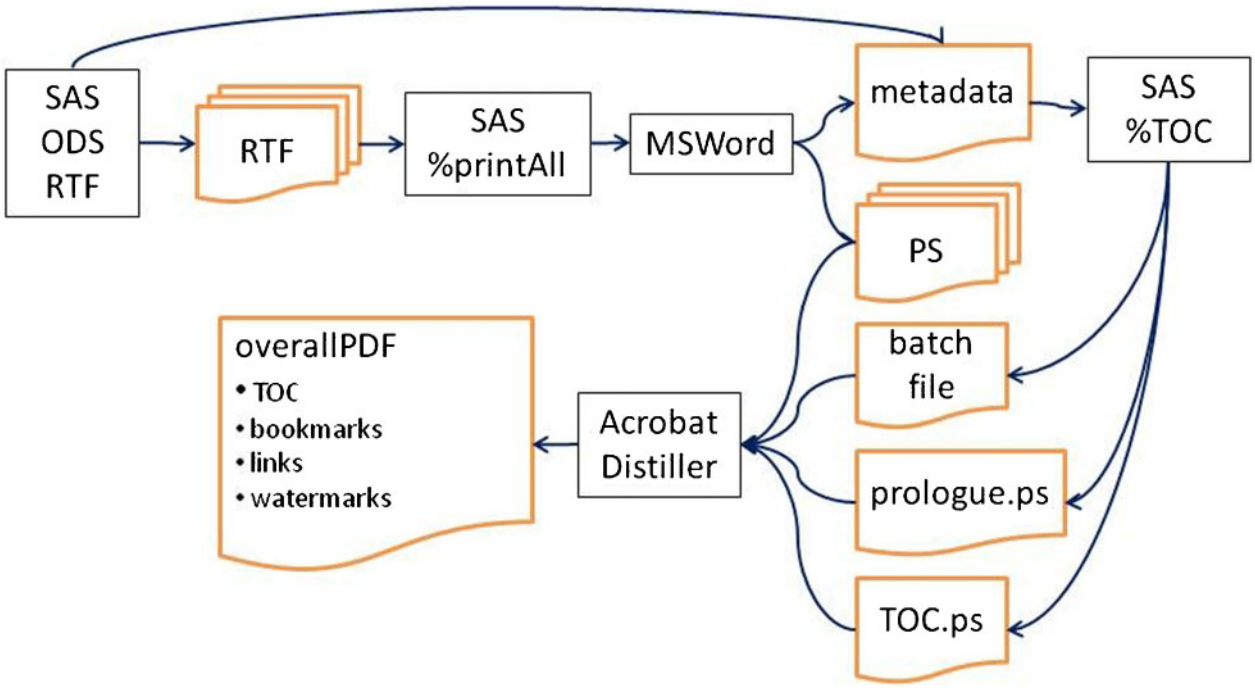

Figure 1 General set-up of the post-production process

Table of Contents

Figure 2 Sample TOC

Setting up A Postscript Printer

PostScript was invented in the early 80s by Adobe Systems as a page description language. It is/was used 'to tell' printers how a printed page should look. Acrobat Distiller is part of the Acrobat Professional package and used to convert PS files to PDF. So, in order to be able to produce PDF files we first need our RTF source converted to PS. This is performed using a virtual PostScript printer. This printer can be chosen like any other 'real' printer in the print menu. But instead of printing on paper it will print PostScript code into a file. Setting up a PostScript printer is as easy as installing any other printer. The following method is using PostScript drivers available in Windows. Right click on 'Printer' in the system settings and choose 'Add Printer'→'Local Printer'. Then choose 'FILE:' in the drop down menu and click on 'Next'. Then, choose a printer which is capable to print PostScript (e.g. HP LaserJet 4L/4ML PostScript) and choose 'Next'. You may choose another name for the new printer but for now let us call it 'PostScript'. Finish the installation by clicking 'Next'. The new printer will now be available in the printers menu like any other printer. However, by preset this printer can only print text. Printouts of figures may look odd. In order to be able to print graphics correctly, right-click on the printer icon and choose 'Preferences'. Choose 'Settings'→'More…'. Under 'PostScript Options' choose 'PostScript Output Options' and set these to 'Optimize Portability'. This

