

Quality PDF Documents: Part I

by John Redmond

formatted December 3, 2009

Contents

Introduction	1
Overview of L ^A T _E X	1
Overview of Lexxia	1
Installing the Software	2
Getting Started	3
Adding Content	3
Structuring Content with Headings	3
Styles in PDF/Lexxia	4
The Document Basics	4
Including Postscript Fonts in L ^A T _E X	5
Lexxia Styles	6
Using Styles	7
Structured Content	9
Mixed Content	9
Lists	9
Tables	10
Images	12
Styling the Document	14
Document Type	14
The Document Options	14
Formatting the Page Headers	17
Formatting the Paragraphs	17
Hyphenation	17
Preparing Source Documents	18
Text Source	18
Whitespace	20
Line Termination	20
Markup	21
XML or XHTML Source	22
Getting Serious	23

Introduction

The intent of this document is to introduce the use of Lexxia to produce \LaTeX documents. It is certainly not a good example of a well-constructed document, simply because it is a *pôt pourri* of most of the styling options. Too many tricks mean bad taste!

The techniques illustrated require the following:

- A computer with a Linux or Windows environment;
- Lexxia for that environment (available without charge);
- A \LaTeX distribution for that environment (available without charge);
- A text editor for developing and editing content.

A \LaTeX distribution can be enhanced with specialist packages, but the techniques used in the present environment are available *as are* in any distribution.

Overview of \LaTeX

\LaTeX is a very big topic, and I must express my humility as I attempt to give an overview of a topic that remains, in very considerable measure, a source of wonder and mystery for me. Nevertheless, I am confident that the interface that Lexxia provides to \LaTeX is powerful, useful and extensible. Its great appeal, I hope, will be the simplicity of the way it creates quite clean and reasonably elegant \LaTeX files. And, because \LaTeX files are simply ASCII text files, they can be edited and refined to satisfy the requirements of the aficionado. For most users, on the other hand, it provides a direct route from text and XML files to elegant and highly presentable PDF documents.

Overview of Lexxia

At heart, Lexxia is a text processor, primarily an XML processor. It can be used for all the usual XML-type things, such as loading and editing documents and transforming them by XSLT, but it has recently come to focus particularly on the generation of high-quality PDF documents.

Before we proceed, I should note that the PDF format has become increasingly important in the general IT mainstream. That much is pretty obvious, but less obvious is the variation in quality of the PDF documents. It is now very easy to export PDF files from, say, a word processor or browser. This achieves the aim of a locked document, but does not necessarily lead to a good PDF document because the quality is limited by the quality of the rendering engine used by the word processor or browser (generally not especially good). So—there really is a need for better ways to build PDF!

Users of \LaTeX will attest that it stands alone as a means to quality PDF documents (and the associated .dvi and .ps documents). But the trouble is that creating the source documents can be rather intimidating. Lexxia aims to make the preparation of \LaTeX documents simple, adaptable and consistent.

Lexxia is a command-line program, and is available free of charge for Linux and Windows. It is very fast (on Linux, it will typically generate \LaTeX source for a 500-page book in well

under 1 second, depending on your hardware—although the Windows version is rather slower at present) and it requires no other software, apart from a \LaTeX distribution (which is readily available free for download—see below).

If we have a source text or XML/XHTML file and we wish to convert it to a \LaTeX file. This takes a single Lexxia command, such as:

```
lexxia source/grimm.txt -Lgrimm.css+result/*.tex
```

Here we use the *grimm.txt* source file from the *source* directory and process it, using additional specifications in the external *grimm.css* stylesheet in the current directory and output the resulting *grimm.tex* in the *result* directory. (Note the use of the * wildcard here; it is a great help for processing multiple input files in a single command line.)

The resulting \LaTeX document is then processed in the usual way:

```
pdflatex result/grimm.tex
```

This command line is usually entered twice, in order to generate a table of contents. The result is a PDF file, *grimm.pdf*, in the current directory. It is as easy as that!

Gotcha 1: This commandline is for a Linux environment when \LaTeX has been *installed*.

Gotcha 1a: For Windows users: your system uses file paths with backslashes. Be sure to use this form, even though some variants of Windows *do* recognise forward slashes.

Furthermore, you may prefer *not* to install either the \LaTeX distribution or Lexxia. This means that you will have to use the full file paths (more below).

Installing the Software

Lexxia is available from my site (www.limpidsoft.com) and \LaTeX is available from many different sites (look for a *TeX Live* package). I suggest that, at least at first, you *not* install either of these. Move Lexxia to your home directory (Linux) or to C: (Windows). When you download \LaTeX , you will probably be asked whether to install it (in Program Files). In Windows, I suggest that you *not* install, but instead copy the (very large) package to C:; in Linux, it is probably best installed (usually to /usr/bin or /usr/local/bin—while you are superuser).

To jump ahead just a little, these choices lead to the following \LaTeX command lines (for the *grimm.tex* file):

In Windows, something like:

```
\texlive\bin\win32\pdflatex grimm
```

and in Linux (because the \LaTeX package has been installed):

```
pdflatex grimm
```

Getting Started

Adding Content

Without this, we go nowhere. In your editor or word processor, develop your content. (If you prefer, you can use a word processor, but be sure to *save as text*. At this stage, give no thought to how the content is to be displayed, with one possible exception: organise your text into paragraphs. For our purposes, a paragraph consists of a sequence of text *without a break from the ENTER key* (just allow the editor to wrap the lines). I find that it helps to separate the paragraphs by one or more blank lines, to give a sense of structure, but this is not necessary.

