An environment for multicolumn output*†

Frank Mittelbach
Email: see top of the source file

Printed October 31, 2025

Abstract

This article describes the use and the implementation of the multicols environment. This environment allows switching between one and multicolumn format on the same page. Footnotes are handled correctly (for the most part), but will be placed at the bottom of the page and not under each column. LaTeX's float mechanism, however, is partly disabled in this implementation. At the moment only page-wide floats (i.e., star-forms) can be used within the scope of the environment.

Preface to versions 1.9 + 2.0

Version 1.9 added tagging support and also a number of smaller enhancements, such as an optional argument to `\columnbreak` to allow for conditional breaks instead of forced ones. The min column depth was also made customizable (previously it was hardwired to the depth of "p") to support special cases and in particular languages that do not have characters with any noticeable depth such as, for example, Japanese.

Verson 2.0 then simplified and improved the mark handling, by fully supporting the new mark mechanism of LaTeX.

Preface to version 1.8

The 1.8 release improves on the balancing approach. If due to a limited number of break points (e.g., due to large objects) the balanced columns exceed the available vertical space, then balancing is canceled and a normal page is produced first. Some overflow is allowed (controlled by the parameter `\maxbalancingoverflow` which defaults to 12pt). This ensures that we only cut a normal page if we get enough material carried over to next page.

Also added was support for `\enlargethispage`. This means it is now possible to request a page to be artificially enlarged or shortened. Note that if you enlarge pages by more than one line you may have to increase the collectmore counter value to ensure that enough material is being picked up.

This command was used on the second page of this manual to shorten it by one line, in order to get rid of a number of widow lines on the following pages.

There are also some small enhancements to the balancing algorithm including a ways to require a minimum number of rows in the result.

Finally, version 1.8 adds the command `\docolaction` to help with more complicated actions that depend on the current col-

*This file has version number v2.0b, last revised 2025/10/21.

†Note: This package is released under terms which affect its use in commercial applications. Please see the details at the top of the source file.

umn. This command expects 3 arguments: code that is executed if we are in the "first" column, code to execute if we end up in any "middle" column (if there are more than two) and finally code to execute if we are in the "last" column. Thus

```
\docolaction{first}
                {middle}{last}
```

would typeset a different word depending the type of column this code is executed. Using it like this is probably pointless, but you can imagine applications like writing something into the nearest margin, etc.

As this feature needs at least two LaTeX runs to produce correct results and as it adds to the processing complexity it is only made available if one add the option `colaction` when loading the package.

Preface to version 1.7 (right to left support)

The 1.7 release adds support for languages that are typeset right-to-left. For those languages the order of the columns on the page also need to be reversed—something that wasn't supported before. The next paragraph demonstrates the result (as it is typeset as if we are writing in a left-to-right language—so read the rightmost column first). The change is initialized via `\RLmulticolcolumns` and returning to left-right (default) is done via `\LRmulticolcolumns`.

For example:

```
\renewcommand \footnoterule{%
  \kern-3pt\hbox to\textwidth
    {\hskip .6\textwidth
     \hrulefill }%
  \kern2.6pt}
```

directions within the columns. As footnotes are typeset in full measure the footnote rule needs to be redefined as if they are below a single column, i.e., using `\textwidth` not `\columnwidth`.

Right-to-left typesetting will only reverse the column orders. Any other support needed will have to be provided by other means, e.g., using appropriate fonts and reversing the writing

Preface to version 1.5 + 1.6

The 1.5 release contains two major changes: multicols will now support up to 10 columns and two more tuning possibilities have been added to the balancing routine. The balancing routine now checks the badness of the resulting columns and rejects solutions that are larger than a certain threshold. At the same time multicols has been upgraded to run under LaTeX 2_ε.

Later changes to 1.5 include `\columnbreak` and `multicols*`.

For version 1.6 micro-spacing around the boxes produced by multicols has been improved to allow for baseline-grid typesetting.

1 Introduction

Switching between two-column and one-column layout is possible in LaTeX, but every use of `\twocolumn` or `\onecolumn` starts a new page. Moreover, the last page of two-column output isn't balanced and this often results in an empty, or nearly empty, right column. When I started to write macros for doc.sty (see "The `doc`–Option", *TUGboat* volume 10 #2, pp. 245–273) I thought that it would be nice to place the index on the same page as the bibliography. And balancing the last page would not only look better, it also would save space; provided of course that it is also possible to start the next article on the same page. Rewriting the index environment was comparatively easy, but the next goal, designing an environment which takes care of footnotes, floats, etc., was a harder task. It took me a whole weekend[1] to get together the few lines of code below and there is still a good chance that I missed something after all.

Try it and, hopefully, enjoy it; and *please* direct bug reports and suggestions back to Mainz.

[1] I started with the algorithm given in the TeXbook on page 417. Without this help a weekend would not have been enough. (This remark was made in the documentation of the initial release, since then several hundreds more hours went into improving the original code.)

2 The User Interface

To use the environment one simply says

```
\begin{multicols}{⟨number⟩}
      ⟨multicolumn text⟩
\end{multicols}
```

where ⟨*number*⟩ is the required number of columns and ⟨*multicolumn text*⟩ may contain arbitrary LaTeX commands, except that floats and marginpars are not allowed in the current implementation[2].

As its first action, the multicols environment measures the current page to determine whether there is enough room for some portion of multicolumn output. This is controlled by the ⟨*dimen*⟩ variable `\premulticols` which can be changed by the user with ordinary LaTeX commands. If the space is less than `\premulticols`, a new page is started. Otherwise, a `\vskip` of `\multicolsep` is added.[3]

When the end of the multicols environment is encountered, an analogous mechanism is employed, but now we test whether there is a space larger than `\postmulticols` available. Again we add `\multicolsep` or start a new page.

It is often convenient to spread some text over all columns, just before the multicolumn output, without any page break in between. To achieve this the multicols environment has an optional second argument which can be used for this purpose. For example, the text you are now reading was started with

```
\begin{multicols}{3}
  [\section{The User
            Interface}] ...
```

If such text is unusually long (or short) the value of `\premulticols` might need adjusting to prevent a bad page break. We therefore provide a third argument which can be used to overwrite the default value of `\premulticols` just for this occasion. So if you want to combine some longer single column text with a multicols environment you could write

```
\begin{multicols}{3}
    [\section{Index}
     This index contains ...]
    [6cm]
 ...
```

The space between columns is controlled by the length parameter `\columnsep`. The width for the individual columns is automatically calculated from this parameter and the current `\linewidth`. In this article a value of 18.0pt was used.

Separation of columns with vertical rules is achieved by setting the parameter `\columnseprule` to some positive value. In this article a value of .4pt was used.

The color of the rules separating the columns can be specified through `\columnseprulecolor`. The default value is `\normalcolor`.

Since narrow columns tend to need adjustments in interline spacing we also provide a ⟨*skip*⟩ parameter called `\multicolbaselineskip` which is added to the `\baselineskip` parameter inside the multicols environment. Please use this parameter with care or leave it alone; it is intended only for package file designers since even small changes might produce totally unexpected changes to your document.

2.1 Balancing columns

Besides the previously mentioned parameters, some others are provided to influence the layout of the columns generated.

Paragraphing in TeX is controlled by several parameters. One of the most important is called `\tolerance`: this controls the allowed 'looseness' (i.e. the amount of blank space between words). Its default value is 200 (the LaTeX `\fussy`) which is too small for narrow columns.

Setting it to 10000 (a.k.a. ∞) means arbitrary bad lines are possible. With that setting LaTeX will make most lines perfect but intermix them with really bad lines. This was the setting originally used by `\sloppy` (nowadays it is a bit more cautious and used 9999 which makes a huge difference).[4]

We therefore use a `\multicoltolerance` parameter for the `\tolerance` value inside the multicols environment. Its default value is 9999 which is less than infinity but 'bad' enough for most paragraphs in a multicolumn environment. Changing its value should be done outside the multicols environment. Since `\tolerance` is set to `\multicoltolerance` at the beginning of every multicols environment one can locally overwrite this default by assigning `\tolerance=`⟨*desired value*⟩. There also exists a `\multicolpretolerance`

[2] This is dictated by lack of time. To implement floats one has to reimplement the whole LaTeX output routine.

