A Comparison of Utilities for converting from PostScript or Portable Document Format to Text

Author: Nicholas Robinson
Email: Nicholas.Robinson@cern.ch

Date: 31-08-2001
CERN-OPEN-2001-065

Introduction

There are many available tools for converting PostScript and/or PDF files into text files. As part of a project involving the autonomous extraction and recognition of citation information from electronic full-text documents, it has been decided that before the actual extraction process can begin, it is first of all necessary to convert the document into some sort of plain text or HTML format. This requirement has resulted in some research into the available tools, followed by testing. It is the purpose of this report to briefly detail the results of this testing, and finally to recommend one of the tools for use above the others.

Conversion Tools Tested

Having carried out a search for the available conversion tools, I became aware of the existence of the following tools:

- pdftotext
- ps2ascii.ps
- pdftohtml
- pdftohtml(.bin)
- pstotext
- prescript

Testing was carried out upon each of these tools, and the details of this testing are recorded in the following sections of this report.

The 'pdftotext' Tool

Source: <http://www.foolabs.com/xpdf/>

The pdftotext tool is available from the above URL. It comes as part of a larger package called XPDF, which is a PDF viewing and manipulation package for use under X-Windows. The pdftotext tool is a command line tool that comes with the XPDF package. The tool pdftohtml, also discussed in this report is based upon the pdftotext tool.

Installing pdftotext

In order to obtain the pdftotext tool it is necessary to download the entire XPDF package. This package comes in the form of a zipped ".tar" file, which contains all of the source code for the XPDF applications. Under UNIX, it is fairly easy to build the package from the source with the “make” command.

Using pdftotext

The pdftotext tool is invoked from the command line, and provides the following usage information:

```
Usage: pdftotext [options] <PDF-file> [<text-file>]
  -f <int>         : first page to convert
  -l <int>         : last page to convert
  -ascii7          : convert to 7-bit ASCII (default is 8-bit ISO Latin-1)
  -raw             : keep strings in content stream order
  -q               : don't print any messages or errors
  -h               : print usage information
  -help            : print usage information
```

It can be seen from the above usage information, that there are 4 options with which the tool can be used. One can specify the range of pages to convert, and also the encoding of the output text (7-bit ASCII or 8-bit ISO Latin-1). It is also possible to run the tool in "quiet" mode using the "-q" flag, hence suppressing any error messages that would otherwise be displayed.

When the tool was tested for this report, it was invoked in the following way:

«pdftotext -raw -q [PDF Filename] -»

This kept strings in the content stream order, allowed any warnings to be suppressed, and also allowed the output of the tool to be printed to the STDOUT file, hence avoiding the intermediate step of making a text file.

When using this tool, I found it to operate fairly quickly. Of all of the PDF to text conversion tools tested by myself, this operated the fastest. For example, in order to convert a PDF file of approximately 182 KB, the tool took just 20 seconds. This is considerably faster than the other tools tested for this purpose.

On the whole, the quality of the plain text produced by the pdftotext tool was very good. There were rarely many words incorrectly broken with spaces. Unfortunately, words that were broken over two lines, using hyphenation, due to word-wrapping in the PDF were not de-hyphenated by this tool. However, this would not be such a problem to predict and correct in a script using the plain text output.

It was also unfortunate that the lines were split by '\n' characters at the places at which there was a wrapped line in the PDF. It would have been nice if lines were not broken due to wrapping as was the case with the "ps2ascii.ps" tool.

On the whole, the tool was very good at separating titles from their following body of text, and respecting any double spacing that existed within the PDF file. An example of this separation can be seen below in Figure 1, which is text extracted from a PDF file by pdftotext.

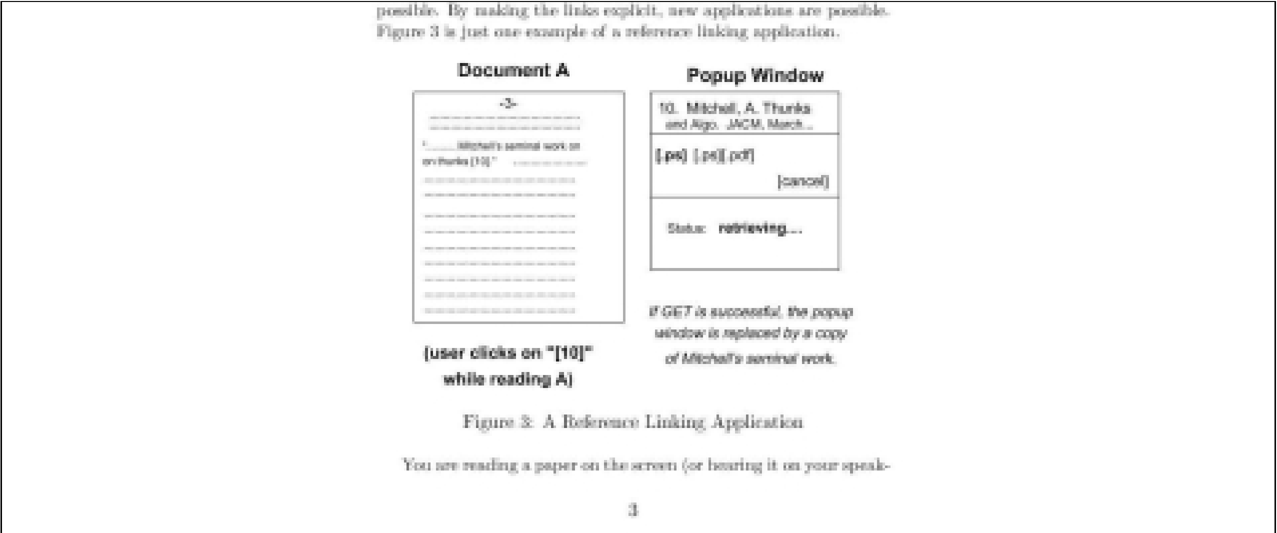
```
...iteration of the method the implementation improves a little. The work done so far
indicates that the architecture and design for the reference linking API are sound.

References
[1] Donna Bergmark. Automatic extraction of reference linking infor-
mation from online documents. Technical Report TR 2000-1821,
Cornell Computer Science Department, October 2000.
[2] Priscilla Caplan and William Arms.          Reference link-
ing for journal articles.         D-Lib Magazine:   The Maga-
zine of Digital Library Research, 5(7/8), July/August 1999.
<http://www.dlib.org/dlib/july99/caplan/07caplan.html>
...
```

Figure 1. Example of text extracted from a PDF file by pdftotext.

Notice from the above extract that the title of the references section is situated on its own line and is therefore easily recognisable as the title of this section.