Now save the content to a file. How you name it is not important, provided that it has the ".txt" extension. This provides Lexxia with a clue as to its physical structure. Assume that the file is "myfile1.txt", and we are to process it with Lexxia:

```
lexxia myfile1.txt -L+result/*.tex
```

Enter this command exactly as shown: the "+" character after the "-L" is important, as we shall see. This command will generate a new file (myfile1.tex) in the result directory. (Windows users: remember to use backslashes!)

We now have a \LaTeX document ready for processing with the pdflatex program, as outlined above (the .tex extension can be ignored):

```
pdflatex myfile1
```

or, in Windows

```
\texlive\bin\win32\pdflatex myfile1
```

At this point, the resulting PDF file will be less than exciting: a wasted page or two at the start, followed by the text from the input text file. But you might note a couple of points:

- The paragraphs *are* depicted pretty much as you expected;
- There is a slight additional space between some of the paragraphs (this is the \LaTeX parskip, which can be controlled);
- For each line in your source, you will get a separate paragraph;
- There are indentations at the starts of paragraphs in the PDF document, but *not* the first paragraph (this is North American best practice—again controllable in \LaTeX).

Structuring Content with Headings

In any non-trivial document, we expect to see the text broken up with headings. Our attention is generally drawn to these by *styling* with increased character size and shape. We would use these features to indicate *levels* of headings. With Lexxia, we simply add one or more dots (full stops or periods) at the starts of the heading lines:

- .Heading Level 1 (Typically the document title)
- ..Heading Level 2 (Typically a chapter or section title), etc.

Add headings to your paragraphs text and process as before. In the internal XML document generated by Lexxia, real structure is added by these headings: each time a dotted heading is encountered, a new block is created and the heading and subsequent content are added to it until another block, or the end of the file, is encountered. Furthermore, there are automatic mechanisms to control how and where these blocks are organised: a level-3 block, for instance, can never be placed inside another level-3 block, while it can, and must, be inside a level-2 block—or, occasionally, directly inside a level-1 block.

You may now be inquisitive enough to see what the internal XML document looks like (you can do this at any time). Use the Lexxia output (-O) command:

```
lexxia myfile1.txt -O
```

This command writes to the standard output a standard XML representation of the internal document. If you want to write it a file (here, myfile1.xml) use:

```
lexxia myfile1.txt -O*.xml
```

The structure that you observe (such as the h2 element inside an L2 block) really *is* important, because it creates distinctive environments that can be identified and styled distinctively.

After you have added some headings, process your source file again:

```
lexxia myfile1.txt -L+result/*.tex
pdflatex myfile1
pdflatex myfile1
```

The PDF document should now show the headings that you have created, styled distinctively, and page headings corresponding to these. (To avert confusion: pages have headers and footers; text may have some interspersed headings in the body of the page.)

Styles in PDF/Lexxia

Almost all aspects of Lexxia's default style are controlled by an internal CSS-style document. In this section, we examine some of the specifications in some detail. (In devising this approach, I have attempted to keep to the style and general flavour of the familiar CSS stylesheets for XML/HTML.)

The Document Basics

These are specified in a single definition:

```
document { class: article; format: portrait; font: palatino; text-align: full; toc-columns: 1;
toc-page-numbering: roman; body-columns: 1; page-numbering: arabic; title: true; title-page:
true; date: formatted; pdf: true; table-style: tabularx; table-placement: p; }
```

Here we define the general shape and appearance of the document without any reference to specific components of it. Much of it should be obvious:

document: the default is 'article'; often 'book' is more appropriate.

format: the default is 'portrait'; otherwise 'landscape'.

font: default is 'palatino'; you might try 'times', 'chancery', 'courier', 'helvetica', 'avantgarde', 'charter'. Changing font size and line spacing comes later.

text-align: 'full' (full justification with hyphenation) is default; 'left' and 'right' are possible.

toc-columns: The number of columns for the table of contents: generally '1' or '2'; '0' means no table of contents.

toc-page-numbering: The numbering of the pages in the table of contents; possible are 'arabic', 'roman' and 'Roman'.

body-columns: The number of columns for the body of the document; defaults to '1'; '2' is sometimes useful.

page-numbering: The numbering of the main body of the document;

title: Set to 'true' to display a title;

title-page: Set to 'true' to display the title on a separate page.

date: In the event of a title, insert this text before the actual date (here it is "formatted").

pdf: Set to 'true' to include pdf links;

table-style: Set to "tabularx" for display of tables; "tabulary" is possibly better, but is typically not included in L^AT_EX distributions;

table-placement: the options are 'h' (here), 't' or 'b' (top or bottom of current page) and 'p' (on a special page); The default is 'p'. This is discussed below.

Including Postscript Fonts in L^AT_EX

It is not necessary to specify any fonts in L^AT_EX, because the default *Computer Modern* fonts are included automatically. But *Postscript* fonts are becoming increasingly attractive because they are generally familiar and scalable. Lexxia provides an easy option to use the Postscript New Font Selection Scheme (PSNFSS) for using the Postscript fonts that you already have on your computer. By default, the following specifications are included in your L^AT_EX preamble:

```
fonts { times: 9/10.5, ptm, m, n;
bookman: 10/11.5, pbk, m, n;
charter: 10/11.5, pch, m, n;
palatino: 10/11.5, ppl, m, n;
avantgarde: 9/10.5, pag, m, n;
chancery: 9/10.5, pzc, m, il;
courier: 9/11, pcr, m, n;
helvetica: 8/9.5, phv, m, n; }
```

Palatino is the default font in Lexxia (simply because I like it!). Note how it is defined in the above specification:

Name: palatino

Font Size: 10pt
Line Spacing: 11.5pt
PSNFSS CodeName: ppl
Font Weight: m (medium)
Font Shape: n (normal)

As we shall see, any of these (or other) fonts can be manipulated by changing their size and shape (such as small, large, Large, italic, sloped and small-caps). I continue to be surprised how rarely I need to do this, as so much of what \LaTeX does is automatic.

