L'extension `booktabs`
Tables scientifiques de qualité avec LaTeX *

Simon Fear
300A route de Meyrin
Meyrin
Switzerland

Généré le 17 juillet 2016

Résumé

Cet article décrit quelques commandes supplémentaires pour améliorer la qualité des tables en LaTeX. Dans ce cadre, des principes sont donnés pour constituer des tables visuellement satisfaisantes. La version de l'an 2000 (1.61) de l'extension `booktabs`, décrite ici, ajoute quelques améliorations à celle de 1995 (1.00), essentiellement la compatibilité avec `longtable`.

Les versions ultérieures (1.618, 1.6180, 1.61803 et 1.618033) ajoutent des correctifs, un support de l'extension `colortbl` et une meilleure compatibilité avec `longtable` [1]

1 Introduction

Les commandes décrites ci-dessous facilitent la production de tables telles qu'elles devraient apparaître dans les livres et journaux scientifiques. Ce qui distingue ces tables de celles que LaTeX produit normalement est la présence par défaut d'un espace au-dessus comme au-dessous des filets ainsi que des filets d'épaisseur variable. Ce qui les distingue encore plus des tables que beaucoup de gens produisent en utilisant *pourtant* LaTeX est l'absence de filets verticaux et de filets doubles.

Je dois faire une distinction claire entre ce que j'appelle une *table formelle*, ensemble de valeurs dans des colonnes titrées, et ce que j'appelle un *tableau*. Ce dernier est le genre de choses présentés dans le manuel LaTeX, de plus en plus fréquent en tant que sortie de systèmes de gestion de bases de données ; il aura

*Ce fichier a pour numéro de version v1.618033 (convergeant vers phi, le nombre d'or) et date 27/04/2016. La première traduction en français de « *Publication quality tables in LaTeX* » a été publiée par Jean-Pierre Drucbert et Mathieu Goutelle le 2 mai 2001 sur la base de la version 1.00.

1. Par Danie Els (`dnjels@sun.ac.za`) en l'absence de l'auteur.

probablement des icônes en abondance et de la couleur sans l'ombre d'un doute. La mise en page d'un tel *tableau* est (heureusement) à usage unique, compte tenu du méli-mélo de commandes que le concepteur essaie de combiner en une configuration sensée. À l'opposé, la mise en page d'une *table* a été établie sur la base de siècles d'expérience et ne devrait être altérée que dans des cas extraordinaires.

Pour illustrer ce propos, considérons ce tableau extrait du manuel LaTeX (page 64 de l'ancienne édition [2]) :

moucherons	gramme	13,65€
	la pièce	,01
gnou	farci	92,50
émeu		33,33
tatou	congelé	8,99

C'est un fatras d'informations, probablement présenté de manière raisonnablement claire ainsi (mais l'émeu est-il farci ou pas ?). Cependant, en tant que table publiée, elle devrait certainement suivre les principes donnés dans la suite de ce manuel :

Élément		
Animal	Description	Prix (€)
Moucheron	le gramme	13,65
	la pièce	0,01
Gnou	farci	92,50
Émeu	farci	33,33
Tatou	congelé	8,99

Cette table formelle a demandé un travail de présentation bien moindre ; nous n'avons pas à construire une nouvelle mise en page pour chaque table que nous constituons. De plus, nous pouvons être quasiment certains que les données ne pourront pas être mal interprêtées car le lecteur n'a pas à apprendre comment lire un nouveau type de présentation.

Malheureusement, la table ci-dessus ne peut pas être produite en LaTeX standard. Une tentative de mise en page peut être faite mais, malgré tous nos efforts, l'utilisation de simples commandes `\hline` donne

Élément		
Animal	Description	Prix (€)
Moucheron	le gramme	13,65
	la pièce	0,01
Gnou	farci	92,50
Émeu	farci	33,33
Tatou	congelé	8,99

2. N.D.T. : table ici traduite.

Notez (si ce n'est pas déjà évident) qu'il n'y a pas assez d'espace entre la ligne du haut et le « É » majuscule de « Élément », et que cela se trouve pour toutes les lignes : comparez avec la version précédente. Qui plus est, les filets du haut et du bas dans la première version sont plus gras que le filet du milieu, qui à son tour est plus gras que le filet mineur en-dessous de « Élément ». Bien sûr, vous *pourriez* redéfinir `\doublerulesep` et ensuite utiliser `\hline\hline` pour obtenir quelque chose donnant presque le même effet, et vous pouvez utiliser des cales (avec la commande par exemple) pour améliorer l'espacement. Mais vous ne devriez pas avoir à vous soucier de telles choses. L'extension `booktabs` définit ses propres commandes pour que ces questions soient traitées automatiquement.