This tool does not separate pages with any kind of marker, except a '\f' (form feed) character. It is easy to make a script recognise this and extract it, but if it is left in the text it will be invisible except to a printer, which will print the text that follows it on a new line.

On the downside of this tool, when it reaches an image that also has text, it simply discards the image, but transforms the text. This can lead to "garbage" text in the output. An example of this can be seen by looking at Figures 2 and 3, below. Figure 2 is a PDF document with an image in it. It has been taken from the paper “An Architecture for Reference Linking”, by Donna Bergmark, William Arms and Carl Lagoze (all of the *Cornell Digital Library Research Group*). Figure 3 is a small portion of text extracted from this PDF document by the pdftotext tool.



possible. By making the links explicit, new applications are possible. Figure 3 is just one example of a reference linking application.

Figure 3: A Reference Linking Application

You are reading a paper on the screen (or hearing it on your speak-

3

Figure 2. An example of a PDF document with images in it.

```
If GET is successful, the popup
window is replaced by a copy
of Mitchell's seminal work.
while reading A)
(user clicks on "[10]"
Document A
-3-

..........................................
..........................................
..........................................
..........................................
..........................................
..........................................
..........................................
..........................................
..........................................
..........................................
..........................................

"...........Mitchell's seminal work on
on thunks [10]."        ....................
Popup Window

Status:    retrieving....
[cancel]
[.ps] [.ps][.pdf]
10.  Mitchell, A. Thunks
and Algo.  JACM, March...
```

Figure 3. when the PDF document is converted into text, the text in the image is converted with it, hence corrupting the true text of the document.

It can be seen that the extracted text shown above should not have been extracted, as it is really part of the image in the PDF document. It should in fact simply be ignored, as when placed into the middle of the extracted text it is meaningless and only serves to confuse matters.

The ‘pdftotext’ Tool: Testing Conclusion

Although this tool has a few disadvantages such as the wrapping of lines with '\n' characters; extracting text from images; the inability to convert PostScript files to text; and leaving hyphenated words in the plain text output, I believe that the advantages it offers are very important. Primarily, the plain text output produced by the tool is of a high quality and the tool is very fast - by far the fastest tool that I have tested. It is also the only tool that seems to be fairly reliable when it comes to separating the titles from the bodies of text that follow them.

As far as the subject of reference extraction is concerned, I believe that this is the most reliable tool that I have tested to date. I believe that it would also be suitable for use if it were simply desirable to create a plain text version of a PDF file for general reading, as the plain text output of the tool is in a very readable format.

Verdict: Recommend for use.

The ‘ps2ascii.ps’ Tool

Source: <http://www.ghostscript.com/>

This tool is shipped with the Aladdin Ghostscript package from the above URL.

Installing ps2ascii.ps

This tool comes as part of the Ghostscript package, and as such, is installed when Ghostscript is installed. I did not install Ghostscript myself, but I imagine that it is a fairly easy process.

Using ps2ascii.ps

The tool can be used in different modes. These modes include displaying the text from the PostScript file, along with the coordinate information for the location of the text upon the page. The following is an extract of some output from this tool.

```
F 172 205 (CMBX12)
S 1402 2998 (1) 97
S 1692 2998 (Bac) 318
S 2005 2998 (kground) 700
F 133 137 (CMR10)
S 1402 2754 (This) 214
S 1652 2754 (rep) 152
S 1808 2754 (ort) 139
S 1986 2754 (co) 102
S 2084 2754 (v) 58
P
```

Figure 4. Sample output from the ps2ascii.ps tool.

It can be seen from the above extract, that there are several fields to each row of output. The first field signifies the type of data contained within the row. If it starts with an "F", it refers to a font change, the first integer refers to the height of the characters in 1/720", and the second integer refers to the minimum width of the characters in 1/720". The text within the parentheses is the name of the font that has been selected. If the field starts with an "S", it means that it is a string of text. In this case, the first integer refers to the X coordinate, and the second integer refers to the Y coordinate of the start of the string. Coordinates are measured in units of 1/720" from the origin, which is at the lower left of the PostScript page. The text within the parentheses in this case is the string text, and the integer after this string refers to the width of the string, which is measured in units of 1/720". It can be seen that there is also a row that consists of a single character "P". This row signifies a page break.

The tool can also be operated in the COMPLEX mode of operation. In this case, it also adds rows for colour changes and indicates the presence of an image.

If either of the two modes described above are used, it is necessary to write a program to interpret this data, and rebuild the information into meaningful text. There is however a third mode of operation to the tool, which is the SIMPLE mode. This mode simply extracts the text from the PostScript, and outputs it in a plain text form. It is this form that I shall now discuss.

The ps2ascii.ps tool is invoked in SIMPLE mode as follows:

```
«gs -q -dNODISPLAY -dNOBIND -dWRITESYSTEMDICT -dSIMPL-c save -f ps2ascii.ps
[filename] -c quit»
```

This will cause the plain text output to be streamed to STDOUT. The tool can convert both PostScript and PDF documents to plain text format.

Unfortunately, while testing the tool, I found it to be fairly slow. For example, a PDF document of approximately 182 KB, took 1 minute and 40 seconds. This is quite a slow conversion for the file, certainly slower than certain other converters were for the same task. Although some of this lack of speed can be attributed to the poor performance of the machine on which I have conducted the tests, this tool still operated more slowly than some of the other tools tested.

The plain text output of the ps2ascii.ps tool is of a fairly high quality. Lines are rather long. In fact, the lines produced by this tool are considerably longer than those produced by any other tool that I have tested. This seems to be due to the attempts made by the tool to guess when a line has actually been broken by an intentional new line, as opposed to when it has simply been wrapped due to long length. This line break guessing can be both a positive factor and a negative factor. On the positive side, it is better if the lines do not have "\n" characters in them when it is really just a case of the line being wrapped due to PostScript page size, and this tool helps to prevent this. On the negative side however, the attempts made by the tool to guess where lines are truly supposed to be broken can be erroneous. It can break lines incorrectly, and in particular, it seems to have difficulties separating a section title from the line that follows it.

Take for example, the following title followed by some example text:

> 6 Architecture Recap
>
> Up to this point, we have discussed the API: its methods and outputs. However, recall that the application shown in Figure 3 requires solving two very difficult problems: analysis, to find the li......

Figure 5. Sample text – a section title followed by the section itself.

It can clearly be seen from the above text that the title line reads "6 Architecture Recap", and is located on its own line, with a blank line separating it from the passage of text that follows it. However, when the ps2ascii.ps tool translated it to plain text, the following was produced:

```
6 Architecture Recap Up to this point, we have discussed the API: its methods
and outputs. However, recall that the application shown in Figure 3 requires
solving two very difficult problems: analysis, to find the li......
```

Figure 6. The plain text produced by ps2ascii for the text shown in figure 5.

