

Another Look at L^AT_EX to SGML Conversion

Sebastian Rahtz

Elsevier Science Ltd

The Boulevard

Langford Lane

Kidlington

Oxford OX5 1GB

UK

Email: s.rahtz@elsevier.co.uk

Abstract

Publishers are starting to use SGML as their permanent form of storage for documents. How can L^AT_EX files be converted to an SGML instance? This paper discusses possible strategies, and describes an implementation by Elsevier Science of a system based on conversion in T_EX itself, and extraction of SGML code from the dvi file.

The problem

The work outlined in this paper concerns the translation of L^AT_EX files into SGML. “Why,” the T_EX user asks, “do you want to do this? L^AT_EX produces nice typeset pages, it’s a worldwide convention for document interchange, and T_EX’s math markup is a standard.” But “Why,” the SGML people ask, “do you want to use L^AT_EX at all? SGML is a real, ISO, standard, with much stronger validation and portability than T_EX, and with a growing set of excellent tools.” The problem arises for us at Elsevier Science because we are caught between two camps. On the one hand, many of our authors use plain T_EX or L^AT_EX to prepare articles, and we are asked to use their files. On the other hand, the majority of our papers come to us in a wide variety of markup and word-processor formats. We must therefore have a common markup format to convert them all into. We *could* simply use each file in its native format, be it T_EX, Word or Quark Xpress, but then there are serious practical difficulties:

1. We recognise, like any serious publisher, the long-term potential of an electronic archive of material, and we want our ultimate archive to conform to the most rigorous standards. We want all our files in the same markup system, so that we can develop tools for alternative publishing media, and tools to quality-check the markup.
2. We do not, by and large, do our own typesetting, but send electronic files to a range of external firms, most of whom prove to be very reluctant to accept T_EX files. We can however achieve efficient handling of our electronic files

by the typesetters if we supply a single type of fully-tagged file which contains all the information the typesetter needs.

3. We publish over 1000 journals, and a single L^AT_EX article may arrive for any of several hundred of these; it is not practical or logical to train all our production editors to deal with L^AT_EX in a sophisticated way, and it can frustrate the process if we have to push occasional articles onto L^AT_EX-experienced staff.
4. On a purely day-to-day basis, we cannot maintain all the possible hardware and software configurations at all production sites, or even keep all our production editors trained to use the many different systems and platforms.

The only possible solutions are therefore either to adopt an *ad hoc*, temporary, workaround like doing all our work in Word or Quark, or to invest in a total conversion to SGML, flexible enough to meet all our known and perceived future needs. Elsevier Science made their decision to adopt SGML some years ago, has developed its own DTDs, and is now engaged in the mammoth task of converting its journal production environment to a system which produces totally electronic files for typesetting or electronic publication. The standards and practices adopted are described in [3].

Consequently, to achieve a complete rigorous quality-controlled archive, L^AT_EX files have to be converted to SGML. But SGML is merely a language for describing markup, and ‘conversion’ could mean little more than syntactic conversion, whereby all L^AT_EX macros were changed to a corresponding SGML tag; thus `\section` is changed to `<section>`.

This is, in effect, the approach adopted by some of the apparent $\LaTeX \Leftrightarrow$ SGML systems — constraining the DTD to mimic \LaTeX . What it does not do is solve the problem of parsing the general \TeX syntax.

Another approach for the particular case of mathematics is to declare a single SGML element `<texmath>`, and leave the math markup exactly as is. This is attractive, but has two drawbacks:

1. Many authors have dependencies in their markup on their own definitions, or external style files; it is not trivial to totally ‘flatten’ all these references, but if they are left in, it is irritating to have to work around it when the document is used later (possibly *much* later, when the style files may have changed...). An even worse case is when the author *changes* definitions half-way through the paper, so that `\MyX` expands to `X_x~y` for the first few equations, but to `X_a~b` for the rest — quite legal, however inelegant.
2. If our archive is to still be useful in 20 years, it needs to be totally internally consistent and complete. The `<texmath>` notation assumes the existence in 20 years of a program able to parse it in the same way as \TeX does now — likely, but not under our control.

We are left with having to genuinely translate arbitrarily-complicated \TeX code into ‘real’ SGML; i.e., our DTD may bear no relation to the logical structure of \LaTeX , and we have to deal properly with mathematics and tables. The remainder of this paper discusses possible solutions, and describes the one we have adopted.

Possible solutions

There are four technologies we can adopt:

1. Throw away the electronic file, and retype the entire text from the printed copy direct into SGML.
2. Strip out all \TeX coding, and treat it like an unknown word-processor — leave all the words intact, but re-key all the markup, maths and tables.