[3] Actually the added space may be less because we use `\addvspace` (see the LaTeX manual for further information about this command).

[4] Look at the next paragraph, it was set with the `\tolerance=10000`.

parameter holding the value for \pretolerance within a multicols environment. Both parameters are usually used only by package or class designers.

Generation of multicolumn output can be divided into two parts. In the first part we are collecting material for a page, shipping it out, collecting material for the next page, and so on. As a second step, balancing will be done when the end of the multicols environment is reached. In the first step TEX might consider more material whilst finding the final column content than it actually uses when shipping out the page. This might cause a problem if a footnote is encountered in the part of the input considered, but not used, on the current page. In this case the footnote might show up on the current page, while the footnotemark corresponding to this footnote might be set on the next one.[5] Therefore the multicols environment gives a warning message[6] whenever it is unable to use all the material considered so far.

If you don't use footnotes too often the chances of something actually going wrong are very slim, but if this happens you can help TEX by using a \pagebreak command in the final document. Another way to influence the behavior of TEX in this respect is given by the counter variable 'collectmore'. If you use the \setcounter declaration to set this counter to ⟨*number*⟩, TEX will consider ⟨*number*⟩ more (or less) lines before making its final decision. So a value of −1 may solve all your problems at the cost of slightly less optimal columns.

In the second step (balancing columns) we have other bells and whistles. First of all you can say \raggedcolumns if you don't want the bottom lines to be aligned. The default is \flushcolumns, so TEX will normally try to make both the top and bottom baselines of all columns align.

If there is only a small amount of material available for balancing then you may end up with very few lines per column. In an extreme case there may be only one line which looks distinctly odd. In that case it might be better to have more material distributed to the earlier columns even if that means that later columns are empty or partially empty. This is controlled through the counter 'minrows' (default 1). If set to a higher value then the balancing will have at least that many rows in the first column (and also all further columns until it runs outs of material).

Additionally you can set another counter, the 'unbalance' counter, to some positive ⟨*number*⟩. This will make all but the right-most column ⟨*number*⟩ of lines longer than they would normally have been. 'Lines' in this context refer to normal text lines (i.e. one \baselineskip apart); thus, if your columns contain displays, for example, you may need a higher ⟨*number*⟩ to shift something from one column into another. A negative value can make sense if you have set minrows and want to locally adjust that.

Unlike 'collectmore,' the 'unbalance' counter is reset to zero at the end of the environment so it only applies to one multicols environment.

The two methods may be combined but I suggest using these features only when fine tuning important publications.

Two more general tuning possibilities were added with version 1.5. TEX allows to measure the badness of a column in terms of an integer value, where 0 means optimal and any higher value means a certain amount of extra white space. 10000 is considered to be infinitely bad (TEX does not distinguish any further). In addition the special value 100000 means overfull (i.e., the column contains more text than could possibly fit into it).

The new release now measures every generated column and ignores solutions where at least one column has a badness being larger than the value of the counter columnbadness. The default value for this counter is 10000, thus TEX will accept all solutions except those being overfull. By setting the counter to a smaller value you can force the algorithm to search for solutions that do not have columns with a lot of white space.

However, if the setting is too low, the algorithm may not find any acceptable solution at all and will then finally choose the extreme solution of placing all text into the first column.

Often, when columns are balanced, it is impossible to find a solution that distributes the text evenly over all columns. If that is the case the last column usually has less text than the others. In the earlier releases this text was stretched to produce a column with the same height as all others, sometimes resulting in really ugly looking columns.

In the new release this stretching is only done if the badness

[5] The reason behind this behavior is the asynchronous character of the TEX *page_builder*. However, this could be avoided by defining very complicated output routines which don't use TEX primitives like \insert but do everything by hand. This is clearly beyond the scope of a weekend problem.

[6] This message will be generated even if there are no footnotes in this part of the text.

of the final column is not larger than the value of the counter finalcolumnbadness. The default setting is 9999, thus preventing the stretching for all columns that TEX would consider infinitely bad. In that case the final column is allowed to run short which gives a much better result.

And there are two more parameters of some experimental nature, one called `\multicolovershoot` the other `\multicolundershoot`. They control the amount of space a column within the multicols environment is allowed to be "too full" or "too short" without affecting the column badness. They are set to 0pt and 2pt, respectively.

Finally, when doing the balancing at the end, columns may become higher than the remaining available space. In that case the algorithm aborts and instead generates a normal page. However, if the amount is not too large, e.g., a line or so, then it might be better to keep everything on the same page instead of starting a new page with just one line after balancing. So the parameter `\maxbalancingoverflow` governs this process: only when the excess gets larger than its value balancing is aborted.

2.2 Not balancing the columns

Although this package was written to solve the problem of balancing columns, I got repeated requests to provide a version where all white space is automatically placed in the last column or columns. Since version v1.5q this now exists: if you use `multicols*` instead of the usual environment the columns on the last page are not balanced. Of course, this environment only works on top-level, e.g., inside a box one has to balance to determine a column height in absence of a fixed value.

2.3 Manually breaking columns

Another request often voiced was: "How do I tell LATEX that it should break the first column after this particular line?". The `\pagebreak` command (which works with the two-column option of LATEX) is of no use here since it would end the collection phase of multicols and thus all columns on that page. So with version 1.5u the `\columnbreak` command was added. If used within a paragraph it marks the end of the current line as the desired breakpoint. You can observe its effect on the previous page where three lines of text have been artificially forced into the second column (resulting in some white space between paragraphs in the first column).

From version 1.9 onwards `\columnbreak` accepts an optional argument (just like `\pagebreak`) in which you can specify the desirability to break the column at that point: supported values are `0` (slightly desirable) to `4` (forced). This version also adds `\newcolumn` which forces a column break but runs the column short (comparable to `\newpage`).

2.4 Floats inside a multicols environment

Within the multicols environment the usual star float commands are available but their function is somewhat different as in the two-column mode of standard LATEX. Stared floats, e.g., `figure*`, denote page wide floats that are handled in a similar fashion as normal floats outside the multicols environment. However, they will never show up on the page where they are encountered. In other words, one can influence their placement by specifying a combination of `t`, `b`, and/or `p` in their optional argument, but `h` doesn't work because the first possible place is the top of the next page. One should also note, that this means that their placement behavior is determined by the values of `\topfraction`, etc. rather than by `\dbl...`.

2.5 Support for right-to-left typesetting

In right-to-left typesetting the order of the columns on the page also need to be reversed, i.e., the first column has to appear on the far right and the last column on the left. This is supported through the commands `\RLmulticolcolumns` (switching to right-to-left typesetting) and `\LRmulticolcolumns` (switching to left-to-right typesetting) the latter being the default.

2.6 Warnings

Under certain circumstances the use of the multicols environment may result in some warnings from TEX or LATEX. Here is a list of the important ones and the possible cause:

`Underfull \hbox (badness ...)`

As the columns are often very narrow TEX wasn't able to find a good way to break the paragraph. Underfull denotes a loose line but as long as the badness value is below 10000 the result is probably acceptable.

`Underfull \vbox ... while \output is active`

If a column contains a character with an unusual depth, for example a '(', in the bottom line

then this message may show up. It usually has no significance as long as the value is not more than a few points.

`LaTeX Warning: I moved some lines to the next page`

As mentioned above, multicols sometimes screws up the footnote numbering. As a precaution, whenever there is a footnote on a page where multicols had to leave a remainder for the following page this warning appears. Check the footnote numbering on this page. If it turns out that it is wrong, you have to manually break the page using `\newpage` or `\pagebreak[..]`.

`Floats and marginpars not allowed inside 'multicols' environment!`

This message appears if you try to use the `\marginpar` command or an unstarred version of the figure or table environment. Such floats will disappear!