In the above example, it can clearly be seen that the line separating the title and the body of text has been completely lost, and the title line is now part of the body of text. The tool seems rather unpredictable in its behaviour of splitting titles from the text that follows them. Sometimes, it has no trouble in recognising that the title is not part of the body of text, and separates the two entities in the same way that they are separated in the PostScript/PDF. Other times however, it is unable to distinguish between the two.

Unfortunately this problem poses a big argument against using this tool in the extraction of references. This is because the reference extraction tool must search for a title to the reference line (a title generally being a short line of text on its own line, e.g. "The References Section"). If the title is placed within the body of the references section itself, it will look to the parser as though it is simply part of the text, and therefore won't pick up on the fact that it is the title to the references section.

The ‘ps2ascii.ps’ Tool: Testing Conclusion

The ps2ascii.ps program has some good points as well as bad points. On its good side, it provides long lines of text, mostly only inserting new line characters when they are encountered within the PostScript/PDF. This is a very desirable feature, as it eliminates all need to rebuild lines at a later stage. The tool is also able to convert both PostScript and PDF documents. The readable quality of the text produced by ps2ascii.ps is also very good. If the user simply wanted to create text versions of PostScript documents, then this tool would be a good choice.

On the down side of the tool however, it is fairly slow, and it has problems differentiating between titles and bodies of text. Unfortunately, this is a big enough disadvantage to make the tool unsuitable for the process of reference extraction, as it is necessary to be able to identify the reference section by title, and be sure that we are not simply identifying the word "reference" within the text.

Another disadvantage of this tool is that it is unable to reliably convert PostScript files created from Microsoft products such as Word.

Verdict: Good, but unsuitable for the reference extraction process.

The ‘pdftohtml’ Tool

Source: <http://www.ra.informatik.uni-stuttgart.de/~gosho/pdftohtml/>

The pdftohtml tool is based on Derek B. Noonburg's XPDF package. According to the authors, the goal of the pdftohtml project is to create a freely available program that enables a PDF file to be converted into an HTML file.

Fuller documentation details are available from the above URL, but in a nutshell, the authors claim that the tool attempts to preserve all links within the PDF; extract the text from the PDF document and place it within an HTML file; attempt to recognise bold and italic areas of text; and display the content of the pages in HTML from first page to last.

The above-mentioned Web site for the pdftohtml tool is quite useful, as it provides much information about the tool, including a list of problems known with the tool.

Unfortunately, the pdftohtml tool is not capable of transforming a PostScript file into HTML. It only works with PDF files.

There are two main distributions of pdftohtml. These are pdftohtml version 0.22, which is the current stable version of the tool and pdftohtml version 0.31, which is the latest, test release. I have tried both, and found them to be fairly similar in their results. In fact, I found the results of pdftohtml version 0.22 to be more suitable for my purposes, so I have decided to discuss this particular version in this report. It is sufficient to say that there is not a great enough difference between the two tools in order to justify a separate discussion of them.

Installing pdftohtml

The pdftohtml tool is downloaded as a zipped ".tar" file. After unzipping and extracting the tar archive, the binaries of the pdftohtml tool can be created by moving to the directory created by extracting the ".tar" file, and invoking the make command. I had no problems with this installation.

Using pdftohtml

The pdftohtml tool has the following usage information:

```
pdftohtml version 0.22
Usage: pdftohtml [options] <PDF-file> [<html-file>]
  -f <int>          : first page to convert
  -l <int>          : last page to convert
  -q                : don't print any messages or errors
  -h                : print usage information
  -help             : print usage information
  -p                : exchange .pdf links by .html
  -c                : generate complex document
  -i                : ignore images
  -noframes         : generate no frames
  -stdout           : use standard output
  -ext <string>     : set extension for images (in the Html-file) (default png)
```

It can be seen that there are many options for the tool, such as telling it to convert certain pages within a range etc. I only made use of certain options when testing the tool. I made use of the "-stdout" option, which allowed me to have the output of the tool streamed directly to the STDOUT stream. I also made use of the "-noframes" option, which allowed me to prevent the tool from creating HTML in which frames would be used. This is because for my purposes, I simply wanted all of the text to appear as one page of HTML, and frames would not allow this. I also used the "-i" flag, which allowed the tool to ignore all images within the document. This was because for my purposes, images were not desired. The "-q" flag was also used, which enabled the suppression of error messages.

The pdftohtml tool consists of 2 main tools. The first is "pdftohtml", which is a shell script that executes "pdftohtml.bin" and then converts all PBM/PPM images into PNG images format using pnmtopng converter. The other tool is the "pdftohtml.bin" tool, which is the actual tool that performs the conversion itself. Since I was not interested in having any images converted, I decided to perform my tests by directly calling the pdftohtml tool.

The tool was invoked as follows:

«**`pdftohtml.bin -i -noframes -stdout [filename]`**»

When using the tool, I found it to be fairly quick. One advantage of using PDF files is that they are frequently smaller in size than their PostScript counterparts. The tool took approximately 40 seconds to convert and stream to STDOUT a file of approximately 182 KB in size. This is fairly fast compared with some of the PostScript converters, which took well over 1 minute to convert the same file from the PostScript format.

On the whole, the quality of the HTML source produced by the pdftohtml tool was quite bad. It seemed to have one word per line and sometimes a tag such as a "<BR>" tag also. This is a shame - it would have been nicer for parsing and for readability if the lines had been broken only where there were HTML "<BR>" tags or "<P>" tags, or for that matter, where there were any other line break tags. This problem is not too difficult to neaten up with a simple script, which removes all of the unwanted new line characters. However, this is yet another step, causing more time overheads.

