Publication quality tables in LaTeX*

Simon Fear
300A route de Meyrin
Meyrin
Switzerland

Printed July 24, 2002

Abstract

This article describes some additional commands to enhance the quality of tables in LaTeX. Guidelines are given as to what constitutes a good table in this context. The latest release (Version 1.61) of the `booktabs` package, described herein, adds some enhancements to the 1995 release (Version 1.00), most notably `longtable` compatibility.

1 Introduction

The routines described below are to enable the easy production of tables such as should appear in published scientific books and journals. What distinguishes these from plain LaTeX tables is the default use of additional space above and below rules, and rules of varying 'thickness'. What further distinguishes them from the tables many people *do* produce using LaTeX is the absence of vertical rules and double rules.

I must draw a clear distinction between what I call here a *formal table*, which is a set of values in labelled columns, as distinct from what I will call a *tableau*. The latter is the kind of thing illustrated in the LaTeX manual, and increasingly common as the output of many database management systems; it will probably have icons in abundance, and no doubt use colour too. The layout of such a *tableau* is determined (hopefully) as a one-off, given a jumble of material the designer is trying to combine into a meaningful configuration. But the layout of a *table* has been established over centuries of experience and should only be altered in extraordinary circumstances.

By way of illustration, consider this tableau from the LaTeX manual (p. 64 old edition):

gnats	gram	$13.65
	each	.01
gnu	stuffed	92.50
emu		33.33
armadillo	frozen	8.99

*This file has version number v1.61 (converging to phi, the golden ratio), last revised 16 August 2000.

This is a hotch-potch of information that is probably reasonably clearly presented as is (but is the emu stuffed or not?). However, as a published table, this should much rather appear along the lines suggested further down the page in the manual:

	Item	
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

It takes much less work to lay this out, as a formal table; we don't have to work out a new layout for everything we do. Moreover, we can be almost certain that the data cannot be misread, because the reader does not have to learn how to read some novel presentation.

The above table cannot be produced in pure LaTeX, unfortunately. It can be laid out as it should be, but despite your best efforts, using plain `\hline` commands produces

	Item	
Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

Note (if it is not already obvious) that there is not enough space between the top line and the capital I of 'Item', and so on for all the lines: contrast with the previous version. Also, in the first version the top and bottom rules (ie lines) are heavier than the middle rule, which is turn heavier than the subrule underneath 'Item'. Of course you *could* redefine `\doublerulesep` and then use `\hline\hline` to get something near the same effect, and you could use struts to improve the spacing. But you should not have to think of such things. The `booktabs` style defines its commands so that such things are taken care of automatically.

In general, I would say that this package is of no interest to those looking for an alternative to PicTeX to conjure up fancy tableaux. Rather, it is a style guide for authors of scientific papers and books as regards table layout. It is not going too far to say that if you cannot create a table using the commands in this package, you should redesign it.

1.1 A note on terminology

In British typesetting, a 'line' is always called a 'rule'. Perhaps confusingly (for historic reasons in fact), the 'thickness' of rule is often referred to as is its 'width' (whereas just about everyone else would call this 'depth' or 'height', if they were thinking of a horizontal rule). A 'thick black line' is called a 'heavy rule'. I have used this terminology in most of the new commands below. If nothing else it avoids confusion with `\hline`.

2 The layout of formal tables

You will not go far wrong if you remember two simple guidelines at all times:

1. Never, ever use vertical rules.
2. Never use double rules.

These guidelines may seem extreme but I have never found a good argument in favour of breaking them. For example, if you feel that the information in the left half of a table is so different from that on the right that it needs to be separated by a vertical line, then you should use two tables instead. Not everyone follows the second guideline: I have worked for a publisher who insisted on a double light rule above a row of totals. But this would not have been my choice.

There are three further guidelines worth mentioning here as they are generally not known outside the circle of professional typesetters and subeditors:

3. Put the units in the column heading (not in the body of the table).
4. Always precede a decimal point by a digit; thus 0.1 *not* just .1.
5. Do not use 'ditto' signs or any other such convention to repeat a previous value. In many circumstances a blank will serve just as well. If it won't, then repeat the value.

Whether or not you wish to follow the minor niceties, if you use only the following commands in your formal tables your reader will be grateful. I stress that the guidelines are not just to keep the pedantic happy. The principal is that enforced structure of presentation enforces structured thought in the first instance.