`Very deep columns! Grid alignment might be broken`

This message can only appear if the option `grid` was chosen. In that case it will show up if a column has a very large depth so that multicols is unable to back up to its baseline. This is only relevant if one tries to produce a document where all text lines are aligned at an invisible grid, something that requires careful adjustment of many parameters and macros, e.g., heading definitions.

2.7 Tracing the output

To understand the reasoning behind the decisions TEX makes when processing a multicols environment, a tracing mechanism is provided. If you set the counter '`tracingmulticols`' to a positive ⟨*number*⟩ you then will get some tracing information on the terminal and in the transcript file:

⟨***number***⟩ = 1**.** TEX will now tell you, whenever it enters or leaves a multicols environment, the number of columns it is working on and its decision about starting a new page before or after the environment.

⟨***number***⟩ = 2**.** In this case you also get information from the balancing routine: the heights tried for the left and right-most columns, information about shrinking if the `\raggedcolumns` declaration is in force and the value of the '`unbalance`' counter if positive.

⟨***number***⟩ = 3**.** Setting ⟨*number*⟩ to this value will additionally trace the mark handling algorithm. It will show what marks are found, what marks are considered, etc. To fully understand this information you will probably have to read carefully through the implementation.

⟨***number***⟩ ≥ 4**.** Setting ⟨*number*⟩ to such a high value will additionally place an `\hrule` into your output, separating the part of text which had already been considered on the previous page from the rest. Clearly this setting should *not* be used for the final output. It will also activate even more debugging code for mark handling.

3 Prefaces to older versions

3.1 Preface to version 1.4

Beside fixing some bugs as mentioned in the multicol.bug file this new release enhances the multicols environment by allowing for balancing in arbitrary contexts. It is now, for example, possible to balance text within a multicols or a minipage as shown in 2 where a multicols environment within a quote environment was used. It is now even possible to nest multicols environments.

The only restriction to such inner multicols environments (nested, or within TEX's internal vertical mode) is that such variants will produce a box with the balanced material in it, so that they can not be broken across pages or columns.

Additionally I rewrote the algorithm for balancing so that it will now produce slightly better results.

I updated the source documentation but like to apologize in advance for some 'left over' parts that slipped through the revision.

A note to people who like to improve the balancing algorithm of multicols: The balancing routine is now placed into a single macro which is called `\balance@columns`. This means that one can easily try different balancing routines by rewriting this macro. The interface for it is explained in table 1. There are several improvements possible, one can think of integrating the `\badness` function of TEX3, define a faster algorithm for finding the right column height, etc. If somebody thinks he/she has an enhancement I would be pleased to learn about it. But please obey the copyright notice and don't change multicol.dtx directly!

The macro `\balance@columns` that contains the code for balancing gathered material is a macro without parameters. It assumes that the material for balancing is stored in the box `\mult@box` which is a `\vbox`. It also "knows" about all parameters set up by the multicols environment, like `\col@number`, etc. It can also assume that `\@colroom` is the still available space on the current page.

When it finishes it must return the individual columns in boxes suitable for further processing with `\page@sofar`. This means that the left column should be stored in box register `\mult@firstbox`, the next in register `\mult@firstbox` + 2, ..., only the last one as an exception in register `\mult@grightbox`.

Table 1: Interface description for `\balance@columns`

3.2 Preface to version 1.2

After the article about the multicols environment was published in *TUGboat* 10#3, I got numerous requests for these macros. However, I also got a changed version of my style file, together with a letter asking me if I would include the changes to get better paragraphing results in the case of narrow lines. The main differences to my original style option were additional parameters (like `\multicoladjdemerits` to be used for `\adjdemerits`, etc.) which would influence the line breaking algorithm.

But actually resetting such parameters to zero or even worse to a negative value won't give better line breaks inside the multicols environment. TEXs line breaking algorithm will only look at those possible line breaks which can be reached without a badness higher than the current value of `\tolerance` (or `\pretolerance` in the first pass). If this isn't possible, then, as a last resort, TEX will produce overfull boxes. All those (and only those) possible break points will be considered and finally the sequence which results in the fewest demerits will be chosen. This means that a value of −1000 for `\adjdemerits` instructs TEX to prefer visibly incompatible lines instead of producing better line breaks.

However, with TEX 3.0 it is possible to get decent line breaks even in small columns by setting `\emergencystretch` to an appropriate value. I implemented a version which is capable of running both in the old and the new TEX (actually it will simply ignore the new feature if it is not available). The calculation of `\emergencystretch` is probably incorrect. I made a few tests but of course one has to have much more experience with the new possibilities to achieve the maximum quality.

Version 1.1a had a nice 'feature': the penalty for using the forbidden floats was their ultimate removal from LATEXs `\@freelist` so that after a few `\marginpar`s inside the multicols environment floats where disabled forever. (Thanks to Chris Rowley for pointing this out.) I removed this misbehavior and at the same time decided to allow at least floats spanning all columns, e.g., generated by the `figure*` environment. You can see the new functionality in table 2 which was inserted at this very point. However single column floats are still forbidden and I don't think I will have time to tackle this problem in the near future. As an advice for all who want to try: wait for TEX 3.0. It has a few features which will make life much easier in multi-column surroundings. Nevertheless we are working here at the edge of TEXs capabilities, really perfect solutions would need a different approach than it was done in TEXs page builder.

The text below is nearly unchanged, I only added documentation at places where new code was added.

`\setemergencystretch`: This is a hook for people who like to play around. It is supposed to set the `\emergencystretch` ⟨*dimen*⟩ register provided in the new TEX 3.0. The first argument is the number of columns and the second one is the current `\hsize`. At the moment the default definition is `4pt` × `#1`, i.e. the `\hsize` isn't used at all. But maybe there are better formulae.

`\set@floatcmds`: This is the hook for the experts who like to implement a full float mechanism for the multicols environment. The `@` in the name should signal that this might not be easy.

Table 2: The new commands of multicol.sty version 1.2. Both commands might be removed if good solutions to these open problems are found. I hope that these commands will prevent that nearly identical style files derived from this one are floating around.

4 The Implementation

We are now switching to two-column output to show the abilities of this environment (and bad layout decisions).

4.1 The documentation driver file

The next bit of code contains the documentation driver file for TEX, i.e., the file that will produce the documentation you are currently reading. It will be extracted from this file by the `docstrip` program. Since this is the first code in this file one can produce the documentation simply by running LATEX on the `.dtx` file.

```
⟨*driver⟩
\documentclass{ltxdoc}
```

We use the `balancingshow` option when loading multicols so that full tracing is produced. This has to be done before the `doc` package is loaded, since `doc` otherwise requires multicols without any options.

```
\usepackage{multicol}[1999/05/25]
\usepackage{doc}
```

First we set up the page layout suitable for this article.

```
\setlength{\textwidth}{39pc}
\setlength{\textheight}{54pc}
\setlength{\parindent}{1em}
\setlength{\parskip}{0pt plus 1pt}
\setlength{\oddsidemargin}{0pc}
\setlength{\marginparwidth}{0pc}
\setlength{\topmargin}{-2.5pc}
\setlength{\headsep}{20pt}
\setlength{\columnsep}{1.5pc}
```

We want a rule between columns.

```
\setlength\columnseprule{.4pt}
```

We also want to ensure that a new multicols environment finds enough space at the bottom of the page.

```
\setlength\premulticols{6\baselineskip}
```

When balancing columns we disregard solutions that are too bad. Also, if the last column is too bad we typeset it without stretch.

```
\setcounter{columnbadness}{7000}
\setcounter{finalcolumnbadness}{7000}
```

The index is supposed to come out in four columns. And we don't show macro names in the margin.

```
\setcounter{IndexColumns}{4}
```

The following redefinitions have to be moved until after the preamble because version 3 of doc resets them after the preamble (this is tmp, because hypdoc is not yet integrated, but as we all know, tmp solutions have a tendency to survive for a long time...).

```
\AddToHook{begindocument}{%
  \let\DescribeMacro\SpecialUsageIndex
  \let\DescribeEnv\SpecialEnvIndex
  \renewcommand\PrintMacroName[1]{}%
}

\CodelineIndex
%\DisableCrossrefs          % Partial index
\RecordChanges              % Change log
```