A sample of some HTML output for the program is shown below:

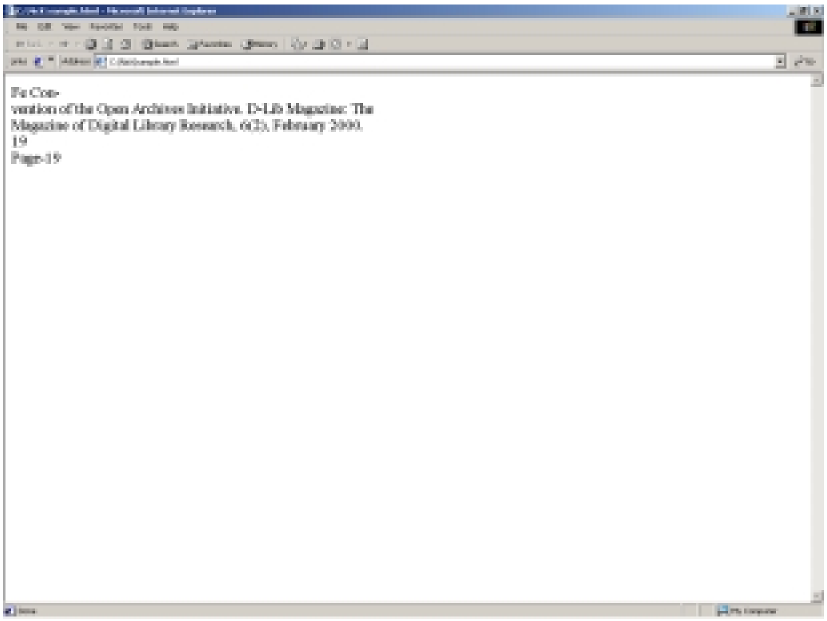
```
Fe
Con-<br>
vention
of
the
Open
Archives
Initiative.
D-Lib
Magazine:
```

```
The<br>
Magazine
of
Digital
Library
Research,
6(2),
February
2000.<br>
19<br>
Page-19
</body>
</html>
```

Figure 7. Sample HTML output from the pdftohtml tool.

Notice that there is more or less one word per line. This is the philosophy of the program - the user can clean up the HTML source as they see fit.

Of course, if the HTML is simply to be displayed in a browser, then this source does not matter, as the browser displays it nicely without all of these breaks in the lines. The image in 'figure 8', below, shows the way in which the article whose source was shown above appears within a browser.



```
Fu Con-
vention of the Open Archives Initiative. D-Lib Magazine: The
Magazine of Digital Library Research, 6(2), February 2000.
19
Page-19
```

Figure 8. Browser representation of the bad quality HTML source produced by pdftohtml – it looks good.

As I have discovered with some of the other converters, when an image that contains text is encountered, the image is discarded, but the text is shown in the output created by pdftohtml. This is a shame, but I believe that it must be unavoidable, as many of the tools do this.

A positive point about this tool is that it marks the pages with information. At the top of each page, it places an anchor whereby the name given is the page number. It also shows the page number at the end of each page. This page numbering and naming makes the recognition of page breaks very easy, and is a nice feature.

During the testing of this program, I did not find many PDF files that the program was unable to convert. The only such files were PDF files of a particularly bad quality, which I suspect were documents that may possibly have contained unknown fonts, so improvisation had to be carried out by the tool that created the PDF. The tool had absolutely no problems converting PDF files that were created from Microsoft Word documents, and the HTML source for PDF files converted from such documents was exactly the same as those converted from non-Microsoft documents.

Although the tool does place bold tags and italic tags into the HTML source, this is not 100% reliable, as the tagging information is a little erratic. Take for example the following extract:

```
<i>Klim</i><i>a-</i>
<i>u</i><i>n</i><i>d</i>
<i>K</i><i>ält</i><i>eanlagen</i><br>
<i>du</i><i>rc</i><i>h</i>
<i>E</i><i>ins</i><i>a</i><i>tz</i>
<i>v</i><i>o</i><i>n</i>
<i>e</i><i>lek</i><i>t</i><i>ronis</i><i>c</i><i>h</i>
<i>geregelt</i><i>e</i><i>r</i>
<i>P</i><i>um</i><i>pen.</i>
<i>KI</i>
<i>Luf</i><i>t-</i>
<i>und</i>
<i>Kältet</i><i>ec</i><i>hni</i><i>k</i><i>,</i>
<i>page</i>
<i>20-</i><br>
<i>23</i><i>,</i>
<i>J</i><i>anuary</i>
```

Figure 9. HTML source produced by pdftohtml. Note the pointless repetition of HTML tagging information.

Notice from the above extract, that the tags are closed and re-opened pointlessly for parts of a word, where it should really be 1 set of tags to enclose the entire word. In fact, often, it should really be 1 set of tags to enclose the entire section of italicised text. Of course this does not make a difference to the browser, but if we are interested in the source, then there is a lot of cleaning to be done upon it.

The 'pdftohtml' Tool: Testing Conclusion

It is a shame that the HTML source produced by this tool is so broken up. It would have been preferable if it was only broken where the relevant HTML break tags were. However, despite this fact, the HTML output is displayed very well in a browser, and so if this is the goal, then this tool would be perfect. It even has the capability to include pictures, make frames, etc.

Unfortunately, although the tool does place bold and italicising mark-up tags into the HTML source, it is not reliable enough for my purposes. I had wanted the tool to markup all headers preferably with "<H>" tags, so that I could recognise section headers etc. However, the tool does not use "<H>" tags, and it sometimes would mark up header, but other times it would not. There is nothing to be gained from the checking for marked-up headers in the source produced, as it uses “<B>” tags to mark headers up, but also marks up normal text with these bold tags, hence destroying any way of uniquely identifying the headers by mark-up.

I would say that as far as PDF-to-HTML converters go, this tool is very good for creating HTML that is intended for display in a browser, and I would strongly recommend the tool if this is the desired operation. However, it must be realised that if the source is wanted for any further examination, it will be necessary to do considerable cleaning on it before it is ready.

Verdict: Recommend for PDF-to-HTML conversions to be viewed in a browser.
Not reliable if mark-up of various sections is required.

The ‘pstotext’ Tool

Source: <http://research.compaq.com/SRC/virtualpaper/pstotext.html>

The pstotext tool is available freely from the above URL. It was written by Andrew Birrell, as a spin off from a project known as "Virtual Paper".

The pstotext tool is primarily a PostScript to text converter, but can also be used for converting PDF documents to text (although the documentation for the tool claims that this is slightly less reliable).

The pstotext tool requires version 3.33 or later of Ghostscript in order to work.

Installing pstotext

The pstotext tool is downloaded from the above URL as a zipped ".tar" file. Its installation is really very simple, it only being necessary to build the tool with the make command.

Using pstotext

The pstotext tool has the following usage information:

```
Usage: pstotext [option|file]...
Options:
  -cork            assume Cork encoding for dvips output
  -landscape       rotate 270 degrees
  -landscapeOther  rotate 90 degrees
  -portrait        don't rotate (default)
  -bboxes          output one word per line with bounding box
  -debug           show Ghostscript output and error messages
  -gs "command"    Ghostscript command
  -                read from stdin (default if no files specified)
  -output file     output results to "file" (default is stdout)
```

Essentially, I used the tool with none of the options in the form:

«**pstotext [filename]**»

This streamed the output to STDOUT, which I found to be a good feature, as it allows the output to be directly captured and manipulated instead of having to be written to a file, which would take up extra time.

I found that the pstotext tool worked fairly quickly. For a PostScript file of around 322 KB, it took approximately 50 seconds to make the conversion to text, including writing it to a file.