En général, cette extension n'a aucun intérêt pour ceux qui cherche une alternative à PicTEX pour générer des tableaux sophistiqués. Elle doit être considérée comme un code typographique pour tables à destination d'auteurs d'articles et de livres scientifiques. Il n'est pas exagéré de dire que si vous ne parvenez pas à créer votre table en utilisant cette extension, vous devriez la revoir en profondeur.

1.1 Note sur la terminologie

En typographie[3], un « trait droit » (*line*) est toujours appelé « filet » (*rule*). Source de confusion éventuelle (pour des raisons historiques), l'« épaisseur » (*thickness*) d'un filet est souvent appelée « largeur » (*width*), alors que tout à chacun l'appellerait « profondeur » ou « hauteur » en pensant à un filet horizontal. Une « ligne noire épaisse » (*thick black line*) est appelée « filet gras » (*heavy rule*). La terminologie anglaise est reprise dans la plupart des noms des nouvelles commandes décrites ci-dessous. Ceci évite au moins la confusion avec `\hline`.

2 Mise en page de tables formelles

Vous ne ferez pas trop d'erreurs si vous gardez à l'esprit à tout moment deux principes simples :

1. Ne jamais, au grand jamais, utiliser de filets verticaux.
2. Ne jamais utiliser de filets doubles.

Ces principes peuvent sembler extrêmes mais je n'ai jamais trouvé une bonne raison pour passer outre. Par exemple, si vous estimez que les informations dans la moitié gauche d'une table sont à ce point différentes de celles de la droite qu'il faut les séparer par une ligne verticale, vous devriez alors plutôt utiliser deux tables. Le second principe n'est pas suivi par tout le monde : j'ai travaillé pour un éditeur qui insistait pour placer un filet double fin au-dessus des rangées de totaux. Ce que je n'aurai pas fait.

Il y a trois autres principes intéressants à mentionner citer ici, ceux-ci étant généralement peu connus en dehors des cercles des typographes et éditeurs professionnels :

3. N.D.T. : le texte d'origine évoque la typographie britannique. Le traduction reprend ici la terminologie française et précise les termes anglais entre parenthèses, ces derniers étant ceux utilisés dans les noms de commande par la suite.

3. Placer les unités dans l'en-tête de colonne (pas dans le corps de la table) ;
4. Faire toujours précéder la virgule décimale par un chiffre, soit, par exemple, 0,1 *au lieu de* ,1 ;
5. Ne pas utiliser de guillemets de répétition (" ou ») ou toute convention analogue pour répéter une valeur précédente. Dans la plupart des cas, un blanc fait aussi bien l'affaire. Si ce n'est pas le cas, répéter la valeur.

Que vous souhaitiez ou pas tenir compte de subtilités mineures, si vous suivez les principes évoqués ci-dessus dans vos tables formelles, votre lecteur vous en sera reconnaissant. Je tiens à préciser que ces principes n'existent pas pour faire plaisir aux tatillons. Le point essentiel est qu'une structure de présentation clarifiée facilite immédiatement la compréhension.

3 Utilisation des nouvelles commandes

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

Dans les cas les plus simples une table commence avec un filet initial, `\toprule`, a une rangée unique d'en-têtes de colonnes, puis un filet de séparation appelé ici `\midrule` ; après les colonnes de données, la table s'achève avec un filet terminal, `\bottomrule`. La plupart des éditeurs de livres rendent les filets `\toprule` et `\bottomrule` plus gras (c'est-à-dire plus larges ou plus sombres ; voir la section 1.1) que le filet intermédiaire `\midrule`. Cependant, lorsque les tables sont composées en très petits caractères, il est parfois impossible de faire cette distinction ; de plus, un bon nombre de journaux utilisent des filets tous de même épaisseur.