Lexxia Styles

Style definitions in Lexxia are of the following general form:

```

featurestyle { define: style; font-weight: bold; text-align:left; }
emphstyle { define: style; font-weight: normal; font-shape: slanted; }
capsstyle { define: style; font-weight: normal; font-shape: small-caps; }
subtitledstyle { define: style; environment: quote; font-weight: bold;
font-shape: italic; }
commentstyle { define: style; environment: quotation; font-size: small; font-weight: normal; }
quotestyle { define: style; environment: quotation; font-weight: normal; font-shape: italic; }
  
```

Styles in Lexxia have the most influence on the appearance of the final document. These definitions are for the default styles in Lexxia. But, before we examine them in detail, we should look at the restrictions that \LaTeX places on command names (because these style definitions are destined to be converted to \LaTeX commands). The rules are:

- The command name must consist solely of alphabetical characters (traditionally these are all lower-case, but many extension packages do include upper-case characters), but *not* digits;
- Or the command name can consist only of non-alphabetical characters, (such as `_` and `.`);
- The names of the styles *do not* need to end in “style”, but I have found the convention useful because it avoids confusion with the names of XML tags—and the possibility of collision with inbuilt \LaTeX command names.

To define a style in Lexxia, we use the syntax:

```
stylename { define: style; ..... }
```

In the definition, we specify:

environment: such as quotation and verse.
font: any of the fonts that are defined as above, eg ‘charter’
font-size: typically large, Large, small

font-weight: normal or bold
font shape: such as italic, slanted and small-caps
color: the color of the font (try “red”, “blue”, etc)

The effects of most of these specifications are pretty obvious, but the environments deserve special mention. All environments share one characteristic: they separate the enclosed text vertically from the preceding and following content and often left and right indenting. Great for featured content, such as the font detail above.

One of the default Lexxia styles is indentstyle:

```
indentstyle { define: style; environment: quotation; font: helvetica; font-size: small; text-align: left; }
```

It sets the font to helvetica, reduces the text size, aligns the text to the left and displays it in a block with left and right indenting (because of the quotation environment). This style has been used consistently in this document to display technical details. Moreover, it can be used for a single string of text or a structured block, such as description (property) list. But we are getting ahead of ourselves. We need to discuss some of the structures (of XHTML and \LaTeX).

Using Styles

There is a set of predefined tag names in Lexxia, and each has an associated, default style. These are specified in the default stylesheet; examples are:

```
subtitledstyle { define: style; environment: quotation; font-weight: bold; font-shape: italic; }
commentstyle { define: style; environment: quotation; font-size: small; font-weight: normal; }
notestyle { define: style; font-size: small; font-shape: slanted; }

author, subtitle { style: subtitledstyle; }
comment { style: commentstyle; }
note { style: notestyle; }
```

Note how a style is defined with the “define” specification and the actual tag name (such as “comment”) is linked to the style (“commentstyle”) with a “style” specification.

We can alter all or any of the style specifications from an optional external stylesheet, so that, for example, the contents of a comment tag could be styled in normal size and small-capitals shape by adding the following declaration to an external stylesheet:

```
commentstyle { font-shape: small-caps; font-size: ""; }
```

This illustrates the setting of the font-shape specification (which had been the default shape because it had not been set in the earlier definition of commentstyle) and cancelling the earlier font-size specification by setting its value to an empty string with “”. (This means that the font size is set at the default size.) There is no specification for an environment, which means that there is no indenting or padding with extra space above and below the text. Contrast this with the definitions of subtitledstyle and commentstyle, each of which has a “quotation” environment to set the enclosed text out among the general text.

(Incidentally, there is a \LaTeX *quote* environment that indents in the same way as ‘quotation’. It differs in the way in which the included paragraphs are formatted: quote indents, while quotation adds space between the paragraphs.)

We now have enough information to design our own (rather silly) style and use it for the default (p) tags:

```
pstyle { environment: quote; font: chancery; font-size: large; color: green; text-align: right; }
p { style: pstyle; }
```

Alternatively, we could have styled these tags directly:

```
p { style: ""; environment: quote; font: chancery; font-size: large; color: green; text-align: right; }
```

To make this work reliably, it is a good idea to cancel any style setting for the tag (by setting it to “”), because there might have been an earlier setting for the style. This illustrates an important point:

Regardless of any specifications for a tag, they are ignored if there is a style setting.

For this reason alone, it is good policy to define a non-trivial style and use it for one or more tags in your source document. But there is another reason, which is best illustrated by an example:

The source text consists of (say) three simple paragraphs

Note how a style is defined with the “define” specification and the actual tag name (such as “comment”) is linked to the style (“commentstyle”) with a “style” specification.

Note how a style is defined with the “define” specification and the actual tag name (such as “comment”) is linked to the style (“commentstyle”) with a “style” specification.

Note how a style is defined with the “define” specification and the actual tag name (such as “comment”) is linked to the style (“commentstyle”) with a “style” specification.

