The cellprops package
CSS-like cell and table properties*

Julien "_FrnchFrgg_" Rivaud†

Released 2021/01/30

1 cellprops documentation

This package reworks the internals of `tabular`, `array`, and similar constructs, and adds a `\cellprops` command accepting CSS-like selectors and properties. It implements the `border-collapse: separate` CSS model.

It depends on mdwtab, xcolor and of course expl3 and xparse.

cellprops default settings mimick the LaTeX layout, that is left and right padding equal to `\tabcolsep` or `\arraycolsep`, zero top and bottom padding, but minimum height and depth corresponding to the table strut box.

I recommend to add globally:

```
\cellprops{ td { padding: 1ex; min-height: 0pt; min-depth: 0pt; } }
```

so that you get better-looking tables by default.

1.1 Examples

To produce the (arguably ugly):

This is text	A_2	A_3	A_4
B1	$Thisismaths$	B_3	
C1	C_2	X	Y
D1	D_2	DX	v
	F	$\int_a^b f(t)dt$	v
Top	Middle	Bottom	Baseline

you can use:

*This file describes v2.0, last revised 2021/01/30.

†E-mail: frnchfrgg@free.fr

```
\[
  \cellprops{
      td {
          padding: 1ex;
          min-height: 0pt;
          min-depth: 0pt;
          border-style: none solid solid none;
          text-align: center;
      }
      table {
          background-color: black!5!white;
      }
      tr:nth-child(even) {
          background-color: black!15!white;
      }
      td:nth-child(even) {
          background-color: yellow!20!white
      }
      tr:nth-child(even) td:nth-child(even) {
          background-color: yellow!50!white;
      }
      tr:first-child td {
          border-top-style: solid;
      }
      td:first-child {
          border-left-style: solid;
          math-mode: text;
          text-align: left;
      }
      tr:nth-child(6) td:nth-child(1) {
          vertical-align: top; % see remark in usage guide
      }
      tr:nth-child(6) td:nth-child(2) {
          vertical-align: middle;
      }
      tr:nth-child(6) td:nth-child(3) {
          vertical-align: bottom; % see remark in usage guide
      }
  }
  \begin{array}{nnnp{5em}}
      This is text & A_2 & A_3 & A_4 \\
      B1 & This is maths & B_3 & \\
      C1 &  C_2  &  X  & Y \\
      D1 &  D_2  &  DX & v \\
      &  F  &  \int_a^b f(t) dt & v \\
      \fbox{Top} & \fbox{Middle} & \fbox{Bottom} & \fbox{Baseline}%
      \textcolor{red}{%
         \kern -9cm\relax
         \vrule height 0.1pt depth 0.1pt width 10cm
         \kern -1cm\relax
```

```
      }\textcolor{blue}{%
         \kern -9cm\relax
         \vrule height 0.52ex depth -0.48ex width 10cm
         \kern -1cm\relax
      } \\
  \end{array}
\]
```

You can also use the `longtable` environment:

aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb
aaaaa	baaaa	caaaaa	dbbbb

This table has been produced by:

```
\cellprops{
  td { border: thin solid black; }
  tr:nth-child(-n+3) { background-color: black!10; }
  tr:nth-child(n+8) { background-color: blue!10; }
  tr:nth-child(4n) td:first-child,
  tr:nth-child(4n+1) td:nth-child(2),
  tr:nth-child(4n+2) td:nth-child(3),
  tr:nth-child(4n+3) td:nth-child(4) {
```

```
      border: thick solid red;
  }
}
\begin{longtable}{nnnn}
  aaaaa & baaaa & caaaaa & dbbbb \\
  ...
  aaaaa & baaaa & caaaaa & dbbbb \\
\end{longtable}
```

1.2 Usage guide

<usage>: `'\cellprops{'` [*<selectors>* `'{'` *<properties>* `'}'`]* `'}'`

<selectors>: *<selector>* [, *<selectors>*]

<selector>: [*<table>* `' '`] [*<tr>* `' '`] [*<td>* `' '`] [*<parbox>*]

<table>: `'table'` [`'.'` *<class>*]* [`':where(.'`*<class>*`')'`]* | `':where('`*<table>*`')'`

<tr>: `'tr'` [*<pseudoclass>*]* | `':where('`*<tr>*`')'`

<td>: `'td'` [*<pseudoclass>*]* | `':where('`*<td>*`')'`

<parbox>: `'p'` | `':where('`*<p>*`')'`

<pseudoclass>:`':where('`[*<pseudoclass>*]*`')'` | `':nth-child('`*<nth>*`')`

<nth>: `'odd'` | `'even'` | *<number>* | *<number>*`'n+'`*<number>* | *<number>*`'n-'`*<number>*

<properties>: [*<property>* `';'`]*

<property>:
`'padding: '` (*<dimension>*) {1,4} |
`'padding-top: '` *<dimension>* |
`'padding-right: '` *<dimension>* |
`'padding-bottom: '` *<dimension>* |
`'padding-left: '` *<dimension>* |
`'min-height: '` *<dimension>* |
`'min-depth: '` *<dimension>* |
`'min-width: '` *<dimension>* |
`'text-align: '` (`'left'` | `'right'` | `'center'`) |
`'vertical-align: '` (`'top'` | `'middle'` | `'baseline'` | `'bottom'`) |
`'math-mode: '` (`'text'` | `'math'` | `'auto'`) |
`'color: '` *<color>* |
`'background-color: '` (*<color>* | `'transparent'`) |
`'border: '` [*<bd-width>*] [*<bd-style>*] [*<color>*] |
`'border-top: '` [*<bd-width>*] [*<bd-style>*] [*<color>*] |
`'border-right: '` [*<bd-width>*] [*<bd-style>*] [*<color>*] |
`'border-bottom: '` [*<bd-width>*] [*<bd-style>*] [*<color>*] |
`'border-left: '` [*<bd-width>*] [*<bd-style>*] [*<color>*] |
`'border-width: '` (*<bd-width>*) {1,4} |
`'border-top-width: '` *<bd-width>*) |

'border-right-width: ' <*bd-width*>) |
'border-bottom-width: ' <*bd-width*>) |
'border-left-width: ' <*bd-width*>) |
'border-style: ' (<*bd-style*>) {1,4} |
'border-top-style: ' <*bd-style*>) |
'border-right-style: ' <*bd-style*>) |
'border-bottom-style: ' <*bd-style*>) |
'border-left-style: ' <*bd-style*>) |
'border-color: ' (<*color*>) {1,4} |
'border-top-color: ' <*color*>) |
'border-right-color: ' <*color*>) |
'border-bottom-color: ' <*color*>) |
'border-left-color: ' <*color*>)

<*color*>: 'inherit' | <*xcolor-expression*> |
'rgb('<*red-0-255*>','<*green-0-255*>','<*blue-0-255*>')' |
'hsl('<*hue-0-360*>','<*sat-0-1*>','<*lum-0-1*>')'

Most of these properties are straight-forward. You should check a CSS documentation to get more information. A very good source is the Mozilla Developer Network.

Here are the supported column types:

- `n`: The most basic cell type, hbox, honoring all properties.
- `l`, `c` and `r`: Same as `n` but with forced `text-align`.
- `Ml`, `Mc` and `Mr`: Same as column `l`, `c` and `r` but enforces `math-mode: math`. The net effect is that `Mc` will create a centered column whose contents are in non-display math mode.
- `T`<*align*>: Same as `M`<*align*> but enforces `math-mode: text`.
- `p{`<*width*>`}`, `m`<*width*> and `b`<*width*>: parbox cell with the corresponding vertical alignment (`\vtop`, `\vcenter` or `\vbox`).
- `*{`<*count*>`}{`<*coltypes*>`}`: same as in array or mdwtab.
- `>{`<*prefix*>`}` and `<{`<*suffix*>`}`: same as in array or mdwtab.
- You can try to use constructs of array or mdwtab, but they might alter the function of cellprops. Most should be fine though.

The intended usage is to use `n`-type columns and set the properties with CSS, but LaTeX-like columns in the preamble are often less verbose.

Details for some properties:

- `math-mode: auto` means that the cell will be in math mode in environments `array`, `matrix`, ..., and in text mode in environments like `tabular`, ...
- `background-color` is only painted on the cell, and `transparent` actually means `inherit` except that if all values encountered are `inherit`/`transparent` no background is painted at all. That means that (currently) you cannot paint a row in some color and rely on transparency to have it bleed through a cell background.

