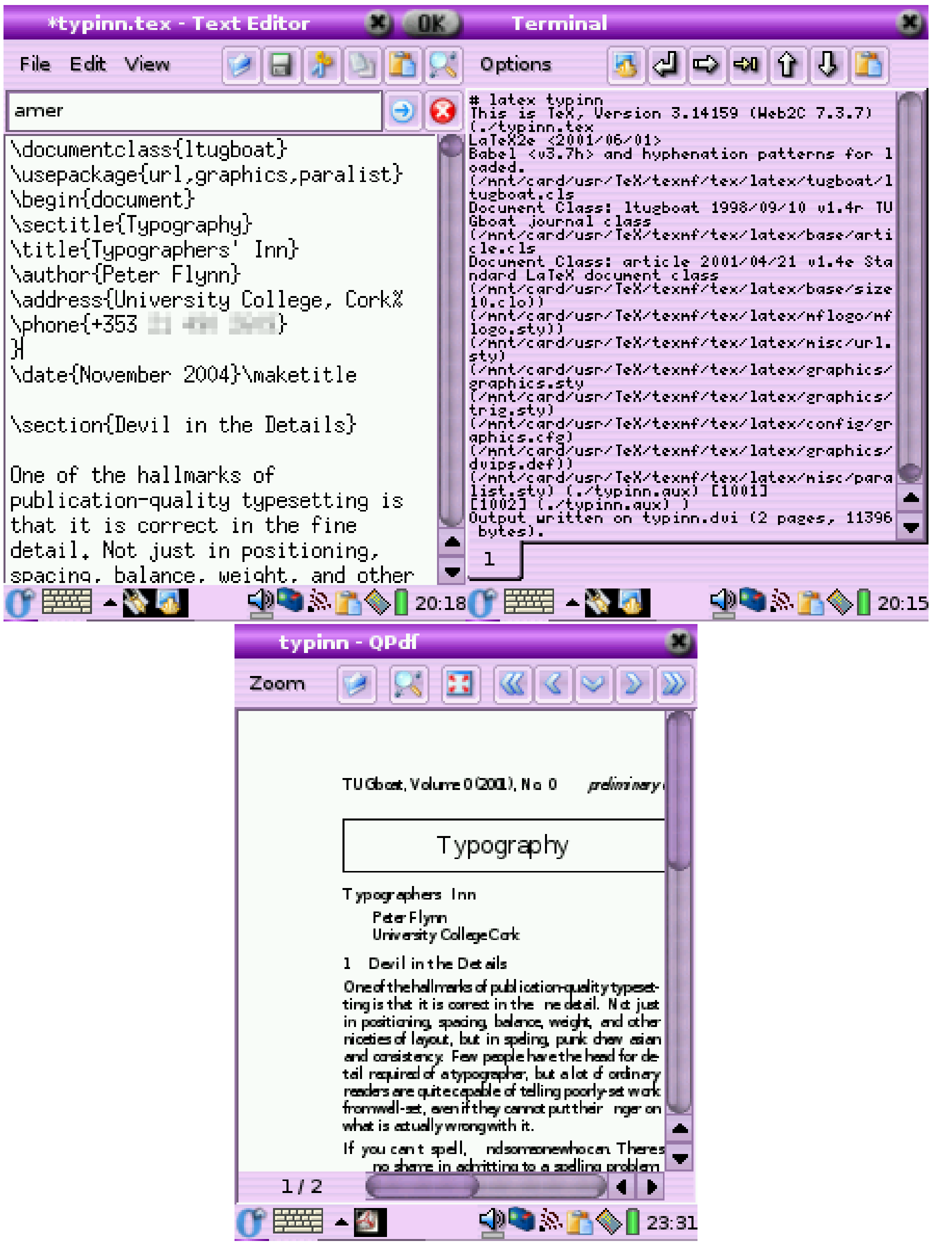
8 Conversion

As we saw right at the start, LaTeX uses plaintext files, so they can be read and written by any standard application that can open text files. This helps preserve your information over time, as the plaintext format cannot be obsoleted or hijacked by any manufacturer or sectoral interest, and it will always be readable on any computer, from your smartphone (LaTeX is available for many handhelds, from old PDAs, see below, to Android devices, see below) through all desktops and servers right up to the biggest supercomputers.

However, LaTeX is intended as the last stage of the editorial process: formatting for print or display. If you have a requirement to re-use the text in some other environment — a database perhaps, or on the Web or a Compact Disc (CD) or Digital Versatile Disc (DVD), or in Braille or voice output — then it should probably be edited, stored, and maintained in something neutral like the Extensible Markup Language (XML), and only converted to LaTeX when a typeset copy is needed.

Although LaTeX has many structured-document features in common with SGML and XML, it can still only be processed by the LaTeX programs. Because its macro features make it almost infinitely redefinable, processing it requires a program which can unravel arbitrarily complex macros, and LaTeX and its siblings are the only programs which can do that effectively.

