

---

CS310: Third Year Project 2002/3

University of Warwick

Coventry, West Midlands, UK

# **PDF to HTML Conversion**

Tamir Hassan

University Number: 0006417

Degree Course: Computer Science

Supervisor: Dr Ranko Lazic

---



## Abstract

This report details the work carried out over the last six months to investigate the problem of converting from PDF to HTML and to develop a piece of software to perform this task. A number of different layouts were investigated, including multi-column newsprint, and the software has been written to understand these layouts and extract the text accordingly. Due to time constraints, certain page features, such as tables, were not studied. Suggestions are included for further development of the project.

## Keywords

- PDF
- HTML
- conversion
- Java
- CSS
- paragraph
- formatting
- columns
- layout

## Note to the reader

All words in *bold italics* are described in the Glossary.

---

## Acknowledgements

I would like to thank the people at IDR solutions for publishing *JPedal* as a free, open source library, without which the project would not have been possible. I would also like to thank Dr Ranko Lazic, my supervisor, for giving me guidance when I needed it and Andrew, for letting me know about *JPedal* in the first place.



## Table of Contents

1	INTRODUCTION.....	1
1.1	Background to PDF .....	1
1.2	Background to HTML .....	2
1.3	Motivation.....	3
2	INVESTIGATION OF EXISTING SOLUTIONS .....	5
2.1	Results with PDF to HTML Recastor .....	5
2.1.1	Problems with converted output .....	6
2.2	Results with pdftohtml.....	7
2.3	Results with pdf2html converter .....	8
2.4	Results with Google.....	9
2.5	Areas of possible improvement .....	9
2.6	Conclusion.....	10
3	PROJECT DECISIONS .....	11
3.1	Change of aims and objectives .....	11
3.2	Implementation decisions .....	11
3.3	Conversion material.....	12
3.4	Program output .....	13
3.5	Choice of language and platform .....	13
3.6	The JPedal Library .....	13
3.7	The front end .....	14
4	DESIGN AND IMPLEMENTATION.....	15
4.1	Class hierarchy.....	15
4.2	Text extraction principles .....	16
4.3	Text merging principles .....	17
4.3.1	Simple text merging .....	18
4.3.2	Sorting text fragments.....	18
4.3.3	Merging process.....	19
4.3.4	nextChar and nextLine methods .....	20
4.4	Column-based layouts.....	21
4.4.1	Original column detection algorithm .....	21
4.4.2	Improved column detection algorithm .....	22
4.4.3	Ordering of text in columns.....	24
4.5	Other layout features .....	25

---

4.5.1	Line spacing.....	25
4.5.2	Styles and formatting.....	27
4.5.3	Symbols .....	28
4.5.4	Hyphenated text.....	29
4.5.5	Indentations.....	29
4.5.6	Forced carriage returns.....	30
4.5.7	Raised and dropped capitals .....	31
4.5.8	Miscellaneous text fragments.....	31
4.5.9	Empty text fragments .....	32
4.6	The front end .....	32
4.6.1	The -c option.....	32
4.6.2	Multiple pages .....	33
<b>5</b>	<b>PERFORMANCE EVALUATION .....</b>	<b>34</b>
5.1	Analysis of converted output .....	34
5.2	Comparison with other methods .....	35
5.3	Examples of converted output.....	35
5.3.1	Simple letter example .....	36
5.3.2	E-book example.....	38
5.3.3	Simple newsletter example.....	40
5.3.4	Complex newsletter example.....	42
<b>6</b>	<b>CONCLUSION.....</b>	<b>44</b>
6.1	Limitation of the implementation.....	44
6.2	Author's assessment of the project.....	44
<b>7</b>	<b>FUTURE DEVELOPMENT .....</b>	<b>46</b>
7.1	Conversion to RTF format .....	46
7.2	Using graphical data on the page .....	46
7.3	Understanding tabular data.....	46
7.4	Inclusion of graphics in converted output.....	46
7.5	Detection of multi-level styles.....	47
	<b>GLOSSARY OF TERMS .....</b>	<b>48</b>
	<b>REFERENCES .....</b>	<b>49</b>
	<b>BIBLIOGRAPHY .....</b>	<b>49</b>
	<b>CONTENTS OF INCLUDED CD .....</b>	<b>50</b>

# 1 Introduction

## 1.1 Background to PDF

PDF started off as an internal project at Adobe based on the early '90s dream of the paperless office. The objective was to create a file format that would allow documents to be distributed throughout the company and viewed on any computer running any operating system. Adobe already had a more or less fitting technology; PostScript, a device and platform independent page description language that was already in widespread use in the printing industry. It was therefore natural that PDF was based on PostScript. Before its official release, PDF was even referred to as Interchange PostScript (IPS) by Adobe.<sup>[1]</sup>

Version 1.0 of PDF was formally released in 1992. Although very similar to PostScript there were a number of differences. One of the most important features was compression, which could typically reduce file size by an order of magnitude, facilitating storage and transmission. Bookmarks and links were also included, although you could only link internally to another page in the document at that time. The Acrobat suite of products followed in 1993 and, at the time, you had to pay £50 for a copy of Acrobat Reader.

During the 1990s the use of the Internet became more widespread and Adobe were in a position to take advantage of this. Future incarnations of the PDF format enabled Adobe to take advantage of the growth of the Internet. Adobe dropped the £50 charge for Acrobat Reader and over 100 million copies were downloaded from the web.<sup>[1]</sup> Support for hyperlinks, scripting and, more recently, tagging were added; the latter being a system of inserting metadata about the paragraph structure to enable re-flowing of text on hand-held devices such as PDAs and e-books.

PDF's biggest advantage is that it is based on a page description language. Therefore any PDF file should display identically on any computer system irrespective of the hardware or operating system being used. Acrobat's printer drivers, *Distiller* and *PDFWriter*, have made creating a PDF as easy as printing a document. This has enabled documents created in a word processor, such as *Microsoft Word*, to be converted to PDF in a single keystroke, ready for transmission or publication to the Web.

## 1.2 Background to HTML

In 1989, Tim Berners-Lee proposed a global hypertext project, to be known as the World Wide Web, while he was working at the CERN particle physics laboratory in Geneva, Switzerland. He wrote the first web server, *httpd*, and the first browser, *WorldWideWeb*, in late 1990, and these programs became available on the Internet at large in the summer of 1991.<sup>[2]</sup> The language used for document exchange was HTML, HyperText Markup Language, and was invented by Berners-Lee for this particular purpose.

As a structured hypertext language, HTML is worlds apart from PDF. Based upon the notion of separating content from presentation, it includes tags to denote paragraphs, heading levels and lists. These tags are understood by the client's browser and used to render the page in an appropriate form on the screen. This is why HTML files often look different when displayed on different platforms or different browsers. HTML even allows the use of external *style sheets* which separate the formatting information in a different file, making it possible to alter the presentation of a whole web site by changing only the style sheet, independently of the content.

As time has passed, technology has moved on and web designers, particularly when working on commercial sites, felt the need to give their sites an original, distinguished look. This required playing tricks with tables and tags such as `<BR>` (*break*) and `&nbsp;` (space) to subvert HTML's principles and get the site to display how they wish. Even then, different browsers often interpreted the HTML differently and it was necessary to design separate sites for different browsers! The "browser war" in the late '90s only added to the confusion as both Microsoft and Netscape invented their own tags, such as the ubiquitous `<BLINK>`, in a hope to gain market share.

Today, HTML is used alongside other technologies, proprietary and open, such as scripting, Flash, streaming audio and video and even PDF, to create the very rich content that we now see on the web. Very often, HTML is not used in the way it was originally intended, and modern HTML files include so much formatting, metadata and other information that they are very difficult to edit. The advent of HTML editing tools, such as *Dreamweaver*, has improved this situation somewhat, but it is still true that editing a modern web site is usually more difficult and time-consuming than it needs to be.

This project has aimed to adhere to HTML's original principles as far as was reasonably practical. For example, the program generates an internal style sheet in the header of the HTML file. However, there is one case where it has been necessary to produce a "fudge": indentations are not supported in HTML (other than in complete paragraphs) and most browsers simply ignore the `<TAB>` tag. Therefore it has been necessary to simulate an indentation by including four spaces (`&nbsp;`). For more information, see section 4.5.5.

### 1.3 Motivation

This motivation for this project arises from the need to convert PDF files to HTML for publishing on a web site. Although PDF files can be viewed with an appropriate plug-in or reader, it is often more appropriate to publish shorter documents in HTML for the following reasons:

- The client must have a copy of a PDF viewer such as Acrobat Reader, which must be executed to view the PDF. Although Adobe's plug-in for popular browsers aims to integrate seamlessly with browsers, there are many inconsistencies between the user interface of a web browser and Acrobat which can be confusing to the user. For example, to print the document the user must click the "printer" icon on the Acrobat toolbar, not the **Print** button on the browser.
- Although PDF files feature compression they are, in general, still significantly larger than the equivalent HTML content. This raises a problem for users with slower Internet connections, particularly those who regularly view web pages without images.
- HTML files can have a "house style" applied to them to allow them to maintain a consistent appearance for a professional appearance. Due to the page-based nature of PDF files, they will always look different to HTML pages, and this gives the impression that they are not part of the main web site. PDF files also do not (and can not) support style sheets and will have to be updated separately if, for example, the web site adopts a new image.
- As the layout of a HTML file is flexible (and dependent on the system on which it is being rendered), it is much easier to edit HTML files, and even make drastic alterations, without impairing the appearance of the page.

PDF files are usually only more appropriate if the content is to be printed, rather than to be viewed on screen. Even in newsprint, which has a very complicated layout that

cannot easily or practically be replicated in HTML, it is best to extract separate articles and publish them in HTML, making the PDF available for download solely for printing purposes.

Unfortunately, the ease of creating PDF files has led to the creation of many PDF documents on the web that would be more appropriate in HTML. This is because many documents are created in word processing packages such as *Microsoft Word* and converted to PDF simply by “printing” them to an Acrobat printer driver.

## 2 Investigation of existing solutions

The following four solutions were found, and were investigated in detail with a variety of PDF files.

- **PDF to HTML Recastor** by Archisoft
- **pdftohtml** sponsored by Lincoln & Co
- **pdf2html** by Twibright Labs
- **Google's View as HTML** feature for cached PDF files

Full URLs to the web sites of these programs are given in the bibliography.

The first, **PDF to HTML Recastor**, is the only commercial solution that had a trial version available for download. The next two are open-source, and **Google's** feature is an example of a server-side implementation on a web site.

The following PDF files were used in the investigation. All the files were downloaded on October 21, 2002.

Title	Type of layout	Location
<i>Boston Sunday Globe, Today, October 20, 2002</i>	Complex newspaper; columns	<i>www.boston.com/globe/acrobat/today.pdf</i>
<i>White Paper: Is the Network Slow Today?</i>	Word-processed document	<i>www.netscout.com/files/artmb_wp.pdf</i>
<i>Connex South Eastern Rail Timetable #5</i>	Tabular	<i>www.connex.co.uk/upload/timetable/PTT05.pdf</i>

As three of the converters, **PDF to HTML Recastor**, **pdftohtml** and **Google** used the same method they generated similar results. To avoid repetition this method is described in detail in the next section only.

### 2.1 Results with PDF to HTML Recastor

This converter generated a series of HTML files from the PDF; one for each page of the document. PNG images were also included for the graphics. The converter also had a feature to generate a page index in a separate frame displayed on the left-hand side of the screen.