Table 1 Sample metadata table

output	prog_name	tabid	tabtitle	orientation	pdfpage	psresult
Table	T_14_01_04	14.1.4	Disposition			T_14_01_04_disp
Table	T_14_01_05	14.1.5	Enrolment by Country and Site			T_14_01_05_enrol
Table	T_14_01_06	14.1.6	Demography			T_14_01_06_demo

ensures that graphics are printed as they appear in the RTF.

All following steps are implemented into macros using the method described by Agator.[3] However, the important bits are shown as open code.

Generating VBSCRIPT Code to Control MSword

As already said above, a virtual PostScript printer can be used like any other 'physical' printer. So, you could now open an RTF file in MSWord, choose the PostScript printer from the printing menu and let the file print as PostScript. However, we surely won't do that manually for all our source files. Instead we will use some Word-VBA (Visual Basic for Applications) to automate this job. There are various techniques available to submit VBA commands. The one shown here uses VBScript[4] to open Word and to submit VBA commands.[5] The script itself is executed using SAS and basically opens the source RTF in MSWord, executes Word-VBA code to gather metadata (number of pages and page orientation) and to print the file to PostScript.

A sample VBScript file would look like the following:

```
Dim arguments
Set arguments=WScript.Arguments
Function DOC2PS( sDocFile, sPSFile )
Dim fso
Dim wdo
Dim wdoc
Dim wdocs
Dim mypages
Dim sTempFile
Dim tempfile
'[1] Create a scripting object to get access to the file
 system object
Set fso = CreateObject('Scripting.FileSystemObject')
'[2] Create a Word object to get access to Word-
 VBA objects
Set wdo = CreateObject('Word.Application')
Set wdocs = wdo.Documents
'[3] Gather information about paths
sDocFile = fso.GetAbsolutePathName(sDocFile)
sFolder = fso.GetParentFolderName(sDocFile)
If Len(sPSFile)=0 Then
   sPSFile = fso.GetBaseName(sDocFile) + '.ps'
   sTempFile = '_temp.txt'
End If
If Len(fso.GetParentFolderName(sPSFile))=0 Then
   sPSFile = sFolder + '\' + sPSFile
   sTempFile = sFolder + '\_temp.txt'
End If
'[4] Open RTF in Word and print it via PostScript
 Printer
Set wdoc = wdocs.Open(sDocFile)
wdo.ActivePrinter = 'PostScript'
wdo.ActiveDocument.PrintOut False , , , sPSFile
'[5] Read out metadata
Set tempfile = fso.CreateTextFile(sTempFile, True)
mypages=wdoc.ComputeStatistics(2)
tempfile.Writeline(mypages)
mypages=wdoc.sections(1).pagesetup.orientation
tempfile.writeline(mypages)
'[6] Housekeeping…
tempfile.close
wdoc.Close 0
wdo.Quit 0
Set wdo = Nothing
Set fso = Nothing
End Function
Call DOC2PS( arguments.Unnamed.Item(0), arguments.
 Named.Item('o') )
```

The script simply consists of a VBScript function (DOC2PS) which needs the location of the RTF source file as argument. After creating the necessary objects (1, 2) the function extracts the root path of the source document and assigns the destination paths for the PS file and a temporary text file (3). After that, it opens the RTF source, assigns the PostScript printer and prints the file to the previously defined PS destination (4). The temporary file is created and the number of pages and the page orientation is read out and saved into the file (5). Once that is performed, all files are closed and all objects are deleted (6). Many more manipulations are possible once the document is opened and you may want to play around with different Word-VBA functions to implement other features.