If we have defined a style for the ‘p’ tag, this is styled in the \LaTeX document as

```
\pstyle {Note how a style is defined with the “define” specification and the actual tag name
(such as “comment”) is linked to the style (“commentstyle”) with a “style” specification.}
```

```
\pstyle {Note how a style is defined with the “define” specification and the actual tag name
(such as “comment”) is linked to the style (“commentstyle”) with a “style” specification.}
```

```
\pstyle {Note how a style is defined with the “define” specification and the actual tag name
(such as “comment”) is linked to the style (“commentstyle”) with a “style” specification.}
```

Here, we have styled a string with a specially designed \LaTeX command (pstyle). The fact that there is a command is unobtrusive, while the text content is set out clearly. If we use direct styling, the \LaTeX document contains:

```
\begin{quote}\chanceryfont \large \color{green}\raggedleft Note how a style is defined with
the "define" specification and the actual tag name (such as "comment") is linked to the style
("commentstyle") with a "style" specification.\end{quote}
\begin{quote}\chanceryfont \large \color{green}\raggedleft Note how a style is defined with
the "define" specification and the actual tag name (such as "comment") is linked to the style
("commentstyle") with a "style" specification.\end{quote}
\begin{quote}\chanceryfont \large \color{green}\raggedleft Note how a style is defined with
the "define" specification and the actual tag name (such as "comment") is linked to the style
("commentstyle") with a "style" specification.\end{quote}
```

Direct styling clutters the text content, making it difficult to read—and maintain. Better avoided, except for possible isolated cases. This is in keeping with the established L^AT_EX concept of “literate programming” (make it elegant and easy to read and understand).

Structured Content

Mixed Content

In writing the source text for this document, *I have used the XHTML technique to emphasise words*. The preceding sentence was input as

```
In writing the source text for this document, <em>I have used the <em>XHTML
technique</em> to emphasise words</em>.
```

It will surprise noone that this line is just a snippet of XHTML syntax, but why change it when it works? The *em* tag is part of the mixed content for the sentence and translates directly to the `\emph{}` command of L^AT_EX. This is the preferred method of emphasis for the printed page, whereas bold text is perhaps more often seen in web pages. In L^AT_EX, emphasis can be nested (as above), with the result that emphasis inside emphasis results in plain style. This is not often required, but it looks consistently good on the printed page.

The alternative of using *bold* type for emphasis generally does not look good on the printed page, as in:

```
In writing the source text for this document, I have used the XHTML technique to emphasise
words.
```

Lists

There is a simple correspondence between XHTML and L^AT_EX lists:

- An unordered list (ul) is converted to an *itemize* structure;
- An ordered list (ol) is converted to an *enumeration* structure;
- An definition list (dl) is converted to an *description* structure;

The first two types are very simple in Lexxia: for this list, simply wrap the lines in a list:

_ul

An unordered list (ul) is converted to an `itemize` structure;

An ordered list (ol) is converted to an `enumeration` structure;

An definition list (dl) is converted to an `description` structure;

—

while a definition list uses a vertical bar to separate the definition term from the data

_dl

term|definition of the term

—

If you know anything about XHTML, you will recognise that lists should contain “li” tags, while here we have unspecified text, which is spread over multiple lines. The Lexxia parser knows what is needed, and converts the “p” tags to “li” tags. The “p” tags inside the definition list are also recast automatically to give the correct XHTML content, with the definition term before the bar character.

Tables

A table is easy to define (it is very similar to a definition list, in that the separation into columns uses the bar character).

_table

Heading 1|Heading 2|heading 3|Heading 4

Content|More content|Longer content|The longest content of all in this column

Content|More content|Longer content|The longest content of all in this column

Content|More content|Longer content|The longest content of all in this column

Content|More content|Longer content|The longest content of all in this column

Foot 4|More Foot|Foot 3|Foot 4

—

The processing of tables is more problematic than lists:

- There are different styles for representing tables, notably *array*, *tabular*, *tabularx* and, wait for it! *tabulary*. The fact that there are different styles for the same sort of structure suggests, correctly, that there is no *one* best way. I have been happy with *tabularx*, in the way it allocates table widths for columns with differing amounts of text content, but *tabulary* is cleaner in the way it handles wide columns (see Part 2 for examples).
- The text alignment within the columns needs to be considered. Unless otherwise specified (see below), alignment of all columns defaults is to the left.
- Tables can be very large and difficult to relate to the general flow of text—and they are likely to overflow the page. It is therefore pretty standard practice to wrap a table in a float (this float is called “table”), which means that L^AT_EX decides where best to place it. (This may be disconcerting at first, but L^AT_EX provides ways of referring to a table that is *just somewhere* in the document.)

- A table may be framed. This is probably instinctive to writers of web pages, but printing best practice seems to suggest otherwise.

```
_table
@cols=rcl
```

- At the XML level, this attaches a `cols="rcl"` attribute to the `table` element. (This is a reminder that we really *are* creating an XML document.)
- This option, which is likely to be different for each table in a document, gives good control over the format of a table. The alignment characters are, pretty obviously, l, c and r for left, center and right alignment; and they apply to the column headers and footers (if any) as well as the column items themselves. To accommodate line wrapping for wide columns, select X. You can specify where vertical bars between any of the columns, such as with `r|cc|l`. Table 1 on page 15 illustrates the use of vertical bars and styled headers and footers (it is deliberately somewhat odd, just to illustrate the options).
- When a table is floated, L^AT_EX takes on the responsibility of placing in the finished document. You can attempt to influence the placement in the document specification:

```
document { table-placement: p; }
```

- The placement options or, more accurately, requests are:
 - h: here (in the document stream);
 - t: top (of the current page);
 - b: bottom (of the current page);
 - p: page (on a special page of L^AT_EX's choosing).
- The table placement specification is part of Lexxia's *global* policy for tables: they are all floated, placed, centered horizontally and either framed or unframed, according to your (global) choice. (A global policy means that they have a consistent style and are located in the final document in a consistent way. To set them on an irregular and inconsistent basis would be confusing for the reader.) A caption (title) and label can (and should) be attached to a table, so that it can be recognised and referred to from the text.
- How best to decide on table placement can be tricky—and somewhat personal. My impression is that, with a small number of small tables, the *t* placement works best for the reader, while many tables and large tables are best with *p* placement.
- Lexxia defines (in the default CSS stylesheet) a custom length (tablewidth) that specifies the width of *all* tables in the document (remember that default settings can be overwritten in an external stylesheet):

```
new-lengths { tablewidth: 0.8\textwidth; }
```

- The setting out of a non-trivial table requires a good deal of processing by L^AT_EX, and the results may sometimes be surprising. An example is a table that has so many wide columns that it cannot be accommodated within a specified table width.