Line numbers are very small for this article.

```
\renewcommand{\theCodelineNo}
  {\scriptsize\rm\arabic{CodelineNo}}
\settowidth\MacroIndent{\scriptsize\rm 00\ }

\begin{document}
  \typeout
   {****************************************
 ^^J* Expect some Under- and overfull boxes.
 ^^J****************************************}
   \DocInput{multicol.dtx}
\end{document}
⟨/driver⟩
```

4.2 Identification and option processing

We start by identifying the package. Since it makes use of features only available in LaTeX 2_ε we ensure that this format is available. (Now this is done earlier in the file.)

```
⟨*package⟩
% \NeedsTeXFormat{LaTeX2e}
% \ProvidesPackage{multicol}[..../../..
%    v... multicolumn formatting]
```

Next we declare options supported by multicols. Two-column mode and multicols do not work together so we warn about possible problems. However, since you can revert to `\onecolumn` in which case multicols does work, we don't make this an error.

```
\DeclareOption{twocolumn}
   {\PackageWarning{multicol}{May not work
             with the twocolumn option}}
```

Tracing is done using a counter. However it is also possible to invoke the tracing using the options declared below.

```
\newcount\c@tracingmulticols
\DeclareOption{errorshow}
    {\c@tracingmulticols\z@}
\DeclareOption{infoshow}
    {\c@tracingmulticols\@ne}
\DeclareOption{balancingshow}
    {\c@tracingmulticols\tw@}

\DeclareOption{markshow}
              {\c@tracingmulticols\thr@@
               \DebugMarksOn
              }
\DeclareOption{debugshow}
              {\c@tracingmulticols5\relax
               \DebugMarksOn
              }
```

The next option is intended for typesetting on a `\baselineskip` grid. Right now it doesn't do anything other than warning if it thinks that the grid got lost.

```
\let\mc@gridwarn\maxdimen
\DeclareOption{grid}
    {\def\mc@gridwarn{\@maxdepth}}
```

Next option enables the `\docolaction` command. As this changes the `.aux` file content this is not automatically enabled.

```
\DeclareOption{colaction}{%
  \def\mc@col@status@write{%
         \protected@write\@auxout{}%
              {\string\mc@col@status
              {\ifmc@firstcol 1\else 2\fi}}%
         \mc@firstcolfalse}%
  \def\mc@lastcol@status@write{%
         \protected@write\@auxout{}%
              {\string\mc@col@status{3}}}%
}
\let\mc@col@status@write\relax
\let\mc@lastcol@status@write\relax

\ProcessOptions
```

4.3 Starting and Ending the multicols Environment

As mentioned before, the multicols environment has one mandatory argument (the number of columns) and up to two optional ones. We start by reading the number of columns into the `\col@number` register.

```
\def\multicols#1{\col@number#1\relax
```

If the user forgot the argument, TeX will complain about a missing number at this point. The error recovery mechanism will then use zero, which isn't a good choice in this case. So we should now test whether everything is okay. The minimum is two columns at the moment.

```
  \ifnum\col@number<\tw@
     \PackageWarning{multicol}%
      {Using `\number\col@number'
       columns doesn't seem a good idea.^^J
       I therefore use two columns instead}%
     \col@number\tw@ \fi
```

We have only enough box registers for twenty columns, so we need to check that the user hasn't asked for more.

```
  \ifnum\col@number>20
     \PackageError{multicol}%
      {Too many columns}%
      {Current implementation doesn't
       support more than 20 columns.%
       \MessageBreak
       I therefore use 20 columns instead}%
     \col@number20 \fi
```

Within the environment we need a special version of the kernel `\@footnotetext` command since the original sets the the `\hsize` to `\columnwidth` which is not correct in the multicol environment. Here `\columnwidth` refers to the width of the individual column and the footnote should be in `\textwidth`. Since `\@footnotetext` has a different definition inside a minipage environment we do not redefine it directly. Instead we locally set `\columnwidth` to

\textwidth and call the original (current) definition stored in \orig@footnotetext. If the multicols environment is nested inside another multicols environment then the redefinition has already happened. So be better test for this situation. Otherwise, we will get a TEX stack overflow as this would generate a self-referencing definition.

```
    \ifx\@footnotetext\mult@footnotetext
    \else
      \let\orig@footnotetext\@footnotetext
      \let\@footnotetext\mult@footnotetext
    \fi
```

Now we can safely look for the optional arguments.

```
\@ifnextchar[\mult@cols{\mult@cols[]}}
```

```
\long\def\mult@footnotetext#1{\begingroup
        \columnwidth\textwidth
        \orig@footnotetext{#1}\endgroup}
```

The \mult@cols macro grabs the first optional argument (if any) and looks for the second one.

```
\def\mult@cols[#1]{\@ifnextchar[%
```

This argument should be a ⟨*dimen*⟩ denoting the minimum free space needed on the current page to start the environment. If the user didn't supply one, we use \premulticols as a default.

```
  {\mult@@cols{#1}}%
  {\mult@@cols{#1}[\premulticols]}}
```

After removing all arguments from the input we are able to start with \mult@@cols.

```
\def\mult@@cols#1[#2]{%
```

First thing we do is to decide whether or not this is an unbounded multicols environment, i.e. one that may split across pages, or one that has to be typeset into a box. If we are in TEX's "inner" mode (e.g., inside a box already) then we have a boxed version of multicols therefore we set the @boxedmulticols switch to true. The multicols should start in vertical mode. If we are not already there we now force it with \par since otherwise the test for "inner" mode wouldn't show if we are in a box.

```
  \par
  \ifinner \@boxedmulticolstrue
```

Otherwise we check \doublecol@number. This counter is zero outside a multicols environment but positive inside (this happens a little later on). In the second case we need to process the current multicols also in "boxed mode" and so change the switch accordingly.

```
  \else
    \ifnum \doublecol@number>\z@
      \@boxedmulticolstrue
    \fi
  \fi
```

Then we look to see if statistics are requested:

```
  \mult@info\z@
      {Starting environment with
       \the\col@number\space columns%
```

In boxed mode we add some more info.

```
        \if@boxedmulticols\MessageBreak
           (boxed mode)\fi
      }%
```

Then we measure the current page to see whether a useful portion of the multicolumn environment can be typeset. This routine might start a new page.

```
  \enough@room{#2}%
```

Now we output the first argument and produce vertical space above the columns. (Note that this argument corresponds to the first optional argument of the multicols environment.) For many releases this argument was typeset in a group to get a similar effect as \twocolumn[..] where the argument is also implicitly surrounded by braces. However, this conflicts with local changes done by things like sectioning commands (which account for the majority of commands used in that argument) messing up vertical spacing etc. later in the document so that from version v1.5q on this argument is again typeset at the outer level.

```
  #1\par\addvspace\multicolsep
```

When the last line of a paragraph had a positive depth then this depth normally taken into account by the baselineskip calculation for the next line. However, the columns produced by a following multicol are rigid and thus the distance from the baseline of a previous text line to the first line in a multicol would differ depending on the depth of the previous line. To account for this we add a negative space unless the depth is -1000pt which signals something special to TEXand is not supposed to be a real depth.

```
  \ifdim \prevdepth = -\@m\p@
  \else
```

The actual generation of this corrective space is a little bit more complicated as it doesn't make sense to always back up to the previous baseline (in case an object with a very large depth was placed there, e.g., a centered tabular). So we only back up to the extend that we are within the \baselineskip grid. We know that the box produced by multicols has \topskip at its top so that also needs to be taken into account.

```
    \@tempcnta\prevdepth
```

```
    \@tempcntb\baselineskip
    \divide\@tempcnta\@tempcntb
    \advance\@tempcnta\@ne
    \dimen@\prevdepth
    \advance\dimen@ -\@tempcnta\baselineskip
    \advance\dimen@ \topskip
    \kern-\dimen@
  \fi
```