At the first glance, the converted documents looked very realistic. The program had no difficulty in converting the simple, word-processed document and even coped with the

multiple columns of the newspaper article. The newspaper article and railway timetable, however, are very complex, and revealed some limitations with the converter. These were very useful in gaining an understanding of how the converter works.

Closer examination of the converted output revealed that this was achieved by using an inline *style sheet* to give an absolute pixel position to each fragment of text. The graphical elements of the page, including images, lines and boxes, were combined into a single image, which was shown as the background of the HTML page.

### 2.1.1 Problems with converted output

The newspaper article was designed for a large (approximately A3-size) page, and could not be viewed on an average computer screen without scrolling. The converter attempted to resize the page to fit in an average browser window (about 800 pixels across). This resulted in text that was too small to read, even with the magnification facility on the converter set to its maximum. As the font sizes were specified in pixels, altering the text size on the browser had no effect.

This problem was exaggerated by the fact that all the text displayed in the HTML file was smaller than the text in the PDF document when viewed at a similar level of magnification. This was understood to have been done to avoid columns of text running into each other when viewed with different fonts or across different platforms.

What was even more striking was that the headline simply appeared as “fghijkl”. During the creation of the newsletter, a custom font for the headline was probably used, in which single characters were mapped to entire words in the headline. As the converter output everything in the Arial font, the headline was displayed as “fghijkl” as shown below. This is a rather special case, but it does illustrate the fact that PDFs can be created in a wide variety of ways, and that it is difficult to design a program to account for all of them.



Fig 2.1: A section of the Boston Sunday Globe newspaper (left) and as converted by the Archisoft converter (right)

A different issue was highlighted by the conversion of the train timetable. In Figure 2.2 below, all the figures should appear under each other. However, the converter has mistakenly detected the circled side-by-side figures as words in a line of text. Rather than place them separately, it has placed them together as a line of text, with each figure separated by a space. The result is that the figures are not in the right place. The figure “1717” should actually appear underneath the figure “1713” in the line above.

New Crossham	dep	1030	1715		
Mottingham	dep	1700	1717		
Lee	dep	1703	1720		
Hither Green	dep	1707	1724		
Barnehurst	dep	1646	1712		
Bexleyheath	dep	1649	1715		
Welling	dep	1652	1718		
Falconwood	dep	1654	1720		
Eltham	dep	1657	1723		
Kidbrooke	dep	1700	1726		
Blackheath	dep	1703	1713	1729	
Lewisham DLR	dep	1706	1712 1717	1730 1734	
St Johns	dep			1736	
New Cross	dep	1716	1735	1739	
Nunhead	dep	1711			

Fig 2.2: A section of the railway timetable (left) and as converted by the Archisoft converter (right)

Although most text fragments in a PDF file consist of an entire line of text, this is not necessarily the case. Changes in font or the inclusion of symbols often necessitate that a separate text fragment is used. Some poorly created PDFs even place each word or each character as a separate text fragment. The feature described above is intended to combine these fragments to create a continuous line of text, improving the appearance when converted to HTML. Unfortunately, it also causes problems for tabular data.

## 2.2 Results with pdftohtml

This converter could generate two types of output, simple and complex. Complex output was generated by using the -c command line parameter. As with the Archisoft

converter, it also had an option to generate a page index in a separate frame displayed on the left-hand side of the screen.

In the complex mode, the results were very similar to those of the Archisoft converter. The only noticeable difference was that the main font used was Times, not Arial. The converter ran into exactly the same difficulties with the newspaper and railway timetable and the results produced were almost identical.

In the simple mode, the converter output all the graphics followed by the text. Text was not positioned absolutely, but output as ordinary paragraph text. A *break* (<BR>) was used at the end of each text fragment and no attempt was made to merge fragments into paragraphs. However, a good attempt at ordering the text was made and, even with the complex newspaper layout, entire articles could be read. There were, however, no headings or paragraph markers to indicate where these articles began. An example of this output is shown below.



Fig 2.3: A section of the Boston Sunday Globe newspaper (left) and as converted by the *pdf2html* converter in simple mode (right)

### 2.3 Results with pdf2html converter

This converter simply used *Ghostscript* to create a PNG image of each page at a resolution appropriate for on-screen viewing. It then generated a series of HTML pages to display each PNG image. The result was perfect, although the newspaper article appeared to small to be readable. With this method of conversion, however, the file size is large and all the benefits of HTML are lost.

## 2.4 Results with Google

As Google only offers this facility for pages stored in its cache, it was not possible to convert the three PDF documents that were used to investigate the previous solutions. However, similar documents were found by searching Google's cache, and showed that the method of conversion is identical to that of the Archisoft converter and pdftohtml in complex mode.

Cosmetically, the output is slightly different as it combines each page into a single HTML file. A  $1 \times 1$  HTML table, displaying the page number, is used to rule a line between each page.

## 2.5 Areas of possible improvement

Investigation of the Archisoft, pdftohtml (complex mode) and Google converters highlighted the following improvements that would improve the result:

- **an option to rasterize text above a threshold size:** as many pages often use decorative fonts for headings, this would enable the page to more closely resemble the original. This feature would also have solved the problem of the custom font for the headline in the newspaper article
- **an option to change the horizontal size of the output:** this may necessitate scrolling, but will allow text to be displayed at a larger and more legible size.
- **improved resizing of images:** when the individual images were combined into one background image, they were resized to appear proportionately to the rest of the page. Unfortunately, the algorithm used by these converters was very simple, using the "nearest neighbour" method. This caused much distortion in the images, particularly as many PDF images are already at a low resolution to minimize the file size. An algorithm utilizing bilinear or bicubic *resampling* would have generated a better result
- **recognition of tabular content:** if the text in the PDF is recognized as belonging to a table (e.g. where successive lines have the same x co-ordinate) the converter should not attempt to merge this material into a single line of text

Investigation of the pdftohtml converter in simple mode made the author aware of another possible approach to the conversion, one that will hereafter be named *intelligent text extraction* in this report. This involves putting the text fragments into the correct order so that they can be merged to create complete paragraphs of text.

Styles and formatting can then be applied to distinguish headlines from body text. A much simpler HTML file can be created this way, which can then be published to a web site.

## 2.6 Conclusion

All the above converters were found to produce good visual results from a variety of different documents. In all cases the method of conversion allowed a “one step” approach to be taken, without any need for configuration by the user. This is useful as these programs are mostly aimed at inexperienced users who do not have the expertise to perform such a task manually.

However, questions arise as to whether the approach of maintaining page layout is the correct approach to take, and under which circumstances the generated HTML files will be useful. With this approach most of the benefits of HTML are lost as text cannot be re-flowed for on-screen viewing and the file cannot easily be incorporated into a web site.

## 3 Project decisions

After four existing solutions to the problem were investigated, it was decided to change the approach of the conversion from attempting to reproduce the page layout to *intelligent text extraction*. The reasons for this decision, and a plan of the work to be carried out, are shown below.

### 3.1 Change of aims and objectives

The original aims of the project were to perform an accurate conversion, maintaining the page layout, fonts, graphics and other elements as closely as possible. After investigating the existing solutions to the problem, it was found that this approach had already been successfully implemented in three different pieces of software.

Although visually accurate, the results with these converters were not very practical. Most of the advantages of the HTML format were lost with this type of conversion; text was too small, could not be re-flowed and the output could not easily be converted into a web-publishable document. It was therefore decided to change the aims of the project to intelligent text extraction; attempting to detect elements such as paragraphs and headings and using HTML's features to represent them in the converted file.

This approach is far more challenging than simply maintaining the page layout as it involves programming a computer to understand the elements of a page in such a way that a human would. This fact had not been fully realized at the time of writing the Progress Report, and it was therefore necessary to further modify the objectives to simplify the implementation so that it would be completed in the time allocated for the project. As a result, the implementation looks solely at the text elements of the PDF.

### 3.2 Implementation decisions

There is a huge variety of documents that are stored in PDF format ranging from simple layouts such as manuals and research papers to complex layouts such as newsletters, catalogues, tables and forms. All the existing solutions performed a “one-step”, layout-independent approach that was performed an accurate conversion reproducing the original layout, thus providing an acceptable result regardless of the type of document.

This one step approach is not possible with intelligent text extraction, as the program itself must understand the particular page layout. Many features in complex layouts

can only be reproduced in HTML by using tricks such as preformatted text, tables or graphics. Simply extracting the text in this case presents many difficulties; for example columns need to be detected and output in the correct order and line ends must be merged to create a continuous flow of text and new paragraphs must be detected. It was therefore decided to present the text as a simple HTML document, similar to the style of a word-processed document.

Each type of layout must be treated differently with this approach and it was decided to group the layouts into two categories: single-column layouts and multi-column layouts. For each of these categories, two PDF files were found and used as the basis of the conversion material for the project.

### 3.3 Conversion material

The PDF files used as the basis of the conversion material are listed below.

#### Single-column:

---

**Title:** Sample letter  
**Author:** Tamir Hassan  
**Filename:** sampleletter.pdf  
**Obtained:** Created in *Microsoft Word* and *Acrobat Distiller* for the purposes of this project  
**Features:** Typical letter layout; includes addresses with *forced carriage returns*

---

**Title:** A Tale of Two Cities (Non-Tagged)  
**Author:** Charles Dickens  
**Obtained:** Downloaded from *planetpdf.com*  
**URL:** [http://www.planetpdf.com/A\\_Tale\\_of\\_Two\\_Cities\\_NT.pdf](http://www.planetpdf.com/A_Tale_of_Two_Cities_NT.pdf)  
**Features:** Header and footer with page numbers; indented paragraphs

---

#### Multi-column:

---

**Title:** Register of English Football Facilities Newsletter – Summer 2001  
**Author:** Register of English Football Facilities  
**Filename:** reffnewsletter.pdf  
**Obtained:** Downloaded from *footballfoundation.org.uk*  
**Features:** Variable line spacing; boxed quotations; variety of fonts and formatting  
**Title:** Witness for Peace Newsletter – Winter 2002

---

**Author:** Witness for Peace  
**Obtained:** Downloaded from *witnessforpeace.org*  
**URL:** [http://witnessforpeace.org/pdf/newsarch/winter\\_02.pdf](http://witnessforpeace.org/pdf/newsarch/winter_02.pdf)  
**Features:** Very complex layout; pictures with captions; boxed articles; headers and footers; variable line spacing; paragraphs with little extra spacing

---

### 3.4 Program output

After analysing the shortcomings of the existing solutions, it was decided that the converted HTML should be clean, legible, correctly structured, and that the correct tags for styles and paragraphs be used where possible. An example of such a HTML file is shown below