Images

In contrast to tables, Lexxia has a relaxed policy for images. At the simplest level, individual images can be displayed differently: either inline or floated. A frame is also an option. Using an enhanced L^AT_EX installation, it is also possible to display images inline on the left or right edge of the page, with the text flowing around them. The Lexxia syntax is simple, eg:

```
_image=../images.image1.jpg
@align=center
@frame=true
```

Gotcha 3: Give the correct path to the (external) image. To understand the problem, look ahead to what we want to do: convert a text or XML file to a L^AT_EX file, and then process this file with `pdflatex`. The file path to the image must be the path relative to the current directory *when you are executing pdflatex*. It is interesting to compare this requirement to the corresponding path in an HTML file: here it is the path *relative to the path of the HTML file*.

Images can, and often should, be scaled in size. Depending on the sources of the image files, the natural sizes may differ greatly, so that it is necessary to scale them on an individual basis, by specifying a scale factor as another attribute to the image:

```
_image=../images.image1.jpg
@align=center
@frame=true
@scale=0.55
```

Alternatively, to set a default image scaling, which is applied in the absence of any individual scale attributes, you can include in the document specifications:

```
document { image-scale: 0.5; image-placement: h; }
```

If this is not specified, and no individual scalings are given, the images are included in their natural sizes.

Just how and where an image is displayed depends on how we specify the alignment. If the alignment is not specified, the image is floated and displayed centered horizontally at a position of L^AT_EX's choosing, with the hint from image placement. If the alignment is specified as *center*, the image is centered horizontally and displayed exactly where it is specified in the source text. This can cause troubles if there is not sufficient space for it to be displayed, with the result that there will be a premature page break (incompletely filled page) with the table or image placed at the top of the following page. This is probably not acceptable. Needless to say, floating some images and centering other images would be an excellent policy for confusing the reader.

As we shall see in the *Part 2* document, we can use an additional style (`picins`) to display images at the side, with text flowing around them. All we need to do there is to specify the alignment as *left* or *right*.

no align specification: the image is floated and centered horizontally;

center: the image is displayed inline and centered horizontally;
frame: the image is framed if this is set to "true";
caption and label: these can be specified if "align" is *not* specified (ie, for a float).

In order to refer to a floated table or image, \LaTeX allows it to have a *caption* and/or a *label*. At very least, a caption (title) should be specified, otherwise the document will be very confusing for a reader. \LaTeX keeps track of the labels, and the floats can be referred to using one or both of:

a *ref* tag: includes the number of the image or table;
 a *pageref* tag: includes the page on which the image or table is displayed.

Gotcha 4: If an image is not floated (given a specified alignment), a label is not added and cannot be referred to explicitly from the text. All tables can, and should, be marked with a caption and label.

To revisit the table above, we add a caption (title) and label:

```
_table
@cols=r|cc|X
@caption=THIS IS A FLOATED TABLE EXAMPLE
@label=EX1
_thead=Heading 1|Heading 2|heading 3|Heading 4
Content|More content|Longer content|The longest content of all in this column
Row 2 Content|More content|Longer content|The long content in this column
Content 3|More content|Longer content|A somewhat longer content in this column
Content 4|More content|Longer content|The longest content of all the rows in this column
_tfoot=Foot 4|More Foot|Foot 3|Foot 4
—
```

Gotcha 5: Make absolutely sure that, if you specify a *cols* attribute, you give a string long enough to match *all* of the columns. It does not seem to matter if you give too many but, if there are not enough, you will get a cryptic error message about an "extra alignment tab".

We have just used commands from the optional *varioref* package, so that the source text file includes the line:

```
and the result is Table <vref>EX1</vref>
```

This is converted to:

```
and the result is Table 2 on page 15.
```

Otherwise, if we have not have the *varioref* package installed, we can use the more common built-in reference commands:

```
and the result is Table <ref>EX1</ref> on page <pageref>EX1</pageref>
```

with a less elegant result if the table happens to be on the same page.

Styling the Document

To date, we have been so involved in the minutiae of a document that we have given no consideration to what shape the overall document has, and what its pages are to look like.

Document Type

The default \LaTeX type is *article*, and this is what you get when you fail to specify otherwise. But this is no bad thing, as it is probably the most versatile of the options.

Consider the structure of the document that you are working with: it could be a postmodern stream-of-consciousness which, I think, means no structure. In the event that this is the case, I probably have no particular advice to give. Typically, though, there *is* structure: if you are talking geography, it is based on countries, continents, etc; history has countries and times; schools have classes and years, plays have acts and scenes, etc. And, more often than not, most structural elements start with a *heading*.

