multicolrule — Decorative rules between columns[*]

Karl Hagen[†]

Released 2019/02/12

Abstract

The multicolrule package lets you customize the appearance of the vertical rule that appears between columns of multicolumn text. It is primarily intended to work with the multicol package, hence its name, but it also supports the twocolumn option and `\twocolumn` macro provided by the standard classes (and related classes such as the KOMA-Script equivalents), as well as the bidi package (and through it, all RTL scripts loaded with polyglossia).

Contents

[*]This file describes version v1.2a, last revised 2019/02/12.
[†]latex@polysyllabic.com

1 Introduction

`line-style=dashed`

In LaTeX, there are two lengths that control the formatting between columns of multicolumn text: `\columnsep` specifies the space between adjacent columns, and `\columnseprule` specifies the width of a solid vertical rule that is placed centered between the columns. The multicol package adds the ability to change the color of the rule, but in both vanilla LaTeX and multicol, the rule itself is drawn directly inside the routines that output the column boxes, and is therefore difficult for users to alter.

Of course it's a legitimate question why anyone should *want* to change this rule, or indeed use one at all, as good typography tends to avoid using large vertical lines.[1] In my own case, I needed to modify the rule because of the requirements of a particular style I was imitating, and having done the hard work of creating the necessary infrastructure for one line style, it was simple to extend the solution to a more general case. I hope someone else will find the options here useful.

The basic line styles that multicolrule makes available are illustrated throughout this guide. The default line-width used is 0.4pt (thin), and the default color is `Maroon`. You can also find examples of rules created with all available options in the file `mcrule-example.pdf`.

`custom-line={\path (TOP) to [ornament=85] (BOT);}, extend-top=-24pt, extend-bot=-8pt`

New for Version 1.2

Version 1.2 adds the ability to define patterns, which are aliases for a series of `\SetMCRule` settings. With patterns, you can change individual separators on the same page. For example, in three-column text, the left separator can differ from the right. You can also alter the appearance of one or more separators anywhere within the environment (see section 3.6).

New for Version 1.1

Version 1.1 supports drawing decorative rules if you have the bidi package loaded, which can occur automatically if you set a right-to-left language with polyglossia. It also provides a mechanism to extend or shrink rules beyond the natural height of the columns, as well as to have the rule fill the available space to the end of the text area (see section 3.5).

1.1 Bugs and Known Limitations

`line-style=dots`

The multicolrule package is written using expl3 syntax, and so requires a less-than-ancient installation of LaTeX. It requires the packages l3keys2e, xparse, xpatch, xcolor, scrlfile, and depending on the mode of operation may also require multicol and tikz. If you have an up-to-date distribution, these requirements should cause no issues.

I am sure that there are bugs that remain to be uncovered, inefficient methods that could stand improvement, and useful features that still need to be implemented. The development code is maintained on github, and you can file feature requests or bug reports there. Alternatively, you can send an email to latex@polysyllabic.com. I welcome contributions for additional styles, especially to provide more options when running the package without tikz.

The following are the issues I'm currently aware of that aren't multicolrule errors but which may cause buggy looking behavior:

[1] See, for example, the remarks in the documentation for the booktabs package.

This package works by patching the output routines of either multicol or the LaTeX kernel, depending on the mode of operation. If bidi is loaded, it will also patch that. It will have no effect if you use a class or package that outputs column text via alternate mechanisms. This includes parcolumns, and probably other classes and packages designed to typeset parallel-column text as well, although I have not done a survey to determine whether this is the case. If you would like support for one of these, please send me an email or file a feature request on github and I'll see what I can do.

The line styles that work by repeating elements in a tiled pattern may have significant gaps at the end of columns, particularly for larger patterns. You can mitigate this problem slightly by tweaking the spaces above and below a pattern, but the basic problem is a side-effect of the way these patterns are implemented (with `\cleaders`), which means that only an integer number of copies can be produced. Lines drawn with tikz do not have this problem.

Extending rules beyond their natural column lengths can seriously mess up the output, including, in certain edge cases, causing multicol to overprint columns or even put them in the margins. The fact that the extended rule affects the vertical layout was a deliberate design decision and is necessary to support the `extend-fill` and `extend-reserve` options. A future version may support drawing the rules to a background layer so that the text is not shifted.

1.2 License

`line-style=dotted, width=ultra-thick`

2 Package Options

2.1 Default Operation

`line-style=dash-dot`

Loading multicolrule with its default settings enables multicol support, and that package will be loaded if it hasn't been already. Note that if you need to pass any parameters to multicol, such as `docolaction`, you should load multicol with the appropriate settings *before* you load multicolrule, as LaTeX does not support reloading packages with different parameters.

2.2 Option 'tikz'

You can use more line styles if you also use the tikz package. Some line styles are only available if tikz is enabled, and others look better with it. The default behavior of multicolrule depends on the status of the tikz package at the time multicolrule is loaded. If multicolrule detects that tikz is already loaded, then tikz support will be enabled by default. Otherwise, you need the `tikz` to enable it. This option also accepts explicit boolean values, so you can pass `tikz=false` if you want to explicitly disable tikz support. If tikz support is not enabled (or if it is explicitly disabled), the line styles marked *tikz only* in section 3.1 will be unavailable and errors will result if you try to use them.

2.3 Option 'twocolumn'

The multicolrule package recognizes the option `twocolumn`, either as a package option or as a global class option. If you pass this option to your document class, you do not need to pass it a second time to the package. It is only necessary to use the package option if you plan to have a predominantly one-column document and use `\twocolumn` to switch temporarily into two-column mode.

Because multicol does not work well with the ordinary two-column mode, multicolrule is only designed to work with one or the other at a time. If you try to use the `twocolumn` option when multicol has already been loaded, you will receive a warning and nothing is guaranteed. But the custom rules will at best only appear in the conventional two-column mode and not within a `multicols` environment.

3 The User Interface

`\SetMCRule {`⟨*key-value list*⟩`}`

`line-style=circles, width=2pt`

The main user command for multicolrule is `\SetMCRule`. It takes one parameter containing a key-value list of all options you want to set. You can issue this command in the preamble or the document body. Changes to the rule settings are local to the current group. For example, if you call `\SetMCRule` inside a `multicols` environment, the rule settings will revert to their previous values once the environment ends. Also note that any changes made with `\SetMCRule` when multiple columns are active will appear starting on the same page as your current location when you issue the command, and will extend the height of the full column box. It is not possible to have a rule change styles in the middle of a page unless you close out one `multicols` environment and begin another.

Table 1 summarizes the keys available in `\SetMCRule`. The functions of each is described in detail in the sections that follow.

