Bijlage 6

Parameterized data for tables in TEX

Dynamics, aha!

Kees van der Laan
`cgl@hetnet.nl`

abstract
The issue of generating and using parameterized data for tables in TEX is elaborated upon. The automatic insertion of markup along with the use of parameterized markup and the `\btable` macro is not the issue, but are prerequisites. `\btable`'s use, along with the automatic insertion of markup in the data as such, can be seen as tools, stepping stones, which made it possible to concentrate on pure data generation, or its use in typesetting tables by TEX.

keywords
BLUe, btable, code tables, data generation, dynamical markup, education, macro writing, parameterization, tables, tail recursion

Introduction

Tables comprise a bewildering variety. A glimpse of this variety was presented at the EuroTeX 92 at Prague, and included as examples in the tables chapter of my PWT.[1]

For this note I like to discriminate between two kind of parametrizations

- parametrization of the markup in tags, like `\ruled`, `\framed`, and similar things, and
- parameterization of the data, that is data which are generated or taken from a database as function of a parameter.

In this note I'll not discuss the parameterization of the markup. That has been done along with my `\btable` macro, which is at the heart of BLUeTEX's tables. However, because `\btable` is used I'll discuss its use.

A little about bordered tables

In BLUeTEX the philosophy behind table markup is to separate markup of the data from the markup of the border, or as I put it to separate the markup of the first column and row from the data proper. The markup for the data proper is independent from whether the table is ruled or not, and so on. This is parameterized in the column and row separators. The reason behind 'bordering' is similar to the reason to treat bordered matrices as a separate group, I guess. A border of a table, especially the first row, is usually more complicated to format than the proper entries of the table. In the PWT user guide I introduced a model for a bordered table.

Use of `\btable` The above ideas about bordering of tables have been implemented in `\btable`, borrowed from BLUeTEX. To explain the coding of the `\btable` macro is not at stake; to tell a little about how to use it might be handy in understanding this note.

In the following example the use of `\btable` and its parameterization of the markup are shown in a nutshell.

Example: Use of `\btable` and parameterizations

11	12
21	22

	Header	
1st row	11	12
2nd row	21	22

Caption

	Header	
1st row	11	12
2nd row	21	22

Footer

It is important to realize that the core, and invariant, markup of all the three representations is just the `\data`[2]

```
\def\data{11\cs12\rs
          21\cs22}
\btable\data
```

where `\cs` denotes column separator, and `\rs` denotes row separator, of which the appearance in print depends on tags like `\framed` or `\ruled`. Whether the table takes an explanatory first row, a caption, rules, or similar things, these have no influence on the markup of the data proper. I consider this as a sort of abstract markup, which is related to dynamical markup, because variations of representation have been accounted for, while the specification of the

1 PWT denotes Publishing with TEX, the user guide which comes with BLUeTEX.
2 The `\cs` and `\rs` can be omitted, see my note on 'Minimal Markup.'
3 This abstraction from plain TEX's & as separator is also useful

data remains invariant. This approach helps among other things to recognize in the markup more easily the number of columns of a table.[3]
The essentials for its use can be distilled from the example given above.[4] The data are expected as replacement text of the macro `\data`, and separated by `\cs` and `\rs`, the (prefab, already provided for) tags for column and row separation. Important is that the table is thought of as a bordered table, similar to the idea of bordered matrix as treated in TUGBOAT. The border consists of the '11'-element first, to be optionally specified as a def `\first`, the border top row to be optionally specified as the def `\header`, and the first column to be optionally specified as replacement text of the def `\rowstblst` as 'groups'[5] without further separation. If the optionals are not defined `\btable` won't complain. (This holds too for `\btablecaption` and `\footer`.) The invoke is just `\btable\data` preceded by optional tags like `\ruled`, `\framed` or similar things, eventally within a math display. So, it looks like an intelligent, robust macro, which won't complain when you specify `\first` without a `\rowstblst` or a `\header`. In that case the results will most likely not look like what you had in mind. `\btable` tries to make something nice out of what you supplied.