A \LaTeX document can be very easy to read—or absolutely impossible, depending on how it is set out. At worst, it can seem to be a rather *flat* document (even though, by its own rules, it is extremely structured). Contrast this with XML, which is best displayed in a very *unflat* way, with line breaks and indentation to help the reader to see elements inside elements inside elements, etc. It is this structured unflatness that is so important for Lexxia processing: it identifies the different levels and classifies the elements within those levels: headings (these are single elements at the starts of the levels), default text, highlight text, structures like tables and lists, etc. (It is possible to process a simple, very flat, text document with Lexxia, but the outcome will not be very interesting.)

\LaTeX documents are based on sections, subsections, subsections and paragraphs, etc. It is worth emphasising at this point that a \LaTeX file is *nothing but an ASCII text file*. A potentially complicated text file at times, but just a text file that can be treated as such—perhaps to edit it in a standard text editor.

The Document Options

The \LaTeX document types are punctuated by headings and other commands of different levels (which, as we have seen, relate to XML nested blocks at different levels):

- article:** document, section, subsection, subsection, paragraph, subparagraph
- book:** document, (part), chapter, section, subsection, subsection, paragraph, subparagraph
- report:** document, (part), section, subsection, subsection, paragraph, subparagraph
- letter:** document, opening, closing (non-standard type)

In truth, there is little difference between the structures of article and book (the two most common types): only the (optional) insertion of *part and chapter* at the top level. Table 3 on the following page shows the how the header structure affects the processing of a text document.

The internal CSS-type stylesheet in Lexxia contains the specifications:

Table 1: Table with Specified Alignment

Heading 1	Heading 2	heading 3	Heading 4
Content	More content	Longer content	The longest content of all in this column
Row 2 Content	More content	Longer content	The long content in this column
Content 3	More content	Longer content	A somewhat longer content in this column
Content 4	More content	Longer content	The longest content of all the rows in this column
Foot 4	More Foot	Foot 3	Foot 4

Table 2: THIS IS A FLOATED TABLE EXAMPLE

Heading 1	Heading 2	heading 3	Heading 4
Content	More content	Longer content	The longest content of all in this row
Row 2 Content	More content	Longer content	The long content in this row
Content 3	More content	Longer content	A somewhat longer content in this row
Content 4	More content	Longer content	The longest content of all the rows in this row
Foot 4	More Foot	Foot 3	Foot 4

Table 3: Headings and Document Types

Dotted File	XHTML Tag	Lexxia ID	Article Use	Book Use
.(header text)	h1	T1	title	title
..(header text)	h2	T2	section	chapter
...(header text)	h3	T3	subsection	section
....(header text)	h4	T4	subsubsection	subsection, etc

```
document { class: article; }
T2 { style: section; }
T3 { style: subsection; }
```

So, to change from our default *article* to *book* type we need to have the following entries in our external CSS stylesheet:

```
document { class: book; }
T2 { style: chapter; }
T3 { style: section; }
```

With the *book* class, we get a PDF document with:

- A separate title page without a header or footer, provided there is a T1-type heading;
- A blank page;
- One or more table of contents pages, with page numberings in roman, starting with “i”;
- Possibly one blank page;
- The content pages of the document, with page numberings in arabic, starting with “1” and headings relating to the content of each page.
- All chapters starting on the recto (odd-numbered) page;
- All pages set out recto/verso for the odd/even-numbering, with side margins appropriate for binding.

In summary, the *book* class is intended for double-sided printing and subsequent binding. The margins alternate and the possible blank pages at the start are used to ensure that the title, table of contents and actual contents start on the right side of the opened book: very professional attention to detail.

It is possible to emulate much of the appearance of a book using the *article* type, but the article type lends itself better to a document with a limited number of sections and, perhaps, a large number of subsections that are not separated by page breaks. It gives you a choice of whether you want a separate title page, whether you really need a table of contents, and whether you wish to separate sections and subsections by page breaks.

Page breaks are easily specified: if, for example, we want to have a page break between sections in an *article* document, the logic goes like this:

```
A section heading comes from a T2 tag (see Table 3 on the previous page);
A T2 tag marks the start of an L2 (level 2) block;
We need a page break at the end of each L2 block.
```

So we add to our external CSS stylesheet:

```
document { class: book; }
T2 { style: chapter; }
T3 { style: section; }
L2 { display: page; }
```

In all, the default *article* class is the place to start.

Formatting the Page Headers

A page can have up to three headers, specified on a yes/no basis, left to right. Consider the following specifications:

```
headers { style: headerstyle; toc: nyn; body: yny; }
headerstyle { font-size: small; font-shape: small-caps; }
```

The first line states that the table of contents section of the document is to have only a center heading, which would be the title of the document, if any. The body of the document is to have left and right headings, corresponding to the (then current) section and subsection (for an article document) or chapter and section (for a book). The page headers have a *headerstyle* style. You can either alter the definition of *headerstyle* or associate the headers with any other suitable style.

Formatting the Paragraphs

By default, paragraph openings are indented, with the exception of the first paragraph of a block. I tend to prefer no indentation, but with an increased gap between the paragraphs. \LaTeX can accommodate this, and the way it does it gives us a glimpse of how the whole document is controlled: by means of lengths. Here, we need to alter two lengths: `parindent` (the length of the indentation) and `parskip` (the excess space to be inserted between the paragraphs). So we add to our stylesheet:

```
lengths { parindent: 0pt; parskip: 8pt plus1pt minus1pt; }
```

Note that `parskip` is set to a *rubber length*, which gives \LaTeX some latitude in inserting the paragraph gaps, with plus and minus limits. *To demonstrate the point, these settings are used in this document.* Finally, there is a related length, *parsep*, that sets the spacings between lines in a *list*.

