Bijlage 8
Minimal markup
expansion in the gullet, aha!

Kees van der Laan
cgl@hetnet.nl

abstract

A plea is made for a reappraisal of TEX's capabilities of expansion in the gullet of minimal marked up scripts into complete marked up scripts. Attention is focused on expansion of implicitly marked up table data by spaces and e-o-l-s into data separated by \cs and \rs, the abstract but explicit column and row separators, respectively. The ultimate aim is that the processing tool can't be distilled from the 'marked up' script.

keywords

BLUe, crosswords, expansion, fifo, look ahead, macro writing, minimal markup, mouth processing, preprocessing, tables, tail recursion

Introduction

Who cares about minimal markup as long as the results in print look beautiful?

Well, . . . I do.

The reason why I care about minimal markup comes from various experiences.

The main reason is: it's elegant, IMHO with all respect, when a user can prepare his copy in such a way that we can hardly tell that he works with (La)TEX. This comes close to abstraction from the processing tool, and I for one consider this beneficial, if not for the entailed flexibility, or the ease in converting a BLUe script into a MAPS submission, with the ultimate goal that conversion is no longer needed.

Second, I like to read marked up scripts next to the results in print. Redundancy, to say the least, in markup does not contribute to readability nor clean scripts.

Third,many a (La)TEX result does not look beautiful at all, and maybe because authors got lost in the complexity of too much and redundant markup.

Fourth, now and then I could just generate the data for tables as function of its size by TEX, and so the user does not have to supply the data for those tables, let alone the markup.[1]

And last but not least, it allows you to become 'lazy,' to be able to forget about most of the details of the tool, but . . . only after you have matured, gained a thorough understanding.

In BLUe I adopted, similar to Knuth, that the \blueheads take as end separator of the heading title a blank line, and so on.[2]

There is one example I'm particular fond of: the minimal markup for crosswords.[3] For crosswords I considered it particularly convenient that a user only has to provide the data between the tags \bdata and \edata,[4] without explicit separators. Just the letters and the * to denote a black cell, with the column separators implicit because the elements consist of only one character, and as row separators the end of lines, e-o-l-s for short. TEX's mouth–gullet while expanding should insert the tags required by subsequent macros.[5] In general I found it beneficial to abstract from the markup of the table data in \cs and \rs, the column and row separators, respectively.

Disclaimer. Alas, as yet I can't read the TEX WEB source[6] so I don't know what \noexpand\cs really entails. Obviously, the gullet refrains from expansion. But eventually it has to be expanded and processed, of course. All below is biased by my intuition and has been verified, interactively.

Example *(Crosswords)* For the discussion at the moment concentrate your attention on how the data between \bdata and \edata, are specified, that is each row as such on a line.

1 See my note 'Parameterized data for tables in TEX.'

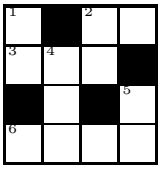
2 A trifle in markup, but it paved the way towards the approach of starting without markup. More advanced is the need for processing on-the-fly, that is with catcodes to be fixed while processing. For that some more markup is needed: the two-part variant consisting of the pair \beginhead \endhead, with its \head{. . . } variant.

3 I know of work done and published in TUGboat by Brian Hamilton Kelley, and I myself published a note about it in the EuroTEX'92 proceedings, in MAPS 92.2, and included a crossword example in the Publishing with TEX user guide, PWT for short, which comes with BLUeTEX. Crossword macros are included as one of the tools in BLUe's tools.dat.

4 An even more minimal variant, a \bluedata macro with a blank line as end separator of the data can be provided on top as follows: \def\bluedata#1\par{\bdata#1\edata}.

5 A white lie. TEX has coupled the mouth–gullet–stomach–gastro-intestinal track by 'pipes.' As soon as some expansion has been done the stomach already starts promulgating it, and so on. In the old computer terminology TEX's mouth functioning can be seen as a preprocessor.