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<STYLE type="text/css">
h1 {font-family: helvetica; font-size:24}
p {font-family: times}
</STYLE>
</HEAD>
<BODY>
<h1>This is a heading </h1>
<p>This is normal text </p>
<p>And this is a separate paragraph </p></BODY>
</HTML>
```

*Fig 3.1: Example of "clean" HTML output*

### 3.5 Choice of language and platform

Java was chosen as it was the main programming language taught at University, with which the author was already familiar. Its multi-platform nature facilitated development, as a combination of Windows, Linux and Solaris platforms was used. It also allows the finished program to be run on any supported platform, which is particularly advantageous as both PDF and HTML are also multi-platform formats. Finally, the *JPedal* library, written in Java, performs the low-level operations on the PDF file itself, allowing the project to concentrate more on the conversion aspects of the problem.

### 3.6 The JPedal Library

JPedal, the Java Pdf Extraction, Decoding and Access Library, was used to access the data from the PDF. JPedal is compatible with PDF files up to version 1.3; the specification for PDF 1.4 has not been fully released by Adobe. Therefore it does not provide access to the tags that are provided in certain e-books that denote paragraph

information. It also has a few other limitations, including that it will not work with encrypted PDF files.

JPedal was written by IDR Solutions as part of their commercial PDF extraction package, *Storypad*. JPedal is, however, published under the Lesser GNU Public License (LGPL) allowing it to be freely used and modified. The source code is provided, which included a generic class, **PdfGenericGrouping**, to extract and group the text. It was decided to extend this class to provide an enhanced class, **PdfGrouping**, to merge paragraphs, detect columns, re-order the text blocks and include style information.

More information about the use of JPedal is given in the next section.

### 3.7 The front end

As the **PdfGrouping** class performed all the processing it was necessary to include a front end, **pdf2html**, to perform all the input, file accesses and output. It has a command line interface that can be called by a script or GUI.

## 4 Design and implementation

This section details the steps taken in designing and implementing the conversion software. As the author had decided to attempt a new approach, it was not possible to produce a complete design at the outset. Instead, small sections of the program were designed and implemented at a time. Many of the later features were included as a result of viewing the program's output and carrying out modifications and improvements where necessary.

Section 4.1 briefly describes the class hierarchy in which the program is organized. Sections 4.2 and 4.3 describe the two main processes in converting from PDF to HTML, text extraction and text merging. The next two sections describe improvements to the implementation to cope with more complex PDF files; section 4.4 describes the modifications made to cope with multi-column layouts and section 4.5 describes a number other features in the layout, such as hyphenation and line spacing, and how they have been understood by the conversion software. The final section, 4.6, describes the front end's role in the implementation.

### 4.1 Class hierarchy

The class **PdfGenericGrouping** was provided in *JPedal* as a base class for which grouping and merging routines could be added. The **PdfGrouping** class was written to extend **PdfGenericGrouping**, adding merging, ordering and other routines necessary for the conversion software, as shown below.

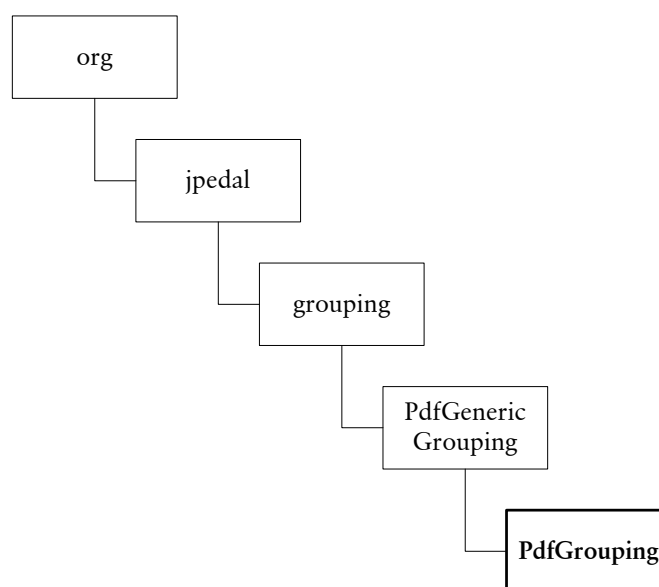


Fig 4.1: PdfGrouping: class hierarchy

The main method, `processPageFragments`, is used to interface between the front end and `PdfGrouping` itself. It was designed to replace the method `decodePageFragments` from `PdfGenericGrouping` although `decodePageFragments` is still accessible if unmerged data is sought.

The only method that has been replaced is `getUnusedFragments` as it was necessary to pass a value, `usedFragments`, in order for it to function properly. Otherwise the code remains identical to that included in `PdfGenericGrouping`.

The front end, `pdf2html`, requires `PdfGrouping` and all its dependencies but is otherwise stand-alone.

## 4.2 Text extraction principles

The general procedure to open a PDF file and extract text fragments with JPedal is shown below.

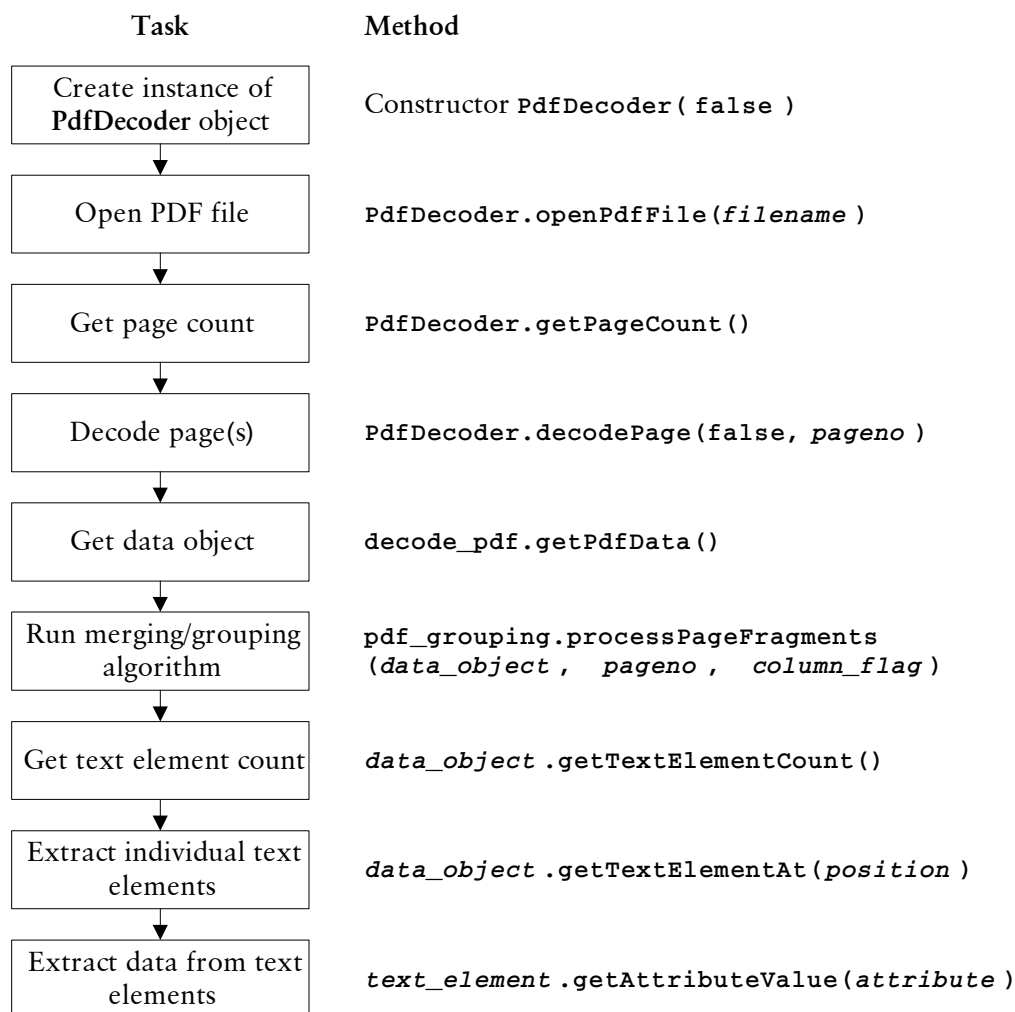


Fig 4.2: Text extraction process

Text in a PDF is held as a series of text fragments. These fragments may be written to the PDF file (and hence extracted by JPedal) in any order. Each text fragment usually contains one full line of text although changes in formatting and the inclusion of certain symbols require the line to be separated into separate fragments. Some PDF file creators place each word or character as a separate fragment.

Each text fragment is held as an XML **Element**, containing the various attributes holding information about the text fragment itself. The only attribute used at this stage was **content**, which was a string including the text embedded in XML/HTML formatting information, START and END tags, as shown below:

```
<~START><FONT face="Minion-Regular" style="font-size:12pt">The quick  
brown fox jumps over the lazy dog</FONT><~END>
```

*Fig 4.3: Example of output from JPedal*

It is therefore necessary to process the string to separate the text from the formatting information and the method **textOf** does this. Other data, such as co-ordinates and font size information, were accessed directly from the arrays by the **PdfGrouping** class.

### 4.3 Text merging principles

The **processPageFragments** method in the **PdfGrouping** class performs the text merging procedure, calling other methods in the **PdfGrouping** class and interfacing between the front end and the library.

In the **PdfGrouping** class text fragment data is held in a number of arrays, each being the same size as the total number of text fragments. The contents of these arrays is updated when the **copyToFragmentArrays** method is called at the beginning of the **processPageFragments** method. Each array holds information about one particular attribute and data about a particular text fragment is held in the same index across all the arrays. Hence **heights[36]**, **f\_start\_font\_size[36]**, **contents[36]** and **text\_length[36]** are all attributes of the same text fragment.

The attributes that were used in the grouping and merging algorithms include:

- **contents[]**: the actual text embedded in XML/HTML formatting information
- **f\_x1[], f\_x2[], f\_y1[], f\_y2[]**: co-ordinates of the bounding box of the text fragment
- **heights[]**: the height of the text fragment
- **f\_start\_font\_size[]** and **f\_end\_font\_size[]**: start and end font sizes respectively

- `f_is_horizontal[]`: a boolean variable set to true if the text is horizontal
- `isUsed[]`: a Boolean variable provided to flag elements that have already been merged into other elements

Once the data has been processed the `writeFromFragmentArrays` method is called to write the modified data back to the `PdfData` object.

### 4.3.1 Simple text merging

The general procedure for merging text fragments is shown below. For simple page layouts this sort is simply in order of Y co-ordinate then X co-ordinate. The text fragments will then be in the correct order so that successive fragments can be appropriately merged together to form a continuous flow of text.

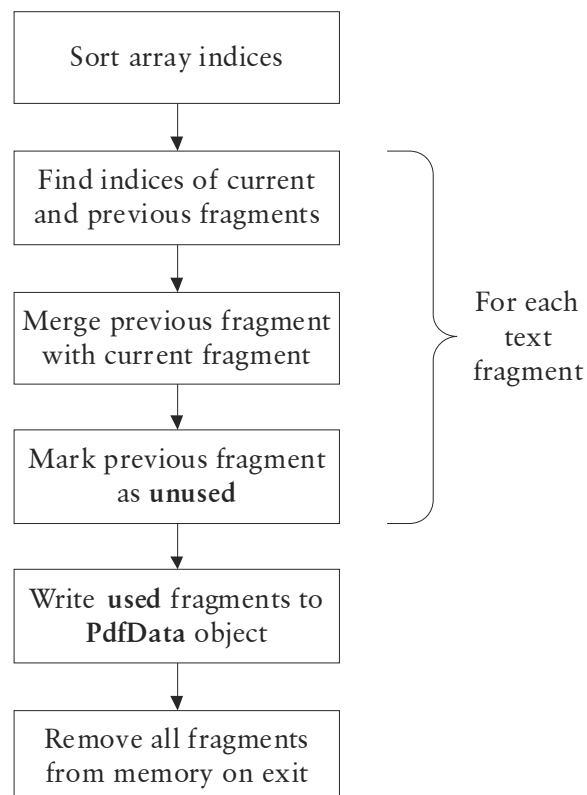


Fig 4.4: Basic text extraction method

### 4.3.2 Sorting text fragments

As the text fragments were held in a series of arrays it would be necessary to sort every single array identically to ensure that all data is preserved. As this method was felt to be inefficient and clumsy, it was decided not to sort the arrays but to create a new array, `Order[]`, which would keep track of the current order of the indices of the

elements. Initially this array was set to count 0, 1, 2, ... , up to the highest numbered text fragment.

The sort was carried out by using a modified version of the `XYComparator` method downloaded from [java.sun.com](http://java.sun.com)<sup>[3]</sup> and calling the `Arrays.sort(Comparator)` method.

A further problem was encountered when investigating the co-ordinates of each text fragment. The vertical co-ordinates of fragments on the same line were found to differ slightly. This was because certain fragments included capital letters, symbols or characters such as “g” which extend beyond the baseline. This sometimes caused the text to be merged in the incorrect order. The figure below illustrates this problem.



*Fig 4.5: Example of words and symbols with different y co-ordinates*

This problem was overcome by the creation of the `sameLine` method which works in two steps. First, both `y1` and `y2` co-ordinates are examined, and returns true if either co-ordinates intersect. Secondly, an “error margin” of 25% of the line height is used and, if the fragments are within this margin, the method also returns true.

A new comparator was created, `YErrComparator`, which calls `sameLine` with two successive fragments and sorts these fragments in X order if it evaluates to true.

### 4.3.3 Merging process

Merging of text fragments is performed by the `mergeTextObjects` method in `PdfGrouping`. Text fragments are processed in the order that they are sorted, and each fragment is merged with the next, building up a continuous page of text. The relationship between two successive text fragments determines how they will be merged, and is ascertained by examining both sets of co-ordinates, as performed in the `sameLine`, `nextChar` and `nextLine` methods.

A simplified version of the procedure for each pair of text fragments is shown in the diagram below. Features such as dropped capitals, hyphenation and variable line spacing have made this process more complex and these are covered in later sections.

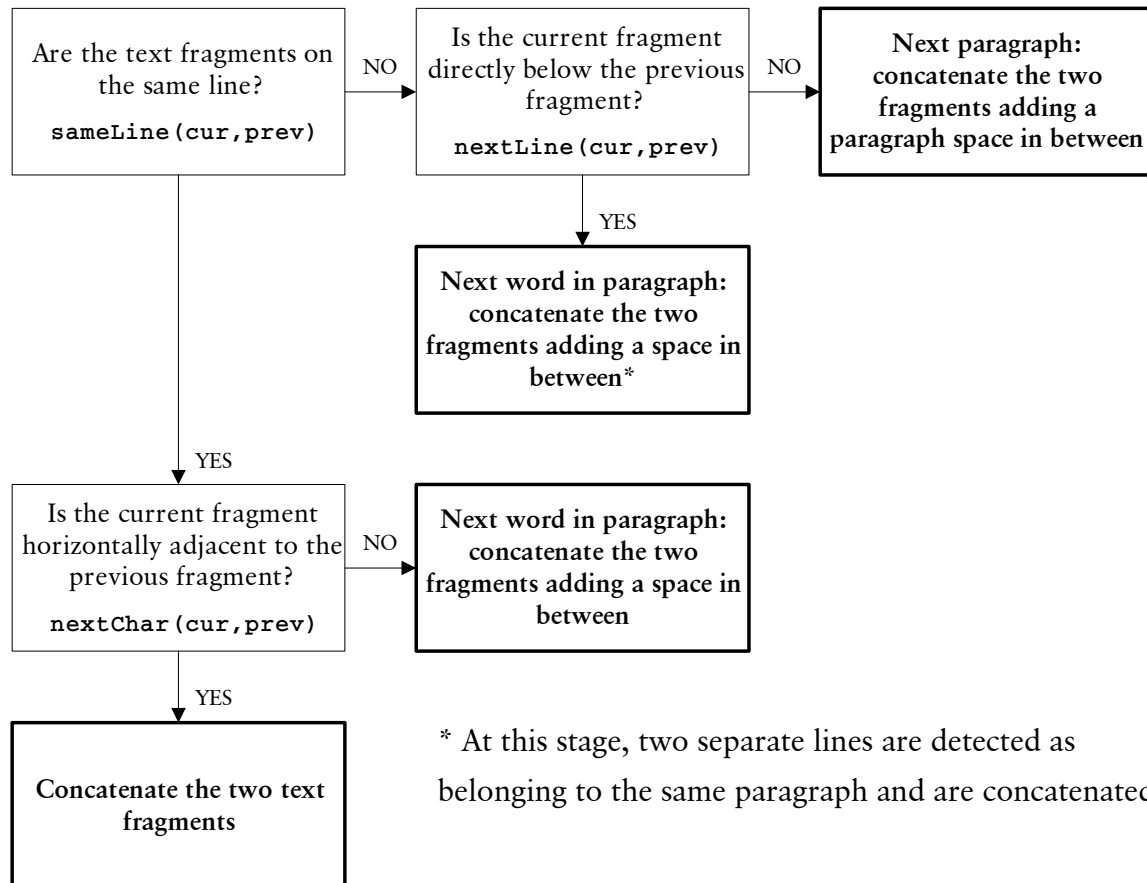


Fig 4.6: Text merging process

#### 4.3.4 nextChar and nextLine methods

The `nextChar` method returns true if two text fragments are horizontally adjacent. It is unnecessary to evaluate vertical co-ordinates as this method is only ever called when `sameLine` has evaluated to true. An “error margin” of 25% of the line height is used, and the `x1` co-ordinate of the current text fragment has to be equal to, or within the error margin of, the `x2` co-ordinate of the previous fragment.

If `nextChar` evaluates to true, the merging algorithm concatenates the two fragments without inserting an intermediate space. Otherwise, it assumes that the two fragments are separate words and inserts a space between them.

The `nextLine` method returns true if two text fragments are vertically adjacent. Originally this method was programmed to return true if the `y1` co-ordinate of the current text fragment was equal to the `y2` co-ordinate of the previous fragment, or within an error margin of 30% of the line height. However, this method was rewritten when the program was improved to allow for varying line spacing. This is covered in section 4.5.1.

If `nextLine` evaluates to true, the simple merging algorithm assumes that the text fragments are separate words that have been wrapped to the next line, and belong to the same paragraph. It therefore concatenates the two fragments, inserting a space in between. Otherwise, it assumes that the two fragments belong to different paragraphs and inserts the appropriate HTML tags to indicate this.

## 4.4 Column-based layouts

This section describes how column-based layouts, such as those of a typical newsletter, were detected and processed so that the text was extracted in the correct order. The general approach was to use the array `group[]` to group the text fragments according to which column they belong, and to output each group (column) of fragments in order from left to right. Since the fragments were already sorted on their vertical co-ordinate, a *stable* sort on the group number was all that is required. Two different methods of grouping the fragments were tried; the original method was found not to work reliably in practice and was replaced by a different method which was found to give better results.

### 4.4.1 Original column detection algorithm

The original method was to partition the page into several blocks which would correspond to separate columns and possibly separate articles, as shown in the figure below. A class called `rectangle` was created to hold the co-ordinates of the bounding boxes of these blocks of text. Each fragment was examined in the order that they were sorted (Y then X) and compared to each bounding box. If the fragment looked like it was part of an existing text block; for example by being directly under it, the bounding box would be grown to encompass the text fragment. If, on the other hand, the text fragment did not fit into any of the existing text blocks, a new text block would be created with the co-ordinates of the bounding box of the text fragment.