Figure 8.1 – LaTeX editing and processing on the Sharp Zaurus 5500 PDA



```
\documentclass{ltugboat}
\usepackage{url,graphics,paralist}
\begin{document}
\sectitle{Typography}
\title{Typographers' Inn}
\author{Peter Flynn}
\address{University College, Cork%
\phone{+353 [redacted]}
}
\date{November 2004}\maketitle

\section{Devil in the Details}

One of the hallmarks of
publication-quality typesetting is
that it is correct in the fine
detail. Not just in positioning,
spacing, balance, weight, and other
```

```
# latex typinn
This is TeX, Version 3.14159 (Web2C 7.3.7)
(./typinn.tex
LaTeX2e <2001/06/01>
Babel <v3.7h> and hyphenation patterns for l
oaded.
(/mnt/card/usr/TeX/texmf/tex/latex/tugboat/l
tugboat.cls
Document Class: ltugboat 1998/09/10 v1.4r TU
Gboat journal class
(/mnt/card/usr/TeX/texmf/tex/latex/base/arti
cle.cls
Document Class: article 2001/04/21 v1.4e Sta
ndard LaTeX document class
(/mnt/card/usr/TeX/texmf/tex/latex/base/size
10.clo))
(/mnt/card/usr/TeX/texmf/tex/latex/mflogo/mf
logo.sty))
(/mnt/card/usr/TeX/texmf/tex/latex/misc/url.
sty)
(/mnt/card/usr/TeX/texmf/tex/latex/graphics/
graphics.sty
(/mnt/card/usr/TeX/texmf/tex/latex/graphics/
trig.sty)
(/mnt/card/usr/TeX/texmf/tex/latex/config/gr
aphics.cfg)
(/mnt/card/usr/TeX/texmf/tex/latex/graphics/
dvips.def))
(/mnt/card/usr/TeX/texmf/tex/latex/misc/para
list.sty) (./typinn.aux) [1001]
[1002] (./typinn.aux) )
Output written on typinn.dvi (2 pages, 11396
 bytes).
```

TUGboat, Volume 0 (2001), No 0 *preliminary*

Typography

Typographers Inn

Peter Flynn
University College Cork

1 Devil in the Details

One of the hallmarks of publication-quality typeset-
ting is that it is correct in the ne detail. Not just
in positioning, spacing, balance, weight, and other
niceties of layout, but in speling, punk chew asian
and consistency. Few people have the head for de-
tail required of a typographer, but a lot of ordinary
readers are quite capable of telling poorly-set work
from well-set, even if they cannot put their nger on
what is actually wrong with it.

If you can t spell, nd someone who can. Theres
no shame in admitting to a spelling problem

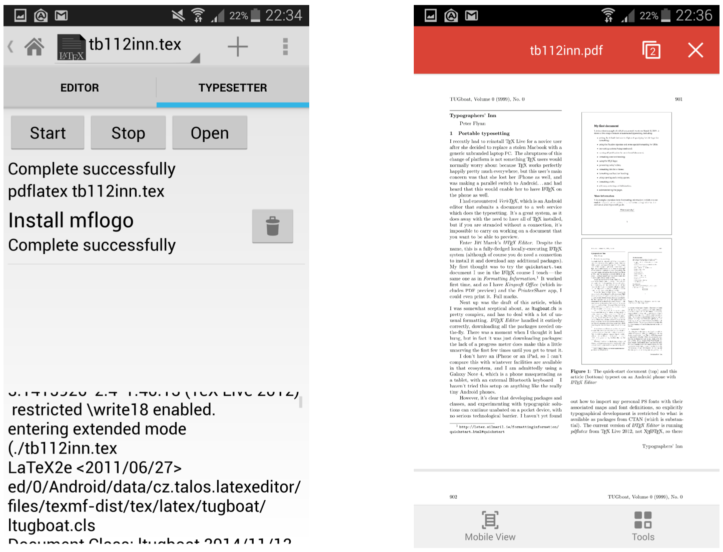


Figure 8.2 – LaTeX editing and processing on the Samsung Galaxy Note 4

```
Complete successfully
pdflatex tb112inn.tex
Install mflogo
Complete successfully
```

```
 restricted \write18 enabled.
entering extended mode
(./tb112inn.tex
LaTeX2e <2011/06/27>
ed/0/Android/data/cz.talos.latexeditor/
files/texmf-dist/tex/latex/tugboat/
ltugboat.cls
```

Like other typesetters and formatters (Quark *XPress*, Adobe *InDesign* and *PageMaker*, *FrameMaker*, Microsoft *Publisher*, *3B2*, etc), LaTeX is largely a one-way street leading to typeset printing or display formatting.

Converting LaTeX to some other format therefore means you will unavoidably lose some formatting, as LaTeX has features that others systems simply don't possess, so they cannot be translated — although there are several ways to minimise this loss or compensate for it. Similarly, converting other formats into LaTeX often means editing back the stuff the other formats omit because they only store appearances, not structure.

Most converters are one-way: that is, they convert into LaTeX or out of LaTeX, and there are several excellent systems for doing the conversion from LaTeX directly to HyperText Markup Language (HTML) so you can at least publish it on the web, as we shall see in below.

Most of the utilities listed below are Open Source or free-to-use software. There are many commercial solutions as well, either the software itself or a service where they do it for you, but they are

Pandoc

There is one system that does conversion in both directions, and includes a huge range of formats: *Pandoc*. This is a large library of *Haskell* routines for handling conversions, with a command-line front end. Supported formats include *Word*, *Libre Office*, LaTeX, *DocBook*, EPUB v3, *InDesign*, *Markdown*, and dozens of others, even JavaScript Object Notation (JSON).