- There are no columns in the CSS object model so you have to use `td:nth-child()` to select a column. Currently, cells spanning several columns actually increase the child count by the number of column they span, so that nth-child can still be used to select columns. This is not consistent with the HTML specification of tables, but acts as if a cell spanning multiple columns was implicitly creating `display: none` empty cell siblings following it.
- Any `:nth-child(`A`n+`B`)` or `:nth-child(`A`n)` or `:nth-child(`B`)` is supported, with arbitrary A and B, including nothing for A (standing for $A = 1$) or just a minus sign (standing for $A = -1$).
- `vertical-align` values `baseline` and `middle` are following the CSS specification. On the other hand, `top` (resp. `bottom`) align the top (resp. bottom) of the cell with the row baseline which gives a different result than CSS if there is mixed alignment in the row. The common case where all cells have `vertical-align: top` or `vertical-align: bottom` behaves as expected from the CSS specification.

  As fas as I can tell, obeying CSS in all cases would require typesetting each row in two passes.
- Class names are matched against the classes defined with `\cellpropsclass`, which takes a space-separated list of classes. By default, `\cellpropsclass` also add as a class the name of the environment that created the tabular-like structure (that is, `tabular` or `array` or `matrix` or similar). If you do not want that behavior, you can use `\cellpropsclass*`. The class list is redefined locally, and cellprops initially calls `\cellpropsclass{}`.
- The `:where()` pseudo-class is the same as in CSS selectors level 4, except that it does not accept spaces nor commas. Its only effect is to neutering the specificity of the selectors within while still keeping their matching conditions. This is mainly useful to write default yet specific rules whose specificity would otherwise dwarf simple rules.

1.3 Compatibility

This package has been tested compatible with diagbox, spreadtab, collcell. Compatibility with longtable has been specifically taken care of, provided cellprops is loaded afterwards. Table packages that only introduce new column types should be loaded after mdwtab, so either you load mdwtab manually and load your package in between mdwtab and cellprops, or you load your package after cellprops (provided it doesn't overwrite the machinery).

1.4 TODO

Add a test suite with compatibility tests. Improve the documentation, and test more LaTeX table constructs and preamble column types.

2 cellprops implementation

```
1 ⟨*package⟩
2 ⟨@@=cellprops⟩
3 \ProvidesExplPackage
```

```
  {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}

\RequirePackage{xparse}
\RequirePackage{xcolor}
\RequirePackage{etoolbox}
```

2.1 Loading and fixing mdwtab

There is a bug in the command `\colpop` of mdwtab: instead of just popping one name in the stack of column sets currently used, it empties it completely because one `\expandafter` is missing. This is proof that not many package authors really use this API as recommended by Mark Wooding... We thus load mdwtab and fix `\colpop`.

```
\RequirePackage{mdwtab}
\cs_set_nopar:Npn \tab@pop #1 { \tl_set:Nx #1 { \tl_tail:N #1 } }
```

2.2 Parsing CSS properties

Properties are parsed once at setting time, by expandable parsers that leave definitions in the input stream. All these resulting definitions are saved in a token list that will be expanded when we need the values. The goal is to have multiple token lists for multiple contexts, yet not to do the full parsing dance once per cell.

\l__cellprops_property_value_<name>_tl
__cellprops_generic_setter:nnn
__cellprops_get_property:n
__cellprops_get_property:nN

We first define a generic setter which just uses `\l__cellprops_property_value_<name>_tl` to store the value of the property. We define getters, one that leaves the value in the stream, and one saving the value in a token list.

```
\cs_new:Nn \__cellprops_generic_setter:nnn {
    \exp_not:N \tl_set:Nn
    \exp_not:c { l__cellprops_property_value_#2_tl }
    {#1 {#3}}
}

\cs_set_nopar:Nn \__cellprops_get_property:n {
    \tl_use:c { l__cellprops_property_value_#1_tl }
}

\cs_new_protected_nopar:Nn \__cellprops_get_property:nN {
    \tl_if_exist:cTF { l__cellprops_property_value_#1_tl } {
        \tl_set_eq:Nc #2 { l__cellprops_property_value_#1_tl }
    }{
        \tl_clear:N #2
    }
}
```

(*End definition for* `\l__cellprops_property_value_<name>_tl` *and others.*)

__cellprops_property_type_<name>:nn
__cellprops_define_properties:nn

The control sequence `\__cellprops_property_type_<name>:nn` holds the setter for the property ⟨*name*⟩. It can be set by the following helper:

```
\cs_new_protected:Nn \__cellprops_define_properties:nn {
    \clist_map_inline:nn {#2} {
        \cs_set:cpn { __cellprops_property_type_##1:nn } {#1}
    }
}
```

(*End definition for* `\__cellprops_property_type_<name>:nn` *and* `\__cellprops_define_properties:nn`.)

`\__cellprops_use_setter:nn` Sometimes we need to use a setter right away rather than save its action somewhere. The following helper does that with an x-expansion.

```
\cs_new:Nn \__cellprops_delegate_setter:nn {
    \use:c {__cellprops_property_type_#1:nn} {#1} {#2}
}
\cs_new_protected:Nn \__cellprops_use_setter:nn {
    \use:x {
        \__cellprops_delegate_setter:nn {#1} {#2}
    }
}
```

(*End definition for* `\__cellprops_use_setter:nn`.)

`\__cellprops_parse_properties:nn` Now we can parse the block of properties for a given selector. The first argument is the token list variable which will ultimately hold the expanded code setting internal variables from the properties. That code will be called when recalling the computed values in a specific context.

```
\cs_new_protected:Nn \__cellprops_parse_properties:Nn {
    \tl_clear:N #1
    \seq_set_split:Nnn \l_tmpa_seq {;} {#2}
    \seq_map_inline:Nn \l_tmpa_seq {
        \tl_if_empty:nF {##1} {
            \exp_args:NNV \seq_set_split:Nnn \l_tmpb_seq \c_colon_str {##1}
            \int_compare:nNnT {\seq_count:N \l_tmpb_seq} = { 2 } {
                \seq_get_left:NN \l_tmpb_seq \l_tmpa_tl
                \exp_args:NNV \str_set:Nn \l_tmpa_str \l_tmpa_tl
                \seq_get_right:NN \l_tmpb_seq \l_tmpa_tl
                \cs_if_exist:cT { __cellprops_property_type_\l_tmpa_str :nn } {
                    \tl_put_right:Nx #1 {
                        \exp_args:NVV \__cellprops_delegate_setter:nn
                            \l_tmpa_str \l_tmpa_tl
                    }
                }
            }
        }
    }
}
```

(*End definition for* `\__cellprops_parse_properties:nn`.)

2.3 Defining new properties

2.3.1 Some helpers

`\__cellprops_fourval_setter:nnnnnn` `\__cellprops_define_fourval_properties:nnnnnn` We first define helpers to parse and define compound properties like `padding` where you can give one to four different values and the missing values are copied from the given ones.

```
\cs_new:Nn \__cellprops_fourval_setter:nnnnnn {
    \__cellprops_fourval_setter_aux:w
        {#1}{#2}{#3}{#4}#6~{\q_no_value}~{\q_no_value}~{\q_no_value}~\q_stop
}
```

```
\cs_new:Npn \__cellprops_fourval_setter_aux:w #1#2#3#4#5~#6~#7~#8~#9\q_stop {
    \__cellprops_delegate_setter:nn {#1} {#5}
    \quark_if_no_value:nTF {#6} {
        \__cellprops_delegate_setter:nn {#2} {#5}
        \__cellprops_delegate_setter:nn {#4} {#5}
    }{
        \__cellprops_delegate_setter:nn {#2} {#6}
        \quark_if_no_value:nTF {#8} {
            \__cellprops_delegate_setter:nn {#4} {#6}
        }{
            \__cellprops_delegate_setter:nn {#4} {#8}
        }
    }
    \quark_if_no_value:nTF {#7} {
        \__cellprops_delegate_setter:nn {#3} {#5}
    }{
        \__cellprops_delegate_setter:nn {#3} {#7}
    }
}

\cs_new_protected:Nn \__cellprops_define_fourval_properties:nnnnnn {
    \__cellprops_define_properties:nn {#1} { #3, #4, #5, #6 }
    \__cellprops_define_properties:nn {
        \__cellprops_fourval_setter:nnnnnn {#3}{#4}{#5}{#6}
    }{
        #2
    }
}
```

(*End definition for* `\__cellprops_fourval_setter:nnnnnn` *and* `\__cellprops_define_fourval_properties:nnnnnn`.)