This approach and the first are not as ridiculous as they sound to \TeX experts; they may well be the cheapest, quickest, most reliable solution for a small quantity of material. The startup cost is zero, and the unit cost may be high — but this may be preferable to a large development cost for a system which is not heavily used and needs maintenance. The reasons *not* to adopt it include the probable gain in processing time of an automatic conversion,

author satisfaction at decreased proof-reading, and the potential high cost of this approach for complex mathematics.

3. Write a new program to parse the \LaTeX , and write SGML. The advantages are:

- Writing parsers is a well understood process, although the ‘traditional’ *lex* and *yacc* tools are necessarily not suited to macro languages like \TeX ;
- We get an efficient, portable program which (if written well) can be maintained.
- Nothing gets into the output that we do not understand.

but there are the following disadvantages:

- Parsing \TeX is notoriously hard, because its interface is a macro programming language, and thus the syntax is extensible.
- There are no fully-successful implementations known, although there are many *partial* relevant solutions (such as Julian Smart’s *tex2rtf*).
- We would have to assume ‘normal’ \LaTeX , and hand-fix any strange files.

This approach could involve a two-stage program, the first attempting (a more or less impossible aim) to regularise the \TeX by expanding macros, and the second doing the parsing and translating. Aspects of this are encapsulated in the Perl script for `latex2html` (discussed in useful detail in [1]), but the philosophy of that system is to convert anything it cannot understand into a picture (via \TeX and PostScript to GIF), which means that it does not really try to be a robust \TeX parser.

4. Work with \TeX itself, to take advantage of the entire macro-processing language and input parser, but fix the output to produce SGML instead. Joachim Schrod [4] has exposed the myth that \TeX cannot be recoded in a more open system, and his implementation would be a good starting point for a \TeX -like program. We can also, however, simply write a special purpose \TeX format which defines all the interesting \TeX and \LaTeX constructs as SGML commands, and extracts the information from the `dvi` file after processing. This is the basis of the approach we are considering using. It has some considerable advantages:

- The task of parsing the \TeX language (expanding the macros) comes ‘for free’;
- The conversion from \LaTeX construct to SGML construct is done in the same language as the original definition; it can also

be customised by existing L^AT_EX-trained staff;

- The system is portable, easily modularised, and can cope with any form of T_EX (with the appropriate macro programming);

while on the downside

- The output is not regularised, since only some items (those foreseen) are trapped; everything else passes through T_EX's processes, which can be both good and bad;
- Writing sophisticated programs in T_EX is not easy, or well-understood, constraining future contract programmers to be T_EX-qualified;
- Some things may not be possible at the macro redefinition stage, notably in maths;
- Extracting material from the dvi file is far from easy.

The third approach has been tried and tested by Bart Wage (Elsevier Amsterdam) for a very large number of relatively simple documents (the front matter of articles), but the program was becoming unwieldy when trying to extend it to the full range of papers we publish. This then was the impetus for the system described in this paper. The approach of rewriting T_EX innards has been largely completed by ICPC (Dublin)¹ with a reworking of the basic WEB code, and this is certainly the most reliable plan of attack; however, it still requires extensive macro programming to get the most from the system, and so the majority of our work is unaffected.

A practical solution

Given the advantages and disadvantages outlined above, we decided to experiment with a T_EX-based approach; we knew that we had the existing traditional parsing program to fall back on, and we knew that ICPC were working on a radical solution altering T_EX. The main reason for wanting T_EX to do the parsing was that our author L^AT_EX markup is extremely variable, and essentially beyond our control; maximum flexibility and ease of *ad hoc* quick fixes was therefore important. It must be stressed that this is a solution tailored to a particular setup; it is not necessarily the best universal solution!

The general outline of our system is shown in Fig. 1. The steps, which are discussed in detail in the next section, are:

1. Roughly clean up the L^AT_EX, and impose some standard markup;
2. attach the `lat2cap` package, and run L^AT_EX (twice, for cross-referencing);
3. extract the SGML text from the DVI file;
4. clean up the SGML and parse it against an intermediate DTD;
5. transform to the final Elsevier article DTD, and parse again.

The justification for the extra stage of intermediate DTD is very dependent on the relationship between the L^AT_EX markup in use, and the DTD. In our case, we found that although we could match up most structures directly, some of the L^AT_EX was not amenable to a one-pass program.