We start a new grouping level to hide all subsequent changes (done in `\prepare@multicols` for example).

```
  \begingroup
    \prepare@multicols
```

If we are in boxed mode we now open a box to typeset all material from the multicols body into it, otherwise we simply go ahead.

```
    \if@boxedmulticols
      \setbox\mult@box\vbox\bgroup
                                \color@setgroup
```

We may have to reset some parameters at this point, perhaps `\@parboxrestore` would be the right action but I leave it for the moment.

```
    \fi
```

We finish by suppressing initial spaces.

```
    \ignorespaces}
```

Here is the switch and the box for "boxed" multicols code.

```
\newif\if@boxedmulticols
\@boxedmulticolsfalse
\newbox\mult@box
```

The `\enough@room` macro used above isn't perfect but works reasonably well in this context. We measure the free space on the current page by subtracting `\pagetotal` from `\pagegoal`. This isn't entirely correct since it doesn't take the 'shrinking' (i.e. `\pageshrink`) into account. The 'recent contribution list' might be nonempty so we start with `\par` and an explicit `\penalty`.[7] Actually, we use `\addpenalty` to ensure that a following `\addvspace` will 'see' the vertical space that might be present. The use of `\addpenalty` will have the effect that all items from the recent contributions will be moved to the main vertical list and the `\pagetotal` value will be updated correctly. However, the penalty will be placed in front of any dangling glue item with the result that the main vertical list may already be overfull even if TEX is not invoking the output routine.

```
\def\enough@room#1{%
```

Measuring makes only sense when we are not in "boxed mode" so the routine does nothing if the switch is true.

```
  \if@boxedmulticols\else
  \par
```

To empty the contribution list the first release contained a penalty zero but this had the result that `\addvspace` couldn't detect preceding glue. So this was changed to `\addpenalty`. But this turned out to be not enough as `\addpenalty` will not add a penalty when `@nobreak` is true. Therefore we force this switch locally to false. As a result there may be a break between preceding text and the start of a multicols environment, but this seems acceptable since there is the optional argument for exactly this reason.

```
  \bgroup\@nobreakfalse\addpenalty\z@\egroup
  \page@free \pagegoal
  \advance \page@free -\pagetotal
```

To be able to output the value we need to assign it to a register first since it might be a register (default) in which case we need to use `\the` or it might be a plain value in which case `\the` would be wrong.

```
    \@tempskipa#1\relax
```

Now we test whether tracing information is required:

```
  \mult@info\z@
      {Current page:\MessageBreak
       height=%
       \the\pagegoal: used \the\pagetotal
       \space -> free=\the\page@free
       \MessageBreak
       needed \the\@tempskipa
            \space(for #1)}%
```

Our last action is to force a page break if there isn't enough room left.

```
  \ifdim \page@free <#1\newpage \fi
\fi}
```

When preparing for multicolumn output several things must be done.

```
\def\prepare@multicols{%
```

We start saving the current `\@totalleftmargin` and then resetting the `\parshape` in case we are inside some list environment. The correct indentation for the multicols environment in such a case will be produced by moving the result to the right by `\multicol@leftmargin` later on. If we would use the value of `\@totalleftmargin` directly then lists inside the multicols environment could cause a shift of the output.

[7] See the documentation of `\endmulticols` for further details.

```
\multicol@leftmargin\@totalleftmargin
\@totalleftmargin\z@
\parshape\z@
```

We also set the register `\doublecol@number` for later use. This register should contain 2 × `\col@number`. This is also an indicator that we are within a multicols environment as mentioned above.

```
\doublecol@number\col@number
\multiply\doublecol@number\tw@
\advance\doublecol@number\mult@rightbox
\mc@prepare@mark@regions
\if@boxedmulticols
\else
```

We add an empty box to the main vertical list to ensure that we catch any insertions (held over or inserted at the top of the page). Otherwise it might happen that the `\eject` is discarded without calling the output routine. Inside the output routine we remove this box again. Again this code applies only if we are on the main vertical list and not within a box. However, it is not enough to turn off interline spacing, we also have to clear `\topskip` before adding this box, since `\topskip` is always inserted before the first box on a page which would leave us with an extra space of `\topskip` if multicols start on a fresh sheet.

```
  \nointerlineskip {\topskip\z@\null}%
  \output{%
    \global\setbox\partial@page\vbox
      {%
```

Now we have to make sure that we catch one special situation which may result in loss of text! If the user has a huge amount of vertical material within the first optional argument that is larger then `\premulticols` and we are near the bottom of the page then it can happen that not the `\eject` is triggering this special output routine but rather the overfull main vertical list. In that case we get another breakpoint through the `\eject` penalty. As a result this special output routine would be called twice and the contents of `\partial@page`, i.e. the material before the multicols environment gets lost. There are several solutions to avoid this problem, but for now we will simply detect this and inform the user that he/she has to enlarge the `\premulticols` by using a suitable value for the second argument.

```
⟨*check⟩
        \ifvoid\partial@page\else
          \PackageError{multicol}%
           {Error saving partial page}%
           {The part of the page before
            the multicols environment was
            nearly full with^^Jthe result
            that starting the environment
            will produce an overfull
            page.  Some^^Jtext may be lost!
            Please increase \premulticols
            either generally or for this%
            ^^Jenvironment by specifying a
            suitable value in the second
            optional argument to^^Jthe
            multicols environment.}
          \unvbox\partial@page
          \box\last@line
        \fi
⟨/check⟩
        \unvbox\@cclv
        \global\setbox\last@line\lastbox
      }%
  }\eject
```

The next thing to do is to assign a new value to `\vsize`. LaTeX maintains the free room on the page (i.e. the page height without the space for already contributed floats) in the register `\@colroom`. We must subtract the height of `\partial@page` to put the actual free room into this variable.

```
  \advance\@colroom-\ht\partial@page
```

Then we have to calculate the `\vsize` value to use during column assembly. `\set@mult@vsize` takes an argument which allows to make the setting local (`\relax`) or global (`\global`). The latter variant is used inside the output routine below. At this point here we have to make a local change to `\vsize` because we want to get the original value for `\vsize` restored in case this multicols environment ends on the same page where it has started.

```
  \set@mult@vsize\relax
```

Now we switch to a new `\output` routine which will be used to put the gathered column material together.

```
  \output{\multi@column@out}%
```

Finally we handle the footnote insertions. We have to multiply the magnification factor and the extra skip by the number of columns since each footnote reduces the space for every column (remember that we have page-wide footnotes). If, on the other hand, footnotes are typeset at the very end of the document, our scheme still works since `\count\footins` is zero then, so it will not change. To allow even further customization the setting of the `\footins` parameters is done in a separate macro.

```
  \init@mult@footins
```

For the same reason (page-wide footnotes), the ⟨*dimen*⟩ register controlling the maximum space used for footnotes isn't changed. Having done this,

we must reinsert all the footnotes which are already present (i.e. those encountered when the material saved in \partial@page was first processed). This will reduce the free space (i.e. \pagetotal) by the appropriate amount since we have changed the magnification factor, etc. above.

```
\reinsert@footnotes
```

Inside multicols a \clearpage is fairly useless as we aren't supporting floats. In fact, it can cause harm as it doesn't know about the \partial@page and may therefore result in making columns too long. So we change that to behave like \newpage but also check if there are any deferred floats. If so, perhaps the user tried to place them through that \clearpage (but that needs to be done before starting the multicols environment.

```
\def\clearpage{%
  \ifx\@deferlist\@empty\else
    \PackageError{multicol}%
     {Deferred floats not cleared}%
     {A \string\clearpage\space inside
      multicols acts like
      \string\newpage\space and doesn't
      clear floats.\MessageBreak
      Move it before the multicols
      environment if you need it.}%
  \fi
  \newpage}%
```

All the code above was only necessary for the unrestricted multicols version, i.e. the one that allows page breaks. If we are within a box there is no point in setting up special output routines or \vsize, etc.

```
\fi
```