Table 1: `\SetMCRule` keys

Key	Purpose
`color`	Set the color of the rule (see sec. 3.2)
`color-model`	Set the color model of the rule (see sec. 3.2)
`custom-line`	Set a custom tikz line for the rule (*tikz only*; see sec. 3.1.1)
`custom-pattern`	Set a custom individual pattern for the rule (see sec. 3.1.1)
`custom-tile`	Set a custom tiling pattern for the rule (see sec. 3.1.1)
`double`	Draw two copies of the rule (see sec. 3.4)
`expand`	Change the extend distance by the same amount at the top and bottom of the column (see sec. 3.5)
`extend-bot`	Set an extra amount to extend the rule at the bottom of the column (see sec. 3.5)
`extend-fill`	Extend rule to the bottom of the text area (*multicol* only; see sec. 3.5)
`extend-reserve`	Space to reserve at bottom of text area when using `extend-fill` (*multicol* only; see sec, 3.5)
`extend-top`	Set an extra amount to extend the rule at the top of the column (see sec. 3.5)
`line-style`	Select the type of rule printed (default=*default*; see sec. 3.1)
`pattern-after`	Number of separators to delay before beginning to use the specified patterns (default=0; see sec. 3.6)

Table 1: \SetMCRule keys (cont.)

Key	Purpose
`pattern-for`	Number times separators to apply the patterns to before returning to default (default=-1; see sec. 3.6)
`patterns`	Specify one or more patterns use to draw rules. (default=*none*; see sec. 3.6)
`single`	Draw a single copy of the rule (*default*; see sec. 3.4)
`repeat`	Set the number of times to draw the rule (see sec. 3.4)
`repeat-distance`	Set the horizontal space between adjacent copies of repeated rules (see sec. 3.4)
`shift`	Shift the extend distance downward; this affects both the top and bottom of the column (see sec. 3.5)
`triple`	Draw three copies of the rule (see sec. 3.4)
`width`	Set the width of the rule (see sec. 3.3)

3.1 Styles with the 'line-style' option

`line-style=solid-circles, width=4pt`

You can choose a style for the rule with the `line-style` key. If the predefined styles are insufficient for your purpose, see section 3.1.1 for different ways to customize the rule in even more radical ways. The width of many line styles scales directly with the setting of `\columnseprule`, the default LaTeX length that controls the width of the column rule, but even those that do not, the width must be non-zero for the rule to display (see section 3.3).

Table 2 summarizes the available line styles. Most of the basic patterns come in three versions, differing only in how closely the pattern is spaced: normal, dense, and loose. These settings parallel those found in tikz and use the same spacing between elements. There are no named settings for double lines and the like because you control that feature separately, with the `repeat` key. All line styles can be repeated as many times as you like (see section 3.4).

Table 2: Styles available for the line-style key

Style	Description
`circles`	A series of hollow circles (*tikz only*)
`dash-dot`	A dash followed by a square dot (*tikz only*)
`dash-dot-dot`	A dash followed by two square dots (*tikz only*)
`dashed`	A series of dashed lines
`default`	A solid rule drawn the same way as the default multicol rule. Does not support extended rules.
`dense-circles`	The same as `circles` but more closely spaced (*tikz only*)
`dense-dots`	The same as `dots` but more closely spaced
`dense-solid-circles`	The same as `solid-circles` but more closely spaced (*tikz only*)
`densely-dash-dot`	The same as `dash-dot` but more closely spaced (*tikz only*)

Table 2: Available line-style settings (cont.)

Style	Description
`densely-dash-dot-dot`	The same as `dash-dot-dot` but more closely spaced (*tikz only*)
`densely-dashed`	The same as `dashed` but more closely spaced
`densely-dotted`	The same as `dotted` but more closely spaced
`dots`	A series of dots drawn with the period (full-stop) of the current font
`dotted`	A series of square dots
`loose-dots`	The same as `dots` but spaced further apart
`loose-circles`	The same as `circles` but spaced further apart (*tikz only*)
`loose-solid-circles`	The same as `solid-circles` but spaced further apart (*tikz only*)
`loosely-dash-dot`	The same as `dash-dot` but spaced further apart (*tikz only*)
`loosely-dash-dot-dot`	The same as `dash-dot-dot` but spaced further apart (*tikz only*)
`loosely-dashed`	The same as `dashed` but spaced further apart
`loosely-dotted`	The same as `dotted` but spaced further apart
`solid`	A solid line, like `default`, but supports extending rules
`solid-circles`	A series of filled circles (*tikz only*)

`line-style=solid`

The `default` and `solid` line styles are nearly the same, except that the `solid` line (as of version 1.1) supports the rule-extension commands described in section 3.5. This means that if you want a solid rule with altered top or bottom extensions, you must explicitly set the line style to `solid`. If you make no calls to `\SetMCRule` after loading multicolrule, the column divider will continue to behave exactly as it does with the ordinary multicol package.

You can alter the rule's width and color either through `\SetMCRule` with the `width` and `color` keys described in sections 3.3 and 3.2, respectively, or directly by changing the value of `\columnseprule` and renewing the `\columnseprulecolor` macro.

The `dots` style and its variants are rendered with a period (.) in the currently active font. This is one of the styles, mentioned above, that do not change their size as the line width increases. The same is true of custom tiles.

The `dotted` styles differ from `dots` in that the former are squares with side lengths equal to `\columnseprule`. This mirrors the behavior of the equivalently named dotted patterns in tikz.

3.1.1 Custom Patterns

`custom-tile` = {⟨*pattern*⟩} {⟨*space above*⟩} {⟨*space below*⟩}

`custom-tile=`
`{\SparkleBold}`
`{16pt}{16pt}`

There are three options to create custom rules with multicolrule. The first is the `custom-tile` key. This creates a rule consisting of vertically stacked boxes of arbitrary content—the tile—running the height of the column separator. The `custom-tile` key takes three parameters, which must all be enclosed brackets and may not be omitted. The first should contain the tokens you want to appear as the content of the tile. The second is a dimension specifying the leading vertical space to apply above each copy of the tile. The third is a dimension specifying the trailing vertical space to insert below each copy of the tile.

The rule in this section uses the `\SparkleBold` symbol from bbding. Notice that when you use the `custom-tile` parameter, of any of the other custom key commands, you do *not* specify a separate `line-style`. If you try to provide both, the last style given in the list will be the one that is kept.

`custom-pattern = {`⟨*pattern*⟩`} {`⟨*shift down*⟩`} {`⟨*shift up*⟩`}`

```
custom-pattern= {\HandRight} {0pt}{0pt}
```

The second custom option is with the `custom-pattern` key. The syntax is identical to that for `custom-tile`, but the content you specify will appear once per page or column pair (if the columns occupy less than a full page). This content will be vertically centered if the second and third parameters are both 0pt. You can shift the content down by increasing the second parameter, and up by increasing the third. The rule in this section uses the `\HandRight` symbol from bbding.