Before trying the systems described in section 8.1 and section 8.2 on page 230, see if *Pandoc* will handle your files.It doesn't have the same levels of speciality as some of the other converters, but it does provide those additional input formats. The exception is probably converting from XML to LaTeX for which a robust Extensible Stylesheet Language 3 (XSLT 3) script is really the only reliable solution.

`pandoc.org/`

mostly aimed at large-scale business conversion, and are ususally too expensive for domestic or academic single documents.

8.1 Converting into LaTeX

Turning other document formats into LaTeX is generally several orders of magnitude easier than the other way round, because almost all other document-handling systems know and understand their own features.

The methods vary from wordprocessors or plugins with a menu entry for File › Save As... › LaTeX, to a custom XSLT script for a bespoke solution.

8.1.1 Conversion from wordprocessors

Several wordprocessor systems can save their text in LaTeX format using the File › Save As... › LaTeX or Export As... menus. A very few actually create LaTeX natively, but there are also some stand-alone converters.

Microsoft Word

Microsoft Word in particular does not have any LaTeX export facility at all. Instead, you can either open the document in one of the other systems listed below, and use that to export into a LaTeX document, or use a converter. If you prefer a commercial solution which runs as a plugin within Word, see the the list item 'GrindEQ' section 8.1.1.2 on the next page or the the list item 'TeX2Word' section 8.2.1 on page 232.

8.1.1.1 Native LaTeX

LyX: LyX is probably the best-known wordprocessor-like interface to LaTeX (not quite WYSIWYG, more What You See Is What You Meant). It is already basically LaTeX internally, and its LaTeX export is very good, offering several flavours (LaTeX, XeTeX, LuaTeX, etc) as well as Word and Libre Office formats.

www.lyx.org/;

Scientific Word: A writing and editing front-end interface to LaTeX specifically designed for math and science papers. With LaTeX as the backend, the companion product *Scientific Workplace* provides computational and plotting facilities.

sciword.co.uk/.

8.1.1.2 Export via plugin

AbiWord: *AbiWord* can load a Word document and provides an extensive list of export formats, so it provides a good pathway for single-file conversion. Export formats include Word, HTML, XHTML, RTF, EPUB v3, *DocBook*, *Libre Office*, and others including LaTeX.

www.abisource.com/;

Libre Office: *Libre Office*[1] has a LaTeX plugin called *Writer2LaTeX*, so it can be used to open Microsoft *Word* documents (as well Office Document Text (ODT) and other formats), and export them to LaTeX.

www.libreoffice.org/;

GrindEQ: *GrindEQ* is a commercial plugin for Microsoft Word to allow the loading and saving of LaTeX documents. It is oriented primarily towards mathematics.

www.grindeq.com/.

Several maths packages like the *EuroMath* editor, and the *Mathematica* and *Maple* analysis packages, can also save material in LaTeX format.

Pandoc: See the Note on p. 220pandoc.

pandoc.org/;

docx2tex: This system converts Microsoft Word's `.docx` to LaTeX (only). It runs in *Java* (standalone or as an XProc pipeline with XML *Calabash*), so it works on all platforms. It is extremely configurable, with customisable directories, and a config file that lets you map your Word styles to LaTeX `\begin` and `\end` commands, just like the old *PCWriTeX* driver.

github.com/transpect/docx2tex/releases.

8.1.1.3 Failing that...

If you can't get the kind of conversion you want from the existing utilities, or you need to make your own (perhaps to embed in another system), these are some alternatives.

Using HTML: Use the File › Save As... › HTML (or export) menu from your wordproessor to save the file as HTML, then rationalise the HTML into the XML version of HTML (XHTML) using Dave Raggett's *HTML Tidy*, and convert the resulting XHTML

[1] The former *OpenOffice* was taken over by Apache, and is no longer regarded as a contender.

file to LATEX with the technique shown below in section 8.1.3 on the next page.

`tidy.sourceforge.net/`;

Using PDF: Apache provides a Java utility called *pdfbox* which can extract the text from a PDF document into HTML format, preserving the bold and italics, which *pdftotext* does not do. This can save a lot of time in post-editing before using the HTML conversion mentioned above.

`pdfbox.apache.org/`;

Using RTF: Among the conversion programs for related formats on CTAN is Ujwal Sathyam and Paul DuBois's *rtf2latex2e*, which converts Rich Text Format (RTF) files (output by many wordprocessors) to LATEX 2$_\varepsilon$. The package description says it has support for figures and tables; equations are read as figures; and it can handle the latest RTF versions from Microsoft *Word* 97/98/2000, *StarOffice* (so presumably *OpenOffice* and *Libre Office*), and other wordprocessors. It runs on Windows and Unix & GNU/Linux systems, including Apple Macintosh OS X

`rtf2latex2e.sourceforge.net/` (also available as package rtf2latex2e from CTAN);

Using Word or ODF: If you can get the files into *Word* or *Libre Office* format (which are both basically Zip files containing XML), write a transformation in XSLT to convert the internal XML directly into LATEX. This is by far the most robust way to do it but it requires that the author or editor has rigorously used Named Styles. Unfortunately, without them, the quality of most wordprocessing files is generally poor when it comes to identifying which bits do what (which is what LATEX needs to know), so some guesswork or heuristics may be needed.

At the extreme end are very simplistic systems that are incapable of outputting any kind of structured document, because they only store what the text looks like (basically, font, size, and style), rather than *why* it's there or what role it fulfils. In those cases you may be able to save the file as a PDF, and use the *pdfbox* utility as in the list item 'Using PDF' above above.