But now we are coming to code that is necessary in all cases. We assign new values to \vbadness, \hbadness and \tolerance since it's rather hard for TEX to produce 'good' paragraphs within narrow columns.

```
\vbadness\@Mi \hbadness5000
\tolerance\multicoltolerance
```

Since nearly always the first pass will fail we ignore it completely telling TEX to hyphenate directly. In fact, we now use another register to keep the value for the multicol pre-tolerance, so that a designer may allow to use \pretolerance.

```
\pretolerance\multicolpretolerance
```

For use with the new TEX we set \emergencystretch to \col@number $\times$ $4pt$. However this is only a guess so at the moment this is done in a macro \setemergencystretch which gets the current \hsize and the number of columns as arguments. Therefore users are able to figure out their own formula.

```
\setemergencystretch\col@number\hsize
```

Another hook to allow people adding their own extensions without making a new package is \set@floatcmds which handles any redefinitions of LATEXs internal float commands to work with the multicols environment. At the moment it is only used to redefine \@dblfloat and \end@dblfloat.

```
\set@floatcmds
```

Additionally, we advance \baselineskip by \multicolbaselineskip to allow corrections for narrow columns.

```
\advance\baselineskip\multicolbaselineskip
```

The \hsize of the columns is given by the formula:

$$\frac{\texttt{\linewidth} - (\texttt{\col@number} - 1) \times \texttt{\columnsep}}{\texttt{\col@number}}$$

The formula above has changed from release to release. We now start with the current value of \linewidth so that the column width is properly calculated when we are inside a minipage or a list or some other environment. This will be achieved with:

```
\hsize\linewidth \advance\hsize\columnsep
\advance\hsize-\col@number\columnsep
\divide\hsize\col@number
```

We also set \linewidth and \columnwidth to \hsize In the past \columnwidth was left unchanged. This is inconsistent, but \columnwidth is used only by floats (which aren't allowed in their current implementation) and by the \footnote macro. Since we want page-wide footnotes[8] this simple trick saved us from rewriting the \footnote macros. However, some applications referred to \columnwidth as the "width of the current column" to typeset displays (the amsmath package, for example) and to allow the use of such applications together with multicol this is now changed.

Before we change \linewidth to the new value we record its old value in some register called \full@width. This value is used later on when we package all columns together.

```
\full@width\linewidth
\linewidth\hsize
\columnwidth\hsize
}
```

[8] I'm not sure that I really want page-wide footnotes. But balancing of the last page can only be achieved with this approach or with a multi-path algorithm which is complicated and slow. But it's a challenge to everybody to prove me wrong! Another possibility is to reimplement a small part of the *fire_up* procedure in TEX (the program). I think that this is the best solution if you are interested in complex page makeup, but it has the disadvantage that the resulting program cannot be called TEX thereafter.

This macro is used to set up the parameters associated with footnote floats. It can be redefined by applications that require different amount of spaces when typesetting footnotes.

```
\def\init@mult@footins{%
    \multiply\count\footins\col@number
    \multiply\skip \footins\col@number
}
```

Since we have to set `\col@umber` columns on one page, each with a height of `\@colroom`, we have to assign `\vsize` = `\col@number` × `\@colroom` in order to collect enough material before entering the `\output` routine again. In fact we have to add another (`\col@number` − 1) × (`\baselineskip` − `\topskip`) if you think about it.

```
\def\set@mult@vsize#1{%
    \vsize\@colroom
    \@tempdima\baselineskip
    \advance\@tempdima-\topskip
    \advance\vsize\@tempdima
    \vsize\col@number\vsize
    \advance\vsize-\@tempdima
```

But this might not be enough since we use `\vsplit` later to extract the columns from the gathered material. Therefore we add some 'extra lines,' one for each column plus a corrective action depending on the value of the '`collectmore`' counter. The final value is assigned globally if `#1` is `\global` because we want to use this macro later inside the output routine too.

```
    \advance\vsize\col@number\baselineskip
    #1\advance\vsize
        \c@collectmore\baselineskip}
```

Here is the dimen register we need for saving away the outer value of `\@totalleftmargin`.

```
\newdimen\multicol@leftmargin
```

In versions prior to 1.8r the balancing at the end of the environment was done by changing the output routine from `\multi@column@out` to `\balance@column@out`. As it turned out that this has a couple of issues when the last columns should not be balanced after all (for example because they contained several `\columnbreak` commands we now stay with one output routine for the environment and only signal that we reached the end of the environment by marking it with a special penalty that we can check for later.

```
\mathchardef\@Mvi=10006  % 10005 is
                         % \columnbreak
```

When the end of the multicols environment is sensed we have to balance the gathered material. Depending on whether or not we are inside a boxed multicol different things must happen. But first we end the current paragraph with a `\par` command.

```
\def\endmulticols{\par
  \if@boxedmulticols
```

In boxed mode we have to close the box in which we have gathered all material for the columns. But before we do this we need to remove any space at the end of the box as we don't want to use this in balancing. Because of the `\color@endgroup` this can't be done later in `\balance@columns` as the color command will hide it.

```
    \remove@discardable@items
    \color@endgroup\egroup
```

Now we call `\balance@columns` the routine that balances material stored in the box `\mult@box`.

```
    \balance@columns
```

After balancing the result has to be returned by the command `\page@sofar`. When the boxed multicol is returned to the page it can happen that it doesn't fit onto it and LaTeX therefore breaks earlier. The problem in that case is that during the generation `\hsize`, etc. got changed and this setting is still in effect right now, and if this boxed multicol is within, say, `multicols*` then its output routine gets very upset. We therefore delay returning the result by saving it in box for now until we have left the group below.

```
    \global\setbox\mc@boxedresult\vbox{%
```

We first update the mark structures and collect all marks that need reinsertion once multicols has finished. Then we output the boxed columns and finally we reinsert the marks.

```
      \mc@handle@marks@and@reinserts
          {in multicol (boxed mode)}%
      \page@sofar
      \mc@reinsert@marks
    }%
```

This finishes the code for the "boxed" case.

```
  \else
```

If there was a `\columnbreak` on the very last line all material will have been moved to the `\colbreak@box`. Thus the galley will be empty and no output routine gets called so that the text is lost. To avoid this problem (though unlikely) we check if the current galley is empty and the `\colbreak@box` contains text and if so return that to the galley. If the galley is non-empty any material in `\colbreak@box` is added in the output routine since it needs to be put in front.

```
    \ifdim\pagegoal=\maxdimen
      \ifvoid\colbreak@box\else
        \mult@info\@ne{Re-adding forced
```

```
                break(s) for splitting}%
      \unvbox\colbreak@box\fi
  \fi
```

If we are in an unrestricted multicols environment we end the current paragraph above with `\par` but this isn't sufficient since TEXs *page_builder* will not totally empty the contribution list.[9] Therefore we must also add an explicit `\penalty`. Now the contribution list will be emptied and, if its material doesn't all fit onto the current page then the output routine will be called before we change it. At this point we need to use `\penalty` not `\addpenalty` to ensure that a) the recent contributions are emptied and b) that the very last item on the main vertical list is a valid break point so that TEX breaks the page in case it is overfull.

```
  \penalty\z@
```

Now it's safe to call the output routine in order to balance the columns. We do this by calling it with a special penalty.

```
  \penalty-\@Mvi
```

If the multicols environment body was completely empty or if a multi-page multicols just ends at a page boundary we have the unusual case that the `\eject` will have no effect (since the main vertical list is empty)—thus no output routine is called at all. As a result the material preceding the multicols (stored in `\partial@page` will get lost if we don't put this back by hand.

```
  \ifvbox\partial@page
       \unvbox\partial@page\fi
  \fi
```

The output routine above will take care of the `\vsize` and reinsert the balanced columns, etc. But it can't reinsert the `\footnotes` because we first have to restore the `\footins` parameter since we are returning to one column mode. This will be done in the next line of code; we simply close the group started in `\multicols`.