Details of the system

Cleaning the L^AT_EX Since our approach is to re-define T_EX macros to produce SGML tags, we could in theory cope with any dialect of T_EX; in practice, however, our articles are structured in a clear, but complicated way, and it is preferable to get the information right before we look at converting it. We have maintained a set of L^AT_EX style files for some years for internal typesetting of many journals, which are used in conjunction with a single public preprint style, which implements all the constructs in a typographically simple way. The complexity is largely in the front matter, as shown in Figs. 2 and 3, where we present the input markup, and one style of typeset output. Almost all author files need some work in this area, so it makes sense to concentrate the translation *assuming* the editing has been done.

A more important consideration is the *richness* of the L^AT_EX. In theory, a valid L^AT_EX file:

```
\documentclass{article}
\begin{document}
  ABOUT CATS

  by

  Seroster

  once upon a time...
\end{document}
```

could be correctly translated to

```
<article>ABOUT CATS<p><p>by<p><p>
Seroster<p>once upon a time...
</article>
```

but this is hardly useful. For an article to have some value in an electronic warehouse, we must minimally identify in an unambiguous way the following elements:

¹ Contact Seamus McCague (seamus@icpc.ie) for details.

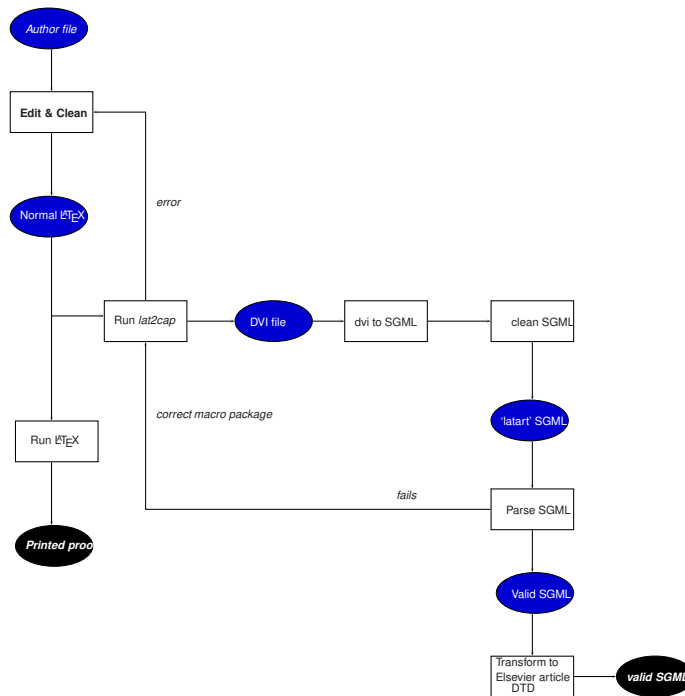


Figure 1: Processing scheme for LaTeX to SGML

1. the authors and their addresses;
2. the title, abstract and keywords;
3. section headings;
4. bibliographical references and citations;
5. figure references.

We may find it useful, in fact, to distinguish 4 ‘types’ of LaTeX file:

Bad It passes through LaTeX, but very low-level visual markup is used; this file says `Figure {\cmbxten 66}`

Clean Composed according to the commands in the LaTeX manual, but without utilizing the abstract features like cross-referencing, floating figures etc.; this file says `Figure~\textbf{66}`;

Rich Uses *all* the commands of the LaTeX manual, at the highest level possible for the context; this file says `\figname~\ref{fig66}`

Over-ripe This file has no spelling or grammatical mistakes and has not used any visual markup; it says `\FIG{66}`, with a single definition in a separate macro package allowing the caption to be placed above or below the figure.

It is clear that only ‘rich’ LaTeX or better can be translated to useful SGML; the alternative is to do all the cleanup at the SGML stage, but this has the disadvantage that we have no macro language; a 100 page LaTeX article with generic, but wrong, markup

may require a single change, the SGML output needs hundreds. This is discussed further below (“Where in the work-flow?”).

The macro package The bulk of our work consists in redefining every possible LaTeX command, at as high a level as possible. We present below selections from the macro package, showing those which are less than obvious.