8.1.2 Bulk conversion

Converting large numbers of related documents using most of the non-graphical (command-line) utilities is often straightforward using a shell script in (eg) *bash* or *Powershell*. At the simplest level it can just be a few lines like

```
for f in *.docx; do
  pandoc -f docx -t latex $f ${f/docx/tex};
done
```

However, if you have large numbers of obsolete Word .doc files (too many to open and save as .docx), you can try to use a specialist conversion tool like EBT's *DynaTag* (supposedly still available from Enigma, if you can persuade them they have a copy to sell you; or you may still be able to get it from Red Bridge Interactive in Providence, RI). It's old and expensive and they don't advertise it, but for the Graphical User Interface (GUI)-driven bulk conversion of consistently-marked *Word* (.doc, *not* .docx) files into usable XML it beats everything else hands down. But whatever system you use, the *Word* files MUST be consistent, though, and MUST use Named Styles from a stylesheet (template), otherwise no system on earth is going to be able to guess what they mean.

There is of course an external way, suitable for large volumes only: send it off to the Pacific Rim to be scanned, retyped, or hand-edited into XML or LaTeX. There are hundreds of companies from India to Polynesia who do this at high speed and low cost with very high accuracy. It sounds crazy when/if document is already in electronic format, but it's a good example of the problem of low quality of wordprocessor markup that this solution exists at all.

8.1.3 Getting LaTeX out of XML

You will have noticed that most of the solutions lead to one place: XML. As explained above and elsewhere, this format is the only one so far devised capable of storing sufficient information in machine-processable, publicly-accessible form to enable your document to be recreated in multiple output formats. Once your document is in XML, there is a large range of software available to turn it into other formats, including LaTeX. Processors in any of the common XML processing languages like XSLT or *Omnimark* can easily be written to output LaTeX, and this approach is extremely common.

Much of this would be simplified if wordprocessors supported native, arbitrary XML/XSLT as a standard feature, because LaTeX output would become much simpler to produce, but this seems unlikely.

However, since the early 2000s the internal format for both *OpenOffice* (now *Libre Office*) and *Word* is now XML. Both `.docx` and `.odf` files are actually Zip files containing the XML document together with stylesheets, images, and other ancillary files. This means that for a robust transformation into LaTeX, you just need to write an XSLT stylesheet to do the job — non-trivial, as the XML formats used are extremely complex, but certainly possible.

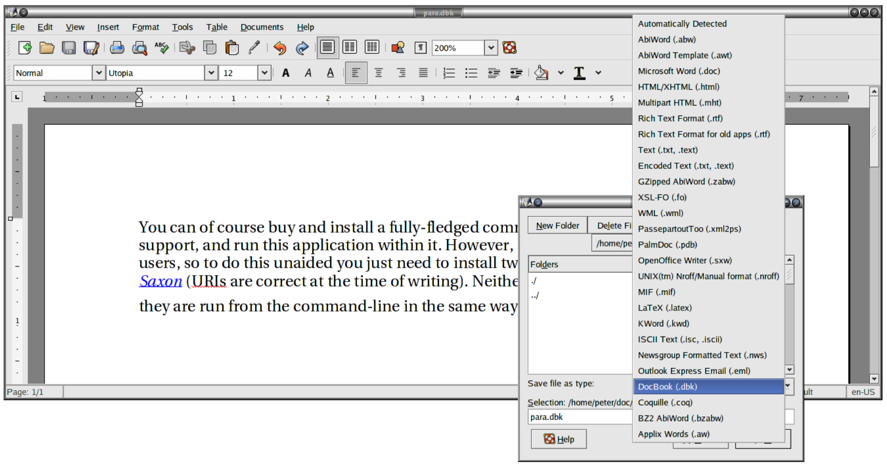
Assuming you can get your document out of its wordprocessor format into XML by some method, here is a very brief example of how to turn it into LaTeX.

You can of course buy any fully-fledged commercial XML editor with XSLT support, and run transformations within it. However, this is beyond the reach of many individual users, although *oXygen* is available at a discounted price to academic sites.

To do the job unaided you need to install three pieces of software: *Java*, *Saxon* or another XSLT processor, and the *DocBook* 5.0 DTD (links are correct at the time of writing). None of these has a graphical interface: they are run from the command-line.

As an example, let's take the a sample paragraph, as typed or imported into *AbiWord* (see below). This is stored as a single paragraph with highlighting on the product names (italics), and the names are also links to their Internet sources, just as they are in this document. This is a convenient way to store two pieces of information in the same place.

Figure 8.3 – Sample paragraph in *AbiWord* being converted to XML



You can of course buy and install a fully-fledged comı
support, and run this application within it. However,
users, so to do this unaided you just need to install tw
Saxon (URIs are correct at the time of writing). Neithe
they are run from the command-line in the same way

Save file as type: DocBook (.dbk)
Selection: /home/peter/doc/
para.dbk

AbiWord can export in DocBook format, which is an XML vocabulary for describing technical (computer) documents — it's what I use for this book. *AbiWord* can also export LaTeX, but we're going to make our own version, working from the XML (Brownie points for the reader who can guess why I'm not just accepting the LaTeX conversion output).