To fix an obscure bug which is the result of the current definition of the `\begin ... \end` macros, we check that we are still (logically speaking) in the multicols environment. If, for example, we forget to close some environment inside the multicols environment, the following `\endgroup` would be incorrectly considered to be the closing of this environment.

```
\@checkend{multicols}%
\endgroup
```

We also set the 'unbalance' counter to its default. This is done globally since LATEX counters are always changed this way.[10]

```
\global\c@unbalance\z@
```

Now it's time to return any footnotes if we are in unrestricted mode. In boxed mode footnotes are kept inside, but in that case we have to first return the saved box to the page and then write another column status into the `.aux` file to support `\docolaction` in case we have nested environments.

```
\if@boxedmulticols
  \unvbox\mc@boxedresult
  \mc@col@status@write
\else
  \reinsert@footnotes
```

We also take a look at the amount of free space on the current page to see if it's time for a page break. The vertical space added thereafter will vanish if `\enough@room` starts a new page.

But there is one catch. If the `\end{multicols}` is at the top of which can happen if there is a break point just before it (such as end ending environment) which was chosen. In that case we would do the next page using the internal `\vsize` for multicol collection which is a disaster. So we better catch this case. Fortunately we can detect it by looking at `\pagegoal`.

```
  \ifdim \pagegoal=\maxdimen
    \global\vsize\@colroom
  \else
    \enough@room\postmulticols
  \fi
\fi
\addvspace\multicolsep
```

There is one more thing to do: the balanced result of the environment is supposed to have a `\prevdepth` of zero as we backed up by its real prevdepth within `\page@sofar`. However if the balancing happened in the output routine then TEX reverts to the `\prevdepth` that was current before the OR once the OR has finished. In short `\prevdepth` is something you can't set globally it is always local to the current list being built. Thus we need to set it back to zero here to avoid incorrect spacing.

```
\prevdepth\z@
```

If statistics are required we finally report that we have finished everything.

[9] This once caused a puzzling bug where some of the material was balanced twice, resulting in some overprints. The reason was the `\eject` which was placed at the end of the contribution list. Then the *page_builder* was called (an explicit `\penalty` will empty the contribution list), but the line with the `\eject` didn't fit onto the current page. It was then reconsidered after the output routine had ended, causing a second break after one line.

[10] Actually, we are still in a group started by the `\begin` macro, so `\global` must be used anyway.

```
\mult@info\z@
   {Ending environment
           \if@boxedmulticols
              \space(boxed mode)\fi
            }}
```

Let us end this section by allocating all the registers used so far.

```
\newcount\c@unbalance
\newcount\c@collectmore
```

In the new LaTeX release `\col@number` is already allocated by the kernel, so we don't allocate it again.

```
%\newcount\col@number
\newcount\doublecol@number
\newcount\multicoltolerance
\newcount\multicolpretolerance
\newdimen\full@width
\newdimen\page@free
\newdimen\premulticols
\newdimen\postmulticols
\newskip\multicolsep
\newskip\multicolbaselineskip
\newbox\partial@page
\newbox\last@line
\newbox\mc@boxedresult
```

And here are their default values:

```
\c@unbalance   = 0
\c@collectmore = 0
```

To allow checking whether some macro is used within the multicols environment the counter `\col@number` gets a default of 1 outside the environment.

```
%\col@number = 1
\multicoltolerance = 9999
\multicolpretolerance = -1
\premulticols = 50pt
\postmulticols= 20pt
\multicolsep = 12pt plus 4pt minus 3pt
\multicolbaselineskip=0pt
```

4.4 The output routines

We first start with some simple macros. When typesetting the page we save the columns either in the box registers 0, 2, 4,... (locally) or 1, 3, 5,... (globally). This is Plain TeX policy to avoid an overflow of the save stack.

Therefore we define a `\process@cols` macro to help us in using these registers in the output routines below. It has two arguments: the first one is a number; the second one is the processing information. It loops starting with `\count@=#1` (`\count@` is a scratch register defined in Plain TeX), processes argument `#2`, adds two to `\count@`, processes argument `#2` again, etc. until `\count@` is higher than `\doublecol@number`. It might be easier to understand it through an example, so we define it now and explain its usage afterwards.

```
\def\process@cols#1#2{\count@#1\relax
      \loop
⟨*debug⟩
      \typeout{Looking at box \the\count@}
⟨/debug⟩
       #2%
      \advance\count@\tw@
      \ifnum\count@<\doublecol@number
    \repeat}
```

We now define `\page@sofar` to give an example of the `\process@cols` macro. `\page@sofar` should output everything prepared by the balancing routine `\balance@columns`.

```
\def\page@sofar{%
```

`\balance@columns` prepares its output in the even numbered scratch box registers. Now we output the columns gathered assuming that they are saved in the box registers 2 (left column), 4 (second column), ... However, the last column (i.e. the rightmost) should be saved in box register 0.[11] First we ensure that the columns have equal width. We use `\process@cols` for this purpose, starting with `\count@ = \mult@rightbox`. Therefore `\count@` loops through `\mult@rightbox`, `\mult@rightbox` + 2,... (to `\doublecol@number`).

```
   \process@cols\mult@rightbox
```

We have to check if the box in question is void, because the operation `\wd`⟨*number*⟩ on a void box will *not* change its dimension (sigh).

```
      {\ifvoid\count@
         \setbox\count@\hbox to\hsize{}%
       \else
         \wd\count@\hsize
       \fi}%
```

Now we give some tracing information.

```
   \count@\col@number \advance\count@\m@ne
   \mult@info\z@
    {Column spec: \the\full@width\space = indent
                  + columns + sep =\MessageBreak
       \the\multicol@leftmargin\space
       + \the\col@number\space
```

[11] You will see the reason for this numbering when we look at the output routines `\multi@column@out` and `\balance@columns@out`.

```
        x \the\hsize\space
        + \the\count@\space
        x \the\columnsep
    }%
```

At this point we should always be in vertical mode.

```
\ifvmode\else\errmessage{Multicol Error}\fi
```

Now we put all columns together in an `\hbox` of width `\full@width` (shifting it by `\multicol@leftmargin` to the right so that it will be placed correctly if we are within a list environment) and separating the columns with a rule if desired.

The box containing the columns has a large height and thus will always result in using `\lineskip` if the normal `\baselineskip` calculations are used. We therefore better cancel that process.

```
    \nointerlineskip
```

As mentioned earlier we want to have the reference point of the box we put on the page being at the baseline of the last line of the columns but we also want to ensure that the box has no depth so that any following skip is automatically starting from that baseline. We achieve this by recording the depths of all columns and then finally backing up by the maximum. (perhaps a simpler method would be to assemble the box in a register and set the depth of that box to zero (not checked).

We need a global scratch register for this; using standard TEX conventions we choose `\dimen2` and initialize it with the depth of the character "p" since that is one of the depths that compete for the maximum.

```
    \setbox\z@\hbox{\multicolmindepthstring}\global\dimen\tw@\dp\z@
    \UseTaggingSocket{page@sofar}%
    \moveright\multicol@leftmargin
     \hbox to\full@width{%
```

If the document is written in a language that is typeset right-to-left then, of course, the multicol columns should be also typeset right-to-left. To support this we call `\mc@align@columns` which with execute different code depending on the typesetting direction.

```
        \mc@align@columns
```

The depths of the columns depend on their last lines. To ensure that we will always get a similar look as far as the rules are concerned we force the depth to be at least the depth of a letter 'p' or more exactly `\multicolmindepthstring` (which is what we set `\dimen2` to above).

```
        \rlap{\phantom \multicolmindepthstring}%
    }%
```

The processed material might consist of a last line with a descender in which case the `\prevdepth` will be non-zero. However, this material is getting reformatted now so that this value is likely to be wrong. We therefore normalize the situation by pretending that the depth is zero. However, if `\page@sofar` is being called inside the OR then setting `\prevdepth` here has no long-lasting effect, we therefore have to repeat this once we return to the main vertical list. Here we set it only for those cases where the command is used within a list and then followed by something else.

```
    \prevdepth\z@
```