On the whole, the quality of the output produced by pstotext was very good. There were rarely many words incorrectly broken with spaces. Lines took the same lengths as they did in the original file (i.e. they are wrapped in the same places as they were in the PostScript file). This wrapping however was done by inserting a new line character into the line, thus breaking it into separate lines. This is perhaps a little unfortunate, as it would be nice in the interests of parsing the results if the tool were to only insert new line characters where there was really supposed to be a newline (not where lines were wrapped for formatting purposes), as it would eliminate the need to rebuild lines at a later stage.

One problem that I did encounter in the output was with the word "different". In the output text from a document, it kept outputting "di#erent" where the word different was supposed to be. I believe that this must be something to do with 2 'f' characters being next to each other - perhaps some sort of character encoding problem. This could cause big problems if the text was to be searched for certain key words, as strange characters in the middle of words in the text could prevent their matching.

Another drawback to the tools output (albeit a small one) was that when a word was split and hyphenated (due to word wrapping in the PostScript), this tool made no effort to remove the hyphenation and place the word back together as one word. This is unfortunate, as it would have helped to improve the quality of the output.

The pstotext tool does not insert any kind of page break information such as a line of hyphens as in the output of the prescript tool. All it does is print the page number (if the document had one), along with a '\f' character, which is a form-feed character, and serves as a page break. This is perhaps unfortunate, as it would aid the clarity of the output to have a line of hyphens or other such characters instead of a '\f' character.

The following extract from the a converted PostScript document shows the quality of text produced by the pstotext tool for PostScript documents:

```
[9] Sandra Payette and Carl Lagoze. Value-added surrogates for dis-
tributed content. D-Lib Magazine: The Magazine of Digital Li-
brary Research, 6(6), June 2000.
18


[10] Andy Quick. Java HTML tidy.
<http://www3.sympatico.ca/ac.quick/jtidy.html>
[11] K. G. Saur. Functional requirements for bibliographic records,
1998. UBCIM Publications - New Series Vol. 19.
[12] Karen Sollins and Larry Masinter. Functional require-
ments for uniform resource names, December 1994.
http://www.ietf.org/rfc/rfc1737.txt.
[13] Elaine Svenonius. The Intellectual Foundation of Information
Organization. M.I.T. Press, 2000.
[14] Herbert Van de Sompel and Carl Lagoze. The Santa Fe Con-
vention of the Open Archives Initiative. D-Lib Magazine: The
Magazine of Digital Library Research, 6(2), February 2000.
19
```

Figure 10. Text produced by the 'pstotext' tool. The quality is fairly good.

Notice in the above figure, that the page number (18) that comes after the 9th reference. This could easily be mistaken for another part of that reference during parsing.

When the tool was tried using PDF files as input, the results of the conversion were fair, but certainly a lack of quality was visible. In fact, in some PDF conversions, the document was cut short. This suggests to me that the pstotext tool possibly has some difficulties in reading the internal references to the objects that make up the PDF file. This does not mean to say that the tool was useless for converting PDF documents to text, just certainly not perfect.

The following extract shows part of a conversion of a PDF document:

```
[37] ATLAS homepage http://atlasinfo.cern.ch:80/Atlas/Welcome.html
[38] ATLAS Trigger/DAQ Prototype-1 homepage http://atddoc.cern.ch/Atlas/
[39] Applications of Corba in the Atlas prototype DAQ, S. Kolos, R. Jones. L.Mapelli, Y. Ryabov,
11th
IEEE NPSS Real Time Conference Proceedings, 1999, pp 469-474
[40] Textor-94 experiment homepage is http://www.fz-juelich.de/ipp
[41] Objectivity / Corba distributed database performance on gigabit SUN-Ultra-10 cluster,
L.Gommans and others, 11th IEEE NPSS Real Time Conference Proceedings, 1999, 442-445
[42] Overview of PHENIX Online System, C.Witzig, 10th IEEE Real Time Conference
Proceedings,
1998, pp 541-543
[43] Use of CORBA in the PHENIX Distributed Online Computing System, E.Desmond and
others,
11th IEEE NPSS Real Time Conference Proceedings, 1999, pp 487-491
[44] BaBar homepage http://www.slac.stanford.edu/BFROOT/
[45] Ambient and Configuration Databases for the BaBar Online System, G. Zioulas and others,
11th
IEEE NPSS Real Time Conference Proceedings, 1999, pp 548-550
```

Figure 11. Text produced by pstotext (converted from a PDF document).

As can be seen from this extract, the quality is fairly high. It is a shame that this can't be guaranteed this every time.

Unfortunately, during testing, I discovered that the pstotext tool was very unreliable when attempting to convert a PostScript file that was created from a Microsoft document. I attempted to convert several PostScript files that had been created from Microsoft Word and PowerPoint files by the CERN Conversion Service, and obtained "garbage" output similar to the following:

```
-



--
--
--
--
--
--
-
--
-



--





.
.
.
-
```

Figure 12. Garbage output obtained when an attempt is made by pstotext to convert a PostScript produced from a Microsoft document.

It can be seen from the above extract, that the output for this Microsoft created PostScript that has been converted to text is completely useless. What is worse is that the tool does not appear to output any sort of error messages saying that it cannot properly understand the file that has been passed to it as input.

The ‘pstotext’ Tool: Testing Conclusion

I have found that the pstotext tool is capable of producing nice output that is fairly easy to parse. There have been a few downfalls with this output, such as the lack of page break information etc, but this is not too crucial, as it still outputs the form-feed character.

However, on the downside of the tool, although often very nice output was obtained from a PDF file conversion, sometimes part of the file would be lost. This makes it partly unreliable for PDF conversions.

The biggest let down of all for the pstotext tool is that it seems to be very unreliable at converting Microsoft PostScript documents to text. This, in my opinion, makes the tool unsuitable for use in a production environment where we cannot determine the sources and creators of the PostScript/PDF files that we want to convert. In fact, it is quite likely that many of the files that we would expect to convert would have been created from Microsoft Word documents, so clearly, this tool is unsuitable.

Verdict: Unsuitable.

The ‘Prescript’ Tool

Source: <http://www.nzdl.org/html/prescript.html>

Prescript is available freely from the above URL. It was written by people at the New Zealand digital Library organisation as a translator to change PostScript documents into Text. It also however offers support for a simple HTML output of the document. Unfortunately, because prescript is a PostScript to text translator, it does not offer PDF to text translation capabilities.

When prescript is used to produce HTML, it has the capability only to introduce certain HTML tags. These are the "<P>", "<BR>", "<HR>" and "<I>...</I>" tags. It also of course inserts the "<HTML>", "<HEAD>", etc tags into the document. According to the NZDL sit, prescript also attempts to support paragraph boundaries detection by using the line spacing and indentation in the document in order to determine paragraph boundaries.