Although *AbiWord*'s default is to output an XML book document type, we'll convert it to a LaTeX article document class. In this example I've changed the linebreaks to keep it within the bounds of the page size of the PDF edition:

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
  "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
<book>
<!-- ================================================== -->
<!-- This DocBook file was created by AbiWord.          -->
<!-- AbiWord is a free, Open Source word processor.     -->
<!-- You may obtain more information about AbiWord
     at www.abisource.com                               -->
<!-- ================================================== -->
<chapter>
  <title></title>
  <section role="unnumbered">
    <title></title>
    <para>You can of course buy and install a fully-fledged
      commercial XML editor with XSLT support, and run this
      application within it. However, this is beyond the
      reach of many users, so to do this unaided you just
      need to install three pieces of software: <ulink
      url="http://java.com/download/"><emphasis>Java</emphasis></ulink>,
      <ulink
      url="http://saxon.sourceforge.net"><emphasis>Saxon</emphasis></ulink>,
      and the <ulink
      url="http://www.docbook.org/xml/4.2/index.html">DocBook
      4.2 DTD</ulink> (URIs are correct at the time of writing).
      None of these has a visual interface: they are run from
      the command-line in the same way as is possible with
      L<superscript>A</superscript>T<subscript>E</subscript>X.</para>
  </section>
</chapter>
</book>
```

The XSLT language lets us create templates for each type of element in an XML document. In our example, there are only three which need handling, as we did not create chapter or section titles (DocBook requires them to be present, but they don't have to be used).

- para, for the paragraph[s];
- ulink, for the URIs;
- emphasis, for the italicisation.

I'm going to cheat over the superscripting and subscripting of the letters in the LaTeX logo, and use my editor to replace the whole thing with the \LaTeX command. In the other three cases, we already know how LaTeX deals with these, so we can write the templates accordingly.

Writing XSLT is not hard, but requires a little learning. The output method here is text, which is LaTeX's file format (XSLT can also output HTML and other flavours of XML).

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">

  <xsl:output method="text"/>

  <xsl:template match="/">
    <xsl:text>\documentclass{article}\usepackage{url}</xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="book">
    <xsl:text>\begin{document}</xsl:text>
    <xsl:apply-templates/>
    <xsl:text>\end{document}</xsl:text>
  </xsl:template>

  <xsl:template match="para">
    <xsl:apply-templates/>
    <xsl:text>&#x0a;</xsl:text>
  </xsl:template>

  <xsl:template match="ulink">
    <xsl:apply-templates/>
    <xsl:text>\footnote{\url{</xsl:text>
    <xsl:value-of select="@url"/>
    <xsl:text>}}</xsl:text>
  </xsl:template>

  <xsl:template match="emphasis">
    <xsl:text>\emph{</xsl:text>
    <xsl:apply-templates/>
    <xsl:text>}</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

1. The first template matches /, which is the document root (before the book start-tag). At this stage we output the text which will start the LaTeX document, \documentclass{article} and \usepackage{url}.

   The apply-templates instructions tells the processor to carry on processing, looking for more matches. XML comments get ignored, and any elements which don't match a template simply have their contents passed through until the next match occurs, or until plain text is encountered (and output).[2]

2. The book template outputs the \begin{document} command, invokes apply-templates to make it carry on processing the contents of the book element, and then at the end, outputs the \end{document} command.

3. The para template just outputs its content, but follows it with a linebreak, using the hexadecimal character code x0A.

[2] Strictly speaking it isn't output at this stage: XML processors build a 'tree' (a hierarchy) of elements in memory, and they only get 'serialised' at the end of processing, into a stream of characters written to a file.

4. The ulink template outputs its content but follows it with a footnote using the \url command to output the value of the url attribute.

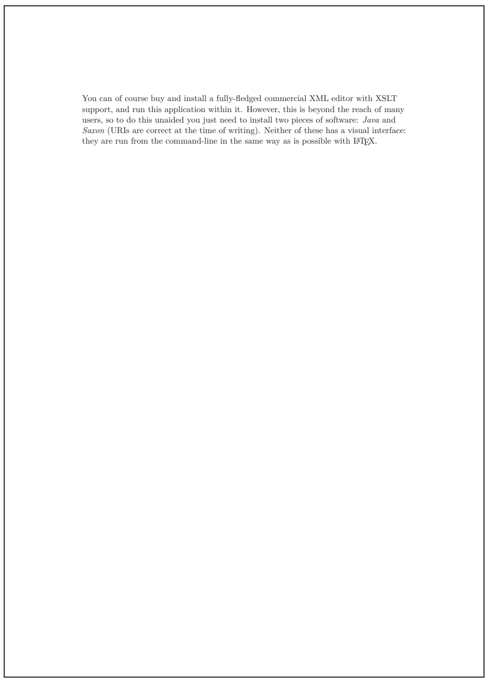
5. The emphasis template surrounds its content with \emph{ and }.

If you run this through *Saxon*, which is an XSLT processor, you can output a LaTeX file which you can typeset (see below).