Basic tools In order to simplify the TeX output, our first aim is to constrain and nullify the vast majority of low-level visual markup code. We set everything in LaTeX 2_ε’s T1 encoding, in a single font which is completely monospaced and allows no hyphenation. The use of T1 means that LaTeX will automatically translate all accented and non-ASCII characters, which may be entered with TeX macros, into 8-bit characters in the output. Our dvi processor will translate these into the appropriate SGML entities or accent tags.

```

1  \def\baselinestretch{1}
2  \DeclareFontShape{T1}{sgml}{m}{n}{<-> sgml}{}
3  \DeclareFontShape{OT1}{sgml}{m}{n}{<-> sgml}{}
4  \fontencoding{T1}\let\fontencoding=@gobble
5  \fontfamily{sgml}\let\fontfamily=@gobble
6  \fontseries{m}\let\fontseries=@gobble
7  \fontshape{n}\let\fontshape=@gobble
8  \fontsize{10}{10pt}\let\fontsize=@gobbletwo
9  \global\let\mathversion@gobble
10 \global\let\getanddefine@fonts@gobbletwo
  
```


```

\begin{document}
\volume{33} \issue{4} \ssdi{93}{E0045F}
\accepted{4 Kal 44} \received{2 Kal 44}
\firstpage{101} \runauthor{Cicero, Caesar and Vergil}
\runtitle{In Catilinam}
\begin{frontmatter}
\title{In Catilinam IV: A murder in 5 acts\thanksref{X}}
\author[Paestum]{Marcus Tullius Cicero\thanksref{Senate}}
\author[Rome]{Julius Caesar}
\author[Baiae]{Publius Maro Vergilius}
\thanks[X]{This is the history of the paper, etc etc}
\address[Paestum]{Buckingham Palace, Paestum}
\address[Baiae]{The White House, Baiae}
\address[Rome]{Senate House, Rome}
\thanks[Senate]{Partially supported by the Roman Senate}
\begin{abstract}
Cum M.~Cicero consul Nonis Decembribus senatum in aede
Iovis Statoris consuleret, quid de iis coniurationis Catilinae
sociis fieri placeret, qui in custodiam traditi essent, factum
est, ut duae potissimum sententiae proponerentur, una D.~Silani consulis
designati, qui morte multandos illos censebat,
altera C.~Caesaris, qui illos publicatis bonis per municipia
Italiae distribueudos ac vinculis sempiternis tenendos existimabat. Cum
autem plures senatores ad C.~Caesaris quam ad
D.~Silani sententiam inclinare viderentur, M.~Cicero ea, quae
infra legitur, oratione Silani sententiam commendare studuit.
\end{abstract}

\begin{keyword}
Cicero; Catiline; Orations
\end{keyword}
\end{frontmatter}

```

Figure 2: Example Elsevier front-matter markup



ELSEVIER

0 0 0 0 - 0 0 0 0 (0 0) X X X X X - X X

A Journal Of Research 33 (1995) 101–105
 © Elsevier Science Limited
 Printed in Great Britain. All rights reserved
 000-0000/95/\$9.50

In Catilinam IV: A murder in 5 acts¹

Marcus Tullius Cicero^{*,2} Julius Caesar[‡] Catullus[‡] Publius Maro Vergilius[†]

^{*} *Buckingham Palace, Paestum*
[†] *The White House, Baiae*
[‡] *Senate House, Rome*

(Received 2 Kal 44; accepted 4 Kal 44)

Cum M. Cicero consul Nonis Decembribus senatum in aede Iovis Statoris consuleret, quid de iis coniurationis Catilinae sociis fieri placeret, qui in custodiam traditi essent, factum est, ut duae potissimum sententiae proponerentur, una D. Silani consulis designati, qui morte multandos illos censebat

Altera C. Caesaris, qui illos publicatis bonis per municipia Italiae distribueudos ac vinculis sempiternis tenendos existimabat. Cum autem plures senatores ad C. Caesaris quam ad D. Silani sententiam inclinare viderentur, M. Cicero ea, quae infra legitur, oratione Silani sententiam commendare studuit.

Key words: Cicero; Catiline; Orations

Figure 3: Elsevier front-matter typeset

```
11 \selectfont
12 \def\selectfont{}
```

Within our SGML DTD there is no requirement for typesize markup, so for authors who use size-changing commands we simply nullify them:

```
1 \let\normalsize\relax
2 \let\tiny\relax
3 \let\scriptsize\relax
4 \let\footnotesize\relax
5 \let\small\relax
6 \let\large\relax
7 \let\Large\relax
8 \let\LARGE\relax
9 \let\huge\relax
10 \let\Huge\relax
```

Writing SGML tags `\SGML` is a special command to emit SGML tags; it is important to do this, rather than use the actual `<`, `>` and `&` characters, as those will be translated to entities by the later processing. Note that our dvi processor will emit the parameter of the `\special` command as literal text.

```
1 \def\ENT#1{\leavevmode\special{&#1};}
2 \long\def\SGML#1{\leavevmode
3 \special{<#1\special{>}}
```

