8 Compatibility

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 Figure 8.1 on page 173, to Android devices, see Figure 8.2 on page 174) through all desktops and servers right up to the biggest supercomputers.

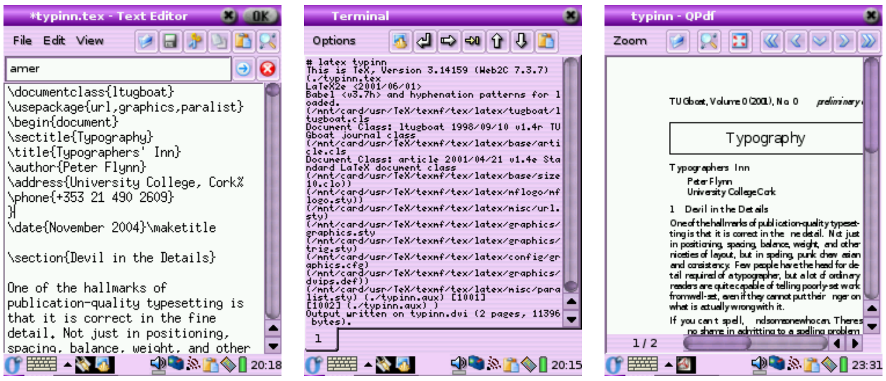


Figure 8.1: LaTeX editing and processing on the Sharp Zaurus 5500 PDA

```
\documentclass{ltugboat}
\usepackage{url,graphics,paralist}
\begin{document}
\sectitle{Typography}
\title{Typographers' Inn}
\author{Peter Flynn}
\address{University College, Cork%
\phone{+353 21 490 2609}
}
\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

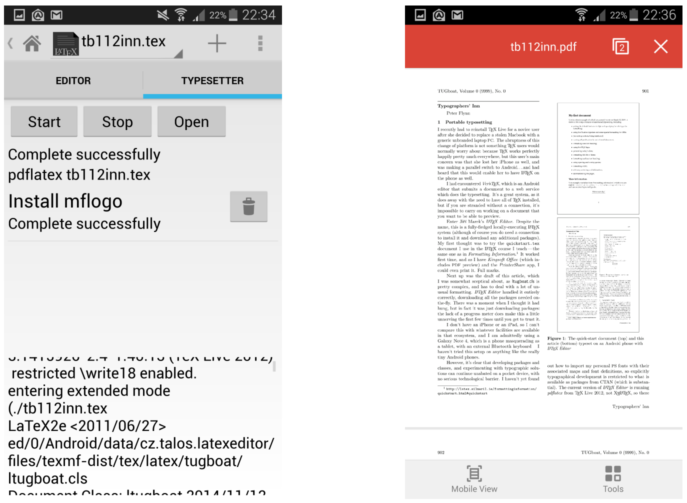
Peter Flynn
University College Cork

1 Devil in the Details

One of the hallmarks of publication-quality typesetting is that it is correct in the ne detail. Not just in positioning, spacing, balance, weight, and other niceties of layout, but in spelling, punk chew asian and consistency. Few people have the head for detail 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. There s 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
Document Class: ltugboat 2014/11/12
```

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 CD-ROM or 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, *PDFLaTeX*, and XeLaTeX 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. 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 section 8.2.

However, there is one system that does both, and includes a huge range of formats: *Pandoc* (pandoc.org/). This is a large library of Haskell routines for handling conversions, with a command-line front end. Supported formats include Word, OpenOffice/Libre Office, DocBook, InDesign, Markdown, and MediaWiki. Before trying the systems described in section 8.1 on page 175 and section 8.2 on page 182, see if *Pandoc* will handle your files. The exception is probably converting from XML to LATEX for which a robust XSLT2 script is really the only reliable solution.

8.1 Converting into LATEX

Before looking at one-way systems, see the earlier note about *Pandoc*.

There are several systems which will save their text in LATEX format. The best known is probably LYX, which is a wordprocessor-like interface to LATEX (not quite WYSIWYG, more What You See Is What You Mean). Both *AbiWord* (Linux and Windows) and *Kword* (Linux) have a very good Save As… LATEX output, and *OpenOffice* (all platforms) has a LATEX plugin, so they can be used to open Microsoft *Word* documents as well as their own format, and convert them to LATEX. Several maths packages like the *EuroMath* editor, and the *Mathematica* and *Maple* analysis packages, can also save material in LATEX format.