`custom-line = {`⟨*draw command*⟩`}`

```
custom-line={\path (TOP) to [ornament=88] (BOT);}
```

The third custom pattern involves setting your own tikz drawing function using the key `custom-line`. The rule in this section is drawn with an ornament from pgfornaments. Obviously, this feature requires tikz support. The value you provide to the `custom-line` key should consist of a tikz command, such as `\draw` or `\path`, without the surrounding `tikzpicture` environment.

Before the drawing command is called, multicolrule will set up a `tikzpicture` with both the x- and y-coordinates scaled to points, and two nodes, named `(TOP)` and `(BOT)`, which are set to the coordinates of the top and bottom of the rule. You can then specify your own `\draw` or `\path` function in whatever way you like. The rule separating these columns was drawn with a decorative element from the pgfornaments package.

This function will use the color set in `\columnseprulecolor` if you don't set it explicitly within the tikz command, but you must provide everything else necessary to draw the line correctly, including the line width. Note that nothing limits you to drawing a picture that fits within the space between the columns. If the rule is wider than the available space, it will be centered between the columns and overlap the text. Normally, of course, that will be undesirable, but you can use it to your advantage in certain cases. The file `mcrule-example.pdf` contains examples showing the effect of a rule that is too wide, as well as a custom rule which includes horizontal rules at the top and bottom of the column.[2]

3.2 Colors

```
line-style=solid,
width=2pt
color-model=cmy,
color={0.7,0.5,0.3}
```

You can set colors for the rule through the `color` and, optionally, the `color-model` keys. multicolrule loads the xcolor package to manage colors, and the `color` parameter accepts any name that xcolor recognizes, either natively or as the result of any names you have defined with `\definecolor` (see the xcolor documentation). Note that if you want to use color names that are defined through the one of xcolor's package options, you must load xcolor before both multicolrule and tikz with the relevant options.

To specify a color by a numeric specification, you use the `color-model` parameter

[2] This latter rule was developed as an answer to StackExchange question 473828.

to specify any color model that xcolor recognizes (rgb, cmy, etc), and `color` to hold the color-specification list. Because that list is itself comma-separated, you must enclose it in brackets.

The current color setting can always be found in `\columnseprulecolor`. If you are running in twocolumn mode without multicol, this command will be provided and colors will work the same way they do with multicol. Note that setting the `color` key causes `\columnseprulecolor` to be redefined within the current group only. If you directly redefine `\columnseprulecolor`, the color of the custom rule will reflect this setting. This way, the settings of any packages that might alter the rule color will be respected.

3.3 Width

`line-style= dash-dot-dot, width=thick`

You can set the width of the rule with the `width` key. Legal values are any explicit dimension or dimension expression, as well as with names that parallel those used by tikz, except that spaces in the key names are replaced with hyphens.

The current width of the rule is kept in `\columnseprule`, just as in vanilla LaTeX, and if it is set separately, the custom rule's width will reflect this change. Note that although some line styles do not depend directly on `\columnseprule` to calculate their actual width, the value of `\columnseprule` must be greater than 0pt for any rule to appear. This behavior is intentional and is in keeping with the way the default column rules work.

Table 3: Sizes of named line widths

Name	Width
`ultra-thin`	0.1pt
`very-thin`	0.2pt
`thin`	0.4pt
`semithick`	0.6pt
`thick`	0.8pt
`very-thick`	1.2pt
`ultra-thick`	1.6pt

3.4 Repeated Rules

`line-style= dash-dot-dot, triple=2pt`

You can draw multiple, adjacent copies of any rule by setting the number of times to draw the rule with the `repeat` key. The space between copies is controlled with the `repeat-distance` key. Initially, this distance is set to `\columnseprule`. Note that you must enter an actual dimension expression for this distance. The names used for line widths are not accepted.

The keys `single`, `double`, and `triple` are shorthand methods to set the number of repeats and the `repeat-distance` at the same time. If you use the key without a value `repeat-distance` is set to `\columnseprule`.

There are no checks made to ensure that repeated rules will fit in the available space between columns, so you should be careful using these commands, especially with thicker rules.

3.5 Extended Rules

line-style=dashed, expand=-16pt

You can specify an additional amount by which the top or bottom of the rule projects beyond the column's natural length with the keys extend-top and extend-bot, each of which can be set to a dimension expression. Extending the top of the rule with a positive dimension will push the columns down from any preceding material. A positive value for extend-bot does the same in the other direction when a column ends in the middle of a page, but the rule will extend into the the bottom margin if the column goes to the end of the page.

Note that positive values for extending the rules should be used with caution, only in situations where you need a special effect for one column (see section 3.6 for a way to limit the change to one or a few columns) or a small multicol environment. Overprinting and other bizarre effects can result from extending the rule in the wrong place. Negative values for both keys may be more generally useful, as they have the effect of shrinking the rule. This behavior is illustrated with the rule for this section.

The expand and shift keys provide shorthands for two common situations. You can use expand to set the same value for extend-top and extend-bot For example, expand=-16pt is equivalent to extend-bot=-16pt, extend-top=-16pt. The shift key moves the rule downward for positive values and upward for negative ones without changing the overall length of the rule. More precisely, shift=x translates to extend-bot=x, extend-top=-x. For example, shift=16pt is equivalent to extend-bot=16pt, extend-top=-16pt.

line-style=solid, extend-fill

The extend-fill key is a boolean option that, when set to true, will extend the rule to occupy any space between the bottom of the columns and the end of the text area. Providing the key with no value is equivalent to extend-fill=true. This option is only relevant for the multicols environment. It will have no effect with either multicols* or the plain LaTeX two-column mode.

If you want text below and on the same page as the multicols environment when using extend-fill, you can reserve space for it with extend-reserve, which takes a dimension expression specifying the vertical space to leave available after the rule. If the value is greater than zero, the height of the extended line will be reduced by the reserved amount plus the value of \multicolsep. In other words, you only have to specify the actual space you need for the text itself, not the space that multicol adds automatically below the columns. Note that if the amount you request for reserved space is less than the amount actually available at the end of the page, the rule will not extend below the columns and you probably will find this material spilling onto the next page anyway.

3.6 Rule Patterns

```
\DeclareMCRulePattern {⟨name⟩} {⟨key-value list⟩}
```

`patterns={right-hand, left-hand}`
`See the code sample below for the definitions of the patterns`

A *pattern* refers to a bundle of settings used by multicolrule. Although you can use patterns as a shortcut to save you a little typing, their main purpose is to let you to alter individual rules within a multicol environment. For example, if you have three-column text, you can make the left rule different from the right one. In two-column text, you can have different rules for alternating pages.

