The colortbl package*

David Carlisle†

2024/07/06

Abstract

This package implements a flexible mechanism for giving colored 'panels' behind specified columns in a table. This package requires the array and color packages.

1 Introduction

This package is for coloring tables (i.e., giving colored panels behind column entries). In that it has many similarities with Timothy Van Zandt's colortab package. The internal implementation is quite different though, also colortab works with the table constructs of other formats besides LaTeX. This package requires LaTeX (and its color and array packages).

First, a standard tabular, for comparison.

```
\begin{tabular}{|l|c|}
 one&two\\
 three&four
 \end{tabular}
```

one	two
three	four

2 The `\columncolor` command

The examples below demonstrate various possibilities of the `\columncolor` command introduced by this package. The vertical rules specified by `|` are kept in all the examples, to make the column positioning clearer, although possibly you would not want colored panels *and* vertical rules in practice.

The package supplies a `\columncolor` command, that should (only) be used in the argument of a `>` column specifier, to add a colored panel behind the specified column. It can be used in the main 'preamble' argument of array or tabular, and also in `\multicolumn` specifiers.

The basic format is:

`\columncolor[`⟨*color model*⟩`]{`⟨*color*⟩`} [`⟨*left overhang*⟩`][`⟨*right overhang*⟩`]`

The first argument (or first two if the optional argument is used) are standard color package arguments, as used by `\color`.

The last two arguments control how far the panel overlaps past the widest entry in the column. If the *right overhang* argument is omitted then it defaults to

*This file has version number v1.0i, last revised 2024/07/06.

†Report issues to `https://github.com/davidcarlisle/dpctex/issues`

left overhang. If they are both omitted they default to `\tabcolsep` (in tabular) or `\arraycolsep` (in array).

If the overhangs are both set to `0pt` then the effect is:

```
|>{\columncolor[gray]{.8}[0pt]}l|
>{\color{white}%
  \columncolor[gray]{.2}[0pt]}l|
```

one	two
three	four

The default overhang of `\tabcolsep` produces:

```
|>{\columncolor[gray]{.8}}l|
>{\color{white}%
  \columncolor[gray]{.2}}l|
```

one	two
three	four

You might want something between these two extremes. A value of `.5\tabcolsep` produces the following effect

```
|>{\columncolor[gray]{.8}[.5\tabcolsep]}l|
 >{\color{white}%
   \columncolor[gray]{.2}[.5\tabcolsep]}l|
```

one	two
three	four

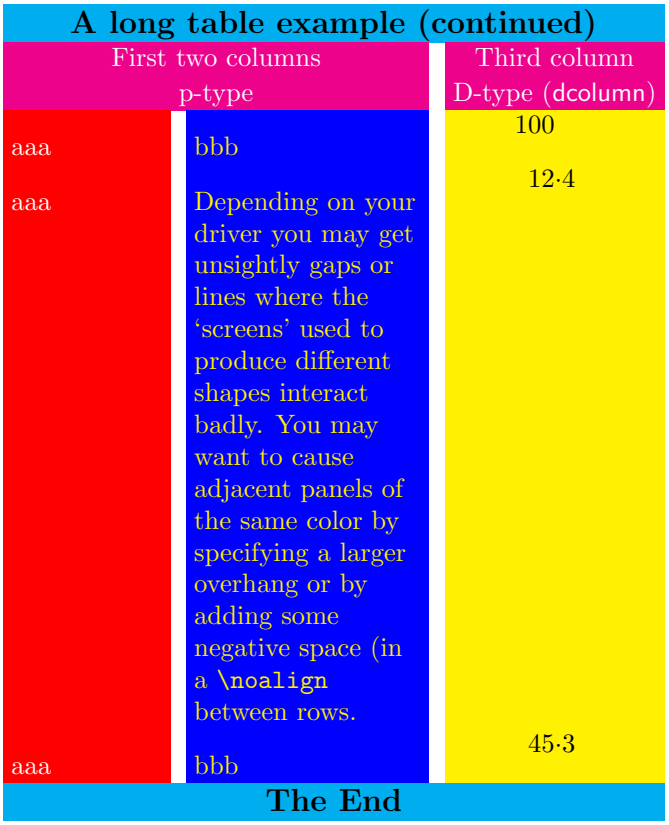
This package should work with most other packages that are compatible with the array package syntax. In particular it works with longtable and dcolumn as the following example shows.

Before starting give a little space: `\setlength\minrowclearance{2pt}`

A long table example

First two columns p-type		Third column D-type (dcolumn)
P-column	and another one	12·34
Total	(wrong)	100·6
Some long text in the first column	bbb	1·2
aaa	and some long text in the second column	1·345
Total	(wrong)	100·6
aaa Note that the colored rules in all columns stretch to accomodate large entries in one column.	bbb bbb	1·345 1·345

Continued. . .