Les commandes de filet de cette extension ont toutes une épaisseur par défaut qui peut être modifiée à l'intérieur du document (de préférence, mais pas obligatoirement, dans le préambule). Pour le filet initial et le filet final, il s'agit de `\heavyrulewidth` et, pour les filets intermédiaires, de `\lightrulewidth` (commandes décrites par la suite). Dans de très rares cas, vous pouvez utiliser les arguments optionnels des commandes de filet ayant la syntaxe formelle suivante :

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

où ⟨*largeur*⟩ est une dimension TEX (par exemple 1pt, .5em, etc.).

Toutes ces commandes de filet se placent immédiatement après la commande `\\` achevant la rangée précédente du tableau (sauf bien sûr pour `\toprule`, qui se place juste après le début de l'environnement `tabular`) ; en d'autres termes, exactement là où LATEX autorise traditionnellement `\hline` ou `\cline`.

`\cmidrule`

Il arrive fréquemment d'avoir besoin d'un filet qui ne s'étend que sur certaines des colonnes, ce que permet `\cmidrule` (équivalent à la commande `\cline` de LATEX). En général, ce filet ne devrait pas recouvrir toute la largeur des colonnes, en particulier lorsqu'un filet `\cmidrule` commence immédiatement après la fin d'un autre (des `\cline` de LATEX peuvent se toucher si vous n'êtes pas extrêmement attentifs à `\extracolsep`). Aussi, des options de raccourcissement vont généralement être utilisées.

Ces options de raccourcissement se placent entre parenthèses (comme ceci), sans aucune espace entre elles. Les valeurs possibles sont `r`, `r{`⟨*largeur*⟩`}`, `l`, `l{`⟨*largeur*⟩`}` ou toute combinaison de quatre valeurs précédentes. `r` et `l` indiquent si les extrémités droite et/ou gauche doivent être rognées tandis que ⟨*largeur*⟩ est une dimension. La commande sans argument explicite est équivalente à `r{\cmidrulekern}`, où `\cmidrulekern` vaut par défaut 0,5 em mais peut être redéfini par l'utilisateur dans le préambule [4].

À titre d'exemple, `(lr{.75em})` génère un filet à extrémité gauche rognée par défaut et à extrémité droite rognée d'exactement de 0,75 em. L'option `(r{.75em}l)` est également valide [5].

La syntaxe complète de la commande est :

`\cmidrule[`⟨*largeur*⟩`](`⟨*rognage*⟩`){`*a*–*b*`}`

où ⟨*largeur*⟩ est une option d'épaisseur de filet, entre crochets (la valeur par défaut étant `\cmidrulewidth`), et le dernier argument, *non optionnel*, donne le numéro des premières et dernières colonnes sur lesquelles s'étend le filet.

Voici un exemple d'utilisation de ces commandes avec le code utilisé pour produire l'exemple de table ci-dessus :

```
\begin{tabular}{@{}llr@{}}  \toprule
\multicolumn{2}{c}{\'{E}lément}           \\ \cmidrule(r){1-2}
Animal     & Description   & Prix (\euro) \\ \midrule
Moucheron  & le gramme     & 13,65        \\
           & la pièce      & 0,01         \\
Gnou       & farci         & 92,50        \\
\'{E}meu   & farci         & 33,33        \\
Tatou      & congelé       & 8,99         \\ \bottomrule
\end{tabular}
```

`\addlinespace` À l'occasion, il peut être pertinent d'insérer un espace supplémentaire entre certaines rangées d'une table ; par exemple, avant la dernière rangée, s'il s'agit d'un total. Ceci s'obtient simplement en insérant :

`\addlinespace[`⟨*largeur*⟩`]`

après la marque d'alignement `\\`. L'effet est alors tout à fait identique à celui de `\\[\defaultaddspace]`, que je trouve plutôt maladroit, entre des lignes de texte ordinaire, et est meilleur que `\\ \\`, qui insère trop d'espace. De même, `\addlinespace` peut être utilisé avant, après ou entre les filets si vous souhaitez contrôler exactement l'espace à insérer. L'espace par défaut placé avant ou après un filet est alors remplacé exactement par `\defaultaddspace` ou l'espace spécifié dans l'argument optionnel [6].

4. Des retours utilisateurs ont suggéré que la valeur par défaut de la version 1.00, 0,25 em, était trop petite. Désolé pour cette perte de rétrocompatibilité. Rappelez-vous que vous pouvez facilement redéfinir `\cmidrulekern` en préambule, ou juste retenir `(r{.25em})` pour retrouver le comportement d'origine.