3 Use of the new commands

`\toprule` `\midrule` `\bottomrule`

In the simplest of cases a table begins with a `\toprule`, has a single row of column headings, then a dividing rule called here a `\midrule`; after the columns of data we finish off with a `\bottomrule`. Most book publishers set the `\toprule` and `\bottomrule` heavier (ie thicker, or darker; see section 1.1) than the intermediate `\midrule`. However, when tables appear in very small typesizes it is sometimes impossible to make this distinction, and moreover quite a few journals routinely use all rules of the same heaviness.

The rule commands here all take a default which may be reset within the document (preferably, but not necessarily, in the preamble). For the top and bottom rules this default is `\heavyrulewidth` and for midrules it is `\lightrulewidth` (fully described below). In very rare cases where you need to do something special, you may use the optional arguments to the rule commands which have formal syntax as follows:

`\toprule[`⟨*wd*⟩`]`
`\midrule[`⟨*wd*⟩`]`
`\bottomrule[`⟨*wd*⟩`]`

where ⟨*wd*⟩ is a TEXdimension (for example 1pt, .5em, etc.).

All the rule commands described here go after the closing \\ of the preceding row (except \toprule, which comes right after the \tabular{} command); in other words, exactly where plain LaTeX allows \hline or \cline.

\cmidrule

Frequently we need a sub-rule to extend over only some of the columns, for which we need a \cmidrule (the analogue of LaTeX's \cline command). Generally, this rule should not come to the full width of the columns, and this is especially the case when we need to begin a \cmidrule straight after the end of another one (LaTeX's \cline**s** crash into each other here if you are not extra careful with \extracolsep). Thus, you will generally want to use the optional 'trimming' commands.

The trimming commands, if used at all, go in parentheses (like this), with no spaces separating them. The possible specifications are `r`, `r{`⟨*wd*⟩`}`, `l` and `l{`⟨*wd*⟩`}`, or any combination of these, where ⟨*wd*⟩ is a dimension, and `r` and `l` indicate whether the right and/or left ends of the rule should be trimmed. The form without explicit argument is equivalent to `r{\cmidrulekern}`, where \cmidrulekern defaults to 0.5 em, but can be set by the user in the preamble.[1]

Here's an illustrative example: `(lr{.75em})` gives you a default left trim and exactly 0.75 em right trim. Equally valid here is `(r{.75em}l)`.[2]

The full syntax of the command is

`\cmidrule[`⟨*wd*⟩`](`⟨*trim*⟩`){`*a*–*b*`}`

where ⟨*wd*⟩ is an optional rule width command, in square brackets [like this] (the default here is \cmidrulewidth), and the last argument, *which is not optional*, gives the column numbers to be spanned.

An example of the commands in use is given by the code used to produce the example table above:

```
\begin{tabular}{@{}llr@{}} \toprule
\multicolumn{2}{c}{Item} \\ \cmidrule(r){1-2}
Animal & Description & Price (\$)\\ \midrule
Gnat  & per gram  & 13.65 \\
      & each      & 0.01 \\
Gnu   & stuffed   & 92.50 \\
Emu   & stuffed   & 33.33 \\
Armadillo & frozen & 8.99 \\ \bottomrule
\end{tabular}
```

\addlinespace

Occasionally we want to put an extra space between certain rows of a table; for example, before the last row, if this is a total. This is simply a matter of inserting

`\addlinespace[`⟨*wd*⟩`]`

after the \\ alignment marker. Between ordinary rows of text, the effect is identical to the ordinary LaTeX usage `\\[\defaultaddspace]`, which I find rather clumsy, and it is better than \\ \\, which inserts too much space. Also, \addlinespace can be used before, after, or between rules if you want to control the exact amount

[1] User feedback suggested the Version 1.00 default, 0.25 em, was too small. Sorry for any loss of backward compatibility. Remember that you can easily set \cmidrulekern in the preamble, or just use `(r{.25em})` to recover the original behaviour.

[2] As a matter of fact, `(lrrlr{.75em})` does the same thing: only the last encountered left and the last encountered right specification are applied.

of space to be inserted. The default space before or after an adjacent rule is replaced by exactly `\defaultaddspace` or the amount of space specified in the optional argument.[3]

4 Abuse of the new commands

Let's face it, nobody can leave well alone, so here are some guidelines and extra commands.