This example shows rather poor taste but is quite colorful! Inspect the source file, `colortbl.dtx`, to see the full code for the example, but it uses the following column types.

```
\newcolumntype{A}{%
   >{\color{white}\columncolor{red}[.5\tabcolsep]%
      \raggedright}%
   p{2cm}}
\newcolumntype{B}{%
   >{\columncolor{blue}[.5\tabcolsep]%
     \color{yellow}\raggedright}
   p{3cm}}
\newcolumntype{C}{%
    >{\columncolor{yellow}[.5\tabcolsep]}%
      D{.}{\cdot}{3.3}}
\newcolumntype{E}{%
   >{\large\bfseries
     \columncolor{cyan}[.5\tabcolsep]}c}
\newcolumntype{F}{%
    >{\color{white}
      \columncolor{magenta}[.5\tabcolsep]}c}
\newcolumntype{G}{%
    >{\columncolor[gray]{0.8}[.5\tabcolsep][\tabcolsep]}l}
\newcolumntype{H}{>{\columncolor[gray]{0.8}}l}
\newcolumntype{I}{%
    >{\columncolor[gray]{0.8}[\tabcolsep][.5\tabcolsep]}%
```

```
D{.}{\cdot}{3.3}}
```

3 Using the 'overhang' arguments for tabular*

The above is all very well for tabular, but what about tabular*?

Here the problem is rather harder. Although TEX's `\leader` mechanism which is used by this package to insert the 'stretchy' colored panels is rather like *glue*, the `\tabskip` glue that is inserted between columns of tabular* (and longtable for that matter) has to be 'real glue' and not 'leaders'.

Within limits the overhang options may be used here. Consider the first table example above. If we use tabular* set to 3 cm with a preamble setting of

```
\begin{tabular*}{3cm}{%
@{\extracolsep{\fill}}
>{\columncolor[gray]{.8}[0pt][20mm]}l
>{\columncolor[gray]{.8}[5mm][0pt]}l
@{}}
```

one	**two**
three	**four**

Changing the specified width to 4 cm works, but don't push your luck to 5 cm...

one	**two**
three	**four**

one	**two**
three	**four**

4 The `\rowcolor` command

As demonstrated above, one may change the color of specified rows of a table by the use of `\multicolumn` commands in each entry of the row. However if your table is to be marked principally by *rows*, you may find this rather inconvenient. For this reason a new mechanism, `\rowcolor`, has been introduced[1].

`\rowcolor` takes the same argument forms as `\columncolor`. It must be used at the *start* of a row. If the optional overhang arguments are not used the overhangs will default to the overhangs specified in any `\columncolor` comands for that column, or `\tabcolsep` (`\arraycolsep` in array).

If a table entry is in the scope of a `\columncolor` specified in the table preamble, and also a `\rowcolor` at the start of the current row, the color specified by `\rowcolor` will take effect. A `\multicolumn` command may contain `>{\rowcolor...` which will override the default colors for both the current row and column.

```
\begin{tabular}{|l|c|}
 \rowcolor[gray]{.9}
 one&two\\
 \rowcolor[gray]{.5}
 three&four
 \end{tabular}
```

one	**two**
three	**four**

5 The `\rowcolors` command

The `\rowcolors` command and its documentation originate in the xcolor package by Dr. Uwe Kern.

`\rowcolors` `[`⟨*commands*⟩`]{`⟨*row*⟩`}{`⟨*odd-row color*⟩`}{`⟨*even-row color*⟩`}`
`\rowcolors*` `[`⟨*commands*⟩`]{`⟨*row*⟩`}{`⟨*odd-row color*⟩`}{`⟨*even-row color*⟩`}`
One of these commands has to be executed *before* a table starts. ⟨*row*⟩ tells the number of the first row which should be colored according to the ⟨*odd-row color*⟩ and ⟨*even-row color*⟩ scheme. Each of the color arguments may also be left empty (= no color). In the starred version, ⟨*commands*⟩ are ignored in rows with inactive *rowcolors status* (see below), whereas in the non-starred version, ⟨*commands*⟩ are applied to every row of the table. Such optional commands may be `\hline` or `\noalign{`⟨*stuff*⟩`}`.

`\showrowcolors` `\hiderowcolors` `\rownum`
The *rowcolors status* is activated (i.e., use coloring scheme) by default and/or `\showrowcolors`, it is inactivated (i.e., ignore coloring scheme) by the command `\hiderowcolors`. The counter `\rownum` (or LaTeX counter `rownum`) may be used within such a table to access the current row number.

At the present time, the `rownum` counter is only incremented in tables using `\rowcolors`.

```
\rowcolors[\hline]{3}{green}{yellow} \arrayrulecolor{red}
\begin{tabular}{ll}
test & row \therownum\\
test & row \therownum\\
test & row \therownum\\
test & row \therownum\\
\arrayrulecolor{black}
test & row \therownum\\
test & row \therownum\\
\rowcolor{blue}
test & row \therownum\\
test & row \therownum\\
\hiderowcolors
test & row \therownum\\
test & row \therownum\\
\showrowcolors
test & row \therownum\\
test & row \therownum\\
\multicolumn{1}%
 {>{\columncolor{red}}l}{test} & row \therownum\\
\end{tabular}
```

test	row 1
test	row 2
test	row 3
test	row 4
test	row 5
test	row 6
test	row 7
test	row 8
test	row 9
test	row 10
test	row 11
test	row 12
test	row 13

test	row 1
test	row 2
test	row 3
test	row 4
test	row 5
test	row 6
test	row 7
test	row 8
test	row 9
test	row 10
test	row 11
test	row 12
test	row 13

6 The `\cellcolor` command

A background color can be applied to a single cell of a table by beginning it with `\multicolumn{1}{>{\rowcolor`..., (or `\columncolor` if no row-color is in effect) but this has some deficiencies: 1) It prevents data within the cell from triggering the coloration; 2) The alignment specification must be copied from the top of the tabular, which is prone to errors, especially for `p{}` columns; 3) `\multicolumn{1}` is just silly. Therefore, there is the `\cellcolor` command, which works like `\columncolor` and `\rowcolor`, but over-rides both of them; `\cellcolor` can be placed anywhere in the tabular cell to which it applies.