5. Pour être tout à fait précis, `(lrrlr{.75em})` génère également le même résultat : seules les dernières options droite et gauche rencontrées sont appliquées.

6. Il s'agit d'un changement par rapport à la version 1.00 où l'espace était parfois *ajoutés* à l'espace autour du filet par défaut.

4 Abus avec les nouvelles commandes

Il faut le reconnaître : tout ceci ne marche parfois pas tout seul. Quelques conseils et commandes supplémentaires sont apportés ici.

Les nouvelles commandes créant des filets n'ont pas la garantie de fonctionner avec `\hline` ou `\cline`, bien que celles-ci restent disponibles et inchangées. Ici, je ne vois aucune raison a priori pour vouloir les mélanger.

Point plus important, les filets engendrés par les nouvelles commandes ne sont pas spécialement pensés pour se connecter aux filets verticaux engendrés par des caractères `|` dans le préambule de la table. Ceci est un choix fonctionnel (voir plus haut). Vous ne devriez pas utiliser de filets verticaux dans les tables, point final.

`\morecmidrules` Si vous ne pouvez pas vous empêcher d'utiliser un filet double, même une construction aussi bizarre que `\toprule\bottomrule\midrule` fonctionne sans provoquer de message d'erreur (tout comme vous pouviez recourir à une double `\hline`). Ces filets sont séparées par l'intervalle classique `\doublerulesep` de LaTeX. Cependant si votre perversion va jusqu'à vouloir des `\cmidrule` doubles, vous aurez besoin de la commande supplémentaire `\morecmidrules` pour le faire correctement, car, normalement, deux commandes `\cmidrule` de suite forment une construction parfaitement correcte demandant deux filets sur la même « ligne ». Ainsi, dans

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

la seconde commande écrit un filet qui vient se superposer exactement sur le premier ; et je suppose que vous souhaitiez plutôt

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

qui donne un filet double pour les colonnes une et deux, séparés de `\cmidrulesep` (comme une `\cmidrule` donne un filet très fin, la valeur ordinaire `\doublerulesep` donnerait probablement un espacement trop grand). Il faut terminer une rangée complète de filets avant de mettre la commande `\morecmidrules`. Notez que `\morecmidrules` n'a aucun effet si elle ne suit pas immédiatement une `\cmidrule` (elle n'est donc pas une commande générale d'espacement).

`\specialrule` Si vous avez l'extraordinaire besoin de spécifier exactement l'espacement entre deux filets à 0.5 em (par exemple), vous pourriez utiliser une construction telle que `\midrule \addlinespace[.5em] \midrule`. Par un rare accès de tolérance, cependant, j'ai également mis à disposition la commande

`\specialrule{`⟨*largeur*⟩`}{`⟨*espace-au-dessus*⟩`}{`⟨*espace-au-dessous*⟩`}`

dans laquelle les trois arguments sont obligatoires (je ne me suis pas soucié d'établir des valeurs par défaut). Si vous utilisez ceci fréquemment, vous n'avez pas compris le but essentiel des conseils donnés ci-dessus. Le filet qui précède n'ajoute pas son espace par défaut et le filet qui suit n'ajoute non plus pas son espace par défaut qui le précède : ainsi vous avez *exactement* l'espacement indiqué dans les arguments [7].

7. Il s'agit d'un changement par rapport à la version 1.00, qui préférait ajouter un espace `\doublerulesep` supplémentaire à chaque fois que c'était possible.

5 Booktabs et l'extension longtable

Si les deux extensions `booktabs` et `longtable` sont chargées, les commandes de filets de `booktabs` peuvent toutes être utilisées exactement comme décrit plus haut dans une table « longtable ».

Il faut mentionner ici un ajout particulier : dans une table « longtable », vous pouvez utiliser des commandes optionnelles de raccourcissement à gauche et à droite qui ne fonctionnent normalement que sur les `\cmidrule`, `\toprule`, `\midrule` et `\bottomrule` (et, si nécessaire, aussi sur les `\specialrule`). Des utilisateurs ayant bidouillé le code de la version précédente pour obtenir une compatibilité avec `longtable` [8] semblent avoir touts aimé disposer de filets raccourcis de 0.5 em. Vous devriez pouvoir obtenir la même chose en faisant de `@{}` le spécificateur de votre dernière colonne. Ceci étant, après avoir revu le reste du code, il était facile d'ajouter un test pour les arguments optionnels, ce que j'ai fait (je n'ai cependant pas fait le développement intégral permettant d'utiliser les arguments de raccourcissement *hors* d'une table « longtable ». Si vous voulez des filets raccourcis, passez toutes vos tables en version `longtable`!)