Headings Headings are treated differently to \LaTeX in the Elsevier DTD; there is only one heading tag, but nesting is used to indicate hierarchy. We need to trace the section depth and close open levels at the right time. The actual *title* of the section is not an attribute or context of the tag, but comes in a separate section title tag. We therefore define a single macro which takes each sectioning command, with a level, and keeps track of what level we are at, ending and starting heading elements as needed.

```
1 \def\@dblst#1#2{\@Section*{#1}{#2}}%
2 \def\@Section{#1}{#2}}
3 ...
4 \def\section{\expandafter\secdef
5 \@dblst{2}{section}}
6 ...
7 \def\@Section#1#2#3{%
8 \ifnextchar[%
9 {\@Section{#1}{#2}{#3}}%
10 {\@Section{#1}{#2}{#3} []}%
11 }
12 \newcount\current@sectionlevel
13 \current@sectionlevel=99
14 \def\@Section#1#2#3[#4]#5{%
15 \ifnum\current@sectionlevel=99\else
16 \loop
17 \ifnum\current@sectionlevel>#2
18 \typeout{Section level #2 inside
19 \the\current@sectionlevel}%
20 \advance\current@sectionlevel by -1
21 \SGML{/sec}%
22 \repeat
```

```
23 \ifnum\current@sectionlevel=#2
24 \SGML{/sec}
25 \fi
26 \fi
27 \global\current@sectionlevel#2
```

As we will see later, the cross-referencing mechanism is altered to use unique tags for all sections, not just those labelled with `\label`.

```
1 \refstepcounter[secr]{#3}%
2 \SGML{sec id=#3.\@currentSlabel}%
3 \SGML{st}#5\SGML{/st}%
4 \SGML{p}%
5 }
```

List environments These can be dealt with straightforwardly, as the match between the DTD and \LaTeX is almost 100%.

```
1 \def\item{%
2 \ifnextchar [ {\@item}{\SGML{li}}%
3 }
4 \def\@item[#1]{\SGML{li id=#1}}
5 \renewenvironment{enumerate}
6 {\SGML{l type=ord}}{\SGML{/l}}
7 \renewenvironment{itemize}
8 {\SGML{l type=unord}}{\SGML{/l}}
9 \renewenvironment{description}
10 {\SGML{l type=def}}{\SGML{/l}}
11 \def\quote{\SGML{qd}}
12 \def\endquote{\SGML{/qd}}
13 \let\quotation\quote
14 \let\endquotation\endquote
15 \def\p{\SGML{p}}
```

Font changes The Elsevier DTD supports a wide range of typeface changes in the same way as \LaTeX does:

```
1 \def\it{\SGML{it}\aftergroup\ENDTAG}
2 \def\bf{\SGML{b}\aftergroup\ENDTAG}
3 \def\s1{\SGML{it}\aftergroup\ENDTAG}
4 .....
5 \def\ENDTAG{\SGML{/}}
6 \let\/=\relax
7 \def\textrm#1{\SGML{rm}#1\SGML{/}}
8 \def\textsf#1{\SGML{ssf}#1\SGML{/}}
9 \def\texttt#1{\SGML{ty}#1\SGML{/}}
10 .....
11 \def\emph#1{\SGML{it}#1\SGML{/}}
```

Bibliographic citations

```
1 \def\citex[#1]#2{%
2 \SGML{bbr id="bib-#2"}#1%
3 }
4 \def\check@bb{%
5 \ifin@bb\SGML{/bb}\in@bbfalse\fi
6 }
7 \def\@lbibitem[#1]#2{%
8 \check@bb
9 \SGML{bb id="bib-#2"}%
10 \global\in@bbtrue
```

```

11 }
12 \def\@bibitem#1{%
13   \check@bb
14   \SGML{bb id="bib-#1"}%
15   \global\@in@bbtrue
16 }
17 \def\thebibliography#1{%
18   \SGML{bm}\SGML{bibl}%
19   \let\\relax
20 }
21 \def\endthebibliography{%
22   \check@bb
23   \SGML{/bibl}%
24 }

```

Cross-referencing Anything which can be referenced advances some counter; we overload this to put in an SGML id, and make a note of that for later use in `\label`. An extra parameter is written to the `.aux` file, adding an identifier to the literal page number and section number. This will fail badly if `\theS<name>` does not expand to a sensible reference. This means that classes or package which introduce new elements need to define an equivalent `\theS<name>` for every `\the<name>`

