The collcell Package

Martin Scharrer

martin@scharrer-online.de

http://www.ctan.org/pkg/collcell/

Version v0.5 – 2011/02/27

Abstract

This package provides macros which collect the cell content of a tabular and provide it to a macro as argument. It was inspired by the `\collect@body` macro defined by the `amsmath` or the `environ` package, which can be used to collect the body of an environment. Special care is taken to remove all aligning macros inserted by tabular from the cell content. The macros also work in the last column of a tabular. They do not support verbatim material inside the cells, except of a special almost-verbatim version of `\verb`.

This package is relatively new and might still not work in all possible situations which can arise in a tabular. The implementation might change in future versions. Please do not hesitate to contact the author about any issue and suggestions.

1 Usage

This package provides the macros `\collectcell` and `\endcollectcell` which are supposed to be used with the `>{ }` and `<{ }` tabular column declarations of the `array` package. This can be done either in the argument of tabular or using `\newcolumntype`.

The following code defines a '`E`' column which passes the contents of its cell to `\usermacro` as an argument. The macro can the process the content as usual.

```
% Preamble:
\usepackage{array}
\usepackage{collcell}
% Preamble or document:
\newcolumntype{E}{>{\collectcell\usermacro}c<{\endcollectcell}}

% Document:
\begin{tabular}{lE}
   A & Example \\ % Same as \usermacro{Example}
   B & Text    \\ % Same as \usermacro{Text}
\end{tabular}
```

For example `\usermacro` could be `\fbox` and wrap the cell content in a frame box. More complicated macros are also supported as long they take one argument. This package was originally programmed to be used with the `\tikztiming` macro of the `tikz-timing` package. This macro takes some complex user input and draws a timing diagram from it

Note that if such a cell contains a tabular environment by itself, the environment must be wrapped in braces '`{ }`' to ensure proper operation.

1.1 Options

The following options are supported:

verb

noverb (Default `noverb`) Enables or disables the definition of a special almost-verbatim version of `\verb`. At the moment the one defined by the `tabularx` package is used, which is therefore loaded when this feature is enabled. Future versions of `collcell` might provide this macro in a different way, so the visual result might be different. The `tabularx` should be loaded explicitly if it is used. This version of `\verb` will read the content first normally, i.e. non-verbatim, and then print the included tokens in a verbatim format. The content must include a balanced number of `{ }` and must not be end with `\`. Macros inside the content will be followed by a space. See the manual of `tabularx` (page 8 in the version from 1999/01/07) for a more detailed description.

robustcr

norobustcr (Default `robustcr`) This options enable or disable the redefinition of `\\` to a robust version, i.e. this macro will be prefixed with eTeX's `\protected` to ensure that it isn't expanded by the underlying `\halign`. If this feature disabled the last cell of a tabular must not be empty or only hold empty macros (like `\empty`).

1.2 Limitations

`\ccunskip`

The macro `\unskip` should not be included inside the cell directly, but only inside a `{ }` group or a macro. Otherwise it will be taken as part of the internal cell code and ignored. Leading spaces will however be removed. This macro can be used as a replacement of `\unskip` inside the cells.

`\cci`

The content of every cell is expanded by TeX itself until the first non-expandable token (macro, character, …) is found. This happens to check if a `\noalign` follows with e.g. is used inside `\hrule` and other rule drawing macros. There is nothing what `collcell` could do about this. If this expansion is unwanted the non-expandable token `\cci` should be placed at the beginning of the cell. This macro will be ignored (discarded) by `collcell` and will not be provided to the user macro (`cci` = collect cell; ignore).

2 Tests and Examples

@*T*@ *ABHLDZ2HZLZAZ5DTESTZZ5DTESTZAZ5DTESTZZ5DTEST*

```
\makeatletter
\newcommand*\Meaning[1]
   {\def\CODE{#1}\texttt{\expandafter\strip@prefix\meaning\CODE}}%
\newcolumntype{F}{>{\collectcell\fbox}l<{\endcollectcell}}%
\newcolumntype{M}{>{\collectcell\Meaning}l<{\endcollectcell}}%
\newcolumntype{T}{>{\collectcell\texttiming}l<{\endcollectcell}}%
\begin{tabular}{@{}F@{}|@{}M@{}|@{}T@{}}
   A & B                        & HLDZ 2{HZLZ} \\
   A & \empty\relax Z5D{TEST}Z & Z5D{TEST}Z  \\
   A & \cci\empty Z5D{TEST}Z & Z5D{TEST}Z  \\
   {\begin{tabular}{cFc} a & b & c \end{tabular}} &
    \relax\begin{quote}AA\end{quote}   & $5+5${C} \\
   A & B   \ccunskip B                  & 3{ttz} \\
\end{tabular}%
```

Example 1: Framebox, texttiming, expanded tokens, sub-tabular

M¿
l¡ F¿
l¡

Multi		single
A	B	C

```
\def\abc{ \empty A & \empty B & \empty C }
\begin{tabular}{MMM}
  \multicolumn{2}{M}{\empty Multi} & \empty single \\
  \abc \\
\end{tabular}
```