[1] At some cost to the internal complexity of this package

7 Coloring rules.

So you want colored rules as well?

One could do vertical rules without any special commands, just use something like `!{\color{green}\vline}` where you'd normally use `|`. The space between `||` will normally be left white. If you want to color that as well, either increase the overhang of the previous column (to `\tabcolsep` + `\arrayrulewidth` + `\doublerulesep`) Or remove the inter rule glue, and replace by a colored rule of the required thickness. So

```
!{\color{green}\vline}
@{\color{yellow}\vrule width \doublerulesep}
!{\color{green}\vline}
```

Should give the same spacing as `||` but more color.

However coloring `\hline` and `\cline` is a bit more tricky, so extra commands are provided (which then apply to vertical rules as well).

8 \arrayrulecolor

`\arrayrulecolor` takes the same arguments as `\color`, and is a global declaration which affects all following horizontal and vertical rules in tables. It may be given outside any table, or at the start of a row, or in a `>` specification in a table preamble. You should note however that if given mid-table it only affects rules that are specified after this point, any vertical rules specified in the preamble will keep their original colors.

9 \doublerulesepcolor

Having colored your rules, you'll probably want something other than white to go in the gaps made by `||` or `\hline\hline`. `\doublerulesepcolor` works just the same way as `\arrayrulecolor`. The main thing to note that if this command is used, then longtable will not 'discard' the space between `\hline\hline` at a page break. (TEX has a built-in ability to discard space, but the colored 'space' which is used once `\doublerulesep` is in effect is really a third rule of a different color to the two outer rules, and rules are rather harder to discard.)

```
\setlength\arrayrulewidth{2pt}\arrayrulecolor{blue}
\setlength\doublerulesep{2pt}\doublerulesepcolor{yellow}
 \begin{tabular}{||l||c||}
  \hline\hline
  one&two\\
  three&four\\
  \hline\hline
 \end{tabular}
```

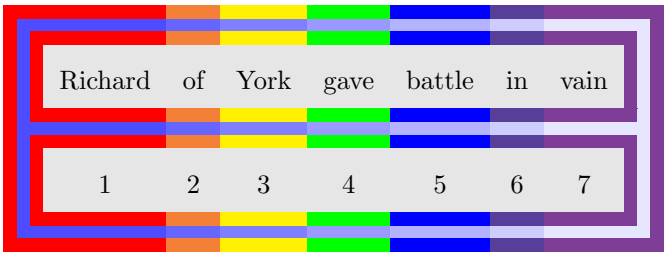
one	two
three	four

10 More fun with \hhline

The above commands work with `\hhline` from the hhline package, however if hhline is loaded in addition to this package, a new possibility is added. You may use `>{...}` to add declarations that apply to the following `-` or `=` column

rule. In particular you may give `\arrayrulecolor` and `\doublerulesepcolor` declarations in this argument.

Most manuals of style warn against over use of rules in tables. I hate to think what they would make of the following rainbow example:




```
\newcommand\rainbowline[1]{%
\hhline{%
  >{\arrayrulecolor   {red}\doublerulesepcolor[rgb]{.3,.3,1}}%
  |#1:=%
  >{\arrayrulecolor{orange}\doublerulesepcolor[rgb]{.4,.4,1}}%
  =%
  >{\arrayrulecolor{yellow}\doublerulesepcolor[rgb]{.5,.5,1}}%
  =%
  >{\arrayrulecolor {green}\doublerulesepcolor[rgb]{.6,.6,1}}%
  =%
  >{\arrayrulecolor  {blue}\doublerulesepcolor[rgb]{.7,.7,1}}%
  =%
  >{\arrayrulecolor{indigo}\doublerulesepcolor[rgb]{.8,.8,1}}%
  =%
  >{\arrayrulecolor{violet}\doublerulesepcolor[rgb]{.9,.9,1}}%
  =:#1|%
  }}
\arrayrulecolor{red}
\doublerulesepcolor[rgb]{.3,.3,1}%
\begin{tabular}{||*7{>{\columncolor[gray]{.9}}c}||}
\rainbowline{t}%
\arrayrulecolor{violet}\doublerulesepcolor[rgb]{.9,.9,1}
Richard&of&York&gave&battle&in&
\multicolumn{1}{>{\columncolor[gray]{.9}}c||}{vain}\\
\rainbowline{}%
1&2&3&4&5&6&
\multicolumn{1}{>{\columncolor[gray]{.9}}c||}{7}\\
\rainbowline{b}%
\end{tabular}
```

11 Less fun with `\cline`

Lines produced by `\cline` are colored if you use `\arrayrulecolor` but you may not notice as they are covered up by any color pannels in the following row. This is a 'feature' of `\cline`. If using this package you would probably better using the `-` rule type in a `\hhline` argument, rather than `\cline`.