Hyphenation

One of the reasons that \LaTeX produces such superb documents is its ability to hyphenate. It has an inbuilt set of rules, but it should be understood that these are rules for *English* and, worse than that, it is *American English*. As a result, there are occasional situations when it cannot find a spot to split a word, so that it overflows the right margin and issues a warning about an “overfull box”. It needs help, which means that you have to insert a hint—either in the source text or the `.tex` file that you are trying to process with \LaTeX . A silly example might be the word

understood, and you would help the hyphenation by converting it to `under\-stood`. This does not happen very often and, in fact, the margin overflows are mostly barely noticeable (such as “2.03pt”). More rarely, you may get an “underfull box” message, when it is not happy about the spacing that it has had to insert in the line. This, too, can usually be remedied by helping with the hyphenation.

A quite different use of hyphens comes with the use of dashes: there are four of them:

A single hyphen: A hyphen (of course)

Two hyphens: An en dash

Three hyphens: An em dash

⋈-⋈: A minus sign that is used only in mathematics mode of \LaTeX .

Preparing Source Documents

Text Source

We need a source file. At first, you are probably better with a text file: just a few paragraphs will do. If you are using a word processor, be sure to save or export it as a *text file* and name it with a `.txt` extension (not the usual `.doc` extension). We want just plain text, with no included markup or styles. Now convert the file to \LaTeX , then PDF:

```
lexxia myfile.txt -L+*.tex
pdflatex myfile
pdflatex myfile
```

Now you have a PDF file (`myfile.pdf`). It will not be very interesting: perhaps a couple of blank pages, then the text from the source file. Not much, but it is a start. Now add some headings at strategic places in the document and reprocess:

```
.Heading 1
..Heading2
...Heading 3
..Heading2a
...Heading 3a
```

At this point, you should have a title, two section headings and two subsection headings: you have added structure and have started to get some styling into your document. More importantly, you have made a start on *text styling*. But, to go much further, you need to know about the Lexxia reserve tags. They are many, and their names are largely self-explanatory:

```
document features: title subtitle author
table group: table thead tbody tfoot th tr td
list group: ul ol li dl dt dd
```

text features: em b
 block features: pre comment note center verb verse quote
 float references: ref pageref vref vpageref url
 image definition: image
 ignore content: ignore

Many of these should be self-explanatory in light of the layout of the present document, but you should experiment with some of the text features, eg, by wrapping some of your existing text inside one, say “note”:

`_note`

This is some meaningless text put inside a wrapper to change its style and indentation. This is some meaningless text put inside a wrapper to change its style and indentation. This is some meaningless text put inside a wrapper to change its style and indentation. This is some meaningless text put inside a wrapper to change its style and indentation. This is some meaningless text put inside a wrapper to change its style and indentation.

At this point, I hope that you might be convinced that it is very easy to generate an elegant document from a text file with very little mark-up. To convince you thoroughly, download the [sources.zip](#) file and expand it in a convenient directory. It contains several text files with familiar names, such as *copperfield.txt* and *grimm.txt*. Process them with Lexxia with default styling and build the pdf files as before. The zip file also contains CSS files, such as *book.css*, so then try processing them with the additional styling, eg:

```
lexxia copperfield.txt -Lbook.css+*.tex
pdflatex copperfield
pdflatex copperfield
```

Now you have a pretty respectable book. (The one thing that really stands out is the messiness of the table of contents in some of the books. This can largely be fixed by changing the toc-columns setting to 2 or 3.) To get a good feel for how to process your own documents, use these source documents and stylesheets as examples.

If you are not aware of its existence, you should visit the Project Gutenberg site at <http://www.gutenberg.org>. There is a whole world there of books and other text, much of it in a pretty rough state—and waiting to be transformed by volunteers into elegant PDF and other formats.

But choose your starting text with some care: there are HTML and plain text documents there. I understand that many of them have been generated by OCR on out-of-copyright books, so they are pretty basic. To work with them, you will need a good text editor. In Linux, the obvious choices are *gedit* and *kedit*, in Windows, the pickings are thinner. Do not even contemplate using *Notepad* because it is a disgrace. *Wordpad* is a little better, but still quite limited. You might try *Crimson Editor*: it is free and pretty good, but still cannot really cope with some of the built-in difficulties with the Gutenberg files.

Bear in mind that we are using Lexxia to load a text file *as an XML document*. At various stages during the document preparation, it is very instructive to load and display what you are working with:

```
lexxia myfile.txt -O
```

With a typical Gutenberg text document, you will initially get a mass of paragraph elements, each corresponding to one of the lines in the text. As we edit to join these, you will see just how they are joined—and how much progress you are making. Once you add a few headers, preferably at different levels, you will see a dramatic change in the document structure.

Whitespace

In the following discussion, I will refer to four types of text whitespace (invisible characters): spaces (' '), tabs ('\t'), carriage returns ('\r') and newlines ('\n'). Of these, carriage returns have long outlived their usefulness in text files, but they persist in files generated in the Windows environment where all lines are terminated with the \r\n pair, whereas in the *nix environment, the termination is with a simple '\n' character. My experience with several of the Gutenberg files has been that they all have the \r\n termination. Why this matters will become clear.

In my experience, the Gutenberg text has consisted of short lines punctuated with blank lines. In other words, a paragraph consists of one or more short lines between the blank lines, whereas the text we are after for Lexxia consists of a *single* line for each paragraph (typically long lines are displayed wrapped in a text editor). To avoid confusion, it is a good idea to show line numbers in the editor to make the line breaks clear (one of the reasons why it is necessary to have a good editor).