You can either save the script somewhere on your machine or hard-code it in a SAS program to create a temporary VBScript file script every time you need it. A major advantage of this technique is that version control is very easy. In order to execute the script defined above the following call needs to be submitted to the Windows command line:

'cscript /nologo path/to/myVBScript.vbs /nologo path/to/myRTF.Doc'

The Cscript command starts the Windows Scripting Host and executes the VBS file/function which needs the name of the source RTF file as argument. The option /nologo prevents display of the logo at run time (by default, the logo is displayed). We will use the SYSTASK statement together with the WAITFOR statement, to make SAS execute the statement:

```
data _null_;
     systask command 'cscript />nologo
     path/to/myVBScript.vbs /nologo
     path/to/myRTF.Doc' taskname=
     PRINT2PS;
     waitfor PRINT2PS TIMEOUT=
     &timeout.;
```

```
      systask command 'taskkill /IM
      WINWORD.EXE /F' ;
run;
```

The first systask statement sends the command to the Windows command line and assigns a name for this task. In theory you could start the script as often as you like, because systask allows that tasks execute asynchronically. However, every task started would need its own Word instance. So, you may run into memory and/or performance issues if you start more than one task at a time. The WAITFOR option is used to define a timeout for the task. This statement will make SAS wait for the script operations to finish before executing the next steps. However, if for some reason the script job does not finish a timeout time should be set. The timeout should have a value around 6000 (seconds), because very large documents (listings!!) need some time until they have opened and printed. The second SYSTASK statement is submitted to close any open Word session. This statement is useful for debugging if SAS and MSWord run on a server and for some reason the script crashes before MSWord has been closed.

Updating the Metadata File

Once the RTF has been printed into PS, we want to save the data from the temporary file created by MSWord which has the information about number of pages and page orientation. There are only two lines: the first shows the number of pages, the data in the second line show a '1' if the page orientation was landscape and '0' if portrait. The temporary file is imported to SAS and the metadata is updated to look like shown in Table 2.

The steps above will be repeated for each RTF source file.

Preparing the TOC

We now have a metadata file with all information needed to produce a TOC and bookmarks in our final merged PDF. First of all, we want to bring the metadata into shape and use it to print the TOC as RTF. Then, we convert it to PS using the same technique, as we did for the other RTF files. The resulting underlying dataset may look like Table 3.

In the final document, we want to see the TOC as a separate document. Page numbers of the TOC are shown as roman literals and the overall pagination of the main documents starts with 1. Every page of the main document will have a page number 'Overall page x of y' including a link back to the TOC. Moreover, each row in the TOC will have a link to the start page of the respective table.

The page numbers displayed in the TOC (variable TOCpage) were simply calculated starting with page 1 and then adding the number of pages in variable pdfpage. We also need to know the total sum of pages up to the current row in the dataset (sum). The total number of pages of the TOC and the total number of pages of the main document will be stored in macro variables &TOCNum and &totalPages.

Variables TOCtxt and TOCpage now form the TOC text and are simply printed as a table using ODS RTF. This ensures that the TOC entries are always in the same location on the page. Knowing that, it is possible to find out by trial and error where the first link must be placed to lie over the TOC text. Variable box is then calculated for all following rows depending of the coordinates of this first link. After converting the RTF to PS we will use the TOC dataset to set up the prologue.ps file which contains the postscript code to insert bookmarks, links, overall pagination and watermarks.