You declare a pattern for a line style with the command `\DeclareMCRulePattern`. The ⟨*name*⟩ should consist of letters and hyphens only. The ⟨*key-value list*⟩ can contain all keys that are valid for `\SetMCRule` with the exception of `patterns`, which is filtered out. In other words, if you put something like `patterns=foo` in the pattern definition, it will be ignored.

Once you have declared a pattern, you can use it as a value for the `patterns` argument of `\SetMCRule`. This key can accept either a single pattern or a comma-separated list of patterns. If you use a comma-separated list, make sure you enclose it in braces.

When a pattern is in effect, its settings are applied on top of the prior settings. If you set the key to an empty list, any patterns currently in effect will be canceled, and multicolrule will revert to the previous settings.

If the `patterns` key contains more than one pattern, multicolrule will cycle through the list of patterns, using one pattern each time a rule is drawn between columns. Note that the patterns do not cycle within a single column separator if you use the `repeat` key. This cycle is global, so if the number column separators is not a multiple of the number of patterns and you start a new `multicols` environment with the same patterns in effect, the cycle will pick up where it left off. Every time you set new patterns, however, the cycle begins anew with the first pattern in the list.

The columns above were defined with the following:
```
\DeclareMCRulePattern{left-hand}{custom-tile={\HandLeft}{8pt}{8pt}}
\DeclareMCRulePattern{right-hand}{custom-tile={\HandRight}{8pt}{8pt}}
\begin{multicols}{3}
  \SetMCRule{patterns={right-hand,left-hand}}
  ...
\end{multicols}
```

`patterns=shrink-me, pattern-for=1`
`See the code sample below for the definition of ‘shrink-me’`

If you want to alter the rule only for certain column separators, you can use the `pattern-after` and `pattern-for` keys, both of which take integer values, in conjunction with `patterns`.

The `pattern-for` key means “use the given pattern or patterns for this many column separators only.” Afterwards, the pattern will be disabled, meaning that it won’t be applied any more and only the settings applied directly will be in effect until it is reset. A negative value to this key means that the patterns will be repeated indefinitely. The default is −1.

The `pattern-after` key means “wait until after this many column separators before starting to apply the pattern. The default is 0. If you use it in conjunction with `pattern-for`, the count of modified column separators begins after the skipped columns.

For example, suppose you have four-column text and want to alter the third column separator on the

[3]Remember that you have one less column separator than you have columns.

first page of the environment only.[3] You could accomplish this task with the code below.

Using predefined patterns adds processing overhead, since they must be applied each time the rule is drawn. Therefore it is more efficient to avoid patterns unless you need to actually change the line style from column to column, although if you compile on a reasonably modern computer, you are unlikely to notice too much delay.

Note that any settings you provide in the same command where you apply a `patterns` key do not alter definition of the pattern, even if they come after the `patterns` key. Such settings will take effect before the pattern is applied and will reappear after the pattern ends, if it does.

Shrinking the final two column separators in four-column text:

```
\DeclareMCRulePattern{shrink-me}{line-style=solid,
   extend-top=-3\baselineskip}
\begin{multicols}{4}
  \SetMCRule{patterns=shrink-me,pattern-after=1,pattern-for=2}
  ...
\end{multicols}
```

4 Implementation

⟨*package⟩

⟨@@=mcrule⟩

4.1 Preliminaries

```
\ProvidesExplPackage {multicolrule} {2019/02/12} {1.2a}
  {Decorative vertical rules between columns}
```

We always need these packages.

```
\RequirePackage{l3keys2e}
\RequirePackage{xpatch}
\RequirePackage{xcolor}
\RequirePackage{scrlfile}
```

Define the messages we use.

```
\msg_new:nnn {multicolrule} {patch-success} {Patched~#1.}
\msg_new:nnn {multicolrule} {patch-failure} {Error~patching~#1.}
\msg_new:nnnn {multicolrule} {tikz-required} {Tikz~required}
{The~'#1'~setting~requires~tikz~to~work.~Either~load~tikz~before~you~load~
  multicolrule~or~use~multicolrule's~'tikz'~package~option.}
\msg_new:nnnn {multicolrule} {multicol-loaded} {Multicol~loaded} {You~are~
 using~the~'twocolumn'~option~with~multicol~already~loaded.~You~will~likely~
  run~into~problems.}
\msg_new:nnnn {multicolrule} {pattern-undefined} {Pattern~undefined}
  {The~multicolrule~pattern~'#1'~has~not~been~defined.}
```

\g__mcrule_twocolumn_bool
\g__mcrule_use_tikz_bool

Flags for package options

```
\bool_new:N \g__mcrule_twocolumn_bool
\bool_new:N \g__mcrule_use_tikz_bool
```

(End definition for `\g__mcrule_twocolumn_bool` *and* `\g__mcrule_use_tikz_bool`*.)*

\l__mcrule_repeat_int
\l__mcrule_repeat_distance_dim

Variables to support repeated copies of the rule.

```
\int_new:N  \l__mcrule_repeat_int
\int_set:Nn \l__mcrule_repeat_int {1}
\dim_new:N  \l__mcrule_repeat_distance_dim
```

(*End definition for* `\l__mcrule_repeat_int` *and* `\l__mcrule_repeat_distance_dim`*.*)

\l__mcrule_extend_top_dim
\l__mcrule_extend_bot_dim
\l__mcrule_extend_fill_bool
\l__mcrule_extend_reserve_dim

Variables to control the distance to extend the rule above and below the natural column height.

```
\dim_new:N  \l__mcrule_extend_top_dim
\dim_new:N  \l__mcrule_extend_bot_dim
\bool_new:N \l__mcrule_extend_fill_bool
\dim_new:N  \l__mcrule_extend_reserve_dim
```

(*End definition for* `\l__mcrule_extend_top_dim` *and others.*)

\l__mcrule_color_name_tl
\l__mcrule_color_model_tl

Keep name and color model so we can set them separately while retaining the value of the other one.

```
\tl_new:N \l__mcrule_color_name_tl
\tl_new:N \l__mcrule_color_model_tl
```

(*End definition for* `\l__mcrule_color_name_tl` *and* `\l__mcrule_color_model_tl`*.*)

\g__mcrule_patterns_prop
\g__mcrule_pattern_count_int
\g__mcrule_pattern_for_int
\g__mcrule_pattern_after_int
\l__mcrule_pattern_list_seq

Variables to support defined patterns.

```
\prop_new:N \g__mcrule_patterns_prop
\int_new:N  \g__mcrule_pattern_count_int
\int_new:N  \g__mcrule_pattern_for_int
\int_new:N  \g__mcrule_pattern_after_int
\seq_new:N  \l__mcrule_pattern_list_seq
```