Pour finir ce point, un point quelque peu technique : dans une table « longtable », `\hline` et `\hline\hline` produisent toutes deux un filet *double* (pour permettre un saut de page à cet endroit). Mais les règles de `booktabs` *ne l'autorisent pas*. Le doublement automatique de `\hline` par `longtable` est discutable, ainsi que le précise d'ailleurs la documentation de l'extension. Mais le doublement des filets par `booktabs` n'a aucun sens. Dans le cas malheureux où un filet est mis par `booktabs` lors d'un saut de page, vous devrez faire les ajustements nécessaires à la main [9] (en général, cela signifie retirer le filet incriminé).

6 Booktabs et l'extension colortbl

`Booktabs` est désormais compatible avec l'extension `colortbl` [10]. La commande `\arrayrulecolor` donne des filets en couleur si l'extension `colortbl` est chargée.

7 Profil technique des commandes

Les nouvelles commandes de filet sont valides au sein des environnements `tabular` (et `array`), des environnements `tabular` et `array` modifiés par l'extension `array` et dans les tables classiques et `longtable` après le chargement de l'extension `longtable`.

Les commandes suivent les règles de placement standard de `\hline`. Il peut y avoir des espaces (incluant un retour à la ligne mais pas deux) entre deux com-

8. Jim Service a été le premier.
9. Point résolu en version 1.618033 (Morten Høgholm).
10. Depuis la version v1.6180.

mandes consécutives de filet [11].

✖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. [12] 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, `\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 [13]. There is a frequent and legitimate reason you might want space below a bottom rule : namely, when there's a table footnote. [14] 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. [15] 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).

11. Un changement bienvenu par rapport à la version 1.00 où des espaces entre ces commandes générait un message d'erreur vraiment déroutant.
12. 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.
13. This is a change from Version 1.00, where there was always a `\belowrulesep`
14. But don't use footnotes, Donald.
15. I don't like either of these. Sort it out in Version 1.618 ?

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 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 `\cmidrules` 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 9). 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 9 ; 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.

8 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 ...

9 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 :

```
\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= }
```

9.1 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 (\ifnum0=`}) 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=\belowrulesep %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]{%
```

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
    \nobreak
    \let\@BTswitch\@BLTrule
  \else
    \let\@BTswitch\@BTnormal
  \fi\fi
```

Call `\@BTswitch` at end of macro

```
  \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`.

```
\@BTswitch}
```

`\CT@arc@` This is support for the colortbl package for colored rules. `\CT@arc@` hold the `\arrayrulecolor` setting.

```
\AtBeginDocument{%
  \providecommand*\CT@arc@{}}%% colortbl support
```

`\@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{%
   {\CT@arc@\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
  \ifnum\@lastruleclass=\@ne\relax\else\vskip\@belowrulesep\fi
  \ifnum0=‘{\fi}}
```

9.2 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
 {\def\@tempb{r}%
   \ifx\@tempa\@tempb
      \global\let\cmrkern@r\cmidrulekern
      \def\cmrsideswitch{\cmrkern@r}%
   \else
      \def\@tempb{l}%
      \ifx\@tempa\@tempb
         \global\let\cmrkern@l\cmidrulekern
         \def\cmrsideswitch{\cmrkern@l}%
      \else
         \global\expandafter\let\cmrsideswitch\@tempa
      \fi
   \fi}}
```

`\cmidrule` `\@cmidrule` `\@@cmidrule` `\@@@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}
```

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

\@cmidrulea 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%
  {\CT@arc@\leaders\hrule \@height\@thisrulewidth\hfill\kern\z@}%
   \hskip\cmrkern@r\cr}%
```

\@cmidruleb

```
\def\@cmidruleb{%
    \multispan\@cmidlb
    \unskip\hskip \cmrkern@l%
   {\CT@arc@\leaders\hrule \@height\@thisrulewidth\hfill\kern\z@}%
    \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⟩

✖

Historique

Index

Les numéros en italique renvoient à la page où se trouve l'entrée correspondante; les numéros soulignés renvoient à la ligne de code de la définition; les numéros en romain renvoient aux lignes de code où l'entrée est utilisée.