In general, most other wordprocessors and DTP systems either don't have the level of internal markup sophistication needed to create a LATEX file, or they lack a suitable filter to enable them to output what they do have. Often they are incapable of outputting any kind of structured document, because they only store what the text looks like, not why it's there or what role it fulfils. There are two ways out of this:

- ☐ Use the File › Save As… menu item to save the wordprocessor file as HTML, rationalise the HTML using Dave Raggett's *HTML Tidy*, and convert the resulting XHTML file to LaTeX with any of the standard XML tools (see below).

☐ Get the files into Word or ODF format, and write a transformation in XSLT to convert the internal XML into LaTeX. This is by far the most robust way to do it, but the quality of most wordprocessing files is poor when it comes to identifying which bits do what, which is what LaTeX needs, so some guesswork or heuristics may be needed.

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 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 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 a fourth way, suitable for large volumes only: send it off to the Pacific Rim to be scanned or retyped 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 the document is already in electronic form, but it's a good example of the problem of low quality of wordprocessor markup that this solution exists at all.

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, the native format for both *OpenOffice* 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.

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_ε. 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, and other wordprocessors. It runs on Macs, Linux, other Unix systems, and Windows.

8.1.1 Getting LaTeX out of XML

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.

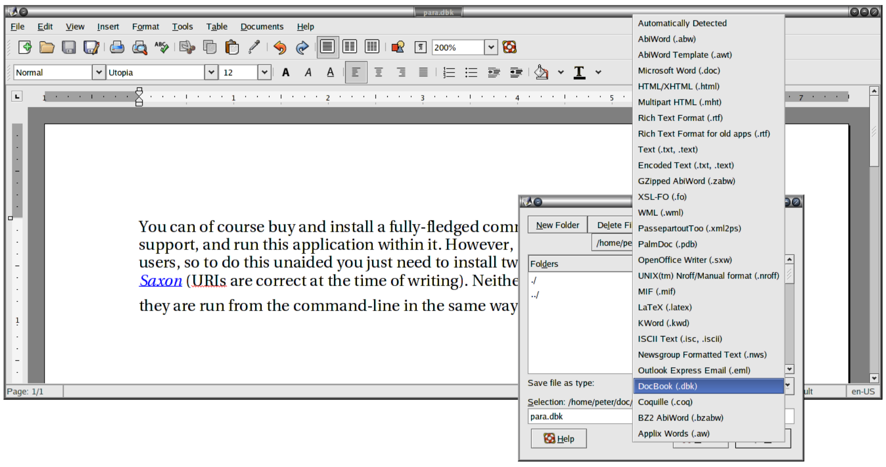
You can of course buy any fully-fledged commercial XML editor with XSLT support, and run transformations within it. However, this is beyond the reach of many individual users, although *oXygen* is available at a discounted price to academic sites.

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 above paragraph, as typed or imported into *AbiWord* (see Figure 8.3 on page 178). 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.

AbiWord can export in DocBook format, which is an XML vocabulary for describing technical (computer) documents — it's what I use for this

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



Save file as type: DocBook (.dbk)

Selection: /home/peter/doc/

para.dbk

You can of course buy and install a fully-fledged comr
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

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).[1]

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` (see the ASCII chart in Table E.1 on page 269).

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 Figure 8.4 on page 182).

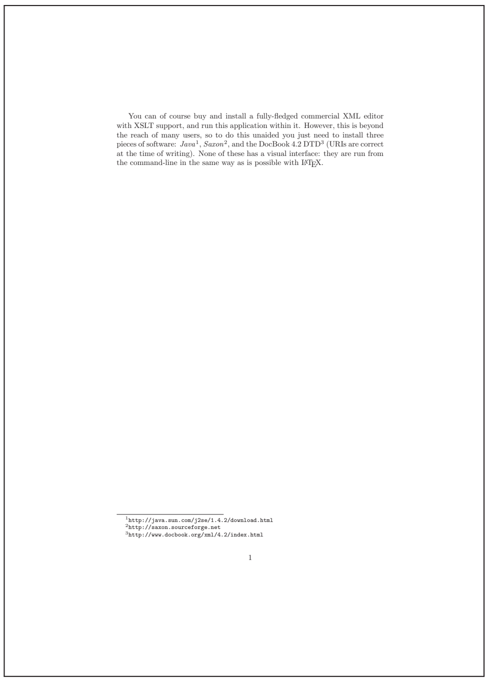
```
$ java -jar saxon9.jar -o para.ltx para.dbk para.xsl
$ pdflatex para.ltx
This is pdfTeX, Version 3.1415926-1.40.10 (TeX Live 2009/Debian)
```