According to the NZDL site, prescript also attempts to de-hyphenate words that have been hyphenated by the PostScript, which could be a fairly useful feature. It also attempts some ligature translation for $\mathrm{T_EX}$ documents.

Prescript requires version 4.01 or higher of the GhostScript utility in order to work. It is also written in the Python language, and so requires the Python interpreter.

There are 2 main versions of prescript available. These are "PreScript 0.1", which is the stable version of the program, and is recommended by the authors as the version that should be used for any serious work that is to be undertaken. There is also however, "PreScript 2.2", which is the latest version of the tool. The authors claim that this version of the tool is a lot faster, and generally better, including better prediction of line, page and paragraph breaks.

Installing prescript

Having downloaded the prescript tool, which came as a "tar" package, a makefile was used in order to install it. However, many things needed to be done manually, such as making the various directories that it needed, as it could not successfully do this during the attempted builds. It was also necessary for me to change the pointer to the python interpreter, etc. It was a little awkward, but no major problems were encountered.

Using prescript:

The tool could be invoked as follows:

```
«prescript <plain|html|arff> <input> [output]»
```

It was necessary to specify for the tool, which format the output should take, the name of the input file (the PostScript file to be converted), and the name of the output file, to which the output was to be written. Unfortunately, it is not possible to tell the utility to simply write its output to the STDOUT stream - it appears to need to actually write it to a file. This is a definite downside to the tool, as for my purposes, I simply want to call the tool from within another program, feeding its output directly to STDOUT, and retrieving it for use by my own program. An intermediate stage of writing a file would be a performance drawback.

Having learned how to use the tool, I tried it with several PostScript files, testing both the HTML output and the plain-text ASCII output. I found that on the whole, the tool gave some very clean and encouraging results.

First of all, the HTML results shall be discussed. I shall also include some short extracts from the converted output of a file so that they can be appreciated within this document.

When viewed within a browser, the HTML results are very nice. Although the text is in fairly short lines, it is well broken up into paragraphs. It is very clear to read in this manner. One problem that was encountered however, was that when there is an image in the PostScript document and this image contains words, the words are unfortunately translated and placed within the output text. Often, this can be nonsense because they mean nothing without the rest of the image, and it would in my opinion have been better to simply leave them out of the translator. Unfortunately, this is a problem common to all of the conversion tools tested for the purposes of this report. It is my feeling that the tools cannot distinguish between text written on top of an image, and text written on the rest of the PS/PDF canvas. Presumably, the image is represented as binary information, but its text remains as a series of characters written on the canvas with the usual operators.

As my goal is the extraction of reference information from the reference section however, it should usually cause no problems. However, because the references section can sometimes contain images and figures, any text from these could pollute the references.

When during the translation process the tool discovers a new page in the PostScript document, it is marked in the HTML output with a "<HR>" tag. This could be very useful, as it would allow any parsing tool to easily and unambiguously identify any new pages in the output. The tool also marks each page with the page number just before the "<HR>" tag.

A downside to the HTML produced is that the prescript tool does not make any effort to mark-up title sections with "<H>" tags or "<B>" tags. This is certainly unfortunate, as it would be very useful for a parser attempting to extract references to have title sections marked up. The authors information about the tool does say that it uses "<I>" tags, but only for marking up header and footer sections. Very disappointing.

Regarding the source of the HTML itself, I can only say that it is very good. With other PostScript to text conversion tools that I have seen, the quality of the output is often messy, with hyphened words frequently occurring, and with words not properly recognised and ending up with spaces in the middle of a word, hence making further parsing difficulties for any tools that use the output to attempt to recognise words and information. With the HTML source produced by the prescript tool however, this is not the case. I did not once see a word that had been erroneously split.

Lines were broken at the end of the line as it appears in the PostScript document. The only reason that these lines were broken at these points in the PostScript document is that the formatting of the text in the PostScript requires lines to be wrapped in order that they fit the page. In our output HTML however, it would be preferable if the lines did not keep this format of line breaks unless there is really supposed to be one. However, with the HTML output, it was not a large problem because with the HTML markup, it would be easy to replace all carriage returns in the text with spaces, unless there was a "<BR>" or "<P>" tag present, in which case the carriage return in the text would be justified.

The following extract shows some HTML source created by the prescript program for a document.

```
<p>[1] Donna Bergmark. Automatic extraction of reference linking information
from online documents. Technical Report TR 2000-1821,
Cornell Computer Science Department, October 2000.
<p>[2] Priscilla  Caplan  and  William  Arms.    Reference  linking
for  journal  articles.    D-Lib Magazine:   The Magazine
of Digital Library Research, 5(7/8), July/August 1999.
&lt;http://www.dlib.org/dlib/july99/caplan/07caplan.html&gt;
<p>[3] James Davis and Carl Lagoze. NCSTRL: design and deployment
of a globally distributed digital library. IEEE Computer, February
1999.
<p>[4] Steve Hitchcock, Les Carr, Wendy Hall, Stephen Harris, S. Probets,
D. Evans, and D. Brailsford. Linking electronic journals:
Lessons from the Open Journal project. D-Lib Magazine: The
```

```
Magazine of Digital Library Research, December 1998.
<p>[5] C. Lagoze and J. Davis. Dienst: An architecture for distributed
document libraries. Communications of the ACM, 38(4):47, April
1995.
<p>[6] Steve Lawrence, C. Lee Giles, and Kurt Bollacker.  Digital libraries
and autonomous citation indexing.  IEEE Computer,
32(6):67{71, 1999. &lt;http://www.researchindex.com&gt;
<p>[7] Norman Paskin. E-citations: actionable identifiers and scholarly
referencing, 1999. &lt;http://www.doi.org/citations.pdf&gt;
<p>[8] S. Payette and C. Lagoze.  Flexible and extensible digital object
and repository architecture (FEDORA). In Second European
Conference on Research and Advanced Technology for Digital Libraries,
Heraklion, Crete, 1998.
<p>[9] Sandra Payette and Carl Lagoze. Value-added surrogates for distributed
content. D-Lib Magazine: The Magazine of Digital Library
Research, 6(6), June 2000.
<p><!--Page No--><p><b><center>18</center></b><p>



<!--End Of Page--><p><hr><p>

<p>[10] Andy    Quick.        Java    HTML     tidy.
&lt;http://www3.sympatico.ca/ac.quick/jtidy.html&gt;
<p>[11] K. G. Saur. Functional requirements for bibliographic records,
1998. UBCIM Publications - New Series Vol. 19.
<p>[12] Karen  Sollins and Larry Masinter.   Functional  requirements
for  uniform  resource  names,   December  1994.
http://www.ietf.org/rfc/rfc1737.txt.
<p>[13] Elaine Svenonius. The Intellectual Foundation of Information
Organization. M.I.T. Press, 2000.
<p>[14] Herbert Van de Sompel and Carl Lagoze. The Santa Fe Convention
of the Open Archives Initiative. D-Lib Magazine: The
Magazine of Digital Library Research, 6(2), February 2000.
<p><!--Page No--><p><b><center>19</center></b><p>
```

Figure 13. A sample of the HTML source created by the prescript tool. It is of a very high quality.