(*End definition for* `\g__mcrule_patterns_prop` *and others.*)

If tikz is already loaded, enable tikz-sensitive line styles unless the user explicitly disables them. If tikz is not already loaded, these functions are disabled unless they are explicitly loaded.

```
\@ifpackageloaded{tikz}
{
  \bool_gset_true:N \g__mcrule_use_tikz_bool
}{}
```

Set up the keys for package options and process them.

```
\keys_define:nn {mcrule-opts}
{
  twocolumn .bool_gset:N = \g__mcrule_twocolumn_bool,
  tikz      .bool_gset:N = \g__mcrule_use_tikz_bool,
  tikz      .default:n   = true,
}
\ProcessKeysOptions{mcrule-opts}
```

4.2 Patching Output Routines

`\__mcrule_column_height:`

Returns the fixed height of the columns. The actual code to calculate the height is set when we set the appropriate mode.

```
\cs_new:Npn \__mcrule_column_height: {}
```

`\__mcrule_column_depth:`

Returns the maximum depth the columns. The actual code to calculate the depth is set when we set the appropriate mode.

```
\cs_new:Npn \__mcrule_column_depth: {}
```

`\__mcrule_column_overflow:`

Returns the amount by which text overflows the bottom of the columns. This situation occurs in multicol when `\maxbalancingoverflow` is greater than 0 (by default it's 12pt) and there would otherwise be a widow at the end of the environment, so we don't check for it in two-column mode.

```
\cs_new:Npn \__mcrule_column_overflow: {0pt}
```

`\__mcrule_patch_mcol_output:N`

Now that we know what mode we're going to run in, we patch the output routine(s) to substitute our custom rule for the vanilla one. Since multicol doesn't fully support twocolumn mode, we patch one or the other, but not both. As of version 1.2, we make `\columnseprulecolor` part of `\mcruledivider` so that we can set the color as part of a style pattern.

```
\cs_new_protected:Npn \__mcrule_patch_mcol_output:N #1
{
  \xpatchcmd{#1} {\columnseprulecolor\vrule\@width\columnseprule}
  {\mcruledivider}
  {\msg_info:nnn {multicolrule} {patch-success} {#1}}
  {\msg_info:nnn {multicolrule} {patch-failure} {#1}}
}
```

`\__mcrule_patch_twocol_output:N`

The same idea as above, only for the vanilla twocolumn mode.

```
\cs_new_protected:Npn \__mcrule_patch_twocol_output:N #1
{
  \xpatchcmd{#1} {\normalcolor\vrule\@width\columnseprule}
  {\mcruledivider}
  {\msg_info:nnn {multicolrule} {patch-success} {#1}}
  {\msg_info:nnn {multicolrule} {patch-failure} {#1}}
}
```

```
\bool_if:NTF \g__mcrule_twocolumn_bool
{
  \@ifpackageloaded{multicol}
  {\msg_warning:nn {multicolrule} {multicol-loaded}}{}
```

Provide the column-color macro from multicol.

```
  \cs_gset:Npn \columnseprulecolor {\normalcolor}
  \cs_gset:Npn \__mcrule_column_height: {\box_ht:N \@outputbox}
  \cs_gset:Npn \__mcrule_column_depth: {\box_dp:N \@outputbox}
  \__mcrule_patch_twocol_output:N \@outputdblcol
```

Now patch the relevant code in `\@outputdblcol`, replacing the hard-coded rule with a macro that we can overwrite.

```
  \__mcrule_patch_twocol_output:N \@outputdblcol
```

bidi has two output routines to patch, and it insists on being loaded after xcolor, tikz, *and* multicol, so it must always be loaded after us. We use `\AfterPackage` from scrlfile to insert the patch if bidi is loaded later on.

```
  \AfterPackage!{bidi}
  {
    \__mcrule_patch_twocol_output:N \RTL@outputdblcol
    \__mcrule_patch_twocol_output:N \LTR@outputdblcol
  }
}
```

Now patch for multicol.

```
{
  \RequirePackage{multicol}
  \__mcrule_patch_mcol_output:N \LR@column@boxes
  \__mcrule_patch_mcol_output:N \RL@column@boxes
```

In LTR mode, we are invoked after a column has been typeset, which destroys their boxes in the process. So the only boxes we can be sure still exist are `\mult@rightbox`, which will be set to column height as all the others, and `\mult@nat@firstbox`, which contains the first column at its natural height.

```
  \cs_gset:Npn \__mcrule_column_height:
  {
    \box_ht:N \mult@rightbox
  }
```

Since the depth can differ from column to column, we use `\dimen\tw@`, which multicol uses to hold the maximum depth of all the columns already typeset, but as it won't have reached the final one yet, we check that too. This could lead to an inconsistent height in the event that there are 3 or more columns and a middle column has a significantly larger depth than either the previous columns or the last column, but for now it does not seem worth accounting for a condition that is likely to be very rare in actual user documents.

```
  \cs_gset:Npn \__mcrule_column_depth:
  {
    \dim_max:nn {\dimen\tw@}{\box_dp:N \mult@rightbox}
  }
```

To avoid widows, multicol allows some overflow, by default up to 12pt, and so it's possible that some text will overflow beyond the fixed bottom of the column. In this case, our rule won't descend far enough. To correct, we measure the column overflow as the difference between the natural height of the first box and the height of the last column, and add that amount if it's greater than 0pt.

```
\cs_gset:Npn \__mcrule_column_overflow:
{
  \dim_max:nn {\box_ht:N \mult@nat@firstbox - \box_ht:N \mult@rightbox}{0pt}
}
```

We need to reissue `\LRmulticolcolumns` to update the actual code in `\mc@align@columns`.

```
    \LRmulticolcolumns
```

The bidi package supplies its own versions of most core multicol functions, including the output boxes. Much of this is unnecessary, as current versions of multicol support printing the columns in right-to-left order, and the effect is to leave the original multicol definitions loaded but unused. As a result, after these changes, the multicol commands `\LRmulticolcolumns` and `\RLmulticolcolumns` have no visible effect. But as bidi also reworks the footnotes extensively, it's easier just to patch the equivalent output routines rather than rewrite it properly.

```
  \AfterPackage!{bidi}
  {
    \cs_gset_eq:NN \LTR@column@boxes \LR@column@boxes
    \cs_gset_eq:NN \RTL@column@boxes \RL@column@boxes
```

While we're at it, we also redefine `\LRmulticolcolumns` and `\RLmulticolcolumns` so they work the way people expect them to.

```
    \cs_gset_eq:NN \LRmulticolcolumns \LTRmulticolcolumns
    \cs_gset_eq:NN \RLmulticolcolumns \RTLmulticolcolumns
  }
}
```