12 The \minrowclearance command

As this package has to box and measure every entry to figure out how wide to make the rules, I thought I may as well add the following feature. 'Large' entries in tables may touch a preceding `\hline` or the top of a color panel defined by this style. It is best to increase `\extrarowsep` or `\arraystretch` sufficiently to ensure this doesn't happen, as that will keep the line spacing in the table regular. Sometimes however, you just want to LaTeX to insert a bit of extra space above a large entry. You can set the length `\minrowclearance` to a small value. (The height of a capital letter plus this value should not be greater than the normal height of table rows, else a very uneven table spacing will result.)

Donald Arseneau's tabls packages provides a similar `\tablinesep`. I was going to give this the same name for compatibility with tabls, but that is implemented quite differently and probably has different behaviour. So I'll keep a new name for now.

13 The Code

```
⟨*package⟩
```

Nasty hacky way used by all the graphics packages to include debugging code.

```
\edef\@tempa{%
  \noexpand\AtEndOfPackage{%
    \catcode`\noexpand\^^A\the\catcode`\^^A\relax}}
\@tempa
\catcode`\^^A=\catcode`\%
\DeclareOption{debugshow}{\catcode`\^^A=9 }
```

All the other options are handled by the color package.

```
\DeclareOption*{\PassOptionsToPackage\CurrentOption{color}}
\ProcessOptions
```

I need these so load them now. Actually Mark Wooding's mdwtab package could probably work instead of array, but currently I assume array package internals so...

```
\RequirePackage{array,color}
```

`\@classz` First define stub for new array package code.

```
\ifx\do@row@strut\@undefined\let\do@row@strut\relax\fi
```

`\@classz` is the main function in the array package handling of primitive column types: It inserts the code for each of the column specifiers, '`clrpmb`'. The other classes deal with the other preamble tokens such as '`@` 'or '`>`'.

```
\def\@classz{\@classx
   \@tempcnta \count@
   \prepnext@tok
```

At this point the color specification for the background panel will be in the code for the '`>`' specification of this column. This is saved in `\toks\@temptokena` but array will insert it too late (well it would work for `c`, but not for `p`) so fish the color stuff out of that token register by hand, and then insert it around the entry.

Of course this is a terrible hack. What is really needed is a new column type that inserts stuff in the right place (rather like `!` but without the spacing that that does). The `\newcolumntype` command of array only adds 'second class'

column types. The re-implementations of \newcolumntype in my blkarray or Mark Wooding's mdwtab allow new 'first class' column types to be declared, but stick with array for now. This means we have to lift the stuff out of the register before the register gets emptied in the wrong place.

```
\expandafter\CT@extract\the\toks\@tempcnta\columncolor!\@nil
```

Save the entry into a box (using a double group for color safety as usual).

```
  \@addtopreamble{%
   \setbox\z@\hbox\bgroup\bgroup
     \CT@everycr{}%
     \ifcase \@chnum
```

`c` code: This used to use twice as much glue as `l` and `r` (1fil on each side). Now modify it to use 1fill total. Also increase the order from 1fil to 1fill to dissuade people from putting stretch glue in table entries.

```
     \hskip\stretch{.5}\kern\z@
     \d@llarbegin
     \insert@column
     \d@llarend\do@row@strut\hskip\stretch{.5}\or
```

`l` and `r` as before, but using fill glue.

```
     \d@llarbegin \insert@column \d@llarend\do@row@strut \hfill
  \or
     \hfill\kern\z@ \d@llarbegin \insert@column \d@llarend\do@row@strut
  \or
```

`m`, `p` and `b` as before, but need to take account of array package update.

```
     \ifx\ar@align@mcell\@undefined
       $\vcenter
        \@startpbox{\@nextchar}\insert@column \@endpbox $
     \else
       \setbox\ar@mcellbox\vbox
         \@startpbox{\@nextchar}\insert@column \@endpbox
       \ar@align@mcell
       \do@row@strut
     \fi
  \or
    \vtop \@startpbox{\@nextchar}\insert@column \@endpbox\do@row@strut
  \or
    \vbox \@startpbox{\@nextchar}\insert@column \@endpbox\do@row@strut
  \fi
```

Close the box register assignment.

```
\egroup\egroup
```

The main new stuff.

```
\begingroup
```

Initalise color command and overhands.

```
  \CT@setup
```

Run any code resulting from `\columncolor` commands.

```
  \CT@column@color
```

Run code from `\rowcolor` (so this takes precedence over `\columncolor`).

```
  \CT@row@color
```

Run code from `\cellcolor` (so this takes precedence over both `\columncolor` and `\rowcolor`).

```
\CT@cell@color
```

This is `\relax` unless one of the three previous commands has requested a color, in which case it will be `\CT@@do@color` which will insert `\leaders` of appropriate color.

```
  \CT@do@color
\endgroup
```

Nothing to do with color this bit, since we are boxing and measuring the entry anyway may as well check the height, so that large entries don't bump into horizontal rules (or the top of the color panels).

```
        \@tempdima\ht\z@
        \advance\@tempdima\minrowclearance
        \vrule\@height\@tempdima\@width\z@
```