`\__cellprops_color_setter:nn` This macro is used to parse color definitions, either named, rgb, or hsl.

```
\tl_const:Nn \c__cellprops_inherit_color_tl { \q_nil }

\cs_new_nopar:Nn \__cellprops_color_setter:nn {
    \str_if_eq:nnTF {#2} {inherit} {
        \__cellprops_generic_setter:nnn \exp_not:n {#1} {\c__cellprops_inherit_color_tl}
    }{
        \str_case_e:nnF { \str_range:nnn {#2} {1} {4} } {
            {rgb(} {
                \__cellprops_generic_setter:nnn \use:n {#1} {
                    \exp_not:n {\color[RGB]} {\str_range:nnn {#2} {5} {-2}}
                }}
            {hsl(} {
                \__cellprops_generic_setter:nnn \use:n {#1} {
                    \exp_not:n {\color[Hsb]} {\str_range:nnn {#2} {5} {-2}}
                }}
        }{
            \__cellprops_generic_setter:nnn \exp_not:n {#1} {
                \color{#2}
            }
        }
    }
}
```

(*End definition for* `\__cellprops_color_setter:nn`.)

`\__cellprops_bgcolor_setter:nn` For background colors, we support `transparent` as an alias for `inherit`.

```
\cs_new_nopar:Nn \__cellprops_bgcolor_setter:nn {
    \str_if_eq:nnTF {#2} {transparent} {
        \__cellprops_color_setter:nn {#1} {inherit}
    }{
        \__cellprops_color_setter:nn {#1} {#2}
    }
}
```

(*End definition for* `\__cellprops_bgcolor_setter:nn`.)

`\__cellprops_linewidth_setter:nn` A setter for line widths that supports common keywords:

```
\cs_new_nopar:Nn \__cellprops_linewidth_setter:nn {
    \str_case:nnF {#2} {
        {thin}   { \__cellprops_generic_setter:nnn \exp_not:n {#1} { \fboxrule} }
        {medium} { \__cellprops_generic_setter:nnn \exp_not:n {#1} { 2\fboxrule} }
        {thick}  { \__cellprops_generic_setter:nnn \exp_not:n {#1} { 3\fboxrule} }
    }{
        \__cellprops_generic_setter:nnn \exp_not:n {#1} {#2}
    }
}
```

(*End definition for* `\__cellprops_linewidth_setter:nn`.)

`\__cellprops_border_setter:nn` `border` and `border-`<side> are compound properties that can define the width, the style and the color. As per the specification, the `border` property always sets all four sides at the same time instead of being a four-valued property.

```
\cs_new_nopar:Nn \__cellprops_border_setter:nn {
    \__cellprops_border_setter_aux:nw
        {#1}#2~{\q_no_value}~{\q_no_value}~\q_stop
}
\cs_new:Npn \__cellprops_border_setter_aux:nw #1#2~#3~#4~#5\q_stop {
    \quark_if_no_value:nTF {#4} {
        \__cellprops_border_setter_isstyle:nTF {#2} {
            \__cellprops_delegate_setter:nn {#1-width} {thin}
            \__cellprops_delegate_setter:nn {#1-style} {#2}
            \quark_if_no_value:nTF {#3} {
                \__cellprops_delegate_setter:nn {#1-color} {inherit}
            }{
                \__cellprops_delegate_setter:nn {#1-color} {#3}
            }
        }{
            \quark_if_no_value:nTF {#3} {
                %% One no-style value, ambiguous. Ignore the property
            }{
                \__cellprops_border_setter_isstyle:nTF {#3} {
                    \__cellprops_delegate_setter:nn {#1-width} {#2}
                    \__cellprops_delegate_setter:nn {#1-style} {#3}
                    \__cellprops_delegate_setter:nn {#1-color} {inherit}
                }{
                    \__cellprops_delegate_setter:nn {#1-width} {#2}
                    \__cellprops_delegate_setter:nn {#1-style} {none}
```

```
                    \__cellprops_delegate_setter:nn {#1-color} {#3}
                }
            }
        }
    }{
        \__cellprops_delegate_setter:nn {#1-width} {#2}
        \__cellprops_delegate_setter:nn {#1-style} {#3}
        \__cellprops_delegate_setter:nn {#1-color} {#4}
    }
}

\cs_new:Npn \__cellprops_border_setter_isstyle:nTF #1 {
    \str_case:nnTF {#1} {
        {none}{} {hidden}{} {dotted}{} {dashed}{} {solid}{}
        {double}{} {groove}{} {ridge}{} {inset}{} {outset}{}
    }
}
```

(*End definition for* `\__cellprops_border_setter:nn`.)

2.3.2 Actual definitions of properties

First some simple-valued properties where we just store the value unexpanded.

```
\__cellprops_define_properties:nn {
    \__cellprops_generic_setter:nnn \exp_not:n
}{
    min-height,
    min-depth,
    min-width,
}
```

`padding` is a compound property for `padding-`<side> which store their value unexpanded.

```
\__cellprops_define_fourval_properties:nnnnnn
    { \__cellprops_generic_setter:nnn \exp_not:n }
    {padding}
    {padding-top}{padding-right}{padding-bottom}{padding-left}
```

Simple-valued properties that store a str value.

```
\__cellprops_define_properties:nn {
    \__cellprops_generic_setter:nnn \tl_to_str:n
}{
    text-align,
    vertical-align,
    math-mode,
}
```

Some simple-valued color properties, using the dedicated parser.

```
\__cellprops_define_properties:nn {
    \__cellprops_color_setter:nn
}{
    color,
}

```

```
\__cellprops_define_properties:nn {
    \__cellprops_bgcolor_setter:nn
}{
    background-color,
}
```

A compound property whose individual sides use the linewidth setter for keyword recognition.

```
\__cellprops_define_fourval_properties:nnnnnn
    { \__cellprops_linewidth_setter:nn }
    {border-width}
    {border-top-width}{border-right-width}
    {border-bottom-width}{border-left-width}
```

A compound property whose individual sides are str values. They could be checked against the list of valid values, but any non-existing one will be ignored anyway due to the way they are implemented.

```
\__cellprops_define_fourval_properties:nnnnnn
    { \__cellprops_generic_setter:nnn \tl_to_str:n }
    {border-style}
    {border-top-style}{border-right-style}
    {border-bottom-style}{border-left-style}
```

A compound property whose individual sides are colors.

```
\__cellprops_define_fourval_properties:nnnnnn
    { \__cellprops_color_setter:nn }
    {border-color}
    {border-top-color}{border-right-color}
    {border-bottom-color}{border-left-color}
```

The five border-specific compound properties are defined here.

```
\__cellprops_define_properties:nn {
    \__cellprops_border_setter:nn
}{
    border, border-top, border-right, border-bottom, border-left
}
```

2.4 Parsing a CSS stylesheet

```
\NewDocumentCommand \cellprops { m } {
    \__cellprops_parse_css:n {#1}
}

\cs_new_protected:Nn \__cellprops_parse_css:n {
    \__cellprops_parse_css:w #1 \q_mark {\q_nil} \q_stop
```

Ensure the already seen specificities is in order without duplicates.

```
    \seq_remove_duplicates:N \l__cellprops_specificities_seq
    \seq_sort:Nn \l__cellprops_specificities_seq {
        \int_compare:nNnTF { ##1 } > { ##2 }
            { \sort_return_swapped: }
            { \sort_return_same: }
    }
}
```

Grab the content up to the first opening brace. That content will be the comma-separated selector list, and the braced content is a block of properties. We can loop until there is no such block remaining.

```
\tl_new:N \l__cellprops_parse_properties_tl
\NewDocumentCommand \__cellprops_parse_css:w { lmu{\q_stop} } {
    \quark_if_nil:nF {#2} {
        \__cellprops_parse_properties:Nn \l__cellprops_parse_properties_tl {#2}
        \clist_map_inline:nn {#1} {
            \tl_if_empty:nF {##1} { \__cellprops_parse_css_addprops:n {##1} }
        }
        \__cellprops_parse_css:w #3 \q_stop
    }
}
```