The new rule commands are not guaranteed to work with `\hline` or `\cline`, although these remain available and unchanged. I cannot foresee any reason to want to mix them.

More importantly the rules generated by the new commands are in no way guaranteed to connect with verticals generated by `{|}` characters in the preamble. This is a feature (see above). You should not use vertical rules in tables, end of story.

`\morecmidrules` If you just cannot stop yourself from using a double rule, even a construction as bizarre as `\toprule\bottomrule\midrule` will work without generating an error message (just as you can double `\hline`). These rules will be separated by the ordinary LaTeX separator `\doublerulesep`. However if your perversion is to want double `\cmidrule`s you will need the extra command `\morecmidrules` to do so properly, because normally two `\cmidrule`s in a row is a sane construction calling for two rules on the same 'rule row'. Thus in

```
\cmidrule{1-2}\cmidrule{1-2}
```

the second command writes a rule that just overwrites the first one; I suppose you wanted

```
\cmidrule{1-2}\morecmidrules\cmidrule{1-2}
```

which gives you a double rule between columns one and two, separated by `\cmidrulesep` (note: since a `\cmidrule` is generally very light, the ordinary `\doublerulesep` is probably too much space). Finish off a whole row of rules before giving the `\morecmidrules` command. Note that `\morecmidrules` has no effect whatsoever if it does not immediately follow a `\cmidrule` (ie it is not a general space-generating command).

`\specialrule` If you find some extraordinary need to specify exactly 0.5 em, say, between two rules, you could use a construction such as `\midrule \addlinespace[.5em] \midrule`. In a rare fit of tolerance, though, I have also provided the command

`\specialrule{`⟨*wd*⟩`}{`⟨*abovespace*⟩`}{`⟨*belowspace*⟩`}`

where all three arguments are mandatory (I couldn't be bothered to program in defaults). If you use this frequently, you have misunderstood the purpose and content of the guidelines given above. A preceeding rule does not add its default space below, and a following rule adds no space above itself, so you get *exactly* the space specified in the arguments.[4]

[3] This is a change from version 1.00, where the space was sometimes *in addition to* default rule space.

[4] This is a change from Version 1.00, which rather liked to add an extra `\doublerulesep` space whenever it could.

5 Booktabs and longtables

If you have both `booktabs` and `longtable` packages loaded, the `booktabs` rule commands can now all be used exactly as described above, within a longtable.

There is an addition worth noting: within a `longtable`, you can use the optional left and right trimming commands, which normally only work for `\cmidrule`s, with `\toprule`, `\midrule` and `\bottomrule` (and if you must, also with `\specialrule`). Users who hacked the previous release for longtable compatability[5] seemed to like all the rules to be right trimmed 0.5 em. I think you can do the same by making `@{}` be the last column specifier. Still, after working out the rest of the code, it was easy to add parsing for the optional arguments, so I did. (I didn't go the whole way and allow the optional trimming *outside* a `longtable`; this would be a huge amount of work. If you must have trimmed rules, make all your tables be `longtable`s!)

A somewhat technical note: within a `longtable`, `\hline` and `\hline\hline` both produce a *double* rule (to allow for page breaks occurring at that point). But the `booktabs` rules do *not*. Longtable's automatic doubling of `\hline` is questionable, even according to the documentation within that package. But doubled `booktabs` rules make almost no sense at all. In the unfortunate event that a `booktabs` rule should occur at a page break, then you will have to make the necessary adjustments by hand. (In general, this will mean deleting the offending rule.)

6 Technical summary of commands

The new rule commands are valid inside the standard `tabular` (and `array`) environment, in the modified `tabular` and `array` of `\usepackage{array}`, and within both standard tables and longtables after `\usepackage{longtable}`.

The commands follow the standard placement syntax of `\hline`. There can be space (including carriage-return, but not two carriage-returns) between successive rule commands.[6]

In what amounts to quite a big change from former releases, within the macro code I now define three classes of rules. (But we don't need these definitions within ordinary use, so I haven't even mentioned them above.) A class 1 rule (otherwise called a 'normal' rule) is any of `\toprule`, `\midrule`, `\bottomrule`, or `\cmidrule`. The class 2 rules are `\specialrule` and `\addlinespace`. Finally, a class 0 rule is none of the preceeding — or in other words, not a rule at all.[7] Note that `\addlinespace` counts as a class 2 rule, not as class 0 text.