It would be safer to leave this boxed, but unboxing allows some flexibilty. However the total glue stretch should either be finite or fil (which will be ignored). There may be fill glue (which will not be ignored) but it should *total 0fill*. If this box contributes fill glue, then the leaders will not reach the full width of the entry. In the case of `\multicolumn` entries it is actually possible for this box to contribute *shrink* glue, in which case the colored panel for that entry will be too wide. Tough luck.

```
        \unhbox\z@}%
  \prepnext@tok}
```

`\CT@setup` Initialise the overhang lengths and the color command.

```
\def\CT@setup{%
  \@tempdimb\col@sep
  \@tempdimc\col@sep
  \def\CT@color{%
    \global\let\CT@do@color\CT@@do@color
    \color}}
```

`\CT@@do@color` The main point of the package: Add the color panels.

Add a leader of the specified color, with natural width the width of the entry plus the specified overhangs and 1fill stretch. Surround by negative kerns so total natural width is not affected by overhang.

```
\def\CT@@do@color{%
  \global\let\CT@do@color\relax
        \@tempdima\wd\z@
        \advance\@tempdima\@tempdimb
        \advance\@tempdima\@tempdimc
        \kern-\@tempdimb
        \leaders\vrule
```

For quick debugging with xdvi (which can't do colors). Limit the size of the rule, so I can see the text as well.

```
^^A                     \@height\p@\@depth\p@
                \hskip\@tempdima\@plus  1fill
        \kern-\@tempdimc
```

Now glue to exactly compensate for the leaders.

```
        \hskip-\wd\z@ \@plus -1fill }
```

`\CT@extract` Now the code to extract the `\columncolor` commands.

```
\def\CT@extract#1\columncolor#2#3\@nil{%
  \if!\noexpand#2%
```

`!` is a fake token inserted at the end.

```
    \let\CT@column@color\@empty
  \else
```

If there was an optional argument

```
    \if[\noexpand#2%
      \CT@extractb{#1}#3\@nil
    \else
```

No optional argument

```
      \def\CT@column@color{%
        \CT@color{#2}}%
      \CT@extractd{#1}#3\@nil
    \fi
  \fi}
```

`\CT@extractb` Define `\CT@column@color` to add the right color, and save the overhang lengths. Finally reconstitute the saved '>' tokens, without the color specification. First grab the color spec, with optional arg.

```
\def\CT@extractb#1#2]#3{%
  \def\CT@column@color{%
    \CT@color[#2]{#3}}%
  \CT@extractd{#1}}%
```

`\CT@extractd` Now look for left-overhang (default to `\col@sep`).

```
\def\CT@extractd#1{\@testopt{\CT@extracte{#1}}\col@sep}
```

`\CT@extracte` Same for right-overhang (default to left-overhang).

```
\def\CT@extracte#1[#2]{\@testopt{\CT@extractf{#1}[#2]}{#2}}
```

`\CT@extractf` Add the overhang info to `\CT@do@color`, for excuting later.

```
{\catcode`\!\active
\gdef\CT@extractf#1[#2][#3]#4\columncolor#5\@nil{%
  \@tempdimb#2\relax
  \@tempdimc#3\relax
  \edef!{\string!}%
  \edef\CT@column@color{%
    \CT@column@color
    \@tempdimb\the\@tempdimb\@tempdimc\the\@tempdimc\relax}%
  \toks\@tempcnta{#1#4}}}%
```

`\CT@everycr` Steal `\everypar` to initialise row colors

```
\let\CT@everycr\everycr
\newtoks\everycr
\CT@everycr{\noalign{\global\let\CT@row@color\relax}\the\everycr}
```

\CT@start

```
\def\CT@start{%
  \let\CT@arc@save\CT@arc@
  \let\CT@drsc@save\CT@drsc@
  \let\CT@row@color@save\CT@row@color
  \let\CT@cell@color@save\CT@cell@color
  \global\let\CT@cell@color\relax}
```

\CT@end

```
\def\CT@end{%
  \global\let\CT@arc@\CT@arc@save
  \global\let\CT@drsc@\CT@drsc@save
  \global\let\CT@row@color\CT@row@color@save
  \global\let\CT@cell@color\CT@cell@color@save}
```

\shortstack \shortstack

```
\gdef\@ishortstack#1{%
  \CT@start\ialign{\mb@l {##}\unskip\mb@r\cr #1\crcr}\CT@end\egroup}
```

\@tabarray array and tabular (delayed for delarray)

```
\AtBeginDocument{%
  \expandafter\def\expandafter\@tabarray\expandafter{%
    \expandafter\CT@start\@tabarray}}
```

\endarray

```
\expandafter\def\expandafter\endarray\expandafter{\endarray\CT@end}
```

\multicolumn \multicolumn Patch \multicolumn to restore color settings. Done this way to work wth different versions depending on the age of the array package.

```
\def\@tempa#1\@arstrut#2\relax{
  \long\def\multicolumn##1##2##3{%
    #1%
```

row@color

```
   \let\CT@cell@color\relax
   \let\CT@column@color\relax
   \let\CT@do@color\relax
   \@arstrut
   #2}}
\expandafter\@tempa\multicolumn{#1}{#2}{#3}\relax
\let\@temp\relax
```