4.3 Creating the Rules

Utility functions for different rule types

`\mcruledivider` This is the function directly called by the patched output routines. Its main function is to call the internal function `\mcrule_divider:`, which contains the actual rule-typesetting instructions, the number of times specified in `\l__mcrule_repeat_int`. multicol puts the rule in a group in order to keep the color contained, which means that any local changes here will be lost at the end of the rule. For this reason, we must set the pattern, if any, here in order to support having different line styles between different columns.

```
\cs_new_protected:Npn \mcruledivider
{
```

If the `pattern-after` counter is set, wait that many iterations of the rule before we apply the patterns.

```
  \int_compare:nNnTF {\g__mcrule_pattern_after_int} > {\c_zero_int}
  {
    \int_gdecr:N \g__mcrule_pattern_after_int
  }
  {
```

Don't change if the pattern is empty or the `pattern-for` counter has expired. The way the logic works here, negative values of `pattern-for` result in an indefinite number of repeats.

```
    \bool_lazy_and:nnT
   {\int_compare_p:nNn {\seq_count:N \l__mcrule_pattern_list_seq} > {\c_zero_int}}
    {! \int_compare_p:nNn {\g__mcrule_pattern_for_int} = {\c_zero_int}}
    {
      \int_gincr:N \g__mcrule_pattern_count_int
      \int_compare:nNnT {\g__mcrule_pattern_count_int} >
```

```
       {\seq_count:N \l__mcrule_pattern_list_seq}
      {
        \int_gset:Nn \g__mcrule_pattern_count_int {\c_one_int}
      }
      \tl_set:Nx \l_tmpa_tl {\seq_item:Nn \l__mcrule_pattern_list_seq
        {\g__mcrule_pattern_count_int} }
      \tl_if_blank:VF \l_tmpa_tl
      {
        \__mcrule_set_pattern:V \l_tmpa_tl
      }
      \int_compare:nNnT {\g__mcrule_pattern_for_int} > {\c_zero_int}
      {
        \int_gdecr:N \g__mcrule_pattern_for_int
      }
    }
  }
```

Now that the pattern has been changed we can set the color.

```
  \columnseprulecolor
```

We only call `\mcrule_divider:` if `\columnseprule` > 0, so that all line styles can be turned off by setting it to 0, just as is the case with the vanilla rules.

```
  \bool_lazy_and:nnT
  {\dim_compare_p:nNn {\columnseprule} > {\c_zero_dim}}
  {\int_compare_p:nNn {\l__mcrule_repeat_int} > {\c_zero_int}}
  {
    \mcrule_divider:
    \prg_replicate:nn {\l__mcrule_repeat_int - \c_one_int}
    {
      \hspace{\l__mcrule_repeat_distance_dim}
      \mcrule_divider:
    }
  }
}
```

(*End definition for* `\mcruledivider`. *This function is documented on page* **??**.)

`\__mcrule_column_total_height:`
`\__mcrule_column_total_depth:`

Get column height and depth with any explicit alterations.

```
\cs_new:Npn \__mcrule_column_total_height:
{
  \dim_eval:n {\__mcrule_column_height: + \__mcrule_column_depth: +
   \__mcrule_extend_column_top: + \__mcrule_column_overflow: + \__mcrule_extend_column_botto
}
\cs_new:Npn \__mcrule_column_total_depth:
{
  \dim_eval:n {\__mcrule_column_depth: + \__mcrule_column_overflow: +
    \__mcrule_extend_column_bottom:}
}
```

`\__mcrule_extend_column_top:`

Currently, the extend amount for the top is just the `\l_@@_extend_top_dim` distance. In the future we may allow more complex criteria, such as by odd or even page, or on a particular page. Although these might theoretically be useful, I'm not going to implement them until someone comes along with a use-case for it.

```
\cs_new:Npn \__mcrule_extend_column_top:
{
  \l__mcrule_extend_top_dim
}
```

`\__mcrule_extend_column_bottom:`

The `extend-fill` option, which is only applicable with multicol, extends the rule from the bottom of the column to the end of the text area, minus whatever reserved space the user specifies. If there's less space available than requested, we give everything we can.

```
\cs_new:Npn \__mcrule_extend_column_bottom:
{
  \bool_lazy_and:nnTF
  {\bool_if_p:n {\l__mcrule_extend_fill_bool}}
  {\bool_not_p:n {\g__mcrule_twocolumn_bool}}
  {
    \dim_compare:nNnTF
   {\@colroom - \__mcrule_column_height: - \__mcrule_extend_reserve:} > {\c_zero_dim}
    {\@colroom - \__mcrule_column_height: - \__mcrule_extend_reserve:}
    {\c_zero_dim}
  }
  {\l__mcrule_extend_bot_dim}
}
```

`\__mcrule_extend_reserve:`

The reserved space is the amount of user-provided space we want, but we also have to account for the space added with `\multicolsep`.

```
\cs_new:Npn \__mcrule_extend_reserve:
{
  \dim_compare:nNnTF {\l__mcrule_extend_reserve_dim} > {\c_zero_dim}
  {\dim_eval:n {\l__mcrule_extend_reserve_dim + \multicolsep}}
  {\c_zero_dim}
}
```

`\mcrule_divider:`

This is the routine that contains the instructions to draw one copy of rule between columns. The default is identical to the original definition used by multicol. It will be reset each time the user calls `\MCSetRule` to specify a new line style.

```
\cs_new:Npn \mcrule_divider: {\vrule\@width\columnseprule}
```

`\__mcrule_pattern:nnn` `\__mcrule_pattern:nnn {⟨pattern⟩} {⟨space above⟩} {⟨space below⟩}`

Typesets a single copy of a pattern, vertically centered, in a vertical box that is the height of the current column. The pattern must be something that can go in a horizontal box. The ⟨*space above*⟩ and ⟨*space below*⟩ arguments must be dimension expressions.

```
\cs_new_nopar:Npn \__mcrule_pattern:nnn #1#2#3
{
  \box_move_down:nn {\__mcrule_column_total_depth:}
  {
    \vbox_to_ht:nn {\__mcrule_column_total_height:}
    {
      \tex_vfill:D
      \tex_kern:D \dim_eval:n {#2} \exp_stop_f:
      \hbox:n{#1}
      \tex_kern:D \dim_eval:n {#3} \exp_stop_f:
      \tex_vfill:D
    }
  }
}
```

`\__mcrule_tile_pattern:nnn` `\__mcrule_tile_pattern:nnn {⟨pattern⟩} {⟨space above⟩} {⟨space below⟩}`