In the following, we first describe each command in 'normal use', meaning that the rule is being used between two lines of text (or more technically, is preceded and followed by a class 0 rule). After that, we will look at the exceptions.

`\toprule[`⟨*wd*⟩`]`

A rule of width ⟨*wd*⟩ (default `\heavyrulewidth`) with `\abovetopsep` space above and `\belowrulesep` extra vertical space inserted below it. By default,

[5] Jim Service was the first

[6] A welcome change from Version 1.00, where space between rule commands generated a very baffling error message.

[7] Except that `\hline` and `\cline` are class 0. Still, there is no reason to lose sleep over this, since one would not want to mix the two rule-drawing systems.

`\abovetopsep` is zero, which seems sensible for a rule designed to go at the top. However, if your tables have captions, it can make sense to use `\abovetopsep` to insert a reasonable amount of space between caption and table, rather than remember to use a `\vspace{}` command in the float.

`\midrule[`⟨*wd*⟩`]`

A ⟨*wd*⟩ (default `\lightrulewidth`) rule with `\aboverulesep` space above it and with `\belowrulesep` space below it.

`\bottomrule[`⟨*wd*⟩`]`

A ⟨*wd*⟩ (default `\heavyrulewidth`) rule with `\aboverulesep` space above it and with `\belowbottomsep` space below it. By default `\belowbottomsep` is zero[8]. There is a frequent and legitimate reason you might want space below a bottom rule: namely, when there's a table footnote.[9] If you don't override the default you could use `\bottomrule \addlinespace[\belowrulesep]` or you could put a suitably sized strut into the footnote text.[10] But the default has to be zero, so that it behaves sensibly in a `longtable` footer.

`\cmidrule[`⟨*wd*⟩`](`⟨*trim*⟩`){`*a*–*b*`}`

A ⟨*wd*⟩ (default `\cmidrulewidth`) rule with `\aboverulesep` space above it (unless following another `\cmidrule`, in which case it is on the same vertical alignment; or if following `\morecmidrules`, separated from a previous `\cmidrule` by `\cmidrulesep`). A `\cmidrule` has `\belowrulesep` below it (unless followed by another `\cmidrule`, in which case the following rule is on the same vertical alignment; or if followed by `\morecmdirules`, when there will be `\cmidrulesep` below it).

The `\cmidrule` spans columns *a* to *b* as specified in the mandatory argument. The optional argument ⟨*trim*⟩, which goes in parentheses if at all, can contain any sequence of the tokens `r`, `l` and `{`⟨*wd*⟩`}`, with the latter setting the kerning to be applied to right or left sides as specified by the immediately preceding token. (There's currently no error checking done here, so be careful to get the syntax right.)

`\morecmidrules`

Instructs LaTeX to begin a new row of `\cmidrules`, separated from the last by `\cmidrulesep`. Has no meaning in any other context.

`\specialrule{`⟨*wd*⟩`}{`⟨*abovespace*⟩`}{`⟨*belowspace*⟩`}`

A ⟨*wd*⟩ rule (note: here this is a mandatory argument) with ⟨*abovespace*⟩ above it and ⟨*belowspace*⟩ below it.

`\addlinespace[`⟨*wd*⟩`]`

Technically this has the same effect as `\specialrule{0pt}{0pt}{`⟨*wd*⟩`}`, i.e. a zero-width rule with no space above and with ⟨*wd*⟩ (default `\defaultaddspace`) space below. This command was primarily designed to add space between rows

[8] This is a change from Version 1.00, where there was always a `\belowrulesep`

[9] But don't use footnotes, Donald.

[10] I don't like either of these. Sort it out in Version 1.618?

in the body of the table, but it may also be used to specify an exact amount of space above or below a class 1 rule.