Preparing the Prologue.ps File

The prologue file is the most important file of the process, since it contains all the formatting information we normally would do manually in Acrobat Professional. Most features are implemented using pdfmark operators which code bookmarks and other features in PS language. A complete guide to pdfmarks can be found in Ref. 6. In the next steps data _null_ will be used to output PS code to the prologue.ps. First of all, we need to initialize the file with some PostScript code to make the file work in

Table 2 Sample metadata table after update

output	prog_name	tabid	tabtitle	orientation	pdfpage	psresult
Table	T_14_01_04	14.1.4	Disposition	LANDSCAPE	1	T_14_01_04_disp
Table	T_14_01_05	14.1.5	Enrolment by Country and Site			T_14_01_05_enrol
Table	T_14_01_06	14.1.6	Demography			T_14_01_06_demo

Table 3 Sample TOC data table

TOCtxt	orientation	pdfpage	TOC page	Sum	box	TOC
Table 14.1.4 Disposition – Double Blind Population	LANDSCAPE	1	1	2	Rect[78 51 100 80]	1
Table 14.1.5 Enrolment by Country and Site – Enrolled Population	PORTRAIT	2	2	4	Rect[112 51 125 80]	1
Table 14.1.6 Demography – Double Blind Population	PORTRAIT	1	4	5	Rect[146 51 150 80]	1

multiple environments and PostScript levels. Its not really mandatory in this project since we want to use the files in Acrobat Distiller only. However, considering possible future use of the files is always advisable:

```
%
% prologue.ps
%
% This file is used to produce annota-
tions, bookmarks and links (Processing
prologue.ps...\n) print
flush

/pdfmark where { pop} { userdict /
pdfmark /cleartomark load put} ifelse
/languagelevel where { pop languagele-
vel} { 1} ifelse
2 lt {
userdict (<<) cvn ([ ) cvn load put
userdict (>>) cvn (] ) cvn load put
} if
```

Then we define a bookmark to the TOC, which always starts on page 1.

```
[ /Page 1 /View [ /Fit] /Title (Table of
Contents) /OUT pdfmark
```

The following lines define the display of page numbers later shown in the status bar in Acrobat Reader. We want to show all TOC pages as roman literals to separate the TOC from the main document. The status bar counts PDF pages from 0. For this reason the number of the first PDF page after the TOC is equal to the total number of TOC pages (&TOCnum).

```
[ { Catalog} << /PageLabels <<
/Nums [ 0 << /S /r >> % Roman literals
starting from physical pdf page 1
&TOCnum << /S /D /St 1 >> % arabic
numeral starting from pdf page &TOCnum
]
>> >>
```

The next line of code will make the document open showing bookmarks and the PDF page fit to screen.

```
[ /PageMode /UseOutlines /Page 0 /View
[ /Fit] /DOCVIEW pdfmark
```

Now we want to include a watermark:

```
<< /BeginPage
{ pop
gsave true setglobal
30 150 moveto 90 rotate .8 setgray %you may
play around with these parameters
/ArialMT 20 selectfont (Draft Copy) show
false setglobal grestore
} >> setpagedevice
/PUT pdfmark
```

In the next step, we use &TOCnum to rotate all TOC pages to landscape orientation.

```
%do i=1 %to &TOCNum;
   [ { Page &i} << /Rotate 90 >> /
PUT pdfmark
%end;
```

The following steps use the page information in the TOC data table row by row. This can be easily achieved with the method described by Agator.[3] All variables in the TOC data table will be converted to macro variables and have the current value of the cell in the current row.

First, we include a bookmark for each row in the metadata table

```
[ /Page &sum /View [ /Fit] /Title
(&TOCtxt) /OUT pdfmark'
```

The following code rotates the actual page if it is landscape and adds the overall pagination by displaying a box with pagination text in the lower left corner of the page. The coordinates of the pagination entered in /Rect were found by trial and error.

```
%do mypages=%eval(&sum) %to %eval(&
sum+&pdfpage-1);
     [ { Page &mypages)} << /Rotate 90 >>
      /PUT pdfmark ';
     [ /Rect [ 568 19 584 526] /Rotate 90' ;
      /Contents (Overall Page %eval(&
      mypages-&TOCNum) of &LastPDF)
      /Color [ 1 1 1]
      /Border [ 0 0 0]
      /DS(font: Arial,sans-serif 8.0pt;
      text-align:left )
      /SrcPg &mypages
      /F 4
      /Subtype/FreeText
      /ANN pdfmark
```