6 METAFONT and PostScript ranked higher on my priority list.



The (left) puzzle above is obtained essentially via[7]

```
\bdata%
P*On
DEk*
*n*S
Edit
\edata \crw
```

Note that the numbers for the clues in the puzzle are generated automatically. The user must take care that these numbers correspond with the clues by using capital characters in the data at the right places—these places get their numbers on turns, rowwise—and that the clues take the correct numbers.

While glancing through this note the markup for the crossword might look so natural to you that you might wonder

> Is that all?

Indeed for a typewriter the above supplied data, for example, is all what is needed, if we consider * as black cell. However, for the puzzle with numbers for the clues inserted we are at loss, definitely when we wish not to bother about it as user. But, ... it needs a touch of knowledge, awareness, wisdom and discipline not to pollute the data by markup, while TEXing for beautiful results, such as black cells, automatic generation and typesetting of numbers for the clues, rules, scaled variants to be placed anywhere on the page, and so on, while preserving simplicity and flexibility.

Because the minimal markup approach in TEX is not widespread, we have to push advanced (plain) TEXies to write the macros for minimal markup. To use the macros once they are there is simple by nature. Although this note also deals with how to write these macros I hope this will not put off the push-the-button type of user.

Why?

In the preface of *The TEXbook* Knuth already stated

> If you are preparing a simple manuscript, you won't need to learn much about TEX at all;

with the consequence of few markup.

Apart from the lovesong on the elegance of minimal markup, I'll unevitably touch on the subject of macro writing. To my experience there are no two TEXies alike who write macros in the same spirit.[8] This is a weakness of macro writing in TEX, IMHO with all respect, because it is much easier to write macros from scratch of your own than to understand the (perceived) idiosyncrazies of your colleagues. Some would call this the strength of TEX's macro language, this richness of possibilities, but I think it really inhibits to create for example a library of TEX macros.[9]

I favour a discipline in TEX macro writing and I hope that this note will contribute towards the macro writing for generating markup automatically, if not for a library of TEX macros. In absence of this library[10] I collect these tools and macros in BLUeTEX, all within the framework of a coherent spirit and consistent philosophy.

I like to code macros for general paradigms in programming, such as FIFO—First-In-First-Out. The FIFO principle implemented as an expandable macro, together with its variants with looking ahead functionality, will be the main TEXniques to be used in this note.[11]

Disclaimer. The quotation taken from the preface of *The TEXbook* continues with

> ... on the other hand some things that go into printing of technical books are inherently difficult, and if you wish to achieve more complex effects you will want to penetrate some of TEX's darker corners.

So, what we can reasonably go for is that

> simple scripts should be marked up simple

and that for inherent difficult scripts complexity will cross our way. But, ... IMHO, with all respect, do hide complexity in macros, such that only the invokes will appear in the markup, or even better that these invokes be inserted automatically by TEX's expansion mechanism, wherever feasible.

7 With default `\puzzletrue`. With `\puzzlefalse` the right figure—the solution—will be obtained.

8 Except for the LATEX team, I presume.

9 The LATEX users are not so much interested in how it is done as long as it does what you want. That is different from what I'm up to. I like to understand and be able to read the code as well.

10 The TEX archive—CTAN—provides mostly style files for LATEX, or other collections like my BLUe. A detailed taxonomy for a library is still lacking. How would you classify FIFO macros for example? (Of course under fifo, but ...)

11 Knuth already had his `\dolist` macro, but because of assignments it is not completely processed in the mouth. Maybe that is not so relevant because of the coupling via pipes of TEX's digestive phases. Knuth's `\dolist` is more general for sure, but for down to earth use my straight implementation of fifo can do a lot. See *The TEXbook*, ex 11.5, or `\ctest` *The TEXbook*, 376. In MAPS 92.2, I have recasted `\dolist` in FIFO terminology, while preserving its assignments, of course, in 'FIFO and LIFO sing the BLUes.'

What is on?