Now we come to the exceptions to the above. We have already seen in the definitions that the type 2 rules are preceded and followed by exactly the amount of space specified by the arguments. That is, a type 2 rule suppresses the space that would normally be generated by a previous type 1 rule (e.g. `\belowrulesep` after a `\toprule`) and replaces it by the argument of the type 2 rule. Similarly, in the combination {type 2 rule}{type 1 rule}, the ordinary space above the type 1 rule (e.g. `\aboverulesep`) is suppressed. But in the combination {type 2 rule}{type 2 rule}, no space is suppressed: the rules will be separated by both the first rule's `{`⟨*belowspace*⟩`}` and the second rule's `{`⟨*abovespace*⟩`}` arguments. Last but not least, the combination {type 1 rule}{type 1 rule} will always give rules separated by `\doublerulesep`, suppressing all normal space generated between the rules (but retaining normal space above the first and below the second).

As an exception to this last exception, 'type 1 rule' excludes `\cmidrule`. Such rules combine with other `\cmidrule`s and `\morecmidrules` in normal use as described above. I don't know and I don't care care what the combination `\toprule\cmidrule{1-2}\midrule` would produce. I can see no excuse for such usage.

The default dimensions are defined at the beginning of the macro description section (Section 8). The user can change these defaults in the preamble, or outside a tabular environment, by simply inserting a command in exactly the same format as in Section 8; the redefinition will stay in effect for the rest of the document or until redefined again. *Inside a table* you would have to make the assignment globally in a `noalign` group: e.g. `\noalign\{\global\abovetopsep=1em\toprule}`. I hope you never have to do that.

7 Acknowledgments

Hugely indebted of course to DEK and Lamport; the optional argument and `\cmidrule` stuff especially was stolen from `latex.sty`. The documentation driver stuff is stolen from the tools package description `dcolumn.dtx` by David Carlisle.

For beta testing and encouragement ...

8 The code

The current version is defined at the top of the file looking something like this

```
⟨*package⟩
%\NeedsTeXFormat{LaTeX2e}
%\ProvidesPackage{booktabs}
%        [\filedate\space version\fileversion]
```

First we set up the new dimensions described above:

```
\newdimen\heavyrulewidth
\newdimen\lightrulewidth
\newdimen\cmidrulewidth
\newdimen\belowrulesep
\newdimen\belowbottomsep
```

```
\newdimen\aboverulesep
\newdimen\abovetopsep
\newdimen\cmidrulesep
\newdimen\cmidrulekern
\newdimen\defaultaddspace
\heavyrulewidth=.08em
\lightrulewidth=.05em
\cmidrulewidth=.03em
\belowrulesep=.65ex
\belowbottomsep=0pt
\aboverulesep=.4ex
\abovetopsep=0pt
\cmidrulesep=\doublerulesep
\cmidrulekern=.5em
\defaultaddspace=.5em
```

And some internal counters of no interest to the end user:

```
\newdimen\@cmidrulewidth
\newcount\@cmidla
\newcount\@cmidlb
\newdimen\@aboverulesep
\newdimen\@belowrulesep
\newcount\@thisruleclass
\newcount\@lastruleclass
\@lastruleclass=0
\newdimen\@thisrulewidth

```

which will be described as needed below.

`\futurenonspacelet` Next we define a very useful macro (more-or-less straight from the TEXbook's Dirty Tricks chapter; documented there). Use `\futurenonspacelet` instead of `\futurelet` when looking for the next (non-space) token after a macro that has an argument. (After a macro without an argument, space is ignored anyway, so `\futurenonspacelet` wouldn't be needed.) This hack allows users to type white space between successive rule commands (which did not work in Version 1.00).

```
\def\futurenonspacelet#1{\def\@BTcs{#1}%
 \afterassignment\@BTfnslone\let\nexttoken= }
\def\@BTfnslone{\expandafter\futurelet\@BTcs\@BTfnsltwo}
\def\@BTfnsltwo{\expandafter\ifx\@BTcs\@sptoken\let\next=\@BTfnslthree
 \else\let\next=\nexttoken\fi \next}
\def\@BTfnslthree{\afterassignment\@BTfnslone\let\next= }
```

8.1 Full width rules

When we are not in a `longtable` environment, we can simply implement the full width rules as a `\hrule` in a `\noalign{}` group. But within a `longtable`, the rule has to be drawn like a `\cmidrule{1-\LT@cols}` (the rationale for this is explained in the `longtable` documentation).