Example 2: Multicolumn, expanded row macro

F	F`F`	$AABABABCBBBCCABC$

```
\begin{tabular}{|F|F|F|}
    \\
    A & \\
    A & B \\
    A & B & \\
    A & B & C \\
    & \\
    & B \\
    & B & \\
    & B & C \\
    & & \\
    & & C \\
    A & B & C
\end{tabular}
```

Example 3: Empty cells, missing ‘\\’ at end

3 Implementation

```
\RequirePackage{array}
\def\collcell@beforeuser{\ignorespaces}
\def\collcell@afteruser{\unskip}

\newif\if@collcell@verb
\newif\if@collcell@robustcr
\@collcell@robustcrtrue
```

3.1 Options

```
\DeclareOption{verb}{\@collcell@verbtrue}
\DeclareOption{noverb}{\@collcell@verbfalse}
\DeclareOption{robustcr}{\@collcell@robustcrtrue}%
\DeclareOption{norobustcr}{\@collcell@robustcrfalse}%
\ProcessOptions\relax
\if@collcell@verb
  \RequirePackage{tabularx}
  \def\collcell@beforeuser{%
    \let\collcell@savedverb\verb
    \let\verb\TX@verb
    \let\TX@vwarn\collcell@vwarn
    \ignorespaces
  }%
  \def\collcell@afteruser{\unskip\let\verb\collcell@savedverb}%
  \def\collcell@vwarn{%
    \PackageWarning{collcell}{\noexpand\verb may be unreliable inside a collected cell}%
  }%
\fi
\if@collcell@robustcr
  \RequirePackage{etoolbox}
  \robustify\@arraycr
\fi
```

3.2 Collect cell content

```
\let\collect@cell@toks\@temptokena
\newcount\collect@cell@count
```

`\collectcell`

#1: user macro(s)
#2: ignored tokens, possible empty

```
\newenvironment{collectcell}{}{}
\def\collectcell#1#2\ignorespaces{%
  \begingroup
  \collect@cell@count\z@
  \collect@cell@toks{}%
  \let\collect@cell@spaces\empty
  \def\collect@cell@end{%
    \expandafter\endgroup
    \expandafter\collcell@beforeuser
    \expandafter\ccell@swap\expandafter{\the\collect@cell@toks}{#1}%
    \collcell@afteruser
  }%
  \collect@cell@look#2%
}
```

\ccell@swap

Swaps the two arguments. The second one (user macro(s)) is added without braces.

```
\def\ccell@swap#1#2{#2{#1}}
```

\endcollectcell

Holds unique signature which will expand to nothing.

```
\def\endcollectcell{\@gobble{endcollectcell}}
```

\collect@cell@look

Looks ahead to the next token and call the next macro to handle it.

```
\def\collect@cell@look{%
  \futurelet\collect@cell@lettoken\collect@cell@look@
}
```

\collect@cell@eatspace

Eats a following space and call the 'look' macro again.

```
\@firstofone{\def\collect@cell@eatspace} {\collect@cell@look}
```

`\collect@cell@look@`

Handles special tokens which should not be read as argument. All other are handled by `\collect@cell@arg`.

```
\def\collect@cell@look@{%
  \cc@case
    \@sptoken{%
      \edef\collect@cell@spaces{\collect@cell@spaces\space}%
      \collect@cell@eatspace
    }%
    \bgroup{\collect@cell@group}%
    \default{\collect@cell@arg}%
  \endcc@case
}
```

`\collect@cell@group`

Tests if the previous discovered `begin-group character {` token was a `\bgroup` or a `{`. In the first case the command sequence is simply added but in the second case the surrounding braces must be added again. The use of `\unexpanded` allows `#` in the cells, e.g. for in-cell macro definitions.

```
\def\collect@cell@group#1{%
  \begingroup
  \edef\@tempa{\unexpanded{#1}}%
  \def\@tempb{\bgroup}%
  \ifx\@tempa\@tempb
    \endgroup
    \collect@cell@addarg{#1}%
  \else
    \endgroup
    \collect@cell@addarg{{#1}}%
  \fi
  \collect@cell@look
}
```

`\collect@cell@addarg`

Adds the given argument to the token list.

```
\def\collect@cell@addarg#1{%
  \expandafter\expandafter\expandafter\collect@cell@toks
  \expandafter\expandafter\expandafter
  {\expandafter\the\expandafter\collect@cell@toks\collect@cell@spaces#1}%
  \let\collect@cell@spaces\empty
}
```

`\collect@cell@addcc`

This macro is called when another `\collectcell` is found in the preamble (at the moment also inside the cell). The argument of it is placed into the token register and all following tokens are placed in an own token list which content is then added with surrounding braces in the outer token list once the `\endcollectcell` is found. TeX scoping mechanism is used for this so only one token register is required.

```
\def\collect@cell@addcc#1{%
  \collect@cell@addarg{#1}%
  \begingroup
  \collect@cell@toks{}%
  \collect@cell@look
}
```