I aim at that the reader after completion will understand how TEX can be used for expansion, to transform for example

ABC ⟶ \ls A \ls B \ls C

or

ABC ⟶ A \cs B \cs C

That is, to insert before each element a list separator—\ls—or to insert between elements a control sequence—\cs. These eventually nested, which is actual for the case of crosswords, and for the data proper part of tables.[12]

Expansion in TEX's mouth, well gullet, is a very powerful tool for transforming the minimal, or implicit, markup into the complete representation as required by subsequent macros, such as \halign. In other words to get the script ready for further execution.

I assumed that the reader starts from the same viewpoint as me: to supply data is one thing, to format them is another. TEX's expansion in the mouth–gullet is a way to bridge the gap.

Minimal markup by Knuth

Knuth in his markup has amply made use of blank lines, see for example *The TEXbook* file. Nice because of the implicit structure, pleasing for the eye.

A TEX paragraph starts usually by its first character and is ended by the first blank line.[13] No explicit markup at all, not to mention that kerning goes automatically!

The markup for a chapter of *The TEXbook* starts with \beginchapter followed by the chapter title and ended by a blank line.

Another occurrence of minimal markup has to do with options. Knuth's \begindisplay \endisplay pair is quite flexible with variations there when needed, which otherwise won't hinder. Just one line? That is fine. More lines? Separate them just by \cr. More columns? OK too, separate columns by &. Then there is the possibility for inclusion by supplying after \begindisplay on the same line.

In general Knuth implemented options via the use of token variables—his \every...-s—to be inspected at appropriate places.

> If you don't need options, Knuth's implementation of them won't hinder, you don't have to pay for what you don't use, also known as Samelson's principle.[14]

Maybe you will consider the above not so relevant, but IMHO, with all respect, when paying attention to minimal markup your scripts will become cleaner and cleaner, if not for clearer and clearer. Moreover, it gives more and more pleasure in (re)reading them. You won't no longer be a victim of the curley braces mania, with its drawback of the 'non-matching braces' error message.

Disclaimer. There is one approach of Knuth which IMO is not minimal. It is about that input has to be repeated when the result in print as well as how this was achieved are displayed, as is the case on many places in *The TEXbook*.

Automatic inclusion of markup for list processing

I needed this for the first time when typesetting the Tower of Hanoi game.[15] Knuth used the active list separator on several occasions, see for example *The TEXbook*, Appendix D.2 List Macros. Maybe, the most notorious practical use is his writing of the answers of the exercises to a file preceded by the list separator \ansno, see *The TEXbook*, 422. As we all know this file is the input for the Appendix A: Answer to All the Exercises. I used the (active) list separator in my database approach of TEX tools and formats, not to mention my literature database, from where references can be selectively loaded, and of course in selecting addresses from my 'addressbook.'

Maybe, the viewpoint that this functionality is of importance to allow minimal markup has not been recognized at large.

Within NTG, next to myself, Piet Tutelaers has used the active list separator, to name but one. He prescribed chess positions in a minimal way, with TEX to expand these into the complete markup. See later on.

12 A prerequisite is to distinguish between the header, first column and the data proper, for the markup of tables. In doing this the generally more complex header part of a table—the legenda—is separated from the data.

13 My Polish friends nicknamed paragraphs by 'From \indent to \par,' in their contribution about how paragraph parameters work for a EuroTEX some years ago.

14 This in contrast with the handling of options by parsing of arguments. In the old LATEX for example markup for options requires embracing them by square brackets. The disadvantage is that these square brackets must be there whether you need an option or not. Definitely redundant, IMHO, with all respect. (I was told that this no longer the case.) In LATEX's graphics chapter brackets like (and) are used to embrace optional items, which is inconsistent and confusing. Inconsistency is inevitable with macros emerging from allover the world. What I borrowed I generally recasted in my BLUe's philosophy in order to enhance consistency, and to allow myself to forget about the details of the tool.

15 An predecessor paper on the issue overlooked the inclusion of (active) list separators and therefore the author had to go explicitly through the data recursively again, all within a LISP flavour. Definitely unintelligible, especially for a non-LISP programmar like me.

What is the problem?

The problem in its simplest form is that we have for example the string 'ABC' and we wish to replace this by '\ls A \ls B \ls C.'

How to do this?

The algorithm is straightforward: walk along the elements and insert \ls before each element. The programming details are a bit tricky in TEX, however.[16] Let us assume that 'ABC' is stored as replacement text of the definition \data, and that we'll deliver the result under the name \markedupdata for educational purposes. The latter allows to use TEX's \show for inspecting the result. For simplicity, I assume further that the elements are not expandable and consist of only single characters.[17]

Example *(Insertion of control sequences before)*

```
\def\fifo#1{\ifx\ofif#1\ofif\fi
  \noexpand\ls{#1}\fifo}
\def\ofif#1\fifo{\fi}
\def\data{ABC} \show\data
\xdef\markedupdata{\expandafter\fifo\data\ofif}
\show\markedupdata
```

Explanantion. I implemented going through the list by tail recursion, via the \fifo macro. Once you understand this macro you are able to program tail recursion on-the-fly. It might be handy to forget about termination first—yes, the infinite loop nightmare—and next concentrate on the termination. I presume that the infinite situation is rather straightforward: all what follows \fifo in the markup is taken one argument at a time, and reinserted preceded by \ls. In order to terminate \fifo we have to append a token to the data, a so-called sentinel in these kinds of loop programming, for the moment without a specific meaning. I chose \ofif. If this \ofif token as argument is recognized then the tail recursion is ended by an invoke of—why not \ofif?—in the true branch. Only now the meaning of \ofif becomes relevant. It should be defined with \fifo as end separator of its argument, in order to 'eat' all tokens up to and including the \fifo token, meaning no next level of tail recursion will take place. However, also the closing \fi is eaten and therefore the replacement text of \ofif must reinsert this token, et voilà.

Remark. TEX's \show control sequence displays the (replacement text of the) macro, supplied as argument, in the log file. By the use of \show we can verify what happened.

Minimal markup in chess

Tutelaers 1992, as mentioned by Eijkhout 1991, faced the problem of inputting a chess position. The problem is characterized by an unspecified number of positions of pieces, with for the pawn positions the identification of the pawn generally omitted. Let us denote the pieces by the capital letters K(ing), Q(ueen), B(ishop), (k)N(ight), R(ook), and P(awn), with the latter symbol, P, default. The position on the board is indicated by a letter a, b, c, . . . , or h, followed by a number, 1, 2, . . . , or 8. The goal is that, for example

```
\pieces Ke1 e2 e4
```

should expand into

```
\piece Ke1 \piece Pe2 \piece Pe4
```

with \piece to be defined for further processing, which is not relevant for our purpose in this note. I assumed a meanest-and-leanest one line input.

The transformation can be done by an appropriate definition of \pieces, and an adaptation of the \fifo template, as follows.

```
\def\pieces#1\par{\xdef\markeduppieces
  {\fifo#1{\ofif} }}
\def\fifo#1 {\ifx\ofif#1\ofif\fi
  \process#1\ssecorp\fifo}
\def\ofif#1\fifo{\fi}
\def\process#1#2#3\ssecorp{\if\relax#3\relax
  \noexpand\piece P#1#2\else
  \noexpand\piece#1#2#3\fi}
%Test
\pieces Ke1 e2 e4 Ra1

\show\markeduppieces
\bye
```

Remarks. A few subtilities have been introduced in the adaptation of the \fifo template. Use has been made of the implicit e-o-l, which is converted by TEX into a space. Therefore the added sentinel \ofif in the replacement text of \pieces must immediately follow #1, no space in between, because this space has already been added by TEX. On the other hand we need a space after the sentinel \ofif and therefore the sentinel is embraced and followed by a space.[18] The argument, #1, as seen by \process is followed by \ssecorp as end separator to cope with the default, implicit identification for pawns. Only recently I polished this variant, sigh, . . . details matter.[19]

16 . . . not that difficult really, once you get the hang of it.

17 Of course there are a lot of variations thinkable, but not so relevant for the basic part, I guess. For example accented characters can be accounted for via modifications, such as enclosing them between braces at the appropriate places. I have omitted those confusing details and will concentrate on the main issues.

18 This sentinel is debraced when taken as argument.

19 This note is all about expanding minimal markup into explicit markup. In practice the creation of an explicit \markeduppieces can be omitted.

Automatic inclusion of markup between elements

Inclusion of markup in between is a little more cumbersome. If we think of processing each element and to insert the in between token after each element, except for the last, then this entails that we must decide for each element whether it is the last one or not, that is to look ahead for the sentinel. Below the control sequence, `\cs`, will be inserted in between.

Example *(Insertion of control sequences in between)*

```
\def\fifo#1#2{#1\ifx\ofif#2\ofif\fi
  \noexpand\cs\fifo#2}
\def\ofif#1\ofif{\fi}
\def\data{ABC}   \show\data
\xdef\markedupdata{\expandafter\fifo\data\ofif}
\show\markedupdata
```

Explanation. As usual `\ofif` has to be appended as sentinel to the data, `\fifo` has two arguments, the second is used to look (ahead) for the sentinel. The first is reinserted and processed—in this simple case just reinserted—and the second is tested against `\ofif` and reinserted at the end of the replacement text. If the test yields true, `\ofif` is invoked which in this variant of looking ahead must 'eat' all tokens up to and including the `\ofif`, that is the reinserted #2. Again a `\fi` must be inserted to compensate for the eaten `\fi`.

Remark. A variant implementation—albeit, more clumsy and less efficient—of the above is to 'look ahead' not just by one element, but to split the total list into the first element and the rest—also called head and tail in programming—where the (shrinking) rest has to be copied each time. This opens the Pandora box of coding variants. For an average TEXie these kinds of variants are quite confusing, I guess. I pay so much attention to as straight as possible implementations in order to use them over and over in my macros, with confidence, enhancing conciseness and more importantly, correctness.

Example *(Splitting into head and tail)*

```
\def\fifo#1#2\ofif{#1\ifx\empty#2\empty\ofif\fi
  \noexpand\cs\fifo#2\ofif}
```

This clumsy code is not only less efficient but also more tricky, see the test for the empty argument, which by the way can be a useful trick at times.

Automatic inclusion of markup for tables

Starting from data as such it will be shown how TEX can insert the markup tags `\cs` and `\rs`, to separate columns and rows.

The basic idea has been borrowed from *The TEXbook*, 249, from the dubble dangerous bend remark about omission of `\cr` in the input data. I have extended this with omission in the input of &.

One-character data with implicit row separators

The idea in its simplest form is to transform for example[20]

```
P*On      P\cs *\cs O\cs n\rs
...   ⟶   ...
Edit      E\cs d\cs i\cs t
```

In each row elements are not separated, the elements consist of just one character. Macros for the above read as follows.

```
\def\fifol#1 #2 {\fifo#1\ofif
   \ifx\lofif#2\lofif\fi
   \noexpand\rs\fifol#2 }
\def\lofif#1\lofif{\fi}
\def\fifo#1#2{%\prc
   #1\ifx\ofif#2\ofif\fi
   \noexpand\cs\fifo#2}
\def\ofif#1\ofif{\fi}
\def\bdata#1\edata
   {\xdef\markedupdata{\fifol#1{\lofif} }}
\bdata P*On
       DEk*
       *n*S
       Edit
\edata%\show\markedupdata%check it
\framed\ruled\btable\markedupdata
```

Explanation. The storing of the (marked up) data is done by `\bdata`, with the sentinel {`\lofif`} added, that is embraced and followed by a space.[21] added. Each row to be further processed by `\fifo` ends by a space, because TEX converts e-o-l-s into spaces, and the sentinel is also followed by a space. The rows, two at a time, are arguments of `\fifol`. The first row is processed. The second row (debraced one level) is tested whether it is the sentinel or not,[22] and reinserted, appended by a space. If the test yields false the process is recursively repeated. If the test yields true the

For fun compare this recent minimal variant with the one I launched in 'FIFO and LIFO sing the BLUes.' Of course I have updated my version of that note, with some more examples I found interesting since then. I'm pondering about my www page with all my notes and BLUe's format. Keep fingers crossed.

20 I abstracted & into `\cs` and `\cr` into `\rs`.

21 If the braces are omitted the space is neglected because TEX neglects spaces after control sequences.

22 Note that the order of the arguments to the test matter.

invoke of \lofif eats all tokens up to and including the (reinserted) \lofif, that is the #2, and compensates for the eaten \fi by the replacement text of \lofif. Similar is the processing of each row, via the invoke of \fifo#1 with the sentinel \ofif added. I like to call the latter nested use of the FIFO idea.
The next step towards typesetting crosswords is that \prc formats the cells in each row, that is insert a \prc before the reinserted element, and give meaning to \prc.

However, in the PWT version of my crossword macros I decided that I could better not use \haling, or my \btable, but process each element directly within a box, and stagger these boxes appropriately, with the total framed. But, the typesetting is not of our concern for the moment, we are concentrating on the expansion of minimal markup, aren't we?

I think that the insertion of the \cs-s and \rs-s by TEX, can be useful. Whatever the value might be, the thinking along the lines I have elaborated on above helped definitely to develop much nicer and clearer codes.

Remarks.

\btable is BLUe's (bordered) table macro on top of plain's \halign. It abstracts from \halign's & and \cr into \cs and \rs, allowing for example the use of flags like \ruled. It is beyond the scope of this note to explain \btable or its use. Roughly speaking, \btable formats the data as you would expect. The above code is not robust with respect to extra blank lines in the data, however.

The code is biased by the e-o-l→ substitution of TEX.

Data with implicit column and row separators

Sometimes it is convenient to supply data for a table as such, row by row with the columns separated by spaces.

To expand table data without markup[23] into marked up data ready for use with TEX, is a bit more cumbersome. Have a try.[24]

Example *(Young tableaux)* The Young tableaux as given for example in the PWT guide, could have been provided, marked up simply, as follows.

```
\bdata 7  8 9 10
       9 11
      16
\edata
```

with (aimed at) results

7	8	9	10
9	11		
16			

The coding for transforming the input as supplied between \bdata and \edata reads as follows, where \obeylines has been used.

```
{\obeylines%
 \gdef\fifol#1
            #2
 {\fifo#1 {\ofif} \ifx\lofif#2\lofif\fi%
    \noexpand\cr\fifol#2
 }%
 \gdef\lofif#1\lofif
 {\fi}%
}
\def\fifo#1 #2 {#1\ifx\ofif#2\ofif\fi
            \noexpand&\fifo#2 }
\def\ofif#1\ofif{\fi}
\def\bdata{\bgroup\obeylines\store}
{\obeylines
 \gdef\store#1\edata{\egroup%
 \xdef\markedupdata{\fifol#1\lofif
}}}
\bdata 7  8 9 10
       9 11
      16
\edata          \show\markedupdata
$$\young\markedupdata$$
```

Explanation. The new issues here are the recognition of e-o-l-s within \obeylines' reign. Maybe you don't know what \obeylines does precisely. Don't worry, whatever it does is fine, as long as we apply its use consequently it should work fine. That is what has been done in the code above, and indeed it works fine, though I know what \obeylines does. The rest is similar to and discussed along with the earlier provided codes.

Remarks. The use of %-s is critical. Extra blank lines are handled robustly: blank cells will show in the table.
Because this note is not about formatting of tables but only on minimal markup and inserting markup automatically, consult the PWT guide for the source and use of \young. Again, its result in print is what you expect, in agreement with tradition.

Table data from a database

As suggested by Wietse Dol the inclusion of markup by TEX can be of practical value too. He told me of his database of table data, which he typesets for the time being by Pascal. Is it feasible to do this by TEX?

The idea is that for example a file contains

23 Meaning just visual ASCII: line by line and within each line elements separated by (one or more) spaces.

24 Hint: In order to discriminate between ordinary spaces and spaces coming from converted e-o-l-s, make use of \obeylines.

```
11 12
21 22
```

and that you will end up with let's say

```
\def\markedupdata{11\cs12\rs21\cs22}
```

That is the data together with markup is the replacement text of `\markedupdata` ready to be used by `\btable`

In principle solution

The insertion of markup is done similar to the example treated above. However, the extra complication is that the data are on a file. Below I read the file line-by-line, and could therefore insert `\rs` naturally, well, . . . more or less.

Assume that the data are in a file called data.

```
\openin1=data
\def\markedupdata{}
\def\addrs{\def\addrs{\ea
  \def\ea\markedupdata\ea{\markedupdata\rs}}}
\loop\read1 to\data
 \ifeof1 \break\fi
 \addrs
 \edef\mdata{\ea\fifo\data{\ofif} }
 \ea\ea\ea\def\ea\ea\ea\markedupdata\ea\ea\ea
     {\ea\markedupdata\mdata}
\pool
\framed\ruled\btable\markedupdata
\bye
```

The above requires the following auxiliaries.

```
\let\ea\expandafter
%Loop macros due to van der Goot
\def\loop#1\pool{#1\loop#1\pool}
\def\break#1\pool{\fi}
%FIFO variant for this case
\def\fifo#1 #2 {#1\ifx\ofif#2\ofif\fi
            \noexpand\cs\fifo#2 }
\def\ofif#1\ofif{\fi}
```

For those who don't have BLUeTEX available what is going on can be followed in the log file.[25] Therefore insert `\show\markedupdata` before the `\pool`, and don't forget to push the return key after the def has been shown in order to continue.

The results for the data

```
1 2 3
21 22 23
31 32 333
```

are

1	2	3
21	22	23
31	32	333

Remarks. Because of the loop I had to do something special with `\addrs`. Maybe it is possible to read the file and deliver the data, with an appropriate separation between the 'rows,' as replacement text of `\data`, let's say. If so we can apply the earlier treated mechanism for inserting markup.[26]

At a lower level there is flexibility in handling the look-and-feel of the typeset data. But as said earlier this is not our concern at the moment. Consult the PWT guide chapter about tables.

Acknowledgements

As usual Jos Winnink proofed the paper and helped me in coercing the note into MAPS format. His remarks and suggestions are always well-taken. To say the least, he reflects what despite the intention did not come across. I consider the approach of a friendly eye better than a referee in warranting quality.

Wietse Dol suggested to consider data stored in a file, or as he put it to typeset tables from a database of table data.

Conclusions

It's hoped for that the use of the FIFO principle, implemented as an expandable macro, for inserting markup during TEX's mouth-gullet processing, will contribute to a more general use of minimal markup.

Minimal markup is the royal road to more readable scripts and alleviates conversion problems such as from a BLUe script into a MAPS submission.

The expansion by TEX's gullet of minimal marked up scripts into completely marked up scripts, made me realize the power and relevance of TEX's gullet expansion capabilities.

> Although TEX has been used abundantly and intensively for nearly 20 years already, the awareness of the elegance and convenience of minimal markup to be expanded by TEX is only just emerging.

What astonishes me still is that it is very hard to really get at the simplests codes. Apparently Knuth had the same experience as can be distilled from the following from the preface in *The TEXbook* and . . . *The* METAFONT*book*

> . . . and there are always better ways to do what you've done before.

My case rests. Have fun, and all the best.

25 It's all about insertion of `\cs` and `\rs` at the appropriate places.

26 A request for this on TEX-nl did not provide an answer; even stronger it was believed it was not possible. Hmmm. . .