Line Termination

Whether or not you include carriage returns in your document is your decision, but it is a good idea to be consistent. If, for example, you edit in Linux a file that has the carriage returns, you should be aware that any *new* line breaks that you might introduce will *not* have the carriage returns. When you subsequently process the file with latex, pdflatex, etc, you will get error messages relating to empty lines. Better avoided.

I now give an overview of how I recast whitespace for Lexxia, using gedit:

1. Replace <specify two spaces> with <specify one space> (remove runs of spaces, and repeat until there are no hits);
2. Replace \r with (nothing) (remove them);
3. Replace \n\n\n with \n\n (remove multiples of more than two blank lines; repeat this command until there are no hits reported);
4. Replace \n\n with a unique string (say "PPPP");
5. Replace \n with <specify one space>; (At this point, there are no line endings in the whole document: the file consists of a single line!)
6. Replace the unique string ("PPPP") with \n\n (With a large file, this can be rather slow, so just trust in your editor!) The overall result is a document with paragraphs that typically extend well past the margins of your editor, causing line wrapping. Between the paragraphs there are two line endings (one is sufficient for our purposes, but two make the whole document much more readable—and editable).

Markup

At this point, we still have a plain text document, with absolutely no markup, so next we look for headings, which we specify with preceding dots (full-stops or periods). We give some thought to the structure that they will impose: a novel, for example, will typically have only a title and a number of chapter titles. So the title gets one dot and each of the chapter headings gets two.

At this point, most of the work has been done, but there are a few minor points to attend to:

- Inline emphasis;
- Block emphasis;
- Quotes.

Emphasis in the body of the text is often delimited with underscore characters, as in “I `_really` `_mean` what I say”. This should be replaced with “I `really` mean what I say”.

To emphasise a block of text, such as a quotation, mark it as a *note* or *comment*. This can be applied to a single sentence as:

```
_note=This is a simple sentence to be highlighted.
```

or to a block of text by bracketing with the same style:

Quotes are best understood and styled by \LaTeX if they are used in the following way:

```
_note
This is a simple sentence to be highlighted.
And this is the second sentence,
and the third.
—
```

To get the best effects from quotes, \LaTeX uses the little-used backtick character (shift-tilde):

- Opening quotes use the backtick character;
- Closing quotes use the normal tick character (apostrophe);
- Double quotes use the backtick or apostrophe doubled.

This is part of the fine control that \LaTeX maintains, but the difference here is not always very obvious. With Palatino font, for example, it is very difficult to pick the difference, but it is quite obvious with Times font.

In summary, the most important alterations of the source document are to the line endings and the headings. With those alone, you can expect a pretty good PDF document!

If you are a Windows user, you will have noted that all this business with whitespace simply cannot be done with any editor that you have. If absolutely desperate, you can resort to manual

joining of lines within a paragraph, using the *delete* or *backspace* key, but that is not an option that would attract me. Line breaks outside of paragraphs are no problem, as Lexxia simply ignores them.

If you are a Linux user and you are preparing files that might be later edited in Windows, give some thought to inserting the carriage return characters to make life easier for Windows users.

- Replace `\n` with `\r\n`.

XML or XHTML Source

Most of the layout/punctuation detail of text source is now irrelevant. So, if you are not daunted by the tag structure of XML source, you should consider it if it is available. (This is largely because most whitespace is ignored by an XML parser.)

More importantly, the structure of an XML file is likely to be much more complex than a marked-up text file. This means that the structure itself becomes the dominant consideration, rather than the selection of reserved tag names, such as “note” and “comment”. Recall that, directly after a source file is loaded, it is analysed by building an internal Schema document and subsequently characterising the tags on the basis of their placement within the document blocks. This means that the styling is based on this characterisation, rather than the tag names—although reserved names are still respected.

The upshot of this is that you should try XML source if it is available, and that you are likely to get a very pleasant shock with the result. Just remember to include the source file extension in the Lexxia command:

```
lexxia newfile.xml -L+*tex
pdflatex newfile
pdflatex newfile
```

Of source, you will want to fine-tune the result with an external CSS file. The Shakespeare files give a superb opportunity to hone your skill with XML source: you will find that it displays best as an *article* document in a two-column format with a specified heading format and—and this is very important—a specified *paragraph format*. This final point is very important: until now, we have been able to assume that blocks of text, as the default category in a document, are to be displayed as *paragraph blocks* with full text justification and separated by whitespace. In a play, on the other hand, the default text is the lines of the speeches. These need to have left-aligned text (full justification can look very bad). And they look even worse if there a paragraph skip between them. These are attended to with the following declarations in the external stylesheet:

```
document { text-align: left; }
p { display: inline; }
```

The second of these specifications overwrites (cancels) the default specification that paragraphs are to be displayed as blocks, with paragraph spacing following.

Getting Serious

The only way to get started is by starting. The ease the way, I provide at my site (www.limpidsoft.com) a zip file with the source text for the present document, as well as several text documents that I have constructed from files at www.gutenberg.org (mostly pretty rough stuff and very much work in progress). The Shakespeare files are readily available (try www.ibiblio.org/bosak).

The source of this document (`aboutpdf1.txt`) and its stylesheet (`about1.css`) should be the most useful, as they illustrate a fair range of detail. Related files (`aboutpdf2.txt` and `about2.css`) introduce extension packages for \LaTeX that provide even greater flexibility. The novels (and their stylesheets), although unfinished, give some indication of how to approach styling in more typical text. In all, they illustrate some good plus a good smattering of the bad and the ugly. All certainly still works in progress, but the road ahead is clearing.

Enjoy—and keep your comments and criticisms coming to john_redmond@optusnet.com.au. I do appreciate your feedback.