Parameterized data

The data for the tables treated in this note are either provided by

- a program, or
- read from a file.

The program will not be recognized as a table, nor will it be easy to tell how many columns and rows the table will consist of.

Why?

Data as programs comes in naturally when we think of a bunch of tables, which are closely related. Examples are the guide cards for a bridge tournament—which actually triggered this note—or Wietse's data—his (employer's database of tables—the real life examples.

Roughly seven years ago, I coded my first dynamical and parameterized 'table' in TEX: the various stadia of the towers of Hanoi game, while attending the advanced TEX class of David Salomon.[6] In that 'markup' I made heavily use of the concept of an active list separator, which I encountered for the first time in TUGBOAT, and have used happily ever since. Moreover, the data are generated and parametrized over the size.

Disclaimer. This note is not about the look-and-feel of tables. The appearance of the discussed tables is straightforward, simple and time-proven, and belong to the class of bordered tables.

Multiplication table

This example was introduced in the TEX world by Pittman in the eighties may serve to illustrate the issue of data generation for tables.[7] Maybe, it forms the simplest example of a parameterized, bordered table.

Example: Multiplication table

×	1	2	3
1	1	2	3
2	2	4	6

What is the problem, doc?

I guess that people would argue that the numbers and lines could just be marked up for in whatever TEX flavour you wish, and that is it. Agreed, that is true. See for example the straight markup à la BLUe given below.[8]

```
\def\first{$\times$}
\def\header{1\cs2\cs3}
\def\rowstblst{12}
\def\data{1\cs2\cs3\rs
          2\cs4\cs6}
\btable\data
```

But,... what if we wish to vary for the size or to vary with respect to the rules of the table? And in general, what can be learned from the example? My answer is positive. Indeed, some paradigms in TEX macro writing can be distilled from the example marked up as function of its size.

for complicated tables with blocks in it. See the PWT guide for examples.

4 There is also a two-part variant of the macro but this will not be used in this note. I have not included the model picture for the `\btable` macro either, because this note as such is not about the macro. For a picture of the model and more elaborate description of the use of `\btable`, if needed, see the chapter on tables of the PWT guide.

5 Well, as arguments for a macro. The empty bordered table, without data is something I did not foresee. One can think of these as a sort of fill-in form, roster or calendar.

6 See TUGboat, 1992, or MAPS 92.1, or the Tables chapter of the PWT guide. If people don't wish to look upon this as a table, that is fine with me. IMHO, with all respect, a taxonomy of tables is not yet in sight, because of the diversity and ambiguity of the subject.

7 Pittman used it to illustrate nested loops in TEX. I consider it as a table with data generated (tail) recursively, which is equivalent to looping.

8 The separation between the header and row-stub-list is default a rule in BLUe. Because of the different functions it is quite natural, IMHO, with all respect, to separate the header and rowstubs from the table proper.

The drawback is that the resulting markup is hardly accessible for an ordinary, non-programmer user. Don't be afraid, hang-on, and go for it.

Tail recursion in TEX

On several occasions I found it rather useful that I knew how to code tail recursion—and its termination—in TEX.[9] The following generates the numbers 0–9. The simplest example I could think of in order to illustrate the idea.

```
\def\0{\the\i\advance\i1 \ifnum\i=10 \9\fi
       \0}
\def\9#1\0{\fi} %The recursion terminator
\0
```

Explanation. The macro `\0` invokes itself, and before doing so formats and increases the value of the counter `\i`. So far an infinite loop, and simple to understand. Now comes the tricky, well unusual, part. The condition is also straightforward but what happens in the true branche? The macro `\9` is invoked, which takes as argument all up to and including `\0`, meaning there is no further invoke, aha. . . termination. Nice, but . . . in the termination process also the `\fi` is eaten, and therefore the replacement text of `\9` must provide a `\fi`. Amazing, isn't it? Confusing? After a while, you will be used to it, don't worry too much, it is not that complicated IMHO, . . . just unusual.