\@classvi Colored rules and rule separations.

```
\def\@classvi{\ifcase \@lastchclass
      \@acol \or
      \ifx\CT@drsc@\relax
        \@addtopreamble{\hskip\doublerulesep}%
      \else
        \@addtopreamble{{\CT@drsc@\vrule\@width\doublerulesep}}%
      \fi\or
      \@acol \or
      \@classvii
      \fi}
```

\doublerulesepcolor

```
\def\doublerulesepcolor#1#{\CT@drs{#1}}
```

\CT@drs

```
\def\CT@drs#1#2{%
 \ifdim\baselineskip=\z@\noalign\fi
  {\gdef\CT@drsc@{\color#1{#2}}}}
```

\CT@drsc@

```
\let\CT@drsc@\relax
```

\arrayrulecolor

```
\def\arrayrulecolor#1#{\CT@arc{#1}}
```

\CT@arc

```
\def\CT@arc#1#2{%
  \ifdim\baselineskip=\z@\noalign\fi
  {\gdef\CT@arc@{\color#1{#2}}}}
```

\CT@arc@

```
\let\CT@arc@\relax
```

hline

\@arrayrule

```
\def\@arrayrule{\@addtopreamble {{\CT@arc@\vline}}}
```

\hline

```
\def\hline{%
  \noalign{\ifnum0=`}\fi
              \let\hskip\vskip
               \let\vrule\hrule
               \let\@width\@height
  {\CT@arc@\vline}%
  \futurelet
   \reserved@a\@xhline}
```

\@xhline

```
\def\@xhline{\ifx\reserved@a\hline
               {\ifx\CT@drsc@\relax
                  \vskip
               \else
                  \CT@drsc@\hrule\@height
               \fi
               \doublerulesep}%
             \fi
      \ifnum0=`{\fi}}
```

\cline `\cline` doesn't really work, as it comes behind the colored panels, but at least make it the right color (the bits you can see, anyway).

```
\def\@cline#1-#2\@nil{%
  \omit
  \@multicnt#1%
```

```
  \advance\@multispan\m@ne
  \ifnum\@multicnt=\@ne\@firstofone{&\omit}\fi
  \@multicnt#2%
  \advance\@multicnt-#1%
  \advance\@multispan\@ne
  {\CT@arc@\leaders\hrule\@height\arrayrulewidth\hfill}%
  \cr
  \noalign{\vskip-\arrayrulewidth}}
```

`\minrowclearance` The row height fudge length.

```
\newlength\minrowclearance
\minrowclearance=0pt
```

`\@mkpream` `\@mkpreamarray` While expanding the preamble array passes tokens through an `\edef`. It doesn't use `\protection` as it thinks it has full control at that point. As the redefinition above adds `\color`, I need to add that to the list of commands made safe.

```
\let\@mkpreamarray\@mkpream
\def\@mkpream{%
    \let\CT@setup\relax
    \let\CT@color\relax
    \let\CT@do@color\relax
    \let\color\relax
    \let\CT@column@color\relax
    \let\CT@row@color\relax
    \let\CT@cell@color\relax
    \@mkpreamarray}
```

`\CT@do@color` For similar reasons, need to make this non-expandable

```
\let\CT@do@color\relax
```

`\rowcolor`

```
\def\rowcolor{%
  \noalign{\ifnum0=`}\fi
  \global\let\CT@do@color\CT@@do@color
  \@ifnextchar[\CT@rowa\CT@rowb}
```

`\CT@rowa`

```
\def\CT@rowa[#1]#2{%
  \gdef\CT@row@color{\CT@color[#1]{#2}}%
  \CT@rowc}
```

`\CT@rowb`

```
\def\CT@rowb#1{%
  \gdef\CT@row@color{\CT@color{#1}}%
  \CT@rowc}
```

`\CT@rowc`

```
\def\CT@rowc{%
  \@ifnextchar[\CT@rowd{\ifnum`{=0\fi}}}
```

`\CT@rowd`

```
\def\CT@rowd[#1]{\@testopt{\CT@rowe[#1]}{#1}}
```

\CT@rowe

```
\def\CT@rowe[#1][#2]{%
  \@tempdimb#1%
  \@tempdimc#2%
  \xdef\CT@row@color{%
    \expandafter\noexpand\CT@row@color
    \@tempdimb\the\@tempdimb
    \@tempdimc\the\@tempdimc
    \relax}%
  \ifnum0=`{\fi}}
```

\@ifxempty {⟨*arg*⟩}{⟨*empty*⟩}{⟨*non-empty*⟩}

Tests without expanding, whether the argument {⟨*arg*⟩} is empty and executes the following code accordingly; {⟨*arg*⟩} must not start with the token \XC@@. Can also be used within \edef.

```
\def\@ifxempty#1{\@@ifxempty#1\@@ifxempty\XC@@}
\def\@@ifxempty#1#2\XC@@
 {\ifx#1\@@ifxempty
  \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}
```

\rowcolors [⟨*commands*⟩]{⟨*row*⟩}{⟨*odd-row color*⟩}{⟨*even-row color*⟩}
\rowcolors* Defines alternating colors for the next tabular environment. Starting with row ⟨*row*⟩, odd and even rows get their respective colors. The color arguments may also be left empty (= no color). Optional commands may be hline or noalign{⟨*stuff*⟩}.