These shenanigans are to make sure section numbers, etc., are always arabic, separated by dots. Who knows how people will set up `\@current label`? If they put spaces in (quite legal) then the processor will get upset.

```

1  \def\pageref#1{\SGMLWarning{pageref}}
2  \let\vref\ref
3  \long\def\footnote#1{%
4    \refstepcounter{fnr}{Sfootnote}%
5    \SGML{fn id=Sfootnote.\@currentSlabel}%
6      #1\SGML{/fn}%
7  }
8  .....
9
10 \def\@setref#1#2#3{%
11   \ifx#1\relax
12   \protect\G@refundefinedtrue
13   ??
14   \@latexwarning{Reference '#3' on page
15     \thepage \space undefined}%
16   \else
17   \special{<}%
18     \expandafter\@secondoftwo#1{%
19       \expandafter#2#1\null}%
20   \special{>}%
21   \fi

```

But all this is very flaky, and open to abuse. Styles like `subeqn` will mess it up, for starters. Appendices are an issue, too. We just hope to cover most situations. We can at least cope with the standard sectioning structure, allowing for `\part` and `\chapter`.

```

1  \@ifundefined{thepart}{\%
2    \newcommand\theSpart{\arabic{part}}}%
3  \@ifundefined{thechapter}{\%
4    \newcommand\theSsection{\arabic{section}}%
5    \newcommand\theSfigure {\arabic{figure}}%
6    \newcommand\theStable  {\arabic{table}}%
7  }{\%
8    \@ifundefined{thepart}%
9      {\newcommand\theSchapter
10       {\arabic{part}.\arabic{chapter}}}%
11     {\newcommand\theSchapter
12      {\arabic{chapter}}}%
13  \newcommand\theSfigure
14    {\theSchapter.\arabic{figure}}%
15  \newcommand\theStable
16    {\theSchapter.\arabic{table}}%
17  \newcommand\theSsection
18    {\theSchapter.\arabic{section}}%
19
20    }
21  ...
22  \newcommand\theSequation
23    {\theSsection.\arabic{equation}}%
24  \newcommand\theStheorem
25    {\theSsection.\arabic{theorem}}%
26  \newcommand\theSthm
27    {\theSsection.\arabic{thm}}%
28  \newcommand\theSenumi
29    {\theSsection.\arabic{enumi}}%
30  ...
31  \newcommand\theSSfootnote
32    {\arabic{Sfootnote}}%
33  ...
34  \let\theSHmpfootnote\theSSfootnote
35  \let\S@refstepcounter\refstepcounter
36  \def\refstepcounter{\@ifnextchar[%
37    {\@refstepcounter}%
38    {\@refstepcounter[]}}%
39  \def\@refstepcounter[#1]#2{%
40    \ifx\#1\\\edef\@sgmlname{#2}%
41    \else
42      \def\@sgmlname{#1}
43    \fi
44    \S@refstepcounter{#2}%
45    \S@makecurrent{\@sgmlname}{#2}%
46  }
47  \def\S@makecurrent#1#2{%
48    \edef\@currentSlabel{%
49      \csname theS#2\endcsname}%
50    \global\edef\@currentSref{%
51      #1 id="#2.\expandafter\strip@prefix
52      \meaning\@currentSlabel"%}
53  }
54  \def\label#1{%
55    \@bsphack
56    \protected@write\@auxout{%
57      {\string\newlabel{#1}{%
58        \@currentSref}}}%

```

```
59 \@esphack
60 }
```

Mathematics and tables Surprisingly, the majority of mathematics requires fairly trivial redefinition; powerful and complicated as TeX's math typesetting is, we simply ignore it.

```
1 \catcode'\^=13 % circumflex for superscript
2 \catcode'\_ =13 % underline for subscript
3 \def\frac#1#2{\SGML{fr}\SGML{nu}#1
4 \SGML{de}#2\SGML{/fr}}
5 \def _#1{\SGML{inf}#1\SGML{/inf}}
6 \def ^#1{\SGML{sup}#1\SGML{/sup}}
```

The real implementation of $\hat{\ }$ in our package is actually rather more complicated than presented here, because it has to cope with markup like this:

```
x^\frac{a}{b}
```

which is in fact only barely tolerated in L^AT_EX. In this situation, the parameter to $\hat{\ }$ ends up as `\frac` unless we take special precautions.² Many macros just output the right SGML entity.

```
1 \def\alpha{\ENT{alpha}}
2 \def\beta{\ENT{beta}}
3 .....
4 \def\vartriangleleft{\ENT{vartriangleleft}}
5 \def\vartriangleright{\ENT{vartriangleright}}
6 \def\vartriangle{\ENT{vartriangle}}
7 \def\veebar{\ENT{veebar}}
8 \def\left#1{\SGML{fen}\SGML{cp style="#1"}}
9 \def\right#1{\SGML{/fen}}
```

The remainder of the mathematical work, including equation labelling and numbering, and tables, involves quite straightforward macro programming, albeit somewhat byzantine at times.