```
$ java -jar saxon-he-10.3.jar -o para.ltx para.dbk para.xsl
$ xelatex para.ltx
This is XeTeX, Version 3.14159265-2.6-0.999991 (TeX Live
  2019/Debian) (preloaded format=xelatex) \write18 enabled.
entering extended mode
LaTeX2e <2020-02-02> patch level 2
L3 programming layer <2020-02-14>
(./para.tex
(/usr/share/texlive/texmf-dist/tex/latex/base/article.cls
  Document Class: article 2019/12/20 v1.4l
  Standard LaTeX document class
(/usr/share/texlive/texmf-dist/tex/latex/base/size11.clo))
(/home/peter/texmf/tex/latex/geometry/geometry.sty
(/usr/share/texlive/texmf-dist/tex/latex/graphics/keyval.sty)
(/usr/share/texlive/texmf-dist/tex/generic/iftex/ifpdf.sty
(/usr/share/texlive/texmf-dist/tex/generic/iftex/iftex.sty))
(/usr/share/texlive/texmf-dist/tex/generic/iftex/ifvtex.sty)
(/usr/share/texlive/texmf-dist/tex/generic/iftex/ifxetex.sty)
(/home/peter/texmf/tex/latex/geometry/geometry.cfg))
(/usr/share/texlive/texmf-dist/tex/latex/base/textcomp.sty)
(/usr/share/texlive/texmf-dist/tex/latex/l3backend/l3backend-xdvipdfmx.def)
(./para.aux)
(/usr/share/texlive/texmf-dist/tex/latex/base/ts1cmr.fd)
*geometry* driver: auto-detecting
*geometry* detected driver: xetex
[1] (./para.aux) )
Output written on para.pdf (1 page).
Transcript written on para.log.
$
```

This is a relatively trivial example, but it serves to show that it's not hard to output LaTeX from XML — this document is produced in exactly this way.

Figure 8.4 – The typeset paragraph and its generated source code



You can of course buy and install a fully-fledged commercial XML editor with XSLT support, and run this application within it. However, this is beyond the reach of many users, so to do this unaided you just need to install two pieces of software: *Java* and *Saxon* (URIs are correct at the time of writing). Neither of these has a visual interface: they are run from the command-line in the same way as is possible with LaTeX.