Typesets multiple copies of pattern, tiled so as to occupy a vertical box that is the height of the current column. The pattern must be something that can go in a horizontal box. The ⟨*space above*⟩ and ⟨*space below*⟩ arguments must be dimension expressions.

```
\cs_new_nopar:Npn \__mcrule_tile_pattern:nnn #1#2#3
{
  \box_move_down:nn {\__mcrule_column_total_depth:}
  {
    \vbox_to_ht:nn {\__mcrule_column_total_height:}
    {
      \tex_cleaders:D \vbox:n
      {
        \tex_kern:D \dim_eval:n {#2} \exp_stop_f:
        \hbox:n{#1}
        \tex_kern:D \dim_eval:n {#3} \exp_stop_f:
      }
    \tex_vfill:D
    }
  }
}
```

`\__mcrule_line_pattern:nnnn`

`\__mcrule_line_pattern:nnnn {⟨tikz-name⟩} {⟨height⟩} {⟨space above⟩} {⟨space below⟩}`

This function can draw a line pattern using either a tikz name or directly (as a tiled pattern). The latter case is currently limited to line patterns that can be described in terms of a solid line of length ⟨*height*⟩ separated by spaces above and/or below the line.

```
\cs_new:Npn \__mcrule_line_pattern:nnnn #1#2#3#4
{
  \bool_if:NTF \g__mcrule_use_tikz_bool
  {
    \__mcrule_pattern_line:n {#1}
  }
  {
    \__mcrule_tile_pattern:nnn {\rule{\columnseprule}{#2}}{#3}{#4}
  }
}
```

`\__mcrule_solid_line:`

Unlike the default solid line, which is created with a simple `\vrule`, this version allows us to extend the line beyond the natural space of the column.

```
\cs_new:Npn \__mcrule_solid_line:
{
 \rule[-\__mcrule_column_total_depth:]{\columnseprule}{\__mcrule_column_total_height:}
}
```

4.3.1 Tikz-only Routines

If we're supporting tikz, make sure it's loaded and redefine the relevant functions. We turn off `expl3` syntax to load the package because tikz relies on 2e catcodes, especially for spaces.

```
\bool_if:NTF \g__mcrule_use_tikz_bool
{
  \ExplSyntaxOff
  \RequirePackage{tikz}
  \ExplSyntaxOn
```

`\__mcrule_tikz_picture:n`

`\__mcrule_tikz_picture:n {⟨draw function⟩}`

Set up the `tikzpicture` environment and declare two nodes, named `(TOP)` and `(BOT)`. This way we can pass a `\draw` routine directly, without worrying about the line's coordinates.

```
\cs_set:Npn \__mcrule_tikz_picture:n #1
{
  \begin{tikzpicture}[x=1pt,y=1pt,inner~sep=0pt,outer~sep=0pt,
   baseline={([yshift=\__mcrule_column_total_depth:]current~bounding~box.south)}]
  \node (TOP) at (0,\__mcrule_column_total_height:) {};
  \node (BOT) at (0,0) {};
  #1
    \end{tikzpicture}
}
```

`\__mcrule_pattern_line:n`

`\__mcrule_pattern_line:n {⟨tikz pattern⟩}`

For the tikz versions of the predefined lines, we just draw a line the length of the column box. ⟨*tikz pattern*⟩ should contain the name of a line style that tikz recognizes.

```
\cs_set:Npn \__mcrule_pattern_line:n #1
{
  \begin{tikzpicture}[x=1pt,y=1pt,inner~sep=0pt,outer~sep=0pt,
   baseline={([yshift=\__mcrule_column_total_depth:]current~bounding~box.south)}]
 \draw[line~width=\columnseprule,#1] (0,\__mcrule_column_total_height:) -- (0,0);
  \end{tikzpicture}
}
```

`\__mcrule_circle:`

Draw a hollow circle with a diameter equal to `\columnseprule`. This will be used as a tile pattern.

```
  \cs_set:Npn \__mcrule_circle:
  {
    \begin{tikzpicture}[x=1pt,y=1pt,inner~sep=0pt,outer~sep=0pt]
    \draw (0,0) circle[radius=.5\columnseprule];
    \end{tikzpicture}
  }
```

`\__mcrule_solid_circle:`

Draw a filled circle with a diameter equal to `\columnseprule`. This will be used as a tile pattern.

```
  \cs_set:Npn \__mcrule_solid_circle:
  {
    \begin{tikzpicture}[x=1pt,y=1pt,inner~sep=0pt,outer~sep=0pt]
    \fill (0,0) circle[radius=.5\columnseprule];
    \end{tikzpicture}
  }
}
```

In case tikz functions are not active, we provide stubs that issue error messages.

```
{
  \cs_set:Npn \__mcrule_tikz_picture:n #1
    {\msg_error:nnn {multicolrule} {tikz-required} {#1}}
  \cs_new:Npn \__mcrule_pattern_line:n #1
    {\msg_error:nnn {multicolrule} {tikz-required} {#1}}
  \cs_new:Npn \__mcrule_circle:
    {\msg_error:nnn {multicolrule} {tikz-required} {circles}}
  \cs_new:Npn \__mcrule_solid_circle:
    {\msg_error:nnn {multicolrule} {tikz-required} {solid-circles}}
}
```

4.4 Color

`\__mcrule_set_rule_color:`

Reset color definition in `\columnseprulecolor` by name or by model and color specification.

```
\cs_new_protected:Npn \__mcrule_set_rule_color:
{
  \tl_if_empty:NT \l__mcrule_color_name_tl
  {
    \tl_set:Nn \l__mcrule_color_name_tl {black}
  }
  \tl_if_empty:NTF \l__mcrule_color_model_tl
  {
    \cs_set:Npn \columnseprulecolor {\color{\l__mcrule_color_name_tl}}
  }
  {
    \cs_set:Npn \columnseprulecolor
    {\color[\l__mcrule_color_model_tl]{\l__mcrule_color_name_tl}}
  }
}
```

4.5 Patterns

`\__mcrule_set_pattern_list:n`

Sets a comma-separated list of patterns as a sequence for later use. The global counter that indicates where we are in the list is also reset here, so setting a list of patterns always means that the next rule will use the first pattern in the list.

```
\cs_new_protected:Npn \__mcrule_set_pattern_list:n #1
{
  \seq_set_split:Nnn \l__mcrule_pattern_list_seq {,} {#1}
  \int_gzero:N \g__mcrule_pattern_count_int
  \int_gzero:N \g__mcrule_pattern_after_int
  \int_gset:Nn \g__mcrule_pattern_for_int {-1}
}
```