TeX accentuation is handled by a similar scheme in our DTD.

```
1 \def\acute#1{\SGML{a}\SGML{ac}#1
2 \SGML{ac}\ENT{acute}\SGML{/a}}
3 \def\grave#1{\SGML{a}\SGML{ac}#1
4 \SGML{ac}\ENT{grave}\SGML{/a}}
5 ...
```

DVI to ASCII We have now written a typeset page of text in a monospace font interspersed with `\special` commands relating to `<` and `>` characters; we are not at all interested in the layout, we just want the words, and all vertical and horizontal spacing turned to simple spaces. There have been a variety of 'dvi2tty' programs written over the years, but most of them are aiming to produce crude ASCII layout; after some experimentation, Geoffrey Tobin's excellent `dv2dt` program was found. This, and a

² We owe the solution to Alan Jeffrey, David Carlisle, Chris Rowley and Michael Downes, almost simultaneously. Seldom can such a concentration of L^AT_EX brain-power have been used to crack such a small nut.

companion `dt2dv`, provides a reliable, and easy to understand, text representation of a dvi file which can even be edited and turned back to dvi. Working with an ASCII representation means that it is easy to check and debug one's work, and writing a parser in Flex is simple. A flavour of the 'dt' language can be gleaned from this example; text is in round brackets, and the `w` commands are horizontal spacing.

```
special1 1 '<'
(fn)
(id=Sfootnote.1)
special1 1 '>'
(Everyone)
w3 218453
(likes)
w0
(cat)
w0
(meat)
special1 1 '<'
(/fn)
special1 1 '>'
w0
special1 1 '<'
(li)
special1 1 '>'
(dogs)
w0
special1 1 '<'
(/1)
special1 1 '>'
```

Our parser needs to read this, output the words and `\special` commands, insert spaces, and convert any non-ASCII characters to SGML entities. Readers familiar with Flex will see from the following fragment how trivial this is:

```
<SKIP>"\"( " BEGIN TEXT ;
<SKIP>"special1 "[0-9]+" "'
{ BEGIN SPECIAL ; } ;
<SPECIAL>"["~']*" { printtext(yytext); };
<SPECIAL>"'" BEGIN SKIP ;
<TEXT>"\" \" BEGIN SKIP ;
....
<SKIP>"s1 249"
printtext("<a<ac>u<ac>&grave;</a>") ;
....
<SKIP>^z { printtext("<P>"); };
<SKIP>^[rwy] { printtext(" "); };
<SKIP>. |
<SKIP>\n { } ;
```

Cleaning the SGML Unfortunately, L^AT_EX has a strange way with paragraphs (as indeed do SGML DTDs), and understanding the vertical and horizontal spacing in a dvi file is slightly fraught; therefore we run a simple cleanup to remove extraneous

<p>s and empty tags, and add some miscellaneous markup.

Transforming the SGML Now to the crux of any L^AT_EX to SGML program — what is the target DTD? Do we go for a translation all the way, or adopt the strategy of the ‘Rainbow’ tools, and work to an intermediate DTD from which we can transform to the desired result? It is important to distinguish between transforming, and upgrading; if we write ‘poor’ SGML, we cannot reliably upgrade it to ‘rich’ SGML, whereas ‘rich’ SGML in dialect A can certainly be transformed to dialect B with no problem. This is one of the tasks for which DSSSL will be useful in the longer term, but for this purpose we use a public domain Perl5 module for manipulating SGML instances. To ensure that the output is correct, the file is parsed using James Clark’s *nsgmls*, and the standard ESIS output from that program is then read by the transformer, writing the final output. This is of course again parsed against the real DTD before the program run is deemed successful.

A simple example may suffice to show why this last transformation stage is necessary. A typical piece of L^AT_EX often contains `eqnarray` structures, or multi-line equations. In L^AT_EX, lines end with a `\\`, and only then do we find if they are numbered. In the Elsevier DTD, this would be represented as a series of math displays nested inside an outer display. No doubt one *could* program this in L^AT_EX, but it is simpler to convert the end of line `\\` to a special tag, and transform the result later.

A harder problem, which we do not in fact face (since we know it can be fixed in the editing phase), is to deal with

```
1 {A \over B} 2
```

with T_EX programming, since in the simple redefinition of macros we do not realise we are in a ‘fraction’ until the group has started, and we have no way of back-tracking to put in a tag at the start of the construct. By contrast, the L^AT_EX notation

```
1 \frac{A}{B} 2
```

is easy to transform immediately to

```
1<fr><nu>A<de>B</fr>2
```

Where in the work-flow?