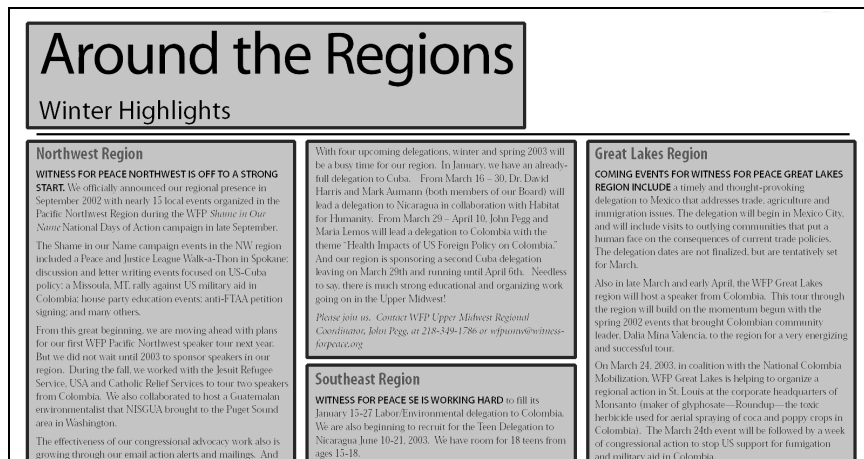


Fig 4.7: Example of newsletter partitioned into columns and articles

In practice this method was unreliable as *miscellaneous* items of text could sometimes be included in other text blocks, severely altering their dimensions and including other columns within the same text block. Sometimes the algorithm recognized the gap between two columns as a gap between two words and grew the text block to encompass the neighbouring column.

After studying the layout of a newsletter it became clear that indicates to the reader that there are two distinct columns is a continuous vertical gap. Hence it was decided to create a new algorithm based on detecting gaps between the columns instead of blocks of text.

#### 4.4.2 Improved column detection algorithm

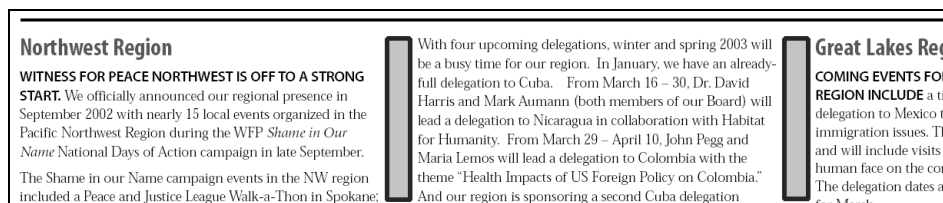
The new method, `findColumnGaps`, used the type `rectangle` to store the co-ordinates of gaps between text fragments rather than the co-ordinates of the text itself. These rectangles were held in a `Vector` named `gaps`.

Text fragments were, again, processed in the order that they had been sorted. When a horizontal gap between successive fragments was found, a `rectangle` was created with the horizontal co-ordinates of the gap, and the highest and lowest vertical co-ordinates of the two text fragments, as shown below.



Fig 4.8: First two steps in gap detection algorithm

After each gap in the text was found, its co-ordinates were compared to those of the rectangles that had already been created. This was performed by the `rectangle.checkAndUpdate` method. If the horizontal co-ordinates were identical or intersected each other, instead of creating a new rectangle, the existing rectangle was “grown” downwards so that its bottom co-ordinates became equal to those of the lowest text block. If the horizontal co-ordinates partly intersected those of the existing rectangle, the rectangle was “shrunk” horizontally so that it no longer intersected the text block, as shown below.



*Fig 4.9: Both rectangles have been “grown” vertically; the left rectangle has also been “shrunk” horizontally.*

If the existing gap was totally intersected by a text block it was grown only to the top of the text block and marked as “closed”. After the gap was marked as closed it would no longer be possible to grow the gap to accommodate further gaps and any further gaps in that horizontal position would lead to the creation of new rectangles, as shown below.



*Fig 4.10: The rectangle on the left has been “closed” such that any future gaps in that horizontal position will lead to the creation of new rectangles*

The `checkAndUpdate` method returns true if it manages to grow the current rectangle to accommodate the gap between the supplied co-ordinates. This informs the `findColumnGaps` method that it is not necessary to create a new `rectangle` object.

In order not to allow small gaps (e.g. between words) to interfere with the algorithm, a threshold height of 36 points and a threshold width of 4 points were set; all rectangles

below the threshold height or width are removed from the vector. This value was taken after finding the smallest likely gap with very small (6 point) newsprint. Newspapers often use very narrow gaps, ruling a line between the two columns to compensate visually for this.

Finally, once the number of rectangles is fixed, they are copied to an array, `gap[]`, and sorted to facilitate grouping of the text fragments in the correct order.

### 4.4.3 Ordering of text in columns

In order to work with the majority of column layouts, it was decided to order the text by column, starting at the left most column, working down the column and moving horizontally onto the next column as shown below.

This is achieved by sorting the `gap[]` array in order of `x2` co-ordinate. This array is then passed to the `groupTextElements` method that assigns a `group[]` to each text fragment based on the horizontally closest `gap` to its left. Text fragments that do not have a gap to their left (i.e. the first column) are assigned to group 0.

The text fragments are then sorted in group order using the `GroupComparator`. As this is a stable sort, the existing Y-then-X order inside each column remains.

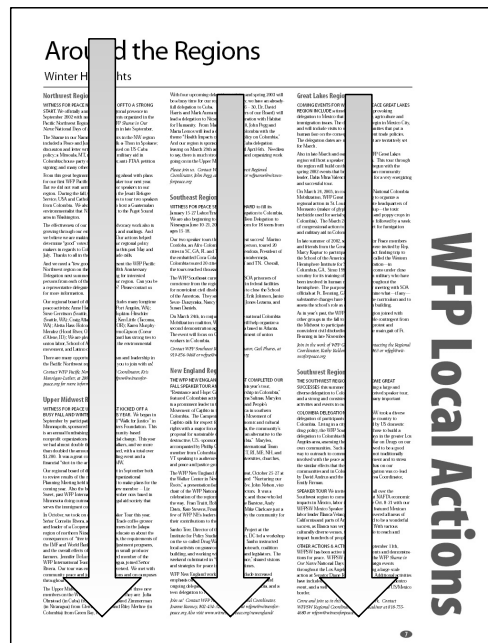


Fig 4.11: Example of column ordering

This method was found to work well for simpler layouts, such as the one above. It did not always work for more complex layouts, particularly where different articles were in columns of different horizontal positions or in boxed sections.

However, it is very difficult to distinguish between different articles on the page, particularly by looking solely at the text itself. One improvement, which is beyond the scope of this project but a suggestion for further work, is to look at other graphical elements, such as lines, boxes and shaded regions.

## 4.5 Other layout features

This section describes how other features of the page layout were understood and handled by the conversion software.

### 4.5.1 Line spacing

Originally the program was written to assume that all the text in the PDF is single spaced. The `nextLine` method was written to return true if the `y1` co-ordinate of the current text fragment was equal to the `y2` co-ordinate of the previous text fragment, or within an error margin of 30% of the line height. However, both the newsletters studied here (as described in section 3.3) contained a large amount of text that was not single-spaced and, in those cases, each line was detected as a separate paragraph.

One early improvement was to create a method, `findModalTextSize`, to process each text fragment and find the most frequently occurring text size in the page. This size would almost definitely correspond to the size of the body text. Another method, `findLineSpacing`, was used to find the average line spacing between successive lines of text of that size. This line spacing was then used to proportionally adjust the error margin that was used by the `nextLine` method to determine whether the next line was part of the same paragraph.

This improvement worked, and enabled text that wasn't single spaced to be converted properly. However, as it worked on a page-by-page basis, it did not cope where different sections of the page had different line spacings or different text sizes, as shown in the example below.

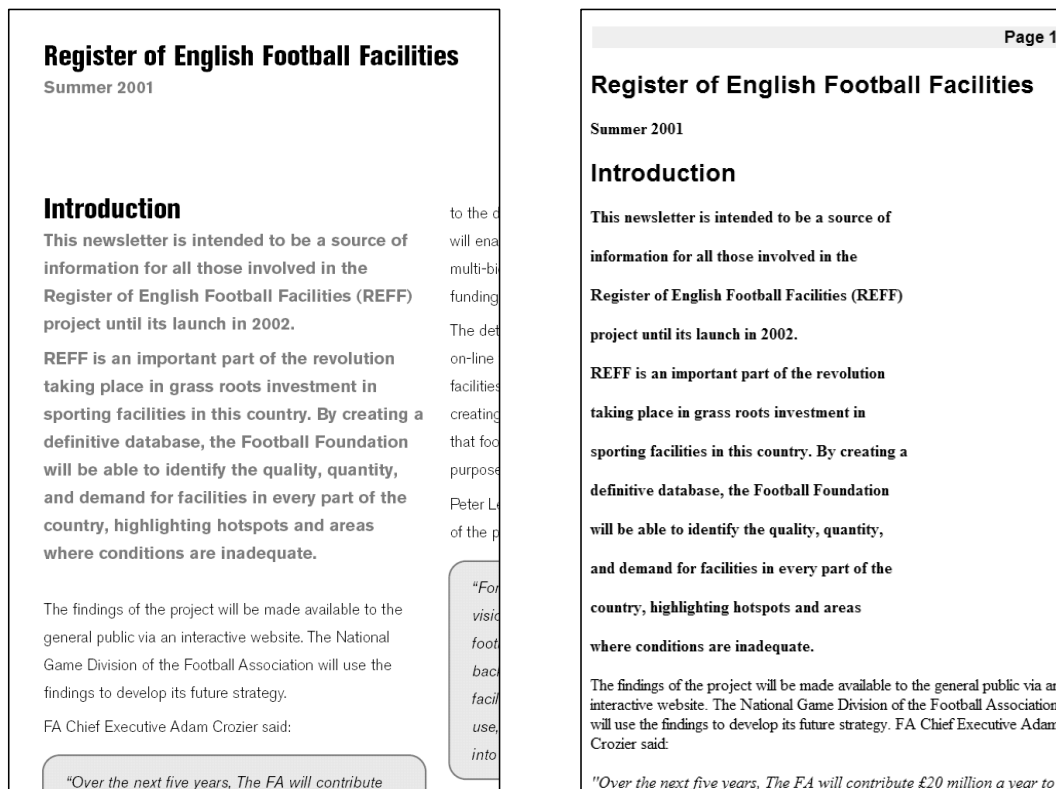


Fig 4.12: Line spacing not correctly detected for introductory text, which is slightly larger than main body text

The solution was to actively track the line spacing as the text fragments are being merged. The `findLineSpacing` method does this by taking the current fragment index and looking at the next two lines. It then returns the smallest of the spaces between the two lines. As the diagram shows below, this ensures that the correct line spacing is always returned.

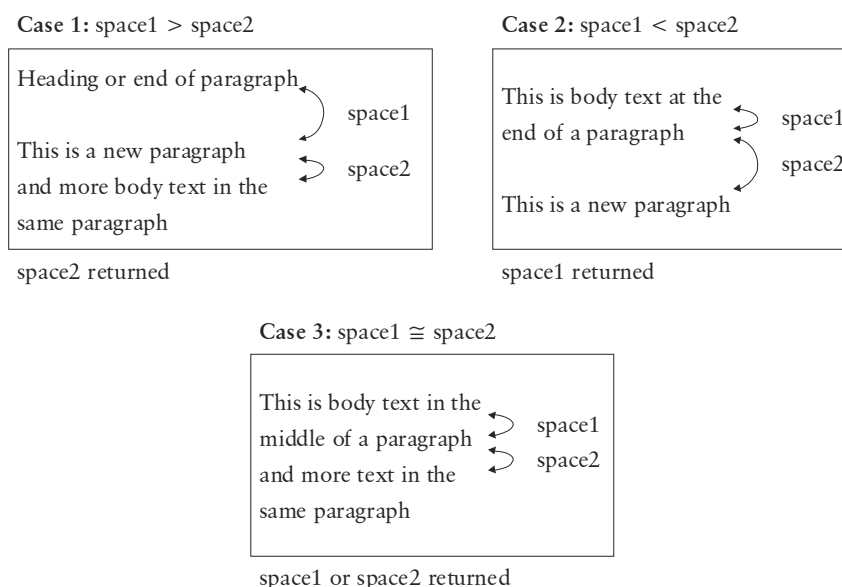


Fig 4.13: Detection of line spacing from the two following lines