Some pseudo-classes generate conditional code for the properties to be applied. Check if such code exists, and wrap the parsed property setters in a `\bool_if:nT`.

```
\tl_new:N \l__cellprops_current_selector_tl
\tl_new:N \l__cellprops_current_selector_check_tl
\cs_new_protected:Nn \__cellprops_parse_css_addprops:n {
    \__cellprops_parse_selector:n {#1}
    \tl_if_empty:NF \l__cellprops_current_selector_tl {
        \tl_set:Nx \l_tmpa_tl
                { l__cellprops_property_group_\l__cellprops_current_selector_tl _tl }
        \tl_if_exist:cF { \l_tmpa_tl } { \tl_clear:c { \l_tmpa_tl } }
        \tl_if_empty:NTF \l__cellprops_current_selector_check_tl {
            \tl_put_right:cV { \l_tmpa_tl } \l__cellprops_parse_properties_tl
        }{
            \tl_put_right:cx { \l_tmpa_tl } {
                \exp_not:N \bool_if:nT {
                    \exp_not:V \l__cellprops_current_selector_check_tl
                }{
                    \exp_not:V \l__cellprops_parse_properties_tl
                }
            }
        }
    }
}
```

Here we parse a selector. These are naturally space-separated, but we first need to detect and normalize constructs like `:nth-child(argument)`. We replace them by `:nth-child{argument}` where the braces will procect any space that can legitimately occur within `argument`.

```
\cs_new_protected:Nn \__cellprops_parse_selector_sanitize:n {
    \exp_args:Nx \__cellprops_parse_selector_sanitize_aux:n
        { \tl_to_str:n{#1} }
}
\cs_new_protected:Nn \__cellprops_parse_selector_sanitize_aux:n {
    \cs_set:Npn \__cellprops_parse_selector_sanitize:w ##1:#1(##2)##3\q_stop
    {
        \quark_if_nil:nTF {##3} {
            ##1
        }{
            ##1:#1{##2}\__cellprops_parse_selector_sanitize:w ##3\q_stop
        }
```

```
    }
    \tl_set:Nx \l__cellprops_current_selector_tl {
        \exp_last_unbraced:NV
            \__cellprops_parse_selector_sanitize:w
            \l__cellprops_current_selector_tl
        :#1()\q_nil\q_stop
    }
}
```

Now that we can sanitize pseudo-classes, parsing the selector is safe. The constructs to protect are `:nth-child()` and `:where()` since other supported pseudo-classes have no argument.

```
\seq_new:N \l__cellprops_current_selector_seq
\seq_new:N \l__cellprops_pseudoclasses_seq
\tl_new:N \l__cellprops_current_element_tl
\tl_new:N \l__cellprops_current_tableclass_tl
\int_new:N \l__cellprops_current_level_int
\int_new:N \l__cellprops_current_specificity_int
\seq_new:N \l__cellprops_specificities_seq
\cs_new_protected:Nn \__cellprops_parse_selector:n {
    \tl_set:Nx \l__cellprops_current_selector_tl { \tl_to_str:n {#1} }
```

The sanitize code is more readable with Expl category colon, so replace it now, instead of defining the method with expand or lccode tricks.

```
    \exp_args:NNV \tl_replace_all:Nnn
        \l__cellprops_current_selector_tl \c_colon_str {:}
    \__cellprops_parse_selector_sanitize:n {nth-child}
    \__cellprops_parse_selector_sanitize:n {where}
    \seq_set_split:NnV \l__cellprops_current_selector_seq {~} \l__cellprops_current_selector
    \tl_clear:N \l__cellprops_current_selector_tl
    \tl_clear:N \l__cellprops_current_selector_check_tl
    \tl_clear:N \l__cellprops_current_tableclass_tl
    \int_set:Nn \l__cellprops_current_level_int {-1}
    \int_zero:N \l__cellprops_current_specificity_int
    \seq_map_inline:Nn \l__cellprops_current_selector_seq {
        \tl_clear:N \l__cellprops_current_element_tl
        \__cellprops_parse_simple_selector:n {##1}
    }
```

Remember the current selector as `<last element>~<specificity>`.

```
    \tl_if_empty:NF \l__cellprops_current_element_tl {
        \tl_if_empty:NF \l__cellprops_current_tableclass_tl {
            \tl_set:Nx \l__cellprops_current_tableclass_tl {
                l__cellprops_active_classes_\l__cellprops_current_tableclass_tl _bool
            }
            \bool_if_exist:cF { \l__cellprops_current_tableclass_tl } {
                \bool_new:c { \l__cellprops_current_tableclass_tl }
            }
            \__cellprops_add_check:x {
                \exp_not:n { \bool_if_p:N }
                {
                    \exp_not:c { \l__cellprops_current_tableclass_tl }
                }
            }
        }
```

```
        \tl_set:Nx \l__cellprops_current_selector_tl {
            \exp_not:V \l__cellprops_current_element_tl
            \exp_not:n {~}
            \int_use:N \l__cellprops_current_specificity_int
        }
        \seq_put_right:Nx \l__cellprops_specificities_seq {
            \int_use:N \l__cellprops_current_specificity_int
        }
    }
}

\tl_new:N \l__cellprops_maybe_element_tl

\cs_new_protected:Nn \__cellprops_parse_simple_selector:n {
```

First we replace `.` with `:.` so that class selectors can be handled with the same code as pseudo-classes. Then we split the current selector item on `:` to get the base element and the pseudo-classes.

```
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_replace_all:Nnn \l_tmpa_tl {.} {:.}
    \seq_set_split:NnV \l__cellprops_pseudoclasses_seq {:} \l_tmpa_tl
    \seq_pop_left:NN \l__cellprops_pseudoclasses_seq \l__cellprops_maybe_element_tl
```

Known type selectors increase the specificity by one.

```
    \str_case:VnTF \l__cellprops_maybe_element_tl {
        { table } { \int_set:Nn \l_tmpa_int { 1 } }
        { tr }    { \int_set:Nn \l_tmpa_int { 2 } }
        { td }    { \int_set:Nn \l_tmpa_int { 3 } }
        { p }     { \int_set:Nn \l_tmpa_int { 4 } }
    }{
        \int_add:Nn \l__cellprops_current_specificity_int { 1 }
    }{
```

An empty type selector is accepted to account for `:where()` pseudo-classes. They will be handled in the loop below, but of course there should be a type selector eventually. Map that case to a big positive next level so that we can detect that and keep the current level as-is.

Unknown type selectors are mapped to big negative levels so that the descendant check will refuse them, unless we are at the very first type selector in which case we map the deprecated environment selector to a class selector.

```
        \tl_if_empty:NTF \l__cellprops_maybe_element_tl {
            \int_set:Nn \l_tmpa_int { 10 }
        }{
            \int_compare:nNnTF \l__cellprops_current_level_int = { -1 } {
                \tl_set_eq:NN
                    \l__cellprops_current_tableclass_tl
                    \l__cellprops_maybe_element_tl
                \tl_set:Nn \l__cellprops_maybe_element_tl { table }
                \int_set:Nn \l_tmpa_int { 1 }
                \int_add:Nn \l__cellprops_current_specificity_int { 11 }
            }{
                \int_set:Nn \l_tmpa_int { -10 }
            }
        }
    }
```

If no type selector has been found yet, record the one we may have found now. Else, refuse to have two in the same selector.

```
    \tl_if_empty:NTF \l__cellprops_current_element_tl {
        \tl_set_eq:NN \l__cellprops_current_element_tl \l__cellprops_maybe_element_tl
    }{
        \tl_if_empty:NF \l__cellprops_maybe_element_tl {
            \int_set:Nn \l_tmpa_int { -10 }
        }
    }
```

If the detected element is a descendent of the previous one, the selector can match; count the element selector specificity and parse its pseudo-classes. If not, the whole selector cannot match and we discard it early.

```
    \int_compare:nNnTF \l_tmpa_int > \l__cellprops_current_level_int {
        \int_compare:nNnT \l_tmpa_int < 10 {
            \int_set_eq:NN \l__cellprops_current_level_int \l_tmpa_int
        }
        \seq_map_inline:Nn \l__cellprops_pseudoclasses_seq {
            \__cellprops_parse_pseudoclass:w ##1{}\q_stop
        }
    }{
        \tl_clear:N \l__cellprops_current_element_tl
    }
```

At the end of the selector parsing, the current element should not be empty.

```
    \tl_if_empty:NT \l__cellprops_current_element_tl {
        \seq_map_break:
    }
}
```

The first argument is the complete pseudo-class up to the opening argument brace (if any), and the second argument is the braced content (if any). The third argument gobbles any trailing garbage.