[1] 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.

```
 restricted \write18 enabled.
entering extended mode
(./para.ltx
LaTeX2e <2009/09/24>
Babel <v3.8l> and hyphenation patterns for english, usenglishmax,
dumylang, nohyphenation, farsi, arabic, croatian, bulgarian,
ukrainian, russian, czech, slovak, danish, dutch, finnish, french,
basque, ngerman, german, german-x-2009-06-19, ngerman-x-2009-06-19,
ibycus, monogreek, greek, ancientgreek, hungarian, sanskrit, italian,
latin, latvian, lithuanian, mongolian2a, mongolian, bokmal, nynorsk,
romanian, irish, coptic, serbian, turkish, welsh, esperanto,
uppersorbian, estonian, indonesian, interlingua, icelandic, kurmanji,
slovenian, polish, portuguese, spanish, galician, catalan, swedish,
ukenglish, pinyin, loaded.
(/usr/share/texmf-texlive/tex/latex/base/article.cls
Document Class: article 2007/10/19 v1.4h Standard LaTeX document class
(/usr/share/texmf-texlive/tex/latex/base/size10.clo))
(/usr/share/texmf-texlive/tex/latex/ltxmisc/url.sty) (./para.aux)
[1{/home/peter/.texmf-var/fonts/map/pdftex/updmap/pdftex.map}]
(./para.aux))
</usr/share/texmf-texlive/fonts/type1/public/amsfonts/cm/cmr10.pfb>
</usr/share/texmf-texlive/fonts/type1/public/amsfonts/cm/cmr6.pfb>
</usr/share/texmf-texlive/fonts/type1/public/amsfonts/cm/cmr7.pfb>
</usr/share/texmf-texlive/fonts/type1/public/amsfonts/cm/cmti10.pfb>
</usr/share/texmf-texlive/fonts/type1/public/amsfonts/cm/cmtt8.pfb>
Output written on para.pdf (1 page, 54289 bytes).
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. In fact there is a set of templates already written to produce LaTeX from a DocBook file at `www.dpawson.co.uk/docbook/tools.html#d4e2905`

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 three pieces of software: *Java*[1], *Saxon*[2], and the DocBook 4.2 DTD[3] (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 LaTeX.

[1] `http://java.sun.com/j2se/1.4.2/download.html`
[2] `http://saxon.sourceforge.net`
[3] `http://www.docbook.org/xml/4.2/index.html`

1

```
\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

Before looking at one-way systems, see the earlier note about *Pandoc* on page 175.

Converting LaTeX to other formats is much harder to do comprehensively. As noted before, the LaTeX file format really requires the LaTeX program itself in order to process all the packages and macros, because there is no telling what complexities authors have added themselves (what a lot of this book is about!).

Many authors and editors rely on custom-designed or homebrew converters, often written in the standard shell scripting languages (Unix shells, Perl, Python, Tcl, etc). Although some of the packages presented here are also written in the same languages, 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 *ad hoc* translations, but it is easier to obtain consistency and a more sophisticated final product using lwarp, *LATEX2HTML* or *TEX4ht* — see below — or one of the other systems available.
- Embedding additional non-standard control sequences in LATEX source code may make it harder to edit and maintain, and will definitely make it harder to port to another system.
- All the above methods (and others) provide a fast and reasonably reliable way to get documents authored in LATEX onto the Web in an acceptable — if not optimal — format.
- *LATEX2HTML* was written to solve the problem of getting LATEX-with-mathematics onto the Web, in the days before MathML and math-capable browsers. *TEX4ht* was written to turn LATEX documents into Web hypertext — mathematics or not. The *lwarp* project aims to allow a rich LATEX document to be converted to a reasonable HTML5 interpretation, with only minor intervention on the user's part.