```
\documentclass{article}\usepackage{url}\begin{document}
You can of course buy and install a fully-fledged commercial XML
editor with XSLT support, and run this application within it. However,
this is beyond the reach of many users, so to do this unaided you just
need to install three pieces of software:
\emph{Java}\footnote{\url{http://java.sun.com/j2se/1.4.2/download.html}},
\emph{Saxon}\footnote{\url{http://saxon.sourceforge.net}}, and the
DocBook 4.2
DTD\footnote{\url{http://www.docbook.org/xml/4.2/index.html}} (links
are correct at the time of writing). None of these has a graphical
interface: they are run from the command-line in the same way as is
possible with \LaTeX.
\end{document}
```

8.2 Converting out of LaTeX

Converting LaTeX to other formats is much harder to do comprehensively than converting *into* LaTeX. As noted before, the LaTeX file format really requires a LaTeX processor in order to handle all the packages and macros, because LaTeX is entirely reprogrammable, and there is therefore no telling what complexities authors have added themselves by redefining things (what a lot of this book is about!).

However, if you have stuck to the standard commands given in the *LATEXbook*, and not defined anything extra or redefined anything else, and have used only a small set of the most common packages, most of the converters shown here will do a good job.

Many authors and editors rely on custom-designed or homebrew converters, often written in the standard shell scripting languages (Unix shell commands, *Perl*, *Python*, *Tcl*, *Lua*, etc). Some of the public packages presented here are also written in the same languages, but they have some advantages and restrictions compared with private conversions:

- ☐ Conversion done with the standard utilities (eg *awk*, *tr*, *sed*, *grep*, *detex*, etc) can be faster for one-off or *ad hoc* transformations, but it is easier to obtain consistency and a more sophisticated final product using a converter written to handle a wider range of LATEX features.

- ☐ Embedding homebrew non-standard control sequences in LATEX source code (a common habit of authors) may be tempting, to make it easier for the author to edit and maintain, but will always make it harder to convert to another system.

- ☐ Most of the converters mentioned here provide a fast and reasonably reliable way to get LATEX documents into *Word*, HTML, and other forms of XML in an acceptable — if not optimal — format, even if your primary target is eventually to convert to TEI, Journal Article Tag Suite (JATS), *DocBook*, or some other vocabulary, because once it's in well-formed XML of one kind, translation to another vocabulary is much easier.

- ☐ Above all, it is essential to understand that no conversion will produce an error-free result except for the most trivial of documents. All other output *will* require post-editing to correct things the converter was unable to handle, add things it missed, change formatting it was unable to apply, and delete things that should not have appeared.

There is a useful discussion of some of the alternatives mentioned below in § 3 of the lwarp package documentation (Dunne 2020, p. 71–73). If you actually want to *author* in a hybrid format that enables XML output (rather than converting existing native LaTeX documents), see below.

8.2.1 Conversion to *Word*

This is the most frequently-requested conversion, and also one of the hardest to do, because there are things LaTeX can do that simply cannot be represented in *Word* in any meaningful manner. However, *Word* uses XML internally, and it is also possible to convert your LaTeX to XHTML and import it (see below).

There are several programs on CTAN to do LaTeX-to-*Word* and similar conversions, but they do not all handle everything LaTeX can produce, and some only handle a subset of the built-in commands of default LaTeX. in particular, however, have a good reputation:

latex2rtf by Wilfried Hennings, Fernando Dorner, and Andreas Granzer translates LaTeX into RTF — the opposite of the *rtf2latex2e* mentioned earlier (RTF can be read by most wordprocessors). This Open Source program preserves layout and formatting for most LaTeX documents using standard built-in commands and obsolete codepages (not Unicode), but it has little support for redefined commands and common packages, including fontspec; and because it doesn't recognise the array its support for advanced column specifications in tables is limited, although it does a good job on simple tables. If you need a conversion for wordprocessors that can't read `.docx` files, this is a good place to start.

`latex2rtf.sourceforge.net/`;

TeX2Word by Kirill A Chikrii for Microsoft Windows is a commercial converter plug-in for *Word* to let it import TeX and LaTeX documents. The author's company claims that 'virtually any existing TeX/LaTeX package can be supported by *TeX2Word*' because it is customisable.

www.chikrii.com/products/tex2word/;

Pandoc: See the Note on p. 220pandoc.

pandoc.org/;

TeX4HT: See the list item 'TeX4HT' section 8.2.2 on page 235.

tug.org/tex4ht/.

One easy route into wordprocessing, however, is the reverse of the procedures suggested in the preceding section: convert LaTeX to HTML, which many wordprocessors can read easily, using any of the packages in section 8.2.2. Once it's in HTML, run it through *Tidy* (see the list item 'Using HTML' section 8.1.1.3 on page 222) to make it well-formed XHTML, add some embedded Cascading Style Sheets (CSS) styling to the header manually, and rename the file to end in the obsolete `.doc` filetype, which can fool *Word* into opening it natively as if it were a *Word* file.

8.2.2 Conversion to HTML

This probably runs *Word* a close second in frequency of demand. Conversion to HTML — or more probably XHTML — lets you put your LaTeX document on the web in a format everyone can read, but as with *Word*, not everything you can do in LaTeX can be represented in HTML, although with CSS3 you can get close.

lwarp: This is a LaTeX *package* for producing HTML v.5 (HTML5) output, using external utility programs for the final conversion of text and images. Strictly speaking this is an authoring package, not a conversion: you have to write your document using the commands defined in the package, rather than normal LaTeX. This makes it hard to use for handling existing documents, as they will need extensive editing before they can be processed. However, the package

Circular conversion

To the best of my knowledge, there is no off-the-shelf system that can convert circularly from LaTeX to *Word* or XML/XHTML *and back* and back again, and back again, without serious loss of formatting.[a] At each conversion, some document features will unavoidably be regularised to conventions of the target format which can no longer be represented in the source format. It may be possible for a very trivial document, but not for any real-life application.

This means that corporate, technical, or academic applications which depend on features of the *Word* interface such as Change Recording or the Style Margin cannot use LaTeX as a distributed editing format. However, after all edits have been made, the *Word* document can of course be converted to LaTeX for final typesetting.

[a] There *was* one once, in the mid-1990s, actually made by Microsoft: *SGML Author for Word*. It wasn't an editor as its name suggested, but a converter that used Named Styles to convert losslessly from SGML to *Word* and back, repeatedly, so that non-tech management could edit tech documents. Just as XML was taking off, they dropped it on the floor. Go figure. See `cora.ucc.ie/bitstream/handle/10468/1690/Human-Interfaces-to-Structured-Documents.pdf#page=393` and `xml.silmaril.ie/downloads/sgml-author-review.pdf` for more details

is under active development and supports a wide range of formatting packages.

The lwarp package is included in all TeX Live-based distributions.

LaTeX2HTML: *LaTeX2HTML*'s main task is to eproduce the document structure as a set of interconnected HTML files, so it is popular for creating multi-page web sites from a single large LaTeX document. It outputs a directory named after the input filename, and all the output files are put in that directory, so the result is self-contained and can be uploaded to a web server as it stands. It supports mathematics via images, and can deal with the built-in commands and a small range of packages.