Notice from the above HTML source, that although each reference line is split into several lines of text (wrapped), each reference is separated from the previous by a "<P>" tag. This would make it very easy for a parser to rebuild the complete reference line, and indeed to separate several reference lines from each other. Notice also, the way that the start of a page is marked with the page number (and there is also a comment to let us know that this is the page number:

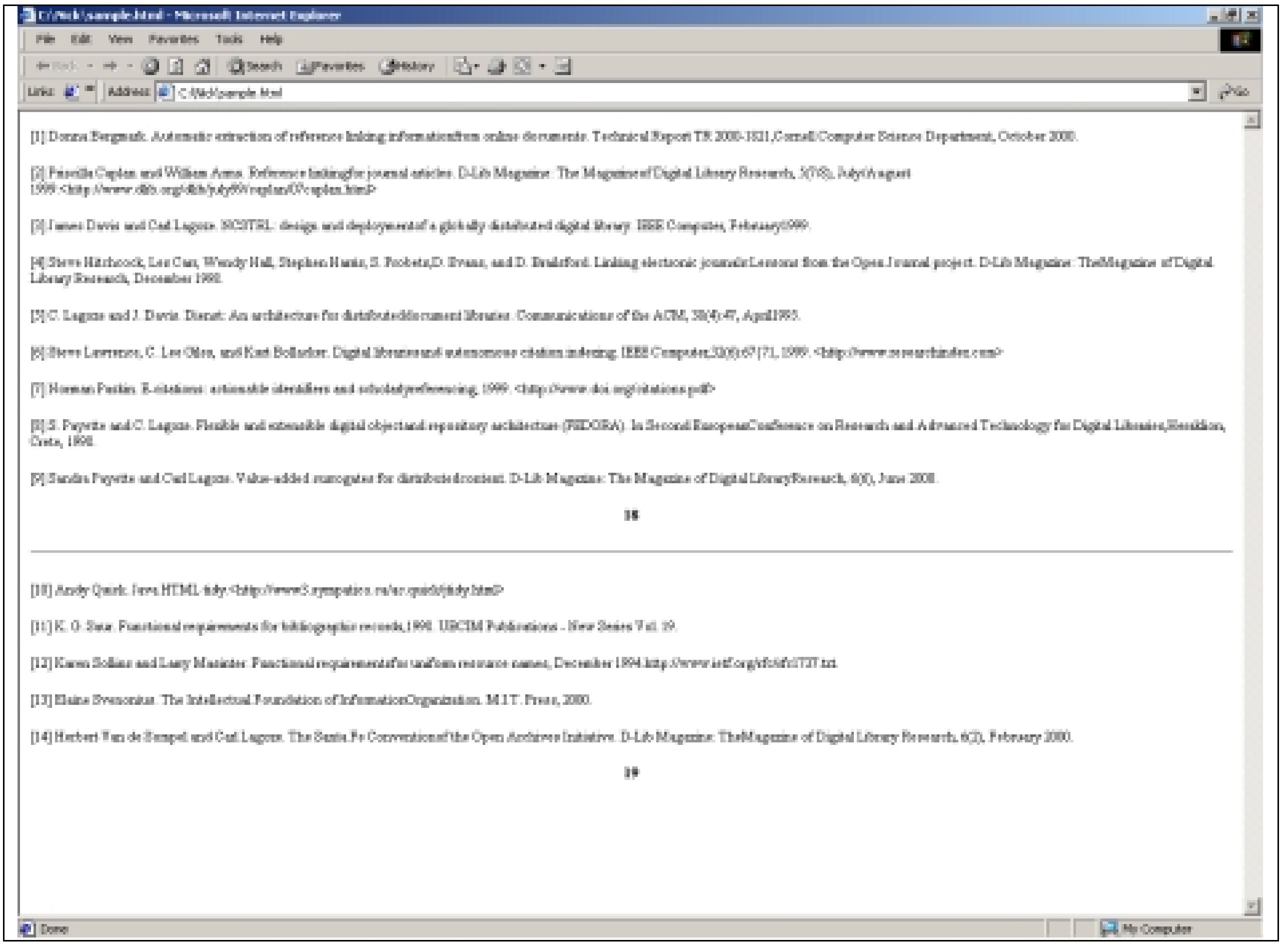
"<p><!--Page No--><p><b><center>19</center></b><p>"

This would make it very easy for a parser to recognise that the page has reached its end, and therefore to recognise the pattern of new lines etc that come with the end of the page, and thus remove them appropriately.

There is also the following end of page comment after the page number has been displayed:

"<!--End Of Page--><p><hr><p>"

Over all, the HTML source produced by prescript is of a high quality. The figure below shows a screen shot of its appearance in a browser:



[1] Donna Bergmark. Automatic extraction of reference linking informationfrom online documents. Technical Report TR 2000-1821,Cornell Computer Science Department, October 2000.

[2] Priscilla Caplan and William Arms. Reference linkingfor journal articles. D-Lib Magazine: The Magazineof Digital Library Research, 5(7/8), July/August 1999 <http://www.dlib.org/dlib/july99/caplan/07caplan.html>

[3] James Davis and Carl Lagoze. NCSTRL: design and deploymentof a globally distributed digital library. IEEE Computer, February1999.

[4] Steve Hitchcock, Les Carr, Wendy Hall, Stephen Harris, S. Probets,D. Evans, and D. Brailsford. Linking electronic journals:Lessons from the Open Journal project. D-Lib Magazine: TheMagazine of Digital Library Research, December 1998.

[5] C. Lagoze and J. Davis. Dienst: An architecture for distributeddocument libraries. Communications of the ACM, 38(4):47, April1995.

[6] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital librariesand autonomous citation indexing. IEEE Computer,32(6):67{71, 1999. <http://www.researchindex.com>

[7] Norman Paskin. E-citations: actionable identifiers and scholarlyreferencing, 1999. <http://www.doi.org/citations.pdf>

[8] S. Payette and C. Lagoze. Flexible and extensible digital objectand repository architecture (FEDORA). In Second EuropeanConference on Research and Advanced Technology for Digital Libraries,Heraklion, Crete, 1998.