If the line spacing is greater than 2.5 times the line height or less than half the line height the line height is instead returned as the line spacing. This is to stop incorrect values due to the line spacing being judged between two lines in different parts of the page or in different columns. If `space1` or `space2` cannot be evaluated, as is the case at the very bottom of the page, the line height is also returned.

The `mergeTextObjects` method keeps track of the current line spacing and recalls the `findLineSpacing` method to obtain a new value every time a new paragraph or column is encountered.

In addition, every time a new line in the PDF is encountered (i.e. `nextLine` evaluates to true) the method `changeLineSpacing` is called to detect whether the line spacing has changed. `changeLineSpacing` returns true if the gap between the two lines is too small, i.e. less than 70% of the current line spacing. This ensures that, if the line spacing is reduced (for example, double spaced text becomes single spaced), the line spacing continues to be tracked correctly, ensuring that new paragraphs are correctly detected.

#### 4.5.2 Styles and formatting

Although JPedal provided access to all the font information in each text fragment, it was decided to use “standard” font families, Times and Helvetica, in order to allow the HTML to display properly on systems that did not have the fonts that were used to create the PDF. Depending on the system, these fonts, or their known equivalents, are available either in bitmap form or as extensively hinted vectors to generate a clear, easy-to-read image at typical screen resolutions.

Two distinct HTML styles were used, `h1` and `p` (standard paragraph), which were chosen based on the size of the text in the PDF. These styles are included in an internal CSS *style sheet* in the header of the HTML file. The threshold between the two sizes was set at 14 points; any text over 14 points in size is therefore classified as a heading. An example of the output of both styles is shown below.

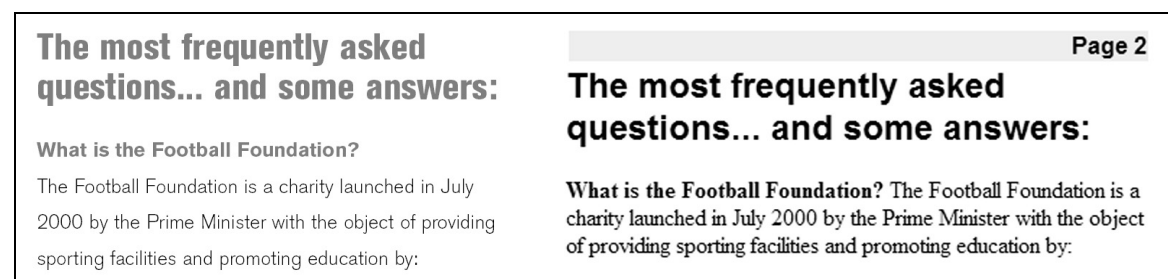


Fig 4.14: Example of formatting information (left) preserved in the HTML conversion (right)

Additional formatting information was retained in body text. JPedal gives the name of the entire font as a single string; if this string included “Bold” or “Italic” the `<B>` and `<I>` tags were used to change the appearance of the text. While sub-headings, which may be under 15 points in size, were sometimes recognized as body text, the bold attribute was still preserved, giving an appropriate appearance in HTML.

This method was found to detect formatted text in most cases, although it did not work for fonts that use a synonym to describe their formatting, such as “Cursive” for italic. It also does not work on Multiple Master fonts which have a continuously variable weight instead of the usual denominations such as “Light”, “Book”, “Medium”, “Bold”, etc.

HTML provides a much wider range of heading styles, and one possible improvement would be to detect various heading levels in a document. As this was not the main focus of this project, this remains as a suggestion for further development.

Formatting information is detected by the `findHeadings` method which works through each text fragment in order, removing all formatting information from the `contents` attribute and setting the boolean `isHeading[]`, `isBold[]` and `isItalic[]` attributes to true or false as appropriate.

These attributes are added to the HTML during merging of the text fragments as this gives an opportunity to ensure that tags are not unnecessarily repeated and that each opening tag (e.g. `<P>`) is met with its respective closing tag (e.g. `</P>`).

### 4.5.3 Symbols

Certain symbols, including typographical quotation marks (e.g. “ and ”) and dashes, were found to display correctly in the Windows environment but not on other systems using other character sets. As HTML is a platform independent format it was decided to replace typographical quotation marks with straight ones and dashes with hyphens with a space on either side. This was performed in the `replaceSymbols` method in the `pdf2html` class.

'Halloa!' the guard replied.	'Halloa!' the guard replied.
'What o'clock do you make it, Joe?'	'What o'clock do you make it, Joe?'
'Ten minutes, good, past eleven.'	'Ten minutes, good, past eleven.'
'My blood!' ejaculated the vexed coachman, 'and not atop of Shooter's yet! Tst! Yah! Get on with you!'	'My blood!' ejaculated the vexed coachman, 'and not atop of Shooter's yet! Tst! Yah! Get on with you!'

*Fig 4.15: Straight quotation marks (left) and typographical quotation marks (right)*

#### 4.5.4 Hyphenated text

As this conversion method creates text that can be re-flowed, words that were hyphenated in the PDF often appear, in the output, at the beginning or in the middle of a line. As newspapers and books contain a large number of hyphenated words it was decided to merge them into single words to make the text more readable.

The method `isHyphenated` takes the indices of two successive text fragments and returns true if:

- the current fragment begins with a letter and
- the previous fragment ends in a letter followed by a hyphen.

The method also strips the hyphen from the end of the previous fragment.

This method is used in the `mergeTextObjects` method at the stage where `nextLine` evaluates to true. If `isHyphenated` evaluates to true the text fragments are simply concatenated, without a space in between.

#### 4.5.5 Indentations

Paragraphs are typically denoted in two ways; either by a larger line space or by an indentation. Some documents even use both.

There is, however, no proper way of performing an indent in HTML and the `<TAB>` tag is ignored by most browsers. It is possible to include a HTML paragraph style to indent the first line of each paragraph but this method always leaves a line space at the end of the paragraph. This method was found to be unsuitable for texts such as the Dickens, where paragraphs can be shorter than one line in length.

The solution was to use four spaces, `&nbsp;&nbsp;&nbsp;&nbsp;`, to simulate an indentation. Although this is a “trick” and therefore not recommended HTML practice, it was the only way to obtain the required result.

Indentations are detected at the merging stage. As each text fragment is processed, the variables `left_margin` and `right_margin` are updated to keep track of the left and right margins of the text. If a new line is encountered that is more than 6 points from the margin it is recognized as an indentation.

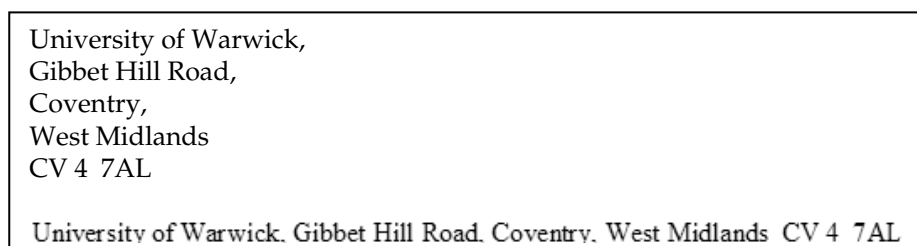
Every time a new column is encountered the margins are reset to allow for the different horizontal position of the new column.

Dropped capitals caused a problem with this method as text adjacent to them is always at a distance from the margin. This was solved by using another variable, `indent_guard`, which is updated in the same way as `left_margin`. However, when two text fragments of differing sizes are encountered, the `indent_guard` is set to the `x1` position of the rightmost fragment, ensuring that any text adjacent to a dropped capital isn't recognized as being indented.

This method was found to give good results. Its only shortcoming is that it cannot detect indented text at the top of the page as, at that stage, it has not processed the text further below to find the left margin.

#### 4.5.6 Forced carriage returns

Forced carriage returns are carriage returns that are included not as a result of word wrapping, but to enforce a particular layout, for example with an address. By default, the paragraph detection algorithm was found to merge each line of the address into a single line as shown below.