In the starred version, ⟨*commands*⟩ are ignored in rows with inactive rowcolors status (see below), whereas in the non-starred version, ⟨*commands*⟩ are applied to every row of the table.

```
\def\rowcolors
 {\@ifstar{\@rowcmdfalse\rowc@lors}{\@rowcmdtrue\rowc@lors}}

\def\rowc@lors{\@testopt{\rowc@l@rs}{}}

\def\rowc@l@rs[#1]#2#3#4%
 {\global\rownum=\z@
  \global\@rowcolorstrue
  \@ifxempty{#3}%
    {\def\@oddrowcolor{\@norowcolor}}%
    {\def\@oddrowcolor{\gdef\CT@row@color{\CT@color{#3}}}}%
  \@ifxempty{#4}%
    {\def\@evenrowcolor{\@norowcolor}}%
    {\def\@evenrowcolor{\gdef\CT@row@color{\CT@color{#4}}}}%
  \if@rowcmd
    \def\@rowcolors
     {#1\if@rowcolors
        \noalign{\relax\ifnum\rownum<#2\@norowcolor\else
                 \ifodd\rownum\@oddrowcolor\else\@evenrowcolor\fi\fi}%
      \fi}%
  \else
    \def\@rowcolors
     {\if@rowcolors
        \ifnum\rownum<#2\noalign{\@norowcolor}\else
        #1\noalign{\ifodd\rownum\@oddrowcolor\else\@evenrowcolor\fi}\fi
      \fi}%
```

```
    \fi
    \CT@everycr{\@rowc@lors\the\everycr}%
    \ignorespaces}
\def\@rowc@lors{\noalign{\global\advance\rownum\@ne}\@rowcolors}
\let\@rowcolors\@empty
```

`\showrowcolors` `\hiderowcolors` Switch coloring mode on/off.

```
\def\showrowcolors{\noalign{\global\@rowcolorstrue}\@rowcolors}
\def\hiderowcolors{\noalign{\global\@rowcolorsfalse\@norowcolor}}
\def\@norowcolor{\global\let\CT@row@color\relax}
\@norowcolor
```

`\if@rowcolors` `\if@rowcmd`

```
\newif\if@rowcolors
\newif\if@rowcmd
```

`\rownum` `\c@rownum` Reserve a counter register. Also alias as a LaTeX counter (but not via `\newcounter` as should not be in the reset list.)

```
\@ifundefined{rownum}{%
   \@ifundefined{c@rownum}%
      {\newcount\rownum\let\c@rownum\rownum}%
      {\let\rownum\c@rownum}%
    }%
   {\let\c@rownum\rownum}
\providecommand\therownum{\arabic{rownum}}
```

`\cellcolor` `\cellcolor` applies the specified color to just its own tabular cell. It is defined robust, but without using `\DeclareRobustCommand` or `\newcommand{}[][]` because those forms are not used elsewhere, and would not work in *very* old LaTeX.

```
\edef\cellcolor{\noexpand\protect
  \expandafter\noexpand\csname cellcolor \endcsname}
\@namedef{cellcolor }{%
  \@ifnextchar[{\CT@cellc\@firstofone}{\CT@cellc\@gobble[]}%
}
\def\CT@cellc#1[#2]#3{%
  \expandafter\gdef\expandafter\CT@cell@color\expandafter{%
    \expandafter\CT@color#1{[#2]}{#3}%
    \global\let\CT@cell@color\relax
}}
\global\let\CT@cell@color\relax
```

`\DC@endright` dcolumn support. the `D` column sometimes internally converts a `c` column to an `r` one by squashing the supplied glue. This is bad news for this package, so redefine it to add negative glue to one side and positive to the other to keep the total added zero.

```
\AtBeginDocument{%
  \def\@tempa{$\hfil\egroup\box\z@\box\tw@}%
  \ifx\@tempa\DC@endright
```

New version of dcolumn, only want to fudge it in the `D{.}{.}{3}` case, not the new `D{.}{.}{3.3}` possibility. `\hfill` has already been inserted, so need to remove 1fill's worth of stretch.

```
    \def\DC@endright{%
```

```
      $\hfil\egroup
    \ifx\DC@rl\bgroup
      \hskip\stretch{-.5}\box\z@\box\tw@\hskip\stretch{-.5}%
    \else
      \box\z@\box\tw@
    \fi}%
  \else
    \def\@tempa{$\hfil\egroup\hfill\box\z@\box\tw@}%
    \ifx\@tempa\DC@endright
```