[9] Sandra Payette and Carl Lagoze. Value-added surrogates for distributedcontent. D-Lib Magazine: The Magazine of Digital LibraryResearch, 6(6), June 2000.

18

[10] Andy Quick. Java HTML tidy.<http://www3.sympatico.ca/ac.quick/jtidy.html>

[11] K. G. Saur. Functional requirements for bibliographic records,1998. UBCIM Publications - New Series Vol. 19.

[12] Karen Sollins and Larry Masinter. Functional requirementsfor uniform resource names, December 1994.http://www.ietf.org/rfc/rfc1737.txt

[13] Elaine Svenonius. The Intellectual Foundation of InformationOrganization. M.I.T. Press, 2000.

[14] Herbert Van de Sompel and Carl Lagoze. The Santa Fe Conventionof the Open Archives Initiative. D-Lib Magazine: TheMagazine of Digital Library Research, 6(2), February 2000.

19

Figure 14. HTML produced by the prescript tool as it appears in a browser.

Text Output

Now that the HTML created by prescript from the PostScript document has been discussed, it is necessary to discuss the text output of the tool. The discussion of the text output of the prescript tool provided here will be fairly short, because the text output is very similar to the HTML output. Essentially, it is the same as the HTML output, but without any markup tags.

The quality of the text output of the tool is essentially very good. Words are formed well, and are not split erroneously with spaces etc in the middle of them, as is the case with some other tools. Care is taken by the tool to ensure that lines are broken in the same places as they are in the postscript. This could possibly lead to difficulties in correctly re-creating lines, as '\n' characters are inserted at the end of each line, which means that a given sentence can be broken up by several '\n' characters.

When the prescript tool encounters a new page, it displays the page number, along with a line of '-' characters. This is very useful, as it allows us to clearly identify a given page. An example of this is shown below.

```
[9] Sandra Payette and Carl Lagoze. Value-added surrogates for distributed
content. D-Lib Magazine: The Magazine of Digital Library
Research, 6(6), June 2000.

18

------------------------------------------------------------------------------


[10] Andy    Quick.      Java   HTML    tidy.
<http://www3.sympatico.ca/ac.quick/jtidy.html>

[11] K. G. Saur. Functional requirements for bibliographic records,
1998. UBCIM Publications - New Series Vol. 19.

[12] Karen  Sollins and Larry Masinter.   Functional  requirements
for  uniform  resource  names,   December  1994.
http://www.ietf.org/rfc/rfc1737.txt.

[13] Elaine Svenonius. The Intellectual Foundation of Information
Organization. M.I.T. Press, 2000.

[14] Herbert Van de Sompel and Carl Lagoze. The Santa Fe Convention
of the Open Archives Initiative. D-Lib Magazine: The
Magazine of Digital Library Research, 6(2), February 2000.

19

------------------------------------------------------------------------------
```

Figure 15. Text produced by the prescript tool.

Notice from the above excerpt that it is very easy to tell where a page begins and where it ends.

The textual output of the prescript tool still included any text that was part of an image, even though the actual image was removed. This is the same situation as with the HTML output.

The Suitability of The ‘prescript’ Tool

When using this tool, I found that it produced results fairly slowly. For a PostScript document of approximately 322 KB, it took 1 minute and 20 seconds to create the text version and write it to a file. Perhaps a contributing factor to this slowness is the fact that the prescript tool insists upon creating a file for the output that it creates - it does not simply stream it to STDOUT.

On the whole, I have found this tool to give very nice results when used to convert most of the PostScript documents that I used to test it. On some documents, the output was of a very poor quality, but I believe that this is because the documents themselves were of a poor quality. For example, the text of the postscript looked very 'rough', as though the document were scanned.

One major disadvantage encountered was that the prescript tool appeared to completely fail with all PostScript documents produced by Microsoft applications (as did all other tools tested). When used to attempt to convert a Microsoft-produced PostScript, the prescript tool would simply produce several error messages and then die. This is obviously a major problem if the tool is to be used to create any sort of tool that would be put into production, as it is quite possible that Microsoft PostScript files will be received for processing, and if the tool that converts the PostScript to text cannot handle any of the Microsoft PostScripts that it receives, it could be disastrous.

The ‘prescript’ Tool: Testing Conclusion

Despite its apparent slowness of operation, the quality of the output from the prescript tool is rather good. I was disappointed with the fact that it could only convert PostScript documents and not PDF documents as well, but I was prepared to live with this fact. However, when I discovered that the tool could not convert PostScript documents that were created from Microsoft products, I was extremely disappointed, and decided that as a result, the tool was unusable for the purposes of CERN. Very upsetting - if it was not for this fatal floor in its operation, I would have selected it, as it gave very nice output.

Verdict: Unsuitable.

Tools for Converting PostScript/PDF to Text: Conclusion

Many conversion tools have been tested with the primary goal of finding a tool for converting PostScript/PDF documents to plain text or HTML format so that the references section can be located and extracted. I have found that there are many positive factors to each tool, along with many negative factors. In general, I found that all tools had similar flaws. The primary flaw discovered was that none of the tools were able to convert PostScript documents that were produced by Microsoft Word. Due to the increasing number of PostScript files submitted to CERN’s archives that were actually produced from Microsoft Word files, this is a major problem, as it means that none of these could be converted to text. This means that it is really necessary to perform the conversion from a PDF document, as it seems to be more stable. Tests were conducted for the conversion of PDF documents to text that were produced from PostScripts that were in turn produced from Microsoft Word files. Problems were not encountered with these PDF file conversions. This unfortunately means that the PostScript conversion tools tested during the course of this research are effectively unusable for the purposes of extracting references at CERN. This is unfortunate, as some good results were seen from PostScript conversion tools, in particular, prescript.

Many other drawbacks were found with all files, including the erroneous breaking of lines, text from images being included in the textual output of the tool when it should not be, occasional erroneous splitting of words, etc. With the PDF files, a common, and extremely unfortunate situation is that the tools seem unable to produce good quality textual output when the quality of the PDF file is bad. This could be a problem because many older PDF files are of a poor quality.

In general, some tools performed far more slowly than others, others produced worse quality output than others, etc. Finally, it was necessary to examine the results attained, and make a choice of which tool to use. The pdftotext tool was chosen due to its quick operation, and the fact that it seemed to produce the best and most reliable output of all of the tools for converting PDF files to text. It will be used at CERN for converting PDF files into the text, which will be parsed in order to locate and extract the reference information. In addition to this, the ps2ascii tool will be used in order to attempt to extract references from a PostScript file, using the PostScript coordinates, in the event of the conversion from PDF to text failing due to a poor quality or corrupted PDF file.