To check your understanding, and demonstrating en-passant that it is worthwhile to spend some time on it, the following example of a loop as a real tail recursion encoding in TEX, due to van der Goot.[10]

```
\def\loop#1\pool{#1\loop#1\pool}
\def\break#1\pool{}
```

The toy example can be marked up by the above loop as follows

```
\loop \the\i \advance\i1
  \ifnum\i=10 \expandafter\break\fi
\pool
```

Remark. Sometimes, especially in nested loops, it is better to have `\fi` as replacement text of `\break` instead of the use of `\expandafter`.

The multiplication table via tail recursion

I like to split up the generation of what is needed in three. How to generate the

- header row
- data proper, and
- first column.

Next to the generation of the data I inserted en-passant the markup tags for column and row separation, `\cs` and `\rs`, respectively.

The header row The first row, which is called `\header` in BLUe, is a straightforward elaboration of the toy example of generating the digits 0-9 via tail recursion, treated above. However, now we must increase globally, because we are within plain TEX's `\halign` reign. The problem is to generate the numbers 1, 2, . . . (up to the order `\n`), separated by the markup for the column separator `\cs`.

```
\def\header{\the\j \global\advance\j1
             \ifnum\j>\n \redaeh\fi
            \cs\header}
\def\redaeh#1\header{\fi}
```

The data proper This is a little harder because we have to apply the tail recursion nested: on the outer level for the rows, and within each row for the columns.

```
\newcount\i%row index
\newcount\j%column index
\newcount\n%order
\newcount\entry
\def\rows{\global\j1 \cols\global\advance\i1
           \ifnum\i>\n \swor\fi
          \rs\rows}
\def\swor#1\rows{\fi}
\def\cols{\entry\i \multiply\entry\j \the\entry
           \global\advance\j1
           \ifnum\j>\n\sloc\fi
          \cs\cols}
\def\sloc#1\cols{\fi}
%
\global\i1 \n3 \framed\btable\rows
```

By the way, `\framed` does what its name suggests: the table is framed.

Are you still there? To finish up the dynamical 'markup' we have to code for the first column, also called row-stub-list in BLUe. This is tricky, admitted. One has to know how `\rowstblst` is processed. Once this is known, it is not that difficult anymore.

```
\def\rowstblst{Anything, just phoney}
\def\nxtrs{\the\i\rss}%overriding definition
```

Explanation. The phoney definition is there for fooling `\btable`, to let it think that a row-stub-list is there. Agreed, a real hack. Then, the real thing, redefine `\nxtrs` to yield the rowstubs on turns.

9 By the way, I had to unlearn never thinking in infinite loops. In this case, however, it was useful to code an infinite situation first and then account for the termination.

10 Loops have been dealt with in my 'Paradigms: Loops,' MAPS 17.

What I needed once

Of late I was asked as a tournement director for a bridge drive and faced the problem to provide for guide cards.[11] The scheme I used—Mitchell—is defined as follows.

For a tournament with n tables and n rounds, that is $2n$ pairs

> at table 1 pair 1 as NZ meets pair 2 as EW,
> at table 2 pair 3 as NZ meets pair 4 as EW, etc.
> After each round NZ moves up and EW moves down 1 table.

In the concrete case n was 14. I took the number of tables (half the number of pairs) as parameter. Below I have taken for concenience $n = 7$.