`\collect@cell@checkcsname`

For support of `\end{tabularx}` without trailing `\\`.

```
\def\collect@cell@checkcsname#1\endcsname{%
  \begingroup
  \expandafter\ccell@swap\expandafter
    {\expandafter,\@currenvir,endtabular,endtabular*,array,tabularx,}%
    {\in@{,#1,}}%
  \ifin@
    \endgroup
    \expandafter\@firstoftwo
  \else
    \endgroup
    \expandafter\@secondoftwo
  \fi
    {\collect@cell@cr\\\csname#1\endcsname}%
    {\collect@cell@addarg{\csname#1\endcsname}\collect@cell@look}%
}
```

`\collect@cell@checkend`

`#1`: The argument of an “end macro
Reads the argument of `\end` and checks if it is identical to the current environment (`tabular`, `array`, `tabularx`, ...). If so the collecting of token is ended, otherwise the `\end` and its argument are added to the

```
\def\collect@cell@checkend#1{%
  \begingroup
  \def\@tempa{#1}%
  \ifx\@tempa\@currenvir
    \endgroup
    \expandafter\@firstoftwo
  \else
    \endgroup
    \expandafter\@secondoftwo
  \fi
    {\collect@cell@cr\\\end{#1}}%
    {\collect@cell@addarg{\end{#1}}\collect@cell@look}%
}
```

`\cc@iftoken`

Compares the `\collect@cell@lettoken` with the token given as argument.

```
\def\cc@iftoken#1{%
  \ifx#1\collect@cell@lettoken
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
```

`\cc@case`

Case statement over `\collect@cell@lettoken`.

```
\def\cc@case{%
  \begingroup
  \let\default= \collect@cell@lettoken
  \cc@@case
}
\def\cc@@case#1{%
  \ifx#1\collect@cell@lettoken
```

```
    \expandafter\cc@@truecase
  \else
    \expandafter\cc@@falsecase
  \fi
}
\def\cc@@truecase#1#2\endcc@case{\endgroup#1}
\def\cc@@falsecase#1{\cc@@case}
```

`\collcell@unskip`

Wrapper around `\unskip` to protect it from the eyes of the token scanner. It is protected to avoid trouble if the user wrongly uses it at the beginning of the cell. The macro is first defined using `\newcommand` to warn the user about name collisions.

```
\newcommand*\ccunskip{}
\protected\def\ccunskip{\unskip}
```

`\cci`

Protected empty macro usable to stop the expansion of tokens at the beginning of the cell. It is ignored (gobbled) by the token scanner. The macro is first defined using `\newcommand` to warn the user about name collisions.

```
\newcommand*\cci{}
\protected\def\cci{}
```

`\collect@cell@cr`

Redefines the table line/row end macro `\cr` so that token collection is restarted after the real `\cr` is expanded and the end material defined by ‘¡‘ is inserted.

This redefinition must be done around some *dirty tricks* otherwise the `\cr` will be wrongly taken as end of the row.

Because the redefinition is done just at the end of a cell inside the group opened by `collcell` it will only be locally.

```
\def\collect@cell@cr{%
  \iffalse{\fi
  \let\collcell@realcr\cr
  \def\cr{%
    \expandafter
    \collect@cell@look
    \collcell@realcr
  }%
  \iffalse}\fi
}
```

`\collect@cell@arg`

Handles the arguments. The first token of the argument is still in the `lettoken` macro which is compared against a list of possible end tokens. Then either the cell end is handled or the argument is added to the token register and the rest of the cell is processed.

```
\def\collect@cell@arg#1{%
  \cc@case
    \\{\collect@cell@cr#1}%
    \end{\collect@cell@checkend}%
    \csname{\collect@cell@checkcsname}%
    \unskip{%
      \let\collect@cell@spaces\empty
      %\collect@cell@addarg{#1}% do not include the \unskip
      \collect@cell@look%
    }%
    \@sharp{%
      \expandafter\collect@cell@addarg\expandafter{#1}%
      \collect@cell@look
    }%
    \collectcell{%
      \advance\collect@cell@count by \@ne
      \collect@cell@addcc%
    }%
    \endcollectcell{%
      \ifnum\collect@cell@count=\z@
        \expandafter\collect@cell@end
      \else
        \expandafter\endgroup
        \expandafter\collect@cell@addarg\expandafter
        {\expandafter{\the\collect@cell@toks}}%
        \advance\collect@cell@count by \m@ne%
        \expandafter\collect@cell@look
      \fi
    }%
    \cci{%
      \collect@cell@look
    }%
    \default{%
      \expandafter\ccell@swap\expandafter
        {\csname endtabular*\endcsname\endtabular\endarray}{\in@{#1}}%
      \ifin@
```

```
          \expandafter\@firstoftwo
      \else
          \expandafter\@secondoftwo
      \fi
      {\collect@cell@cr\\#1}%
      {%
        \collect@cell@addarg{#1}%
        \collect@cell@look
      }%
    }%
 \endcc@case
}
```