If the pseudo-class starts with a `.` this is a class selector. Else, this is an unknown pseudo-class and we reject the complete selector.

```
\NewDocumentCommand \__cellprops_parse_pseudoclass:w { lmu{\q_stop} } {
    \str_case:nnF { #1 } {
        {first-child} { \__cellprops_parse_selector_nth:n {1} }
        {nth-child}   { \__cellprops_parse_selector_nth:n {#2} }
        {where}       { \__cellprops_parse_where:n {#2} }
    }{
        \str_if_eq:eeTF { \str_head:n { #1 } } {.} {
            \tl_set:Nx \l__cellprops_current_tableclass_tl {
                \str_tail:n { #1 }
            }
            \int_add:Nn \l__cellprops_current_specificity_int { 10 }
        }{
            \tl_clear:N \l__cellprops_current_element_tl
            \seq_map_break:n { \seq_map_break: }
        }
    }
}

\str_const:Nn \c__cellprops_parse_n_str {n}
\int_new:N \l__cellprops_nth_coeff_int
```

```
\int_new:N \l__cellprops_nth_offset_int
\cs_new_protected:Nn \__cellprops_parse_selector_nth:n {
```

Count the pseudo-class in the specificity.

```
    \int_add:Nn \l__cellprops_current_specificity_int { 10 }
```

Now parse the nth-child argument:

```
    \str_case:nnF {#1} {
        {even} { \str_set:Nn \l_tmpa_str {2n} }
        {odd}  { \str_set:Nn \l_tmpa_str {2n+1} }
    }{
        \str_set:Nn \l_tmpa_str {#1}
    }
    \exp_args:NNV
        \seq_set_split:NnV \l_tmpa_seq \c__cellprops_parse_n_str \l_tmpa_str
    \seq_pop_right:NN \l_tmpa_seq \l_tmpa_tl
    \tl_if_empty:NTF \l_tmpa_tl {
        \int_zero:N \l__cellprops_nth_offset_int
    }{
        \int_set:Nn \l__cellprops_nth_offset_int { \l_tmpa_tl }
    }
    \seq_get_left:NNTF \l_tmpa_seq \l_tmpa_tl {
        \tl_if_empty:NTF \l_tmpa_tl {
            \int_set:Nn \l__cellprops_nth_coeff_int {1}
        }{
            \exp_args:NV \tl_if_eq:nnTF \l_tmpa_tl {-} {
                \int_set:Nn \l__cellprops_nth_coeff_int {-1}
            }{
                \int_set:Nn \l__cellprops_nth_coeff_int { \l_tmpa_tl }
            }
        }
    }{
        \int_zero:N \l__cellprops_nth_coeff_int
    }
```

At last, generate the condition code.

```
    \str_case:Vn \l__cellprops_current_element_tl {
        {tr} { \__cellprops_generate_check_nth:n {\g__cellprops_row_int} }
        {td} { \__cellprops_generate_check_nth:n {\g__cellprops_col_int} }
    }
}

\cs_new_protected_nopar:Nn \__cellprops_generate_check_nth:n {
    \int_compare:nNnTF \l__cellprops_nth_coeff_int = { 0 } {
        \__cellprops_add_check:x {
            \exp_not:n { \int_compare_p:nNn #1 = }
            {
                \exp_not:V \l__cellprops_nth_offset_int
            }
        }
    }{
        \tl_set:Nx \l_tmpb_tl {
            {
                \exp_not:n { #1 - }
                \exp_not:V \l__cellprops_nth_offset_int
            }{
```

```
                \exp_not:V \l__cellprops_nth_coeff_int
            }
        }
        \__cellprops_add_check:x {
            \exp_not:N \bool_lazy_and_p:nn {
                \exp_not:n { \int_compare_p:nNn 0 = }
                {
                    \exp_not:N \int_mod:nn
                    \exp_not:V \l_tmpb_tl
                }
            }{
                \exp_not:n { \int_compare_p:nNn 0 < }
                {
                    \exp_not:N \int_div_truncate:nn
                    \exp_not:V \l_tmpb_tl
                    \exp_not:n { + 1 }
                }
            }
        }
    }
}

\cs_new_protected:Nn \__cellprops_parse_where:n {
    \use:x {
        \exp_not:n {
            \__cellprops_parse_simple_selector:n { #1 }
            \int_set:Nn \l__cellprops_current_specificity_int
        }
        {
            \int_use:N \l__cellprops_current_specificity_int
        }
    }
}

\cs_new_protected:Nn \__cellprops_add_check:n {
    \tl_if_empty:NTF \l__cellprops_current_selector_check_tl {
        \tl_set:Nn \l__cellprops_current_selector_check_tl { #1 }
    }{
        \tl_set:Nx \l__cellprops_current_selector_check_tl {
            \exp_not:N \bool_lazy_and_p:nn {
                \exp_not:V \l__cellprops_current_selector_check_tl
            }{
                \exp_not:n { #1 }
            }
        }
    }
}
\cs_generate_variant:Nn \__cellprops_add_check:n {x}

\seq_new:N \l__cellprops_classes_seq
\NewDocumentCommand \cellpropsclass { sm } {
    \seq_set_split:Nnn \l__cellprops_classes_seq {~} { #2 }
    \IfBooleanF {#1} {
        \seq_put_right:Nn \l__cellprops_classes_seq { \@currenvir }
```

```
    }
}
\cellpropsclass{}

\cs_set_protected:Nn \__cellprops_recall_properties:n {
    \seq_map_inline:Nn \l__cellprops_specificities_seq {
        \tl_if_exist:cT { l__cellprops_property_group_#1~##1_tl } {
            \tl_use:c { l__cellprops_property_group_#1~##1_tl }
        }
    }
}

\dim_new:N \l__cellprops_colsep_dim
\dim_new:N \l__cellprops_strut_ht_dim
\dim_new:N \l__cellprops_strut_dp_dim

\ExplSyntaxOff
\cellprops{
    :where(td) {
        padding: 0pt \csname l__cellprops_colsep_dim\endcsname;
        min-height: \csname l__cellprops_strut_ht_dim\endcsname;
        min-depth: \csname l__cellprops_strut_dp_dim\endcsname;
        min-width: 0pt;
        text-align: left;
        vertical-align: baseline;
        math-mode: auto;
        color: inherit;
        background-color: transparent;
        border: thin none inherit;
    }
    :where(tr) {
        color: inherit;
        background-color: transparent;
    }
    :where(table) {
        padding: 0pt; % No change at load time
        color: inherit;
        background-color: transparent;
    }
}
\ExplSyntaxOn

\int_new:N \g__cellprops_row_int
\int_new:N \g__cellprops_col_int
\bool_new:N \g__cellprops_inrow_bool
\bool_gset_false:N \g__cellprops_inrow_bool

\box_new:N \l__cellprops_cell_box
\skip_new:N \l__cellprops_left_skip
\skip_new:N \l__cellprops_right_skip
\dim_new:N \g__cellprops_ht_dim
\dim_new:N \g__cellprops_dp_dim
\tl_new:N \g__cellprops_borders_tl

```

```
\tl_new:N \l__cellprops_restore_tl

\dim_new:N \l__cellprops_tablepadding_top_dim
\dim_new:N \l__cellprops_tablepadding_bottom_dim
\tl_new:N  \l__cellprops_color_tl
\tl_new:N  \l__cellprops_bgcolor_tl

\seq_new:N \l__cellprops_classes_at_start_seq

\cs_new_protected:Nn \__cellprops_array_init: {
    \tl_set:Nx \l__cellprops_restore_tl {
        \bool_if:NTF \g__cellprops_inrow_bool {
            \exp_not:n {\bool_gset_true:N \g__cellprops_inrow_bool}
        }{
            \exp_not:n {\bool_gset_false:N \g__cellprops_inrow_bool}
        }
        \exp_not:n { \int_gset:Nn \g__cellprops_row_int }
            { \int_use:N \g__cellprops_row_int }
        \exp_not:n { \int_gset:Nn \g__cellprops_col_int }
            { \int_use:N \g__cellprops_col_int }
        \exp_not:n { \dim_gset:Nn \g__cellprops_ht_dim }
            { \dim_use:N \g__cellprops_ht_dim }
        \exp_not:n { \dim_gset:Nn \g__cellprops_dp_dim }
            { \dim_use:N \g__cellprops_dp_dim }
        \exp_not:n { \tl_gset:Nn \g__cellprops_borders_tl }
            { \exp_not:V \g__cellprops_borders_tl }
    }
```