Now after typesetting the box we back up to its baseline by using the value stored in `\dimen2` (which will hold the largest depth found on any column).

```
\kern-\dimen\tw@
```

However, in case one of the columns was unusually deep TEX may have tried some corrective actions in which case backing up by the saved value will not bring us back to the baseline. A good indication for this is a depth of `\@maxdepth` though it is not an absolute proof. If the option `grid` is used `\mc@gridwarn` will expand to this, otherwise to `\maxdimen` in which case this warning will not show up.

```
\ifdim\dimen\tw@ > \mc@gridwarn
  \PackageWarning{multicol}%
    {Very deep columns!\MessageBreak
     Grid alignment might be broken}%
\fi
}
```

The default minimum depth of each column corresponds to the depth of a 'p' in the current font. This makes sense for Latin-based languages and was hard-wired initially, but for Asian languages it is better to use a zero depth (and alternatively one might want to use the depth of a strut or a parentheses). So we now offer a way to adjust this while maintaining backward compatibility. Use `\renewcommand` to alter it.

```
\def\multicolmindepthstring{p}
```

By default the vertical rule between columns will be in `\normalcolor`.

```
\def\columnseprulecolor{\normalcolor}
```

Before we tackle the bigger output routines we define just one more macro which will help us to find our way through the mysteries later. `\reinsert@footnotes` will do what its name indicates: it reinserts the footnotes present in `\footinbox` so that they will be reprocessed by TEX's *page_builder*.

Instead of actually reinserting the footnotes we insert an empty footnote. This will trigger insertion mechanism as well and since the old footnotes are still in their box and we are on a fresh page `\skip footins` should be correctly taken into account.

```
\def\reinsert@footnotes{\ifvoid\footins\else
        \insert\footins{}\fi}
```

This curious definition is used as the space at the bottom of a column if we implement `\raggedcolumns`. Normally one only appends `\vfill` in that case but this is actually wrong for columns that are more or less full: by adding a glue at the bottom such a column doesn't have any depth any more but without it the material would be allowed a depth of `\@maxdepth`. So we allow shrinking by that amount. This only makes a difference if the box would otherwise become overfull and shrinking never exceeds the specified value, so we should be fine.

```
\def\vfilmaxdepth{\vskip \z@ \@plus .0001fil
                            \@minus \@maxdepth}
```

Now we can't postpone the difficulties any longer. The `\multi@column@out` routine will be called in two situations. Either the page is full (i.e., we have collected enough material to generate all the required columns) or a float or marginpar or a `\clearpage` is sensed. In the latter case the `\outputpenalty` is less than -10000, otherwise the penalty which triggered the output routine is higher. Therefore it's easy to distinguish both cases: we simply test this register.

```
\def\multi@column@out{%
  \ifnum\outputpenalty <-\@M
```

If this was a `\clearpage`, a float or a marginpar we call `\speci@ls`

```
    \speci@ls \else
```

otherwise we construct the final page. For the next block of code see comments in section 7.2.

```
    \ifvoid\colbreak@box\else
      \mult@info\@ne{Re-adding forced
                break(s) for splitting}%
      \setbox\@cclv\vbox{%
       \unvbox\colbreak@box
       \penalty-\@Mv
       \unvbox\@cclv}%
    \fi
```

Let us now consider the normal case. We have to `\vsplit` the columns from the accumulated material in box 255. Therefore we first assign appropriate values to `\splittopskip` and `\splitmaxdepth`.

```
    \splittopskip\topskip
    \splitmaxdepth\@maxdepth
```

We also need to restrict `\boxmaxdepth` so that reboxing is not generating boxes with arbitrary depth.

```
    \boxmaxdepth\@maxdepth
```

Then we calculate the current column height (in `\dimen@`). Note that the height of `\partial@page` is already subtracted from `\@colroom` so we can use its value as a starter.

```
    \dimen@\@colroom
```

But we must also subtract the space occupied by footnotes on the current page. Note that we first have to reset the skip register to its normal value. Again, the actual action is carried out in a utility macro, so that other applications can modify it.

```
    \divide\skip\footins\col@number
    \ifvoid\footins \else
       \leave@mult@footins
    \fi
```

And there is one more adjustment that we have to make: if the user has issue a `\enlargethispage` command then the height the `\@kludgeins` box will be the negation of the size by which the page should be enlarged. If the star form of this command has been used then we also need to shrink the resulting column.

That local change will be reverted at the end of the output routine So for the next page the original state will be reestablished. However, in theory there is a possibility to sneak in a whole multicols environment into the running header definition. If that happens then it will also be affected by this change—too bad I think.

```
    \ifvbox \@kludgeins
      \advance \dimen@ -\ht\@kludgeins
```

The star form of `\enlargethispage` makes the width of the box greater than zero (sneaky isn't it?).

```
      \ifdim \wd\@kludgeins>\z@
         \shr@nkingtrue
      \fi
    \fi
```

Now we are able to `\vsplit` off all but the last column. Recall that these columns should be saved in the box registers 2, 4,... (plus offset).

```
    \process@cols\mult@firstbox{%
         \setbox\count@
             \vsplit\@cclv to\dimen@
```

If `\raggedcolumns` is in force we add a `vfill` at the bottom by unboxing the split box. But we need to unbox anyway to ensure that at the end of the box we do not have unwanted space. This can sneak in, in certain situations, for example, if two lists follow each other and we break between them. While such space is usually zero it still has an effect because it

hides depth of the last line in the column and that will result in incorrect placement.

```
      \setbox\count@
           \vbox to\dimen@
            {\unvbox\count@
             \ifshr@nking
              \vfilmaxdepth\fi}%
}%
```

Then the last column follows.

```
\setbox\mult@rightbox
    \vsplit\@cclv to\dimen@
\setbox\mult@rightbox\vbox to\dimen@
       {\unvbox\mult@rightbox
        \ifshr@nking\vfilmaxdepth\fi}%
```

Having done this we hope that box 255 is emptied. If not, we reinsert its contents.

```
\ifvoid\@cclv \else
    \unvbox\@cclv
    \ifnum\outputpenalty=\@M
    \else
       \penalty\outputpenalty
    \fi
```

In this case a footnote that happens to fall into the leftover bit will be typeset on the wrong page. Therefore we warn the user if the current page contains footnotes. The older versions of multicols produced this warning regardless of whether or not footnotes were present, resulting in many unnecessary warnings.

```
      \ifvoid\footins\else
        \PackageWarning{multicol}%
         {I moved some lines to
          the next page.\MessageBreak
          Footnotes on page
          \thepage\space might be wrong}%
      \fi
```

If the 'tracingmulticols' counter is 4 or higher we also add a rule.

```
      \ifnum \c@tracingmulticols>\thr@@
                    \hrule\allowbreak \fi
\fi
```

With a little more effort we could have done better. If we had, for example, recorded the shrinkage of the material in `\partial@page` it would be now possible to try higher values for `\dimen@` (i.e. the column height) to overcome the problem with the nonempty box 255. But this would make the code even more complex so I skipped it in the current implementation.

Now we use LaTeX's standard output mechanism.[12] Admittedly this is a funny way to do it.

Within the OR `\boxmaxdepth` needs to be unrestricted so we set it back now as it was changed above.

```
\boxmaxdepth\maxdimen
\setbox\@cclv\vbox
   {%
```

If we make a page while still inside the multicols environment we have to handle column and page mark structures.

```
      \mc@handle@col@andpage@marks
         {in multicol OR (full page)}%
    \unvbox\partial@page
    \page@sofar
    }%
```

The macro `\@makecol` adds all floats assigned for the current page to this page. `\@outputpage` ships out the resulting box. Note that it is just possible that such floats are present even if we do not allow any inside a multicols environment.

```
\@makecol\@outputpage
```

Now we reset `\@colroom` to `\@colht` which is LaTeX's saved value of `\textheight`. We also have to reset the recorded position of