In order to allow for both, all the rule macros have to open a `\noalign` group immediately, while they work out whether they have been called within a `longtable`; if you don't do this, TEX's underlying `\halign` process gets hiccups. I use LATEX's dirty trick (`\ifnum=0‘}`) to fool the parser that the bracket

count is OK. The bracket really gets closed after all the skipping at the end of the `\@BTendrule` macro.

The class 1 rules, and `\specialrule`, really only differ in the defaults for space above and below, and the width, passed to a common routine, `\@BTrule`, described below. The spaces, `\@aboverulesep` and `\@belowrulesep`, are set within the `\noalign` group, so are inherited by `\@BTrule`. Similarly, `\@BTrule` knows as much as it needs to about the routine that called it by examining the inherited `\@thisruleclass`. The optional width argument is parsed by `\@BTrule` after being set to default if absent.

`\toprule` `\midrule` `\bottomrule` `\specialrule`

```
\def\toprule{\noalign{\ifnum0=`}\fi
  \@aboverulesep=\abovetopsep
  \global\@belowrulesep=\aboverulesep %global cos for use in the next noalign
  \global\@thisruleclass=\@ne
  \@ifnextchar[{\@BTrule}{\@BTrule[\heavyrulewidth]}}
\def\midrule{\noalign{\ifnum0=`}\fi
  \@aboverulesep=\aboverulesep
  \global\@belowrulesep=\belowrulesep
  \global\@thisruleclass=\@ne
  \@ifnextchar[{\@BTrule}{\@BTrule[\lightrulewidth]}}
\def\bottomrule{\noalign{\ifnum0=`}\fi
  \@aboverulesep=\aboverulesep
  \global\@belowrulesep=\belowbottomsep
  \global\@thisruleclass=\@ne
  \@ifnextchar[{\@BTrule}{\@BTrule[\heavyrulewidth]}}
\def\specialrule#1#2#3{\noalign{\ifnum0=`}\fi
  \@aboverulesep=#2\global\@belowrulesep=#3\global\@thisruleclass=\tw@
  \@BTrule[#1]}
```

`\addlinespace` An `\addlinespace` is essentially a zero-width rule with zero space above and argument (or default) space below. But because the rule is not actually drawn, but is just a `\vskip`, there is no need to check if we're in a longtable, so we don't need to call `\@BTrule` as for 'real' rules. But we do share the `\@BTendrule` lookahead and flagsetting code (described below), and the `\vskip` is done there.

```
\def\addlinespace{\noalign{\ifnum0=`}\fi
  \@ifnextchar[{\@addspace}{\@addspace[\defaultaddspace]}}
\def\@addspace[#1]{\global\@belowrulesep=#1\global\@thisruleclass=\tw@
  \futurelet\@tempa\@BTendrule}
```

`\@BTrule` All the rules (except `\addlinespace`) share this code.

```
\def\@BTrule[#1]{%
  \global\@thisrulewidth=#1\relax
```

Save the width argument (if the user didn't give one, then the calling routine will have called `\@BTrule` with the default) in a global variable for later use when drawing the rule.

```
  \ifnum\@thisruleclass=\tw@\vskip\@aboverulesep\else
```

Specialrules always insert specified space above. (Note: addlinespaces don't come here).

```
  \ifnum\@lastruleclass=\z@\vskip\@aboverulesep\else
  \ifnum\@lastruleclass=\@ne\vskip\doublerulesep\fi\fi\fi
```

After text (last rule class 0), precede the rule by `\aboverulesep`; but if immediately after a previous rule, insert a `\doublerulesep`.

Now we work out, by a very nasty hack, if we're within a `longtable`. It's easy if `\longtable` isn't even defined: then we can't be. But it is not enough just to check if `longtable` is loaded — we might be within an ordinary table rather than a longtable. So we look to see if `\hline` has been re-defined from its LaTeX definition to be the same as `\LT@hline`. (Longtable currently does this redefinition when it opens a `longtable` environment, but not globally, so it is cleared it when the environment closes.) Another package could potentially do this! And `longtable` might change the way it implements this! So, it is not entirely safe, but I have found no better way so far.