```
University of Warwick,  
Gibbet Hill Road,  
Coventry,  
West Midlands  
CV 4 7AL  
  
University of Warwick. Gibbet Hill Road. Coventry. West Midlands CV 4 7AL
```

*Fig 4.16: Address merged into a single line*

It is not always possible to tell whether a carriage return is forced or a result of word wrapping. However, when a line is much shorter than the width of the page, it is usually as a result of a forced carriage return (or being the last line in a paragraph).

Different methods of detecting forced carriage returns were used on single- and multi-column layouts. With single-column layouts the width of the entire text across the page was calculated by the `findTextWidth` method. Multi-column layouts are more complex and the text width only for a particular column was sought. As the left and right margins are already being tracked during the merging process (see section 4.5.5) the text width can simply be calculated by deducting the left margin from the right margin.

A forced carriage return is detected if the current line was less than 70% of the full text width, or 60% of the column text width for multi-column layouts. The smaller figure was used for multi-column layouts as they usually have fewer words per line, increasing the probability of false positives. This detection occurs in the `mergeTextObjects` method when a new line in the same paragraph is reached (i.e. `nextLine` evaluates to true).

#### 4.5.7 Raised and dropped capitals

Raised and dropped capitals were catered for by altering the `findHeadings` method to assign the capital the `p` (normal paragraph) style even though the character is usually over 14pt in size. Raised and dropped capitals are detected by a text fragment containing one character followed by another text fragment on the same line using the `sameLine` method. As the `sameLine` method evaluates to true whenever the coordinates intersect it will always be true for raised or dropped capitals.

Once the character is assigned the paragraph style it is successfully merged into the following paragraph by the `mergeTextObjects` method.

Dropped capitals also caused a problem with the algorithms used for the detection of indentations in text; these are documented in section 4.5.5.

#### 4.5.8 Miscellaneous text fragments

Miscellaneous text fragments include page numbers, copyright notices and other small text fragments that are not a main part of the page content itself. These objects often appeared in the middle of the flow of text, usually between two different columns, interrupting the flow of the text. It was therefore decided to attempt to detect these items and move them to the bottom of the page.

Miscellaneous items are detected by the `isMiscellaneous` method which returns true if the item is not within 24 points of the boundary of any other text block, or if it includes vertical-running text.

After each text fragment has been assigned a `group[]`, the `processPageFragments` method creates a new group and assigns all miscellaneous fragments to it. This ensures that, when the items are all sorted by group, the miscellaneous items will be moved to the bottom of the page.

### 4.5.9 Empty text fragments

Empty text fragments are fragments that do not contain any text, or a string of spaces. These are sometimes inserted by word processors between paragraphs, for example. Although these fragments do not display any text, their existence could cause problems with the paragraph and column detection methods.

They are therefore detected by the `isEmpty` method and removed from the sort order in the `processPageFragments` method, before any column grouping or merging takes place.

## 4.6 The front end

The front end was designed to take two command line parameters, an input file and an output file. If the second parameter is omitted it generates the output file name by removing the “.pdf” extension if there is one and appending the “.html” extension. If there are any errors with the parameters it displays an error message and exits.

The tasks performed by the front end are as follows:

- open input file
- get `PdfData` object
- enumerate pages in PDF file
- append header (including style information) to output string
- for each page:
  - append a “Page *x*” heading if there is more than one page
  - call `PdfGrouping.processPageFragments` and append the result
- append `</HTML>` to output string
- write output string to output file

### 4.6.1 The `-c` option

The `-c` option was included to instruct the converter to attempt to detect columns; without this option column detection is automatically turned off. This was because column detection occasionally detected false positives in single-column layouts, resulting in the text being output in the wrong order. In general, a better result can be obtained for single-column layouts by omitting the `-c` option.

This option is passed to `processPageFragments` as a boolean parameter, `detect_columns`, which determines whether `groupTextElements` will be called.

## 4.6.2 Multiple pages

The conversion software has been designed to convert each page independently from each other, as the process of correctly decoding the data from a single page had already presented a significant challenge. Each page is therefore decoded in turn. If there is more than one page in the PDF file, the front end adds a “**Page *x***” heading to the beginning of each page in the HTML output.

## 5 Performance evaluation

This section evaluates the results given by the conversion software. Annotated examples of the program's output from each of the four PDF files are given, together with an analysis of the results. These results are then compared to the results given by the existing solutions which are covered in section 2.

### 5.1 Analysis of converted output

Overall, the conversion software was found to produce good results with simple and moderately complex documents, including the four PDF files that comprise the conversion material chosen in section 3.3.

In most cases, simple multi-column layouts were correctly detected and the columns were output in the correct order. More complex layouts, such as the *US Military Force in Columbia* article in section 5.5, where an article spans two columns, gave less successful results. Occasionally, elements on the page such as headers, footers and captions, appeared in between the columns instead of being recognized as *miscellaneous* and moved to the bottom of the page. One major improvement, which is given as a suggestion for further work, would be to look at graphical elements of the page such as lines and rectangles. These elements often indicate to the reader where different articles start and end.

Most features of the page layout, as described in section 4.5, were detected and handled correctly. In particular, the line spacing detection algorithm worked very successfully and did not result in any unwanted new paragraphs. Some of the feature detection methods, such as the hyphenation detection, corrected some errors but caused others. This was because, where it met a double barrel word that was wrapped to the next line (such **horse-pistols** as described in section 5.3.2) it would still remove the hyphen and merge the two parts of the word into one. From the information available to the program it is impossible to tell whether the hyphen should remain in place or be removed. Only by understanding the text itself can this decision be made. One solution here would be to use a dictionary of hyphenated words, although this was beyond the scope of the project.

Similarly, the detection of *forced carriage returns* depended on the ratio of the line width to the text width. Therefore, forced carriage returns are not recognized in longer lines and are merged into a complete line of text. Again, without understanding

the meaning of the text itself it is impossible to tell whether a carriage return is forced or a result of word wrapping.

Ultimately, when a PDF is created from a word processing or desktop publishing package, some information is lost. Some of this information is required when converting the page to another layout. This project has attempted to intelligently “guess” some of this data and, based on these guesses, perform the conversion to a new layout. Because of this, it is highly unlikely that any program to convert PDF to HTML will ever be written to work perfectly.

## 5.2 Comparison with other methods

Although the results with the other converters were more accurate, the results with the *intelligent text extraction* method were far better in a practical sense. Text was always displayed at a readable size which could be resized by the user if necessary. The ability to re-flow the text also aided on-screen reading. In all cases the converted output more closely resembled a word processed document than the original, and one of the suggestions given for further work is to extend the converter to convert to RTF (Microsoft Rich Text) format.

In terms of web use, the files generated by this method are far more suitable for publication to a web site. In order to match the appearance of an existing web site, all that is necessary is to remove the *style sheet* from the header and replace it with a link to an existing CSS file on the web site. Alternatively, to use only part of the document, the relevant HTML can be copied out and pasted into another web page.

It is worth noting, however, that the method used here is far more complex and needs to be written specifically for each type of page layout. While the other converters will give acceptable results for almost any PDF document, a large number of PDFs will not work with this method at all. There are also cases where it is preferable to preserve the layout of the page, and use of the other method may be preferred for this very reason. Ultimately, the choice of method depends on the source files and the intended result.

## 5.3 Examples of converted output

Annotated examples of the converted output are shown overleaf, together with copies of the original PDF files. The simple layouts were converted with no command line options; the complex layouts were converted with the `-c` option.

### 5.3.1 Simple letter example

The diagram shows a letter with five numbered callouts:

- 1** points to the recipient's address: Tamir Hassan, Flat 3, 91 Victoria Terrace, Royal Leamington Spa, Warwickshire, CV 31 3AB.
- 2** points to the date: 25 April 2003.
- 3** points to the sender's address: Recruitment Unit, Acme Bank plc, 1 Canada Square, Canary Wharf, Docklands, London, E 14 1AA.
- 4** points to the subject line: **Re: Summer Analyst Scheme**.
- 5** points to the signature: Yours faithfully, Tamir Hassan, Encs.

The letter text is as follows:

Tamir Hassan  
Flat 3  
91 Victoria Terrace  
Royal Leamington Spa  
Warwickshire  
CV 31 3AB

25 April 2003

Recruitment Unit  
Acme Bank plc  
1 Canada Square  
Canary Wharf  
Docklands  
London  
E 14 1AA

Dear Sirs

**Re: Summer Analyst Scheme**

Please find enclosed signed copies of my contract, Declaration of Personal Interests form, Personal Details form, Medical Questionnaire and Equal Opportunities Monitoring form.

I have spoken to Maureen Chambers regarding tax documentation and have enclosed the P38 (Student Employees declaration) and P60 (End of Year Certificate 2002); this being the latest P60 that I have received.

Please could I ask that the P60 be returned to me, at the above address, as soon as possible as I will require it to reclaim tax from the Inland Revenue.

I have already provided a copy of my passport and have been advised that it is not necessary to include it here.

I hope that I have included all of the required documentation and look forward to receiving full confirmation of your offer of employment.

Yours faithfully

Tamir Hassan  
Encs

- 1 Forced carriage returns in the addresses have been detected and reproduced correctly.

The fact that one of the addresses was at a different horizontal position did not cause any problems.

- 2 Bold text detected and reproduced correctly

- 3 The extra line spaces after both these successive lines of text have caused the program to interpret this as double spaced text.

The two lines have not been merged into a single paragraph as the program has correctly detected forced carriage returns

- 4 New paragraphs correctly detected and reproduced

- 5 The large space after **Yours faithfully** is too large to confuse the line spacing detection algorithms. Therefore single line spacing is assumed and **Encs** is correctly positioned in a new paragraph

Tamir Hassan  
Flat 3  
91 Victoria Terrace  
Royal Leamington Spa  
Warwickshire  
CV 31 3AB

25 April 2003

Recruitment Unit  
Acme Bank plc  
1 Canada Square  
Canary Wharf  
Docklands  
London  
E 14 1AA

Dear Sirs

**Re: Summer Analyst Scheme**

Please find enclosed signed copies of my contract, Declaration of Personal Interests form, Personal Details form, Medical Questionnaire and Equal Opportunities Monitoring form.

I have spoken to Maureen Chambers regarding tax documentation and have enclosed the P38 (Student Employees declaration) and P60 (End of Year Certificate 2002); this being the latest P60 that I have received.

Please could I ask that the P60 be returned to me, at the above address, as soon as possible as I will require it to reclaim tax from the Inland Revenue.

I have already provided a copy of my passport and have been advised that it is not necessary to include it here.

I hope that I have included all of the required documentation and look forward to receiving full confirmation of your offer of employment.

Yours faithfully

Tamir Hassan

Encs

### 5.3.2 E-book example

*A Tale of Two Cities*

companions. In those days, travellers were very shy of being confidential on a short notice, for anybody on the road might be a robber or in league with robbers. As to the latter, when every posting-house and ale-house could produce somebody in 'the Captain's' pay, ranging from the landlord to the lowest stable non-descript, it was the likeliest thing upon the cards. So the guard of the Dover mail thought to himself, that Friday night in November, one thousand seven hundred and seventy-five, lumbering up Shooter's Hill, as he stood on his own particular perch behind the mail, beating his feet, and keeping an eye and a hand on the arm-chest before him, where a loaded blunderbuss lay at the top of six or eight loaded horse-pistols, deposited on a substratum of cutlass.

The Dover mail was in its usual genial position that the guard suspected the passengers, the passengers suspected one another and the guard, they all suspected everybody else, and the coachman was sure of nothing but the horses; as to which cattle he could with a clear conscience have taken his oath on the two Testaments that they were not fit for the journey.

'Wo-ho!' said the coachman. 'So, then! One more pull and you're at the top and be damned to you, for I have had trouble enough to get you to it!—Joe!'

10 of 670

---

*A Tale of Two Cities*

'Halloa!' the guard replied.  
 'What o'clock do you make it, Joe?'  
 'Ten minutes, good, past eleven.'  
 'My blood!' ejaculated the vexed coachman, 'and not atop of Shooter's yet! Tst! Yah! Get on with you!'

The emphatic horse, cut short by the whip in a most decided negative, made a decided scramble for it, and the three other horses followed suit. Once more, the Dover mail struggled on, with the jack-boots of its passengers squashing along by its side. They had stopped when the coach stopped, and they kept close company with it. If any one of the three had had the hardihood to propose to another to walk on a little ahead into the mist and darkness, he would have put himself in a fair way of getting shot instantly as a highwayman.

The last burst carried the mail to the summit of the hill. The horses stopped to breathe again, and the guard got down to skid the wheel for the descent, and open the coach-door to let the passengers in.

'Tst! Joe!' cried the coachman in a warning voice, looking down from his box.  
 'What do you say, Tom?'  
 They both listened.  
 'I say a horse at a canter coming up, Joe.'

11 of 670

- 1 Hyphenated text correctly detected and merged, although this does not take into account the context of the hyphenation.

In this case the word **horse-pistols** was not hyphenated due to word wrapping and should remain hyphenated even when the text is reflowed.

- 2 As there is no extra gap between paragraphs, the paragraph recognition fails to recognize the new paragraph.

Fortunately the indentation is recognized and preserved so that the reader identifies the following text as belonging to a new paragraph

- 3 Curly quotation marks replaced with straight ones and the dash is replaced with a spaced-out hyphen to allow for display on different character sets

- 4 Header and footer correctly detected as miscellaneous items and moved to bottom of page

- 5 As the first lines of the page are indented, the program does not know the correct margins of the text until it has already processed the indented lines. Therefore these lines do not appear indented in the conversion.

- 6 Indentations correctly detected and handled

**Page 10**

companions. In those days, travellers were very shy of being confidential on a short notice, for anybody on the road might be a robber or in league with robbers. As to the latter, when every posting-house and ale-house could produce somebody in 'the Captain's' pay, ranging from the landlord to the lowest stable non-descript, it was the likeliest thing upon the cards. So the guard of the Dover mail thought to himself, that Friday night in November, one thousand seven hundred and seventy-five, lumbering up Shooter's Hill, as he stood on his own particular perch behind the mail, beating his feet, and keeping an eye and a hand on the arm-chest before him, where a loaded blunderbuss lay at the top of six or eight loaded horsepistols, deposited on a substratum of cutlass.

The Dover mail was in its usual genial position that the guard suspected the passengers, the passengers suspected one another and the guard, they all suspected everybody else, and the coachman was sure of nothing but the horses; as to which cattle he could with a clear conscience have taken his oath on the two Testaments that they were not fit for the journey.

'Wo-ho!' said the coachman. 'So, then! One more pull and you're at the top and be damned to you, for I have had trouble enough to get you to it! - Joe!'

10 of 670

A Tale of Two Cities

---

**Page 11**

'Halloa!' the guard replied.

'What o'clock do you make it, Joe?'

'Ten minutes, good, past eleven.'

'My blood!' ejaculated the vexed coachman, 'and not atop of Shooter's yet! Tst! Yah! Get on with you!'

The emphatic horse, cut short by the whip in a most decided negative, made a decided scramble for it, and the three other horses followed suit. Once more, the Dover mail struggled on, with the jack-boots of its passengers squashing along by its side. They had stopped when the coach stopped, and they kept close company with it. If any one of the three had had the hardihood to propose to another to walk on a little ahead into the mist and darkness, he would have put himself in a fair way of getting shot instantly as a highwayman.