github.com/latex2html/latex2html/;

TeX2page: This converts Plain TeX LaTeX. or Texinfo documents to HTML. Complex requirements can be configured in the TeX2page extension language (Common Lisp or Scheme). The authors have tried to make running TeX2page as similar as running LaTeX as possible.

github.com/ds26gte/tex2page;

TeX4HT: (TeX-for-HyperText) is an Open Source program which converts TeX and LaTeX documents to various kinds of XML and to *Libre Office* format (among others), which *Word* can open.

It operates differently from most other converters: It uses the TeX/LaTeX program itself to process the file, and handles conversion in a set of postprocessors for the common LaTeX packages. It can also output to XML, including TEI, *DocBook*, and the *Libre Office* and *Word* XML formats, and it can create Texinfo-format manuals.

By default, documents retain the single-file structure of the original, but there is a set of configuration directives to make use of the features of hypertext and navigation, and to split files for ease of use on the web.

tug.org/tex4ht/;

This is an Open Source translator from LaTeX to HTML by Luc Maranget at the *Institut national de recherche en sciences et technologies du numérique* [originally Institut de recherche en informatique et en automatique] (INRIA) in Paris. runs on Unix & GNU/Linux systems, and supports most of LaTeX 2_ε, including macro definitions. and outputs HTML5. It can be customised via style files using LaTeX code.

pauillac.inria.fr/~maranget/hevea/;

This is an Open Source translator running on most platforms, predominantly for converting mathematical LaTeX documents into HTML. works with both Plain TeX and LaTeX. Instead of using images of equations, it claims to translate them to actual HTML.

The author's link at `hutchinson.belmont.ma.us/tth/` seems to be dead, but the package is available as tth from CTAN.

GELLMU: Generalized Extensible LaTeX-Like MarkUp (GELLMU) is a LaTeX-like markup for authoring structured document types which can be converted to HTML, *DocBook*, TEI, or GELLMU's own LaTeX-like document type 'article'. The source language markup most closely resembles actual LaTeX source markup: much of the markup vocabulary is the same as that of actual LaTeX.

Documents are processed via the *SGMLS* Perl module and *elisp* routines in *Emacs*, and can output LaTeX, classic HTML, or XHTML+MathML.

Available as package gellmu from CTAN;

plasTeX: plasTeX is a LaTeX document processing framework. It comes bundled with an XHTML renderer (including multiple themes), as well as a way to simply dump the LaTeX document to a generic form of XML. Other renderers can be added, including Unix & GNU/Linux *man* pages, *Docbook*, and EPUB v3.

It works by processing LaTeX documents into XML Document Object Model (DOM)-like objects which can be used to generate various types of output. Many options can be set, including controlling splitting into multiple files and adding CSS files.

`plastex.github.io/plastex/`.

8.2.3 Conversion to XML

LaTeXML: LaTeXML provides a conversion to an intermediate XML vocabulary which can be used to create industry publishing

XML formats such as DocBook, TEI, and JATS, and even XHTML for EPUB v3.

LaTeXML is at math.nist.gov/~BMiller/LaTeXML/.

Tralics: *Tralics* comes from the Apics and Marelle teams at INRIA. It creates an XML document of its own design, representing everything it finds in the LaTeX file, using an error element type for anything it cannot handle. In a way it is similar to LaTeXML but using a different vocabulary, and it too has an extensive configuration mechanism to tune it for specific types or classes of document.

www-sop.inria.fr/marelle/tralics/;

TeX4HT: See the list item 'TeX4HT' section 8.2.2 on page 235.

tug.org/tex4ht/;

Pandoc: See the Note on p. 220pandoc.

pandoc.org/.

8.2.4 Conversion to plain text

When all else fails, you can always convert your document to plain, unmarked text.

Text extraction: If you have the full version of Adobe *Acrobat Reader* (or one of several other commercial PDF products), you can open a PDF file created by LaTeX, select and copy all the text, and paste it into your wordprocessor, and it will retain some common formatting of headings, paragraphs, and lists. This still requires the text to be edited into shape, but enough of the formatting should be preserved to make it worthwhile for short documents.

Otherwise, use the Open Source *pdftotext* utility to extract everything from the PDF file as plain (paragraph-formatted) text, and open that in a plain-text editor.

If you need HTML, there is a Java utility from Apache, the web server project, called *pdfbox* (see the list item 'Using PDF' section 8.1.1.3 on page 223), which can extract the text from a PDF document in HTML format, preserving the bold and italics, which can save a lot of time.

Last resort: strip the markup: At worst, the *detex* program on CTAN will strip a LaTeX file of all markup and leave just the raw unformatted text, which can then be re-edited. There are also programs to extract the raw text from DVI and PostScript (PS) files.

8.2.5 Authoring with LaTeX and XML

A number of systems have been mentioned which allow you to write your documents in a slightly different way to standard LaTeX but still using the same syntax (the list item 'GELLMU' section 8.2.2 on page 236 and the list item 'lwarp' section 8.2.2 on page 233, for example). There is also a document class called **internet** but reports indicate that it is no longer being developed.

8.3 Going beyond LaTeX

The reader will have deduced by now that while LaTeX is possibly the best programmable typesetting system around, the LaTeX file format is not generally usable with anything except the LaTeX program. LaTeX was originally written in the mid-1980s, about the same time as the Standard Generalized Markup Language (SGML), but the two projects were not connected. However, TeX and LaTeX have proved such useful tools for formatting SGML and more recently XML that many users chose this route for their output, using conversions written in the languages already mentioned in section 8.2 on page 230.

Unfortunately, when the rise of the Web in the early 1990s popularised SGML using the HTML, browser writers deliberately chose to encourage authors to ignore the rules of SGML. Robust auto-converted formatting therefore became almost impossible except via the browsers' low-quality print routines.

It was not until 1995–7, when the XML was devised, that it again became possible to provide the structural and descriptive power of SGML but without the complex and rarely-used options which had made standard SGML so difficult to program for.

XML is now becoming the principal system of markup. Because it is based on the international standard (SGML), it is not proprietary, so it has been implemented on most platforms, and there is lots of free software supporting it as well as many commercial packages. Like SGML, it is actually a meta-language to let you define your own markup, so it is much more flexible than HTML. Implementations of the companion Extensible Stylesheet Language (XSL) provide one route to PDF via Formatting Objects with Extensible Stylesheet Language: Formatting Objects (XSLFO) but at the expense of reinventing many of the wheels which LaTeX already possesses, so the sibling XSLT (Transformations) language can be used instead to translate to LaTeX source code, as shown in the example in section 8.1.3 on page 224. This is usually much faster than writing your own formatting from scratch, and it means that you can take full advantage of the packages and typographic sophistication of LaTeX. A similar system is used for the Linux Documentation Project, which uses SGML transformed by the Document Style Semantics and Specification Language (DSSSL) to TeX

The source code of this book, available online at `www.ctan.org/tex-archive/info/beginlatex/src/` includes XSLT which does exactly this.