We set up `\@BTswitch` to call `\@BTnormal` or `\@BLTrule`, as appropriate, then call it.

```
\ifx\longtable\undefined\let\@BTswitch\@BTnormal\else
\ifx\hline\LT@hline\let\@BTswitch\@BLTrule\else\let\@BTswitch\@BTnormal\fi\fi
\@BTswitch}
```

`\@BTnormal` This is when we're *not* within a longtable. We are already in a `\noalign` group, all we need do is draw an `\hrule` and gobble any trailing spaces, then call the closing routine with `\@tempa` set equal to the next token in the document.

```
\def\@BTnormal{\hrule
  \@height \@thisrulewidth\futurenonspacelet\@tempa\@BTendrule}
```

`\@BLTrule` This is for full width rule within a `longtable`. First we check if a kerning argument has been used; if so let `\@@BLTrule` read it, else call `\@@BLTrule` with an empty string:

```
\def\@BLTrule{\@ifnextchar({\@@BLTrule}{\@@BLTrule()}}
```

`\@@BLTrule`

```
\def\@@BLTrule(#1){\@setrulekerning{#1}%
\global\@cmidlb\LT@cols
```

The `\@setrulekerning` routine parses the kerning argument tokens and sets global kerning widths accordingly (or to defaults, if user hasn't set them explicitly). The global assignment to `\@cmidlb` sets up the column count for the `\@cmidruleb` macro, which is shared with `cmidrules`.

```
\ifnum0=‘{\fi}%
```

Close the currently open `\noalign` group. Within a `longtable`, rules are all to be drawn as leaders within a text box that is `\LT@cols` columns wide.

```
\@cmidruleb
```

Draw the rule. We share the `\@cmidruleb` code with ordinary `\cmidrule`s.

```
\noalign{\ifnum0=‘}\fi
```

We have to open a new noalign immediately else TeXwill start a new text box where we don't want one. Then, after gobbling any unwanted white space, we call the closing routine.

```
\futurenonspacelet\@tempa\@BTendrule}
```

`\@BTendrule` We look one step ahead (token is in `\@tempa`) to see if another rule follows (shame on user!). If so, we set `\@lastruleclass` equal to `\@thisruleclass` (thus setting it up for the following rule). If there isn't a following rule, we clear

`\@lastruleclass` (ie set it to zero), which isn't technically true since we have just drawn a rule, but sets it up correctly for the next rule encountered, which must be following some intervening text.

```
\def\@BTendrule{\ifx\@tempa\toprule\global\@lastruleclass=\@thisruleclass
  \else\ifx\@tempa\midrule\global\@lastruleclass=\@thisruleclass
  \else\ifx\@tempa\bottomrule\global\@lastruleclass=\@thisruleclass
  \else\ifx\@tempa\cmidrule\global\@lastruleclass=\@thisruleclass
  \else\ifx\@tempa\specialrule\global\@lastruleclass=\@thisruleclass
  \else\ifx\@tempa\addlinespace\global\@lastruleclass=\@thisruleclass
  \else\global\@lastruleclass=\z@\fi\fi\fi\fi\fi\fi
  \ifnum\@lastruleclass=\@ne\relax\else\vskip\@belowrulesep\fi
  \ifnum0=`{\fi}}
```

8.2 Special subrules

`\@setrulekerning` The following code parses the trimming arguments (if there are any) for `\cmidrule` or a `\BLTrule`. The rule will be trimmed left and right by `\cmrkern@l` and `\cmrkern@l`, which are zero by default, set to `\cmidrulekern` by the plain `(lr)` arguments, or user set as in `(r{.5em})`. We parse token by token through the arguments. The tokens `r` and `l` cause `\cmrkern@r` or `\cmrkern@l` to be set to `\cmidrulekern`. There is no lookahead to see if a width is the next token; this strategy is efficient for the plain commands, while inefficient for the qualified commands, but more importantly it is much easier to program. Tokens `r` and `l` also set `\cmrswitch` so that if the next token turns out to be `{`⟨*wd*⟩`}` then the kerning will be done on the side currently specified. I have been too lazy to program an error message should one encounter tokens other than `r`, `l` or `{`⟨*wd*⟩`}`.

```
\def\@setrulekerning#1{\global\let\cmrkern@l\z@
  \global\let\cmrkern@r\z@
  \@tfor\@tempa :=#1\do
  {\if\@tempa r\global\let\cmrkern@r\cmidrulekern
      \def\cmrsideswitch{\cmrkern@r}\else
     \if\@tempa l\global\let\cmrkern@l\cmidrulekern
      \def\cmrsideswitch{\cmrkern@l}\else
     \global\expandafter\let\cmrsideswitch\@tempa
     \fi\fi}}
```