The last burst carried the mail to the summit of the hill. The horses stopped to breathe again, and the guard got down to skid the wheel for the descent, and open the coach-door to let the passengers in.

'Tst! Joe!' cried the coachman in a warning voice, looking down from his box.

'What do you say, Tom?'

They both listened.

'I say a horse at a canter coming up, Joe.'

11 of 670

A Tale of Two Cities

### 5.3.3 Simple newsletter example

**1** Register of English Football Facilities  
Summer 2001

**2** Introduction  
This newsletter is intended to be a source of information for all those involved in the Register of English Football Facilities (REFF) project until its launch in 2002.  
REFF is an important part of the revolution taking place in grass roots investment in sporting facilities in this country. By creating a definitive database, the Football Foundation will be able to identify the quality, quantity, and demand for facilities in every part of the country, highlighting hotspots and areas where conditions are inadequate.

The findings of the project will be made available to the general public via an interactive website. The National Game Division of the Football Association will use the findings to develop its future strategy.  
FA Chief Executive Adam Crozier said:

**3** *Over the next five years, The FA will contribute £20 million a year to the Football Foundation for its charitable work. Before new and improved community facilities can be provided it is essential to know what is already out there. That's why this project is so important.*

The project will be funded directly by the Football Foundation, with the backing of its funding partners the FA Premier League, The Football Association, Sport England and the DCMS. It will provide a comprehensive survey of the estimated 70,000 pitches in England to identify which are used for football and other sports, the numbers of games played and the quantity and quality of the other facilities on the site. The data compiled will lead

to the development of County Facility Strategies, which will enable the targeting of priority areas, encouraging multi-bids from local councils and County FAs, getting funding to where it is needed most.  
The details provided in the REFF Project will be available on-line for members of the public to find their local sports facilities and will be constantly revised and up-dated, creating a comprehensive and fully inclusive database so that football facilities can be better used for charitable purposes.  
Peter Lee, Chief Executive of the Football Foundation, said of the project:

**5** *"For the first time we will have a comprehensive vision of the state of the grass roots of community football. The REFF project will be the essential backdrop to all our work in transforming sporting facilities in our parks and schools for charitable use, enabling the Foundation to direct resources into areas crying out for support."*

The project is being delivered by a team of consultants led by PricewaterhouseCoopers and also including PMP Consultancy, a niche sports and leisure group whose staff and associates are based across England. More details of the key people on the ground are included in this newsletter.

**6**

**4**

**7**

**8**

**9**

**10**

**The most frequently asked questions... and some answers:**

**What is the Football Foundation?**  
The Football Foundation is a charity launched in July 2000 by the Prime Minister with the object of providing sporting facilities and promoting education by:

- putting in place a new generation of modern sporting facilities in parks, local leagues and schools
- providing capital and revenue support for the running of grass roots football and other sports
- strengthening the links between football and the community and to harness its potential as a force for good in society

The FA Premier League, The Football Association, Sport England and Government fund the Foundation to the tune of over £200 million over a four-year period.

**Is The Football Association involved in this project?**  
The FA is one of the funders of the Football Foundation and from the outset, contributed time and other resources to the Foundation's work by helping to develop the scope for the project, advising on the selection of the consultants and serving as a key member of the Steering Group guiding the work.

**Are you targeting particular areas?**  
The audit will cover the whole of England; there is no pre-conception that specific areas will be targeted for funding as a result of the work.

**How long will the project take?**  
We are currently talking to the various stakeholders, including local authorities, sporting and educational institutions, the MoD and other major landowners on which football facilities are located. Site visits and benchmarking must be completed by September 2001. Issues papers, drawing together the findings from the audit, will be prepared in September and October and distributed to interested parties. The completed Register and County Facility Strategies will be submitted to the Football Foundation by December 2001, with a public launch anticipated in the New Year.

**Why should owners of football facilities return the questionnaires? Many Leisure Departments in local authorities are overworked and underfunded.**  
The benefits of the project will be wide ranging and the Football Foundation hopes to be able to provide funding to a large number of community sport schemes. The data collected will complement the LA's own work on developing Asset Registers and Playing Pitch Strategies. We appreciate the time and budgetary constraints local authorities may have to work to and the REFF team will offer every assistance in the completion of the questionnaires.

**How long will the database remain relevant?**  
The database will be designed to allow constant updating and improvement. With up-to-date information, it should remain relevant and operable for the foreseeable future. The Football Foundation is appointing specific regional staff, part of whose role will be to ensure that the Register remains current.

- 1 Headings detected correctly
- 2 Line spacing correctly detected, although the text is larger (and therefore more spaced) than the body text
- 3 Curly quotation marks replaced with straight ones to allow for display on different character sets
- 5 Italic text correctly detected. Although paragraph is indented horizontally this does not cause a problem with the methods for column recognition or single-line indentations
- 6 Heading detected correctly and merged into single line
- 7 Bullets not recognized as they are graphics not symbols. Indentation of the paragraph, as before, does not cause a problem
- 8 New paragraphs correctly detected
- 10 Bold text correctly detected

Page 1

## Register of English Football Facilities

Summer 2001

### Introduction

This newsletter is intended to be a source of information for all those involved in the Register of English Football Facilities (REFF) project until its launch in 2002.

REFF is an important part of the revolution taking place in grass roots investment in sporting facilities in this country. By creating a definitive database, the Football Foundation will be able to identify the quality, quantity, and demand for facilities in every part of the country, highlighting hotspots and areas where conditions are inadequate.

The findings of the project will be made available to the general public via an interactive website. The National Game Division of the Football Association will use the findings to develop its future strategy. FA Chief Executive Adam Crozier said:

*"Over the next five years, The FA will contribute £20 million a year to the Football Foundation for its charitable work. Before new and improved community facilities can be provided it is essential to know what is already out there. That's why this project is so important."*

The project will be funded directly by the Football Foundation, with the backing of its funding partners the FA Premier League, The Football Association, Sport England and the DCMS. It will provide a comprehensive survey of the estimated 70,000 pitches in England to identify which are used for football and other sports, the numbers of games played and the quantity and quality of the other facilities on the site. The data compiled will lead

to the development of County Facility Strategies, which will enable the targeting of priority areas, encouraging multi-bids from local councils and County FAs, getting funding to where it is needed most. The details provided in the REFF Project will be available on-line for members of the public to find their local sports facilities and will be constantly revised and up-dated, creating a comprehensive and fully inclusive database so that football facilities can be better used for charitable purposes.

Peter Lee, Chief Executive of the Football Foundation, said of the project:

*"For the first time we will have a comprehensive vision of the state of the grass roots of community football. The REFF project will be the essential backdrop to all our work in transforming sporting facilities in our parks and schools for charitable use, enabling the Foundation to direct resources into areas crying out for support."*

The project is being delivered by a team of consultants led by PricewaterhouseCoopers and also including PMP Consultancy, a niche sports and leisure group whose staff and associates are based across England. More details of the key people on the ground are included in this newsletter.

Page 2

### The most frequently asked questions... and some answers:

**What is the Football Foundation?** The Football Foundation is a charity launched in July 2000 by the Prime Minister with the object of providing sporting facilities and promoting education by:

- putting in place a new generation of modern sporting facilities in parks, local leagues and schools providing capital and revenue support for the running of grass roots football and other sports strengthening the links between football and the community and to harness its potential as a force for good in society

The FA Premier League, The Football Association, Sport England and Government fund the Foundation to the tune of over £200 million over a four-year period.

**Is The Football Association involved in this project?**  
The FA is one of the funders of the Football Foundation and from the outset, contributed time and other resources to the Foundation's work by helping to develop the scope for the project, advising on the selection of the consultants and serving as a key member of the Steering Group guiding the work.

**Are you targeting particular areas?** The audit will cover the whole of England; there is no pre-conception that specific areas will be targeted for funding as a result of the work.

**How long will the project take?** We are currently talking to the various stakeholders, including local authorities, sporting and educational institutions, the MoD and other major landowners on which football facilities are located. Site visits and benchmarking must be completed by September 2001. Issues papers, drawing together the findings from the audit, will be prepared in September and October and distributed to interested parties. The completed Register and County Facility Strategies will be submitted to the Football Foundation by December 2001, with a public launch anticipated in the New Year.

**Why should owners of football facilities return the questionnaires?** Many Leisure Departments in local authorities are overworked and underfunded. The benefits of the project will be wide ranging and the Football Foundation hopes to be able to provide funding to a large number of community sport schemes. The data collected will complement the LA's own work on developing Asset Registers and Playing Pitch Strategies. We appreciate the time and budgetary constraints local authorities may have to work to and the REFF team will offer every assistance in the completion of the questionnaires.

**How long will the database remain relevant?** The database will be designed to allow constant updating and improvement. With up-to-date information, it should remain relevant and operable for the foreseeable future. The Football Foundation is appointing specific regional staff, part of whose role will be to ensure that the Register remains current.

- 4 Columns output in correct order but a new paragraph is incorrectly inserted where the paragraph flows from one column to the next
- 9 Forced carriage returns not always detected; this is not always possible. In these two cases the line width is approaching the text width



in the informal economy or in a textile assembly maquila, an industry that exploits Nicaragua's "comparative advantage": cheap labor

**This little bean will become the greatest threat that Nicaragua will ever face. - economist Alvaro Fonseca**

provided by a desperate workforce. And, of course, another option would be emigrating to Costa Rica or the United States in search of enough income to send home to support her family. But in spite of all the negative effects for ordinary Nicaraguans like Genara, the U.S. government continues to push to formalize free trade agreements that will impact countries like Nicaragua for years to come.

Do As I Say, Not As I Do

In February 2002 President Bush announced his plan to expand free trade throughout Central America under CAFTA, as a stepping stone to the hemisphere-wide Free Trade Area of the Americas (FTAA). Secretary of State Colin Powell clearly declared the intent of the FTAA to "assure for American corporations control of the territory that runs from the North Pole to the Antarctic free access without any hindrance or difficulty for our products, services, technology and capital through the hemisphere." Free trade agreements seek to open foreign markets by eliminating tariffs and "barriers" to trade, under the pretext of fair and equal competition in open markets.

While Bush and Powell began promoting CAFTA in Central America, Congress was beginning to debate this year's farm bill, which will give U.S. farmers \$100 billion dollars in subsidies over the next 8 years. Small farmers in both Nicaragua and the U.S. are losers in the current system. U.S. subsidized agriculture does not generally benefit small U.S. farmers either, as they receive a mere 16% of all subsidies, while they have to compete with the mega-production of the much more heavily subsidized corporate agribusinesses.

Thus, U.S. - grown products - especially those farmed primarily on large corporate farms - such as corn and rice, can be exported at artificially depressed prices to countries like Nicaragua. These prices do not reflect the real cost of production. "We would have to see if a farmer from the great prairies of the United States in his

In rural Nicaragua, the whole family participates in the cultivation of red beans.

air conditioned tractor would be capable of producing without subsidies at the same costs as farmers in the valley of Jalapa, Esteli, Muy Muy, or other rural areas of Nicaragua," challenges Alvaro Fonseca. As free trade policies become more comprehensive and are expanded throughout the hemisphere, in a country where the average annual family income is about \$430, Nicaraguan farmers will be forced to compete on an even larger scale with U.S. farmers who receive thousands of dollars per year in subsidies.

In the name of free market development, the United States has pushed to open foreign markets to U.S. exports and investment. Free trade agreements force countries like Nicaragua to lower tariffs and to eliminate subsidies and other "barriers" to trade. But, as Nicaraguan economist Carlos Pacheco points out, "The attitude of the United States towards free trade has been do as I say, not as I do."

Although the U.S. pledged to reduce agricultural subsidies during the last round of World Trade Organization negotiations, the recent Farm Bill flew in the face of that commitment. Furthermore, the recent increase represents just one of many that have occurred over the last decade, as the U.S. and other industrialized nations continue to increase their level of farm subsidies. Given the clout of the U.S. in the major International Financial Institutions, and given the large scale of the U.S. economy, the United States has not been forced to keep its own promises or follow its own rules. By subsidizing U.S. agriculture and maintaining some tariffs, the U.S. continues to protect itself from the international "barrier-free" competition it promotes and demands of its impoverished neighbors.

Genara is thankful that up until now her family has been able to survive by producing the food they need and selling some of their crop to local markets. "This year will be better than the last, and right now that's all we can hope for." Small farmers confront the challenges of Mother Nature to provide for their families and local communities. A strong local food economy has allowed small Nicaraguan farmers to get by, while the U.S. government's free trade agenda threatens the food security of Nicaragua, and the whole of Latin America.

Where would Genara turn without a local market for her beans? Failing farms will continue to produce the mass migration of unemployed workers to the cities and to the United States, hoping to support their families and communities with the hard won wages they send home.

Only the strong survive in export economies, and impoverished Nicaraguan farmers are not equipped with the necessary technology, financing, and protection needed during the transition to a new style of production to compete with larger economies. The logic of importing products such as Rojo Chiquito to Nicaragua that could be grown locally, puzzles professor Bayardo Ortiz. "Why bring beans from another place to replace our traditional bean? Why don't they help our farmers to plant the bean that our land has produced for centuries? Why would you bring them from another place when we have the seed and the land here? Why don't we grow our own beans so

continued on page 9

Melinda St. Louis

**Nicaragua**

## Notes

1. Titles detected correctly. As the converter does not distinguish between different heading levels, the same style is used for both headings
2. Dropped capital detected and handled correctly
3. This quotation is included in between the paragraphs, just like in the original. For on-screen reading it is usually more appropriate to include the quotation elsewhere on the page.
4. As this text uses a Multiple Master font it has not been possible to detect it as bold. This is described in detail in section 4.5.2
5. The footnote and picture caption have not been detected as miscellaneous items as they are very close to the main text
6. These miscellaneous items have been correctly detected and handled
7. As the space between this text block and the previous text block coincide with the space between each column, part of the text has been incorrectly detected as belonging to the next column
8. Despite there being very little extra spacing between paragraphs, these paragraphs have been correctly detected and handled
9. This sub-article caused problems as it spans over 2 columns. It is included after the first column of the first article and followed by the remaining columns