A second box representing the link back to the TOC is added after that.

```
      [ /Rect [ 568 19 584 526] /Rotate 90
      /Color [ 0 0 0
      /Border [ 0 0 0]
      /SrcPg &mypages
      /Page &TOC
      View [ /Fit]
      /Subtype /Link
      /ANN pdfmark
%end;
```

The code changes slightly for portrait orientation, because coordinates are different and there is no need to rotate the box. So, the first two lines change to

```
[ { Page &mypages)} << /Rotate 0 >> /PUT
pdfmark
[ /Rect [ 19 12 277 27] /Rotate 90
```

The last step for each row in the TOC dataset is the output of the link to appear on the TOC PDF page.

We have the /Rect argument already available in the dataset, so the code changes to:

```
[ /&box. /Rotate 90
/Color [ 0 0 0]
```

```
/Border[ 0 0 0]
/SrcPg &toc.
/Page &sum
/View[ /Fit]
/Subtype /Link
/ANN pdfmark
```

Going the Extra Mile: Implementing Sublevels in Bookmarks

All steps up to now can easily be implemented in almost every environment as all information is gathered during post-production. But sublevel bookmarks are a different story mainly because you need to know where in the actual document a new subcategory begins.

Bookmarks with sublevels are introduced by defining the number of sublevel bookmarks in the parent bookmark. This is simply done by adding parameter /Count to the code defining the parent bookmark. Macro variable &cntsublinks represents the number of sublevel bookmarks. If the number of sublevel bookmarks is negative like in the example below, sublevel bookmarks are shown collapsed when the document is opened.

```
[ /Count -&cntsublinks /Page &sum /View
[ /Fit] /Title (&TOCtxt) /OUT pdfmark
```

Below the parent bookmark you now add all links you want to see as sub-level bookmarks:

```
[ /Page &_subpage /View[ /Fit] /Title
(&_linkname) /OUT pdfmark
```

The code of the child bookmark looks the same as for a generic bookmark. The real challenge is to calculate &cntsublinks, &_subpage and &_linkname. Since this information cannot be retrieved from the RTF output we need to go back to the dataset used to produce the original RTF output. Calculating the destination page of the child bookmark is rather easy if a page count variable (mypagebreakvar) is already available in your source dataset. You then simply need to extract the records from the dataset where a new sublevel begins using something like:

```
%*sort original data by sub-level vari-
able;
%sort(in=originaldata, out=sublinks,
by=sublevel);
data sublinks;
   set sublinks;
   by sublevel;
      if first.sublevel then do;
         _subpage=mypagebreakvar +
         &sum -1; _linkname=sublevel ;
         output;
      end;
run;
```

Dataset sublinks will have a record for each sub-bookmark. Variable _subpage is calculated by adding the page count in your original dataset (mypagebreakvar) to the destination page of the parent bookmark (&sum). Variable _linkname forms the title of the sublevel bookmark and is filled with the value of the by-variable. The actual values of _subpage and _linkname will be used in macro variables &_subpage and &_linkname in the sublevel bookmark. Finally, we calculate the number of records (&cntsublinks) to go into the /Count parameter in the parent bookmark definition:

```
%*retrieve number of sublinks to be cre
ated;
proc sql noprint;
      select count(*)
      into :cntsublinks from sublinks;
quit;
```

This example now consists of a parent bookmark with a number of child bookmarks. Introducing more sublevels with their own child bookmarks uses the same approach. You simply use the /Count parameter whenever you want to introduce a new parent bookmark with sub-levels.