`\cmidrule` The `\cmidrule` re-uses `\@lastruleclass` in an entirely different way from the full width rules. (Maybe I should have used a different flag; it seemed efficient at the time ...). This is (left) set to one if you are in the middle of a row of `\cmidrules`, or starting a new one (with `\morecmidrules`). Otherwise, when `\@lastruleclass` is zero, we precede the rule with `\aboverulesep`.

```
\def\cmidrule{\noalign{\ifnum0=`}\fi
    \@ifnextchar[{\@cmidrule}{\@cmidrule[\cmidrulewidth]}}
\def\@cmidrule[#1]{\@ifnextchar({\@@cmidrule[#1]}{\@@cmidrule[#1]()}}
\def\@@cmidrule[#1](#2)#3{\@@@cmidrule[#3]{#1}{#2}}
```

The above is fiddling around to set defaults for missing optional arguments. We also pass to `\@@@cmidrule` in a different order, namely `[a-b]{width required}{kerning commands}` (this being the order in which the arguments are actually processed):

```
\def\@@@cmidrule[#1-#2]#3#4{\global\@cmidla#1\relax
    \global\advance\@cmidla\m@ne
```

```
    \ifnum\@cmidla>0\global\let\@gtempa\@cmidrulea\else
    \global\let\@gtempa\@cmidruleb\fi
    \global\@cmidlb#2\relax
    \global\advance\@cmidlb-\@cmidla
```

This has set up a switch (`\@gtempa`) to call the relevant routine, `\@cmidrulea` or `\@cmidruleb`, depending on whether we start from column one or not.

```
    \global\@thisrulewidth=#3
```

That is, set per default or given argument. Then parse any trimming arguments to set, globally, `\cmrkern@r` and `\cmrkern@l` accordingly:

```
    \@setrulekerning{#4}
```

Now insert space above if needed, close the `\noalign`, then switch to appropriate rule drawing routine as determined above (`\let` to `\@gtempa`):

```
    \ifnum\@lastruleclass=\z@\vskip \aboverulesep\fi
    \ifnum0=‘{\fi}\@gtempa
```

Having now drawn the rule, open another `\noalign`, and call the closing routine:

```
    \noalign{\ifnum0=‘}\fi\futurenonspacelet\@tempa\@xcmidrule}
```

In this closing routine, see if another `\cmidrule` follows; if so, backspace vertical so it will line up with the one you just drew, and setting `\@lastruleclass` to 1 will suppress adding space above the next. If a `\morecimdrules` follows, we add (positive) `\cmidrulesep` (and again set `\@lastruleclass` to one). Otherwise this is the last rule of the current group and we can just add `\belowrulesep`. Finally, we close the `\noalign`.

```
\def\@xcmidrule{\ifx\@tempa\cmidrule\vskip-\@cmidrulewidth
    \global\@lastruleclass=\@ne\else
    \ifx\@tempa\morecmidrules\vskip \cmidrulesep
    \global\@lastruleclass=\@ne\else
    \vskip \belowrulesep\global\@lastruleclass=\z@\fi\fi
    \ifnum0=‘{\fi}}
```

This code (called below) actually draws the rules. They are drawn as boxes in text, rather than in a `\noalign` group, which permits the left and right kerning.

```
\def\@cmidrulea{\multispan\@cmidla&\multispan\@cmidlb
    \unskip\hskip \cmrkern@l\leaders\hrule \@height\@thisrulewidth\hfill
    \hskip \cmrkern@r\cr}
\def\@cmidruleb{\multispan\@cmidlb
    \unskip\hskip \cmrkern@l\leaders\hrule \@height\@thisrulewidth\hfill
    \hskip \cmrkern@r\cr}
```

`\morecmidrules` This is really a dummy command; all the work is done above within the `\cmidrule` routine. We look one step ahead there to see if a `\morecmidrules` follows the current `\cmidrule`, and if so set the flag. Otherwise, `\morecmidrules` itself does nothing.

```
\def\morecmidrules{\noalign{\relax}}
```

⟨/package⟩