Unset all previous active classes, and set current ones as active.

```
    \seq_map_inline:Nn \l__cellprops_classes_at_start_seq {
        \bool_set_false:c { l__cellprops_active_classes_##1_bool }
    }
    \seq_set_eq:NN \l__cellprops_classes_at_start_seq \l__cellprops_classes_seq
    \seq_map_inline:Nn \l__cellprops_classes_at_start_seq {
        \bool_set_true:c { l__cellprops_active_classes_##1_bool }
    }
```

To count rows and columns.

```
    \int_gzero:N \g__cellprops_row_int
    \bool_gset_false:N \g__cellprops_inrow_bool
    \tl_gclear:N \g__cellprops_borders_tl
    \cs_set_eq:NN \__cellprops_orig_tab@readpreamble:n \tab@readpreamble
    \cs_set_eq:NN \tab@readpreamble \__cellprops_readpreamble:n
```

Zero `\col@sep` but remember its value for the default padding.

```
    \dim_set_eq:NN \l__cellprops_colsep_dim \col@sep
    \dim_zero:N \col@sep
```

Also ignore `\*extrasep` dimensions that are not part of cellprop interface and should be replaced by CSS equivalents.

```
    \dim_zero:N \tab@extrasep
    \group_begin:
        \__cellprops_recall_properties:n {table}
        \dim_gset:Nn \g_tmpa_dim { \__cellprops_get_property:n {padding-top} }
        \dim_gset:Nn \g_tmpb_dim { \__cellprops_get_property:n {padding-bottom} }
        \__cellprops_update_colors:
```

```
        \tl_gset_eq:NN \g_tmpa_tl \l__cellprops_color_tl
        \tl_gset_eq:NN \g_tmpb_tl \l__cellprops_bgcolor_tl
    \group_end:
    \dim_set_eq:NN \l__cellprops_tablepadding_top_dim \g_tmpa_dim
    \dim_set_eq:NN \l__cellprops_tablepadding_bottom_dim \g_tmpb_dim
    \tl_set_eq:NN \l__cellprops_color_tl \g_tmpa_tl
    \tl_set_eq:NN \l__cellprops_bgcolor_tl \g_tmpb_tl
    \dim_set:Nn \l__cellprops_strut_ht_dim { \box_ht:N \@arstrutbox }
    \dim_set:Nn \l__cellprops_strut_dp_dim { \box_dp:N \@arstrutbox }
    \box_clear:N \@arstrutbox
}

\cs_set_nopar:Nn \__cellprops_array_startcontent: {
    \hlx{s[\l__cellprops_tablepadding_top_dim]}
}

\cs_new_protected_nopar:Nn \__cellprops_maybe_startrow: {
    \bool_if:NF \g__cellprops_inrow_bool {
        \bool_gset_true:N \g__cellprops_inrow_bool
        \int_gincr:N \g__cellprops_row_int
        \int_gset_eq:NN \g__cellprops_col_int \c_one_int
        \dim_gzero:N \g__cellprops_ht_dim
        \dim_gzero:N \g__cellprops_dp_dim
    }
}

\cs_new_protected_nopar:Nn \__cellprops_maybe_endrow: {
    \bool_if:NT \g__cellprops_inrow_bool {
        \__cellprops_every_cell_end:
        \bool_gset_false:N \g__cellprops_inrow_bool
    }
}

\cs_new_protected_nopar:Nn \__cellprops_every_cell_end: {
    \int_gincr:N \g__cellprops_col_int
}

\cs_set_protected_nopar:Nn \__cellprops_readpreamble:n {
    \cs_set_eq:NN \tab@readpreamble \__cellprops_orig_tab@readpreamble:n
```

`\tab@multicol` is inserted at the beginning of a each row, and by `\multicolumn` after its `\omit`. We use it to ensure that the row is initialized correctly however it starts (normally or with a `\multicolumn`).

`\tab@tabtext` is inserted at the end of every cell but the last one, so we should ensure that its effect is applied at the end of the row; `\__cellprops_maybe_endrow` will take care of that.

```
    \tl_put_left:Nn \tab@multicol {\__cellprops_maybe_startrow:}
    \tl_put_left:Nn \tab@tabtext {\__cellprops_every_cell_end:}
    \tab@readpreamble{#1}
    \exp_args:Nx \tab@preamble
        { \the\tab@preamble \exp_not:N\__cellprops_maybe_endrow: }
}
```

The color inheritance is handled with `\l__cellprops_inherit_color_tl`, `\l__cellprops_color_tl` and `\l__cellprops_bgcolor_tl`. The role of `\__cellprops_`-

`update_color:Nn` is to set the inherit fallback to the already existing value of `#1` then set `#1` to the CSS value, which can be the inherit variable.

```
\cs_new_protected_nopar:Nn \__cellprops_update_color:Nn {
    \__cellprops_get_property:nN {#2} \l_tmpa_tl
    \exp_args:NV \tl_if_eq:NNF \l_tmpa_tl \c__cellprops_inherit_color_tl {
        \tl_set_eq:NN #1 \l_tmpa_tl
    }
}

\cs_new_protected_nopar:Nn \__cellprops_update_colors: {
    \__cellprops_update_color:Nn \l__cellprops_color_tl {color}
    \__cellprops_update_color:Nn \l__cellprops_bgcolor_tl {background-color}
}
```

Patch the `\@array`, `\LT@array`, `\@mkpream`, `\endarray` and `\endlongtable` commands, so that we can properly setup our line and column counting system. This is the most brittle part of cellprops, and subject to compatibility problems with other packages that patch those (hyperref in particular).

```
\AtEndPreamble{%
\cs_set_eq:NN \__cellprops_orig_array:w \@array
\cs_set_protected_nopar:Npn \@array[#1]#2 {
    \__cellprops_array_init:
    \__cellprops_orig_array:w [#1]{#2}
    \__cellprops_array_startcontent:
}

\cs_set_eq:NN \__cellprops_orig_LTmkpream:n \@mkpream
\cs_set_protected_nopar:Npn \@mkpream#1 {
    \group_end:
    \__cellprops_array_init:
    \group_begin:
    \__cellprops_orig_LTmkpream:n {#1}
}

\cs_set_eq:NN \__cellprops_orig_LTarray:w \LT@array
\cs_set_protected_nopar:Npn \LT@array [#1]#2 {
    \__cellprops_orig_LTarray:w [#1]{#2}
    \__cellprops_array_startcontent:
}

\cs_new_nopar:Nn \__cellprops_end_array:n {
    \tl_if_empty:NF \g__cellprops_borders_tl { \\ }
    \crcr
    \hlx{s[\l__cellprops_tablepadding_bottom_dim]}
    #1
    \tl_use:N \l__cellprops_restore_tl
}

\cs_set_eq:NN \__cellprops_orig_endarray: \endarray
\cs_set_nopar:Npn \endarray {
    \__cellprops_end_array:n { \__cellprops_orig_endarray: }
}
\cs_set_eq:NN \endtabular \endarray
\cs_set_eq:cN {endtabular*} \endarray
```

```
\cs_set_eq:NN \__cellprops_orig_endLT: \endlongtable
\cs_set_nopar:Npn \endlongtable {
    \__cellprops_end_array:n { \__cellprops_orig_endLT: }
}

\cs_new_protected_nopar:Nn \__cellprops_cr:n {
    \__cellprops_maybe_endrow:
    \tl_if_empty:NF \g__cellprops_borders_tl {
        \cr
        \noalign{\nobreak}
        \tl_use:N \g__cellprops_borders_tl
        \tl_gclear:N \g__cellprops_borders_tl
    }
    \cr
    \__cellprops_fix_valign_end:n {#1}
    \use_none:n
}

\cs_set_protected_nopar:Npn \tab@tabcr #1#2 { \__cellprops_cr:n {#2} }
\cs_set_protected_nopar:Npn \@xargarraycr #1 { \__cellprops_cr:n {#1} }
\cs_set_protected_nopar:Npn \@yargarraycr #1 { \__cellprops_cr:n {#1} }
\tl_if_exist:NT \LT@echunk {
    \tl_put_left:Nn \LT@echunk {
        \tl_if_empty:NF \g__cellprops_borders_tl { \\ }
    }
}

\cs_set_eq:NN \__cellprops_orig_multicolumn:w \multicolumn
\cs_set:Npn \multicolumn#1#2#3 {
    \__cellprops_orig_multicolumn:w {#1}{#2}{#3}
    \int_gadd:Nn \g__cellprops_col_int {#1}
    \tl_gput_right:Nx \g__cellprops_borders_tl {
        \prg_replicate:nn {#1 - 1} {\span\omit}
    }
    \ignorespaces
}

}

\cs_new_nopar:Nn \__cellprops_fix_valign_end:n {
    \noalign{
        \dim_set:Nn \l_tmpa_dim {#1}
        \skip_vertical:n {\l_tmpa_dim}
        \exp_args:NV \tl_if_eq:nnTF \tab@hlstate {b} {
            \dim_gadd:Nn \tab@endheight { \g__cellprops_dp_dim + \l_tmpa_dim }
        }{
            \int_compare:nNnT \g__cellprops_row_int = \c_one_int {
                \dim_gadd:Nn \tab@endheight { \g__cellprops_ht_dim }
            }
        }
    }
}
```