Preparing the Batch.PS File

We now have all PS files needed to produce our final merged file. What is missing now is a batch file to tell Acrobat Distiller where to look up the single PS files. This step is rather simple. We just need a bit of postscript code to enable Acrobat Distiller to stack multiple files.

```
/prun{ /mysave save def
dup = flush
RunFile
clear cleardictstack
mysave restore
} def
```

After that, path references to each of the PS files are created such as the documents are expected to appear in the final PDF:

```
(S:/myPathToPostScriptFiles/
prologue.ps) prun
(S:/myPathToPostScriptFiles/TOC.ps)
prun
(S:/myPathToPostScriptFiles/file1.ps)
prun
(S:/myPathToPostScriptFiles/file2.ps)
prun
(S:/myPathToPostScriptFiles/file3.ps)
prun
                    …
```

This batch file will now be used as input for Acrobat Distiller.

Acrobat Distiller

There is a slight workaround needed if Acrobat Distiller version 9 or higher is used to produce a PDF from multiple PS files. For data safety reasons Acrobat opens in a mode that cannot access files outside its installation folder. This default can be changed by starting Acrobat Distiller via the win-

dows command line. You can also save the following code as. bat file and use it to start Acrobat Distiller:

```
cd 'path to my acrodist.exe\Adobe\
Acrobat 9.0\Acrobat'
start acrodist /F
exit
```

The secret bit is option /F which makes Acrobat Distiller open in the non-secure mode. Once the batch.ps file is copied into the In-folder of the watched folder area the merging process starts. Depending on the number and size of the underlying PS files it takes less than a minute to produce the merged file. You should now check the final PDF document whether all bookmarks are set correctly. When you open the PDF and click a bookmark the first time it takes some seconds until the document has been initialized and the requested action is executed. However, after that all links will work instantly after being clicked.

During testing the batch script we found that for some reason watermarks are not produced if file prologue.ps is referenced in the batch file only (all other features work). If you want to have watermarks in the final PDF you should tick option 'Use prologue.ps and epilogue.ps' in the preferences menu of Acrobat Distiller and move the prologue.ps file to the parent folder of the In/Out folders.

Alternative Approach

There are other software packages (e.g. Ghostscript) which can also be used to create a merged PDF file. Although not fully tested a general command line for Ghostscript looks as follows:

```
Gswin32c.exe^
-o c:/path/to/output.pdf^
[…more parameters (optional)…]^
/path/to/prologue.ps^
/path/to/TOC.ps^
/path/to/file1.ps^
/path/to/file2.ps^
/path/to/file3.ps
```

Conclusion

Setting up a merged PDF file from multiple RTF documents can be achieved without the need to change existing workflows or to buy, install and validate additional software. This paper combines several approaches already published to serve as a reference document for programmers who like to test or implement this method in their organisation.

Acknowledgement

The author would like to thank Matthew Travell for reviewing the manuscript.

References

1 Jansen L. Creating PDF™ Documents including Links, Bookmarks and a Table of Contents with the SAS® Software. PharmaSUG 2001: Proceedings of the Pharmaceutical SAS Users Group Conference; 2001 May 20–23; Boston, MS, USA.
2 Verrill D, Yeh E, Kasichainula S. Automating the creation of a single bookmarked PDF document from multiple SAS® ASCII and PostScript® output files. PharmaSUG 2003: Proceedings of the Pharmaceutical SAS Users Group Conference; 2003 May 4–7; Miami, FL, USA.
3 Agator P. Convert dataset variables into macro variables: a new approach with the CALL SET routine and the SCL FETCH function. Pharm Program. 2010;**3**;84–8.
4 VBScript reference. Available from: http://technet.microsoft.com/en-us/library/ee198844.aspx, [Accessed 4 November 2012].
5 Suodenjoki M. Word Doc to PDF Conversion. Available from: http://www.suodenjoki.dk/us/productions/articles/word2pdf.htm, [Accessed 4 November 2012].
6 Adobe pdfmark Reference Manual, Technical Note #5150. Available from: http://partners.adobe.com/public/developer/en/acrobat/sdk/pdf/pdf_creation_apis_and_specs/pdfmarkReference.pdf, [Accessed 4 November 2012].