There is a very useful list of all the alternatives in the lwarp package documentation (Dunne, 2018, p. 55–57)

8.2.1 Conversion to *Word*

There are several programs on CTAN to do LATEX-to-*Word* and similar conversions, but they do not all handle everything LATEX can throw at them, and some only handle a subset of the built-in commands of default LATEX. Two in particular, however, have a good reputation, although I haven't used either of them (I tend to stay as far away from *Word* as possible):

- ☐ *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, and this program preserves layout and formatting for most LaTeX documents using standard built-in commands.

- ☐ Kirill A Chikrii's *TeX2Word* for Microsoft Windows is a converter plug-in for *Word* to let it open 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.

One easy route into wordprocessing, however, is the reverse of the procedures suggested in the preceding section: convert LaTeX to HTML, which many wordprocessors read easily. The following sections cover two packages for this. Once it's in HTML, you could run it through *Tidy* to make it XHTML, add some embedded styling using Cascading Style Sheets (CSS), and rename the file to end in `.doc`, which can fool *Word* into opening it natively.

8.2.2 The lwarp package

This LaTeX package produces HTML5 output directly, using external utility programs only for the final conversion of text and images. Mathematics may be represented by SVG files or *MathJax*.

The lwarp package is under active development and supports a wide range of formatting packages, but no attempt has been made to force LaTeX to provide for every HTML-related possibility, as HTML cannot exactly render every possible LaTeX concept.

8.2.3 LaTeX2HTML

As its name suggests, *LaTeX2HTML* is a system to convert LaTeX structured documents to HTML. Its main task is to reproduce the document structure as a set of interconnected HTML files. Despite using Perl, *LaTeX2HTML* relies very heavily on standard Unix facilities like the *NetPBM* graphics package and the pipe syntax. Microsoft Windows is not well suited to this kind of composite processing, although all the required facilities are available for download in various forms and should in theory allow the package to run — but reports of problems are common.

- The sectional structure is preserved, and navigational links are generated for the standard Next, Previous, and Up directions.
- Links are also used for the cross-references, citations, footnotes, ToC, and lists of figures and tables.
- Conversion is direct for common elements like lists, quotes, paragraph-breaks, type-styles, etc, where there is an obvious HTML equivalent.
- Heavily formatted objects such as math and diagrams are converted to images.
- There is no support for homebrew macros.

There is, however, support for arbitrary hypertext links, symbolic cross-references between 'evolving remote documents', conditional text, and the inclusion of raw HTML. These are extensions to LaTeX, implemented as new commands and environments.

LaTeX2HTML outputs a directory named after the input filename, and all the output files are put in that directory, so the output is self-contained and can be uploaded to a server as it stands.

8.2.4 TeX4ht

TeX4ht operates differently from *LaTeX2HTML*: it uses the TeX program to process the file, and handles conversion in a set of postprocessors for the common LaTeX packages. It can also output to XML, including Text Encoding Initiative (TEI) and DocBook, and the OpenOffice and WordXML formats, and it can create TeXinfo format manuals.

By default, documents retain the single-file structure implied by the original, but there is again a set of additional configuration directives to make use of the features of hypertext and navigation, and to split files for ease of use. This is a most powerful system, and probably the most flexible way to do the job.

8.2.5 Extraction from PostScript and PDF

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 *PDFLaTeX*, select and copy all the text, and paste it into *Word* and some other wordprocessors, and retain some common formatting of headings, paragraphs, and lists. Both solutions still require the wordprocessor text to be edited into shape, but they preserve enough of the formatting to make it worthwhile for short documents. Otherwise, use the *pdftotext* program to extract everything from the PDF file as plain (paragraph-formatted) text.

8.2.6 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.

8.3 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 182.

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 a direct route to PDF but at the expense of reinventing most of the wheels which LATEX already possesses, so the sibling XSLT can be used instead to translate to LATEX source code, as shown in the example in section 8.1.1 on page 177. This is usually much faster than writing your own formatting from scratch in XSL, and it means that you can take full advantage of the packages and 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.