Reset `\firsthline` and `\lasthline` to `\hline` because the version from array which

might be loaded already will mess up the spacing and is unneeded anyway.

```
\cs_set_eq:NN \firsthline \hline
\cs_set_eq:NN \lasthline \hline

\colpush{tabular}

\coldef n{\tabcoltype{
    \__cellprops_begincell:n{}
}{
    \__cellprops_endcell:
}}
\coldef l{\tabcoltype{
    \__cellprops_begincell:n
        {\__cellprops_use_setter:nn {text-align} {left}}
}{
    \__cellprops_endcell:
}}
\coldef c{\tabcoltype{
    \__cellprops_begincell:n
        {\__cellprops_use_setter:nn {text-align} {center}}
}{
    \__cellprops_endcell:
}}
\coldef r{\tabcoltype{
    \__cellprops_begincell:n
        {\__cellprops_use_setter:nn {text-align} {right}}
}{
    \__cellprops_endcell:
}}
\coldef M#1{\__cellprops_MTcol:nn {math}{#1}}
\coldef T#1{\__cellprops_MTcol:nn {text}{#1}}
\cs_new_protected_nopar:Nn \__cellprops_MTcol:nn {
    % TODO: error if align not l, c, or r
    \exp_args:Nx \tabcoltype {
        \exp_not:N \__cellprops_begincell:n {
            \exp_not:n {\__cellprops_use_setter:nn {math-mode} {#1} }
            \exp_not:n {\__cellprops_use_setter:nn {text-align}} {
                \str_case:nn {#2} {
                    {l} {left}
                    {c} {center}
                    {r} {right}
                }
            }
        }
    }{
        \__cellprops_endcell:
    }
}

\coldef p#1{\tabcoltype{
    \__cellprops_begin_par_cell:nn \vtop {#1}
}{
    \__cellprops_end_par_cell:n {}
}}
```

```
\coldef m#1{\tabcoltype{
    \__cellprops_begin_par_cell:nn {\c_math_toggle_token\vcenter} {#1}
}{
    \__cellprops_end_par_cell:n{\c_math_toggle_token}
}}
\coldef b#1{\tabcoltype{
    \__cellprops_begin_par_cell:nn \vbox {#1}
}{
    \__cellprops_end_par_cell:n {}
}}


\colpop

\cs_new_protected_nopar:Nn \__cellprops_begincell:n {
    \__cellprops_begin_raw_cell:n {
        #1
        \hbox_set:Nw \l__cellprops_cell_box
        \str_case_e:nnF {\__cellprops_get_property:n {math-mode}} {
            { text } { \tab@btext }
            { math } { \tab@bmaths }
        }{% any other treated as |auto|
            \tab@bgroup
        }
    }
}

\cs_new_protected_nopar:Nn \__cellprops_endcell: {
    \str_case_e:nnF {\__cellprops_get_property:n {math-mode}} {
        { text } { \tab@etext }
        { math } { \tab@emaths }
    }{% any other treated as |auto|
        \tab@egroup
    }
    \hbox_set_end:
    \__cellprops_end_raw_cell:
}

\cs_new_protected_nopar:Nn \__cellprops_begin_par_cell:nn {
    \savenotes
    \__cellprops_begin_raw_cell:n{
        \hbox_set:Nw \l__cellprops_cell_box
        #1
        \bgroup
        \hsize#2\relax
        \@arrayparboxrestore
        \global\@minipagetrue
        \everypar{
            \global\@minipagefalse
            \everypar{}
        }
        \__cellprops_recall_properties:n {p}
    }
}
```

```
\cs_new_protected_nopar:Nn \__cellprops_end_par_cell:n {
    \ifhmode\@maybe@unskip\par\fi
    \unskip
    \egroup
    #1
    \hbox_set_end:
    \__cellprops_end_raw_cell:
    \spewnotes
}

\cs_new_protected_nopar:Nn \__cellprops_begin_raw_cell:n {
    \group_begin:
    \__cellprops_recall_properties:n {tr}
    \__cellprops_update_colors:
    \__cellprops_recall_properties:n {td}
    \__cellprops_update_colors:
    % Additional init code
    #1
    % Install the cell color
    \__cellprops_update_colors:
    \tl_use:N \l__cellprops_color_tl
}

\cs_new_protected_nopar:Nn \__cellprops_make_solid_hborder:nnn {
    \group_begin:
        \hbox_set_to_wd:Nnn \l_tmpa_box {1pt} {
            \hss
            \hbox:n {
                #3 % install color
                \vrule height~\dim_eval:n{#1+#2}
                    ~depth~-\dim_eval:n{#2}
                    ~width~3pt
            }
            \hss
        }
        \box_set_ht:Nn \l_tmpa_box { \c_zero_dim }
        \box_set_dp:Nn \l_tmpa_box { \c_zero_dim }
        \kern 1pt
        \box_use:N \l_tmpa_box
        \xleaders
            \box_use:N \l_tmpa_box
            \skip_horizontal:n {-4pt~plus~1fil}
        \box_use:N \l_tmpa_box
        \kern 1pt
        \skip_horizontal:n {0pt~plus~-1fil}
    \group_end:
}
\cs_new_protected_nopar:Nn \__cellprops_make_solid_vborder:nnn {
    \group_begin:
        \hbox_set_to_wd:Nnn \l_tmpa_box {0pt} {
            \hbox:n {
                #3 % install color
                \vrule height~\dim_eval:n{#2}~width~\dim_eval:n{#1}
            }
```

```
            \hss
        }
        \box_set_ht:Nn \l_tmpa_box { \c_zero_dim }
        \box_set_dp:Nn \l_tmpa_box { \c_zero_dim }
        \box_use:N \l_tmpa_box
    \group_end:
}
\clist_map_inline:nn {
    dotted, dashed, solid, double,
    groove, ridge, inset, outset
}{
    \cs_set_eq:cN {__cellprops_make_#1_hborder:nnn} \__cellprops_make_solid_hborder:nnn
    \cs_set_eq:cN {__cellprops_make_#1_vborder:nnn} \__cellprops_make_solid_vborder:nnn
}

\dim_new:N \l__cellprops_border_width_dim
\str_new:N \l__cellprops_border_style_str
\tl_new:N \l__cellprops_border_color_tl
\cs_new_protected_nopar:Nn \__cellprops_get_border_info:n {
    \dim_set:Nn \l__cellprops_border_width_dim {\__cellprops_get_property:n {border-
#1-width}}
    \__cellprops_get_property:nN {border-#1-style} \l_tmpa_tl
    \exp_args:NNV \str_set:Nn \l__cellprops_border_style_str \l_tmpa_tl
    \tl_clear:N \l__cellprops_border_color_tl
    \cs_if_exist:cTF {__cellprops_make_\l__cellprops_border_style_str _hborder:nnn} {
        \__cellprops_update_color:Nn \l__cellprops_border_color_tl {border-#1-
color}
    }{
        \dim_zero:N \l__cellprops_border_width_dim
    }
}

\cs_new_protected_nopar:Npn \__cellprops_make_hborder:nnnn #1 {
    \use:c { __cellprops_make_#1_hborder:nnn }
}
\cs_new_protected_nopar:Npn \__cellprops_make_vborder:nnnn #1 {
    \use:c { __cellprops_make_#1_vborder:nnn }
}

\cs_new_protected_nopar:Nn \__cellprops_end_raw_cell: {
    % Here \l__cellprops_cell_box must contain the contents of the cell
    %
    % Prepare the borders token list
    \int_compare:nNnT \g__cellprops_col_int = 1 {
        \tl_gclear:N \g__cellprops_borders_tl
    }
    \tl_gput_right:Nx \g__cellprops_borders_tl {
        \tl_if_empty:NF \g__cellprops_borders_tl { \exp_not:n {&} }
        \exp_not:n { \omit \kern \c_zero_dim }
    }
    % Handle min-height, min-depth and vertical-align