Mitchell 14, 7 rounds

T1	T2	T3	T4	T5	T6	T7
1–2	3–4	5–6	7–8	9–10	11–12	13–14
13–4	1–6	3–8	5–10	7–12	9–14	11–2
11–6	13–8	1–10	3–12	5–14	7–2	9–4
9–8	11–10	13–12	1–14	3–2	5–4	7–6
7–10	9–12	11–14	13–2	1–4	3–6	5–8
5–12	7–14	9–2	11–4	13–6	1–8	3–10
3–14	5–2	7–4	9–6	11–8	13–10	1–12

How to program this table? Of course we can provide all the data explicitly, with the advantage of simplicity and readability, but with the disadvantage of susceptibility for errors and that it has to be redone for each value of the parameter. More elegant is to program the table in the same spirit as the multiplication table above. Because it is so similar I'll just give the code, modulo some syntactic sugar.

```
\newcount\NZ\newcount\EW
\newcount\nz\newcount\ew
\newcount\i\newcount\j
\newcount\n\newcount\twon
\NZ3 \EW0 \i0 \j0 \n7 \twon2 \multiply\twon\n
\def\btablecaption{Mitchell 14, 7 rounds}
\def\header{\global\advance\j1 T\the\j
        \ifnum\j=\n \global\j0 \redaeh\fi
        \cs\header}
\def\redaeh#1\header{\fi}
%data
\def\rows{\global\advance\i1 \global\j0
   \global\advance\NZ-2
   \ifnum1>\NZ  \global\advance\NZ\twon  \fi
   \nz\NZ
   \global\advance\EW2
   \ifnum\EW>14 \global\advance\EW-\twon \fi
   \ew\EW
   \cols
   \ifnum\i=\n\swor\fi
   \rs\rows}
\def\swor#1\rows{\fi}
\def\cols{\the\nz--\the\ew
   \global\advance\nz2
   \ifnum\nz>\twon \global\advance\nz-\twon \fi
   \global\advance\ew2
   \ifnum\ew>\twon \global\advance\ew-\twon \fi
   \global\advance\j1
   \ifnum\j=\n\sloc\fi
   \cs\cols}
\def\sloc#1\cols{\fi}
%There you go
\ruled\framed\btable\rows
```

In a Mitchell scheme the pairs don't really need a guide card, because of the simple process of going from one table to the next, c.q. the previous. The guide card is generally given to each pair in order that they know at which table in what direction they play in each round, despite the heat of the tournament. Below such a guide card has been included.

Example: Guide card parameterized over pair number

Paar 13 – NZ

Ronde	Tafel	Tegen	Spellen
1	7	14	25–28
2	8	18	1–4
3	9	22	5–8
4	10	26	9–12
5	11	2	13–16
6	12	6	17–20
7	13	10	21–24

Remark. It is tempting to ponder about such a scheme as a database from which the required tables can be generated via for example the commands

11 Guide cards guide pairs through the tournement, prompted by the competition and the scheme chosen. One class is called the Mitchell scheme.

```
\mitchell{14}
\pair13\mitchell{14}
```

Font chart alternative

Along with BLUe Sky's TEXtures I found an alternative concise coding for the 'font charts.' In TUGBOAT the code tables are parameterized over the fonts and the entries are generated by `\normalchart`. Below a variant of '`\normalchart`' is included.[12]