Old dcolumn code.

```
      \def\DC@endright{%
        $\hfil\egroup%
        \hskip\stretch{.5}\box\z@\box\tw@\hskip\stretch{-.5}}%
    \fi
  \fi}
```

hhline support (almost the whole package, repeated, sigh).

```
\AtBeginDocument{%
  \ifx\hhline\@undefined\else
\def\HH@box#1#2{\vbox{{%
  \ifx\CT@drsc@\relax\else
    \global\dimen\thr@@\tw@\arrayrulewidth
    \global\advance\dimen\thr@@\doublerulesep
    {\CT@drsc@
     \hrule \@height\dimen\thr@@
     \vskip-\dimen\thr@@}%
  \fi
  \CT@arc@
  \hrule \@height \arrayrulewidth \@width #1
  \vskip\doublerulesep
  \hrule \@height \arrayrulewidth \@width #2}}}
\def\HH@loop{%
  \ifx\@tempb`\def\next##1{\the\toks@\cr}\else\let\next\HH@let
  \ifx\@tempb|\if@tempswa
          \ifx\CT@drsc@\relax
           \HH@add{\hskip\doublerulesep}%
          \else
           \HH@add{{\CT@drsc@\vrule\@width\doublerulesep}}%
           \fi
          \fi\@tempswatrue
          \HH@add{{\CT@arc@\vline}}\else
  \ifx\@tempb:\if@tempswa
          \ifx\CT@drsc@\relax
           \HH@add{\hskip\doublerulesep}%
          \else
           \HH@add{{\CT@drsc@\vrule\@width\doublerulesep}}%
           \fi
              \fi\@tempswatrue
      \HH@add{\@tempc\HH@box\arrayrulewidth\arrayrulewidth\@tempc}\else
  \ifx\@tempb##\if@tempswa\HH@add{\hskip\doublerulesep}\fi\@tempswatrue
         \HH@add{{\CT@arc@\vline\copy\@ne\@tempc\vline}}\else
  \ifx\@tempb~\@tempswafalse
           \if@firstamp\@firstampfalse\else\HH@add{&\omit}\fi
```

```
              \ifx\CT@drsc@\relax
                \HH@add{\hfil}\else
                 \HH@add{{%
                   \CT@drsc@\leaders\hrule\@height\HH@height\hfil}}%
               \fi
                 \else
\ifx\@tempb-\@tempswafalse
          \gdef\HH@height{\arrayrulewidth}%
          \if@firstamp\@firstampfalse\else\HH@add{&\omit}\fi
             \HH@add{{%
               \CT@arc@\leaders\hrule\@height\arrayrulewidth\hfil}}%
                          \else
\ifx\@tempb=\@tempswafalse
     \gdef\HH@height{\dimen\thr@@}%
     \if@firstamp\@firstampfalse\else\HH@add{&\omit}\fi
     \HH@add
        {\rlap{\copy\@ne}\leaders\copy\@ne\hfil\llap{\copy\@ne}}\else
```

Stop the backspacing for `t` and `b`, it messes up the underlying color.

```
\ifx\@tempb t\HH@add{%
  \def\HH@height{\dimen\thr@@}%
  \HH@box\doublerulesep\z@}\@tempswafalse\else
\ifx\@tempb b\HH@add{%
  \def\HH@height{\dimen\thr@@}%
  \HH@box\z@\doublerulesep}\@tempswafalse\else
\ifx\@tempb>\def\next##1##2{%
   \HH@add{%
    {\baselineskip\p@\relax
     ##2%
    \global\setbox\@ne\HH@box\doublerulesep\doublerulesep}}%
     \HH@let!}\else
\ifx\@tempb\@sptoken\let\next\HH@spacelet\else
\PackageWarning{hhline}%
    {\meaning\@tempb\space ignored in \noexpand\hhline argument%
     \MessageBreak}%
\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
\next}
\lowercase{\def\HH@spacelet} {\futurelet\@tempb \HH@loop}
\fi}
```

longtable support.

```
\ExplSyntaxOn
```

Stub tag support if tagging has not been enabled.

```
\cs_if_exist:NF\tag_mc_begin:n{
  \cs_new:Npn\tag_mc_begin:n#1{}
  \cs_new:Npn\tag_mc_end:{}
}
\AtBeginDocument{
    \def\LT@@hline{%
      \ifx\LT@next\hline
        \global\let\LT@next\@gobble
        \ifx\CT@drsc@\relax
          \gdef\CT@LT@sep{%
```

```
            \noalign{\penalty-\@medpenalty\vskip\doublerulesep}}%
        \else
          \gdef\CT@LT@sep{%
            \multispan\LT@cols{%
              \tag_mc_begin:n{artifact}
              \CT@drsc@\leaders\hrule\@height\doublerulesep\hfill
              \tag_mc_end: \int_gdecr:N \g__tbl_row_int
            }\cr}%
        \fi
      \else
        \global\let\LT@next\empty
        \gdef\CT@LT@sep{%
          \noalign{\penalty-\@lowpenalty\vskip-\arrayrulewidth}}%
      \fi
      \ifnum0=`{\fi}%
      \multispan\LT@cols
       {\tag_mc_begin:n{artifact}
        \CT@arc@\leaders\hrule\@height\arrayrulewidth\hfill
        \tag_mc_end: \int_gdecr:N \g__tbl_row_int
       }\cr
      \CT@LT@sep
      \multispan\LT@cols
       {\tag_mc_begin:n{artifact}
        \CT@arc@\leaders\hrule\@height\arrayrulewidth\hfill
        \tag_mc_end: \int_gdecr:N \g__tbl_row_int
       }\cr
      \noalign{\penalty\@M}%
      \LT@next}
    }
\ExplSyntaxOff
```

⟨/package⟩