`\__mcrule_set_pattern:n`

Set the keys an individual pattern. To avoid potential recursion and loops, we filter out the key ‘pattern’ when it appears in a pattern definition.

```
\cs_new_protected:Npn \__mcrule_set_pattern:n #1
{
  \prop_get:NnNTF \g__mcrule_patterns_prop {#1} \l_tmpa_tl
  {
    \keys_set_filter:nnV {mcrule} {patterns} \l_tmpa_tl
  }
  {
    \msg_error:nnn {multicolrule} {pattern-undefined} {#1}
  }
  \tl_set:Nn \l_tmpa_tl {\prop_item:Nn \g__mcrule_patterns_prop {#1}}
}
\cs_generate_variant:Nn \__mcrule_set_pattern:n {V}
```

4.6 Key-Values

Set up all the key definitions. For the line styles, this involves resetting `\mcrule_divider:` to an appropriate value.

```
\keys_define:nn {mcrule}
{
  extend-top                       .dim_set:N  = \l__mcrule_extend_top_dim,
  extend-bot                       .dim_set:N  = \l__mcrule_extend_bot_dim,
  extend-fill                      .bool_set:N = \l__mcrule_extend_fill_bool,
  extend-fill                      .default:n = true,
  extend-reserve                 .dim_set:N  = \l__mcrule_extend_reserve_dim,
  expand                           .code:n = {
    \dim_set:Nn \l__mcrule_extend_bot_dim {#1}
    \dim_set:Nn \l__mcrule_extend_top_dim {#1}
  },
  shift                            .code:n = {
    \dim_set:Nn \l__mcrule_extend_bot_dim {#1}
   \dim_set:Nn \l__mcrule_extend_top_dim {\fp_to_dim:n {-1 * \l__mcrule_extend_bot_dim}}
  },
  line-style                       .choice:,
  line-style / default             .code:n = \cs_set:Npn \mcrule_divider:
    {\vrule\@width\columnseprule},
  line-style / solid               .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_solid_line:},
  line-style / dots                .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {.}{1pt}{1pt}},
  line-style / dense-dots          .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {.}{1pt}{0pt}},
  line-style / loose-dots          .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {.}{2pt}{2pt}},
  line-style / circles             .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {\__mcrule_circle:}{1pt}{1pt}},
  line-style / dense-circles       .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {\__mcrule_circle:}{1pt}{0pt}},
  line-style / loose-circles       .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {\__mcrule_circle:}{2pt}{2pt}},
  line-style / solid-circles       .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {\__mcrule_solid_circle:}{1pt}{1pt}},
  line-style / dense-solid-circles .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {\__mcrule_solid_circle:}{1pt}{0pt}},
  line-style / loose-solid-circles .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_tile_pattern:nnn {\__mcrule_solid_circle:}{2pt}{2pt}},
  line-style / dotted              .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_line_pattern:nnnn {dotted}{\columnseprule}{1pt}{1pt}},
  line-style / densely-dotted      .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_line_pattern:nnnn {densely~dotted}{\columnseprule}{1pt}{0pt}},
  line-style / loosely-dotted      .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_line_pattern:nnnn {loosely~dotted}{\columnseprule}{2pt}{2pt}},
  line-style / dashed              .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_line_pattern:nnnn {dashed}{3pt}{1.5pt}{1.5pt}},
  line-style / densely-dashed      .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_line_pattern:nnnn {densely~dashed}{3pt}{1pt}{1pt}},
  line-style / loosely-dashed      .code:n = \cs_set:Npn \mcrule_divider:
    {\__mcrule_line_pattern:nnnn {loosely~dashed}{3pt}{3pt}{3pt}},
line-style / dash-dot            .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_pattern_line:n{dash~dot}},
line-style / densely-dash-dot    .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_pattern_line:n{densely~dash~dot}},
line-style / loosely-dash-dot    .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_pattern_line:n{loosely~dash~dot}},
line-style / dash-dot-dot         .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_pattern_line:n{dash~dot~dot}},
line-style / densely-dash-dot-dot .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_pattern_line:n{densely~dash~dot~dot}},
line-style / loosely-dash-dot-dot .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_pattern_line:n{loosely~dash~dot~dot}},
color                             .code:n = {
  \tl_set:Nn \l__mcrule_color_name_tl {#1}
  \__mcrule_set_rule_color:
},
color-model                       .code:n = {
  \tl_set:Nn \l__mcrule_color_model_tl {#1}
  \__mcrule_set_rule_color:
},
custom-line         .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_tikz_picture:n {#1}},
custom-pattern      .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_pattern:nnn #1},
custom-tile         .code:n = \cs_set:Npn \mcrule_divider:
  {\__mcrule_tile_pattern:nnn #1},
width               .choice:,
width / ultra-thin  .code:n = \dim_set:Nn \columnseprule {0.1pt},
width / very-thin   .code:n = \dim_set:Nn \columnseprule {0.2pt},
width / thin        .code:n = \dim_set:Nn \columnseprule {0.4pt},
width / semithick   .code:n = \dim_set:Nn \columnseprule {0.6pt},
width / thick       .code:n = \dim_set:Nn \columnseprule {0.8pt},
width / very-thick  .code:n = \dim_set:Nn \columnseprule {1.2pt},
width / ultra-thick .code:n = \dim_set:Nn \columnseprule {1.6pt},
width / unknown     .code:n = \dim_set:Nn \columnseprule {#1},
repeat              .int_set:N   = \l__mcrule_repeat_int,
repeat-distance     .dim_set:N   = \l__mcrule_repeat_distance_dim,
single              .meta:n      = {
  repeat = 1,
  repeat-distance = #1
},
single              .default:n   = \columnseprule,
double              .meta:n      = {
  repeat = 2,
  repeat-distance = #1
},
double              .default:n   = \columnseprule,
triple              .meta:n      = {
  repeat = 3,
  repeat-distance = #1
},
triple              .default:n   = \columnseprule,
patterns            .code:n      = \__mcrule_set_pattern_list:n {#1},
patterns            .groups:n    = {patterns},
  pattern-after       .int_gset:N  = \g__mcrule_pattern_after_int,
  pattern-for         .int_gset:N  = \g__mcrule_pattern_for_int,
}
```

4.7 User Interface

\SetMCRule Set all keys for multicolrule

`\SetMCRule {⟨key-value list⟩}`

All we do here is pass the argument to expl3's key-setting routine.

```
\NewDocumentCommand{\SetMCRule}{m}
{
  \keys_set:nn {mcrule} {#1}
}
```

(*End definition for* `\SetMCRule`. *This function is documented on page* ??.)

\DeclareMCRulePattern Declare a new style pattern.

`\DeclareMCRule {⟨name⟩} {⟨key-value list⟩}`

If a pattern of that name exists, it will be overwritten silently.

```
\NewDocumentCommand{\DeclareMCRulePattern}{m m}
{
  \prop_gput:Nnn \g__mcrule_patterns_prop {#1} {#2}
}
```

(*End definition for* `\DeclareMCRulePattern`. *This function is documented on page* ??.)

Change History

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.