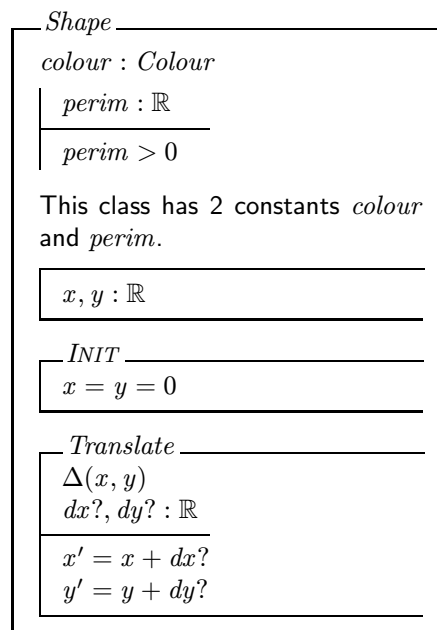
Once the technical problem is solved, we still have to consider where in the work-flow it is used. These are some of the issues we have to face in a production environment:

1. Who runs the converter? Disk administrator, desk editor or special L^AT_EX editor? Do we send it all to an outside contractor?

2. What if it goes wrong? Are problems fixed in L^AT_EX or SGML? Does the SGML editing tool permit illegal SGML to be imported?
3. Is copy-editing done in SGML or L^AT_EX? Some regular markup by the author may well be far easier to change once in L^AT_EX than the expanded form in SGML.

There are also some miscellaneous issues to consider:

1. What about author graphics created in T_EX? These can be very sophisticated. The best solution would be to run the article through L^AT_EX and extract the pages as PostScript output, but this requires some confidence with, and knowledge of, T_EX.
2. What do we do with L^AT_EX constructs which have no corollary in the Elsevier DTD? Two obvious examples are chemical structures, and the the formal Z schemas used in computer science, like this:



Conclusions

There remain, of course, three questions:

1. Does this approach work, other than as a toy?
2. Are we actually using it?
3. Where can it be found?

In our experience, no T_EX-related project works 100% reliably without manual intervention; this one is no exception. The approach works, but it is likely that a sophisticated T_EX programmer could quickly produce a file which produced inappropriate output. However, we do claim that it succeeds in its aim,

and that we have a usable, maintainable, and flexible tool.

To enable the generation of SGML from L^AT_EX, we are now seriously testing this tool, and using it in pilot projects; it remains to be seen whether or not it will move into full production, or whether the maintenance implications will be deemed too inefficient for the potential volume of material.

The approach outlined in this paper relies almost entirely on T_EX programming, and the principles followed are all demonstrated above. The complete style file has *not* been placed in the public domain, because it represents a not inconsiderable (ongoing) investment by Elsevier Science, and because it is not a general-purpose tool to translate L^AT_EX to arbitrary DTDs. However, those interested in discussing this approach are invited to contact Sebastian Rahtz directly. T_EX wizards will learn that many problems are still to be solved, or dealt with in a considerably more elegant way.

Acknowledgements

The genesis of this project was work done by Michel Goossens at CERN in 1993 to convert L^AT_EX documents to HTML, using the ‘dvi to ASCII’ program of Alexander Samarin and Basil Malyshev. At Elsevier, detailed discussion with Bart Wage turned the idea into reality, and he wrote the second half of the system. The L^AT_EX package shares material with the author’s *hyperref* package (see [2]). Peter Flynn and Jonathan Fine, in their different ways, have made contributions to the T_EX and SGML relationship which impacted on this work. Joachim Schrod read this paper and gave extremely helpful feedback on it, and the subject in general. Later versions of the package will hopefully take advantage of his many good suggestions.

References

- [1] Michel Goossens and Janne Saarela, “From L^AT_EX to HTML and Back”, *TUGboat*, 16(3), 1995.
- [2] Yannis Haralambous and Sebastian Rahtz, ‘L^AT_EX, hypertext and PDF, or the entry of T_EX into the world of hypertext’, *TUGboat*, 16(2), 1995.
- [3] Martin Key, ‘Theory into Practice: working with SGML, PDF and L^AT_EX at Elsevier Science’, *Baskerville* 5 (2), 1995.
- [4] Joachim Schrod ‘Towards interactivity for T_EX’, *TUGboat* 15(3), 1994 (Proceedings of the 1994 T_EX Users Group Annual meeting), 309–317.