    % wrong values are treated as |baseline|.
    \box_set_ht:Nn \l__cellprops_cell_box {
        \dim_max:nn
```

```
            {\box_ht:N \l__cellprops_cell_box}
            {\__cellprops_get_property:n {min-height}}
    }
    \box_set_dp:Nn \l__cellprops_cell_box {
        \dim_max:nn
            {\box_dp:N \l__cellprops_cell_box}
            {\__cellprops_get_property:n {min-depth}}
    }
    \str_case_e:nn {\__cellprops_get_property:n {vertical-align}} {
        { top } {
            \hbox_set:Nn \l__cellprops_cell_box {
                \vbox_top:n {
                    \kern 0pt\relax
                    \box_use_drop:N \l__cellprops_cell_box
                }
            }
        }
        { bottom } {
            \hbox_set:Nn \l__cellprops_cell_box {
                \vbox:n {
                    \box_use_drop:N \l__cellprops_cell_box
                    \kern 0pt\relax
                }
            }
        }
        { middle } {
            \hbox_set:Nn \l__cellprops_cell_box {
                \dim_set:Nn \l_tmpa_dim {
                    (\box_dp:N \l__cellprops_cell_box
                    - \box_ht:N \l__cellprops_cell_box
                    + 1ex) / 2
                }
                \raisebox{\l_tmpa_dim}{\box_use_drop:N \l__cellprops_cell_box}
            }
        }
    }
    % Handle padding-top and border-top
    \__cellprops_get_border_info:n {top}
    \box_set_ht:Nn \l__cellprops_cell_box {
        \box_ht:N \l__cellprops_cell_box
        + (\__cellprops_get_property:n {padding-top})
        + \l__cellprops_border_width_dim
    }
    \dim_compare:nNnT \l__cellprops_border_width_dim > \c_zero_dim {
        \tl_gput_right:Nx \g__cellprops_borders_tl {
            \exp_not:N \__cellprops_make_hborder:nnnn
                { \exp_not:V \l__cellprops_border_style_str }
                { \dim_use:N \l__cellprops_border_width_dim }
                {
                    \exp_not:n { \g__cellprops_dp_dim + \g__cellprops_ht_dim - }
                    \dim_use:N \l__cellprops_border_width_dim
                }
                { \exp_not:V \l__cellprops_border_color_tl }
        }
```

```
    }
    % Handle padding-bottom and border-bottom
    \__cellprops_get_border_info:n {bottom}
    \box_set_dp:Nn \l__cellprops_cell_box {
        \box_dp:N \l__cellprops_cell_box
        + (\__cellprops_get_property:n {padding-bottom})
        + \l__cellprops_border_width_dim
    }
    \dim_compare:nNnT \l__cellprops_border_width_dim > \c_zero_dim {
        \tl_gput_right:Nx \g__cellprops_borders_tl {
            \exp_not:N \__cellprops_make_hborder:nnnn
                { \exp_not:V \l__cellprops_border_style_str }
                { \dim_use:N \l__cellprops_border_width_dim }
                { \exp_not:n { 0pt } }
                { \exp_not:V \l__cellprops_border_color_tl }
        }
    }
    % To fix vertical alignment later
    \dim_gset:Nn \g__cellprops_ht_dim {
        \dim_max:nn
            {\g__cellprops_ht_dim}
            {\box_ht:N \l__cellprops_cell_box}
    }
    \dim_gset:Nn \g__cellprops_dp_dim {
        \dim_max:nn
            {\g__cellprops_dp_dim}
            {\box_dp:N \l__cellprops_cell_box}
    }
    % Handle padding-left and border-left
    \__cellprops_get_border_info:n {left}
    \skip_set:Nn \l__cellprops_left_skip
        {\__cellprops_get_property:n {padding-left} + \l__cellprops_border_width_dim}
    \dim_compare:nNnT \l__cellprops_border_width_dim > \c_zero_dim {
        \tl_gput_right:Nx \g__cellprops_borders_tl {
            \exp_not:N \__cellprops_make_vborder:nnnn
                { \exp_not:V \l__cellprops_border_style_str }
                { \dim_use:N \l__cellprops_border_width_dim }
                { \exp_not:n { \g__cellprops_dp_dim + \g__cellprops_ht_dim } }
                { \exp_not:V \l__cellprops_border_color_tl }
        }
    }
    \tl_gput_right:Nx \g__cellprops_borders_tl {
        \exp_not:n {
            \skip_horizontal:n {0pt~plus~1fil}
            \kern \c_zero_dim
        }
    }
    \__cellprops_get_border_info:n {right}
    \skip_set:Nn \l__cellprops_right_skip
        {\__cellprops_get_property:n {padding-right} + \l__cellprops_border_width_dim}
    \dim_compare:nNnT \l__cellprops_border_width_dim > \c_zero_dim {
        \tl_gput_right:Nx \g__cellprops_borders_tl {
            \exp_not:N \skip_horizontal:n
                { - \dim_use:N \l__cellprops_border_width_dim }
```

```
            \exp_not:N \__cellprops_make_vborder:nnnn
                { \exp_not:V \l__cellprops_border_style_str }
                { \dim_use:N \l__cellprops_border_width_dim }
                { \exp_not:n { \g__cellprops_dp_dim + \g__cellprops_ht_dim } }
                { \exp_not:V \l__cellprops_border_color_tl }
            \exp_not:N \skip_horizontal:n
                { \dim_use:N \l__cellprops_border_width_dim }
            \exp_not:n { \kern \c_zero_dim }
        }
    }
    % Handle hpadding and halign
    \skip_set:Nn \l_tmpa_skip {
        \dim_max:nn
            {0pt}
            { (\__cellprops_get_property:n {min-width})
                - \box_wd:N \l__cellprops_cell_box }
    }
    \skip_add:Nn \l_tmpa_skip {
        1sp plus 1fil
    }
    \str_case_e:nnF {\__cellprops_get_property:n {text-align}} {
        { right } {
            \skip_add:Nn \l__cellprops_left_skip { \l_tmpa_skip }
        }
        { center } {
            \skip_add:Nn \l__cellprops_left_skip { \l_tmpa_skip / 2 }
            \skip_add:Nn \l__cellprops_right_skip { \l_tmpa_skip / 2 }
        }
    }{% any other treated as |left|
        \skip_add:Nn \l__cellprops_right_skip { \l_tmpa_skip }
    }
    \kern\c_zero_dim
    \tl_if_empty:NF \l__cellprops_bgcolor_tl {
        \group_begin:
        % Paint a background with leaders
        \tl_use:N \l__cellprops_bgcolor_tl % install the color
        \skip_set:Nn \l_tmpa_skip {
            \l__cellprops_left_skip
            + \box_wd:N \l__cellprops_cell_box
            + \l__cellprops_right_skip
        }
        \leaders
            \vrule
            \skip_horizontal:N \l_tmpa_skip
        \skip_horizontal:n {-\l_tmpa_skip}
        \group_end:
    }
    \skip_horizontal:N \l__cellprops_left_skip
    \box_use_drop:N \l__cellprops_cell_box
    \skip_horizontal:N \l__cellprops_right_skip
    \kern\c_zero_dim
    \group_end:
}
```

⟨/package⟩