Є$_{5}$ Ђ$_{6}$ Ћ$_{7}$ њ$_{8}$ љ$_{9}$ џ$_{10}$ э$_{11}$ і$_{12}$ є$_{13}$ ђ$_{14}$ ћ$_{15}$ Ю$_{16}$
Ж$_{17}$ Й$_{18}$ Ё$_{19}$ Ѵ$_{20}$ Ѳ$_{21}$ Ѕ$_{22}$ Я$_{23}$ ю$_{24}$ ж$_{25}$ й$_{26}$ ё$_{27}$
ѵ$_{28}$ ѳ$_{29}$ ѕ$_{30}$ я$_{31}$ ¨$_{32}$!$_{33}$ ”$_{34}$ Ѣ$_{35}$ ˘$_{36}$ %$_{37}$ ´$_{38}$ ’$_{39}$ ($_{40}$
)$_{41}$ *$_{42}$ ѣ$_{43}$,$_{44}$ -$_{45}$.$_{46}$ /$_{47}$ 0$_{48}$ 1$_{49}$ 2$_{50}$ 3$_{51}$ 4$_{52}$ 5$_{53}$
6$_{54}$ 7$_{55}$ 8$_{56}$ 9$_{57}$:$_{58}$;$_{59}$ «$_{60}$ ı$_{61}$ »$_{62}$?$_{63}$ ˘$_{64}$ А$_{65}$ Б$_{66}$
Ц$_{67}$ Д$_{68}$ Е$_{69}$ Ф$_{70}$ Г$_{71}$ Х$_{72}$ И$_{73}$ Ј$_{74}$ К$_{75}$ Л$_{76}$ М$_{77}$
Н$_{78}$ О$_{79}$ П$_{80}$ Ч$_{81}$ Р$_{82}$ С$_{83}$ Т$_{84}$ У$_{85}$ В$_{86}$ Щ$_{87}$ Ш$_{88}$
Ы$_{89}$ З$_{90}$ [$_{91}$ “$_{92}$]$_{93}$ Ь$_{94}$ Ъ$_{95}$ ‘$_{96}$ а$_{97}$ б$_{98}$ ц$_{99}$ д$_{100}$
е$_{101}$ ф$_{102}$ г$_{103}$ х$_{104}$ и$_{105}$ ј$_{106}$ к$_{107}$ л$_{108}$ м$_{109}$ н$_{110}$ о$_{111}$
п$_{112}$ ч$_{113}$ р$_{114}$ с$_{115}$ т$_{116}$ у$_{117}$ в$_{118}$ щ$_{119}$ ш$_{120}$ ы$_{121}$
з$_{122}$ –$_{123}$ —$_{124}$ №$_{125}$ ь$_{126}$ ъ$_{127}$

The above has been obtained as follows.

```
\font\test=wncyb10 \relax
\def\c#1{\setbox0=\hbox{\test\char#1}%
 \ifdim\wd0>0pt\box0\lower3pt
               \hbox{\fiverm\the#1}
 \fi}
\noindent{\count0=0
 \loop\c{\count0}
   \ifnum\count0<128 \advance\count0 by1
 \repeat}
```

Data from a file

Suppose you have data on a file separated by spaces and by lines. How to typeset these with TEX?

I did read the data line by line to insert the `\rs` and for each line between the data I inserted the `\cs`. For a tiny in-principle example it worked. This has more to do with automatically insertion of markup by TEX than with data generation.

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-driven' tables, or as he did in Pascal to typeset tables from a database of table data.

Conclusions

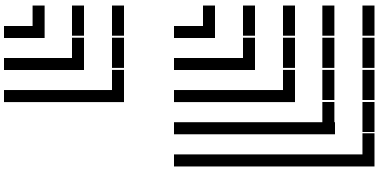
The unusal, dynamical and parameterized markup of tables is completely different from the example material as treated in TUGBOAT in the alignment chapter.[13] I encountered in practice the need for parameterization table markup next to the generation of the data, when I had to create bridge fill-in forms, score sheets, and guide cards for each pair, all of varying size.

In the PWT guide the Pascal triangle, for example, has been marked up as function of its order. Crosswords are parameterized over whether the puzzle or the solution is wanted, and are data-driven. Moreover, the following gnomons, which have been discussed in TUGboat some years ago, can be found in the PWT guide as another example of generated data and unusual markup in general, not to mention the included charts and trees. Do have a try in parameterization of your table markup and generation of data.

1	3	5
1	4	7
1	5	9

1	3	5	7	9
1	4	7	10	13
1	5	9	13	17
1	6	11	16	21
1	7	13	19	25

Or its graphical counter part.



And, . . . let me know about your examples. I welcome comments.

My case rests. Have fun, and all the best

12 I don't know who the author is, I'm sorry, but found it handy myself to find out what should be keyboarded or put in a chardef for unusual fonts.

13 However, the markup for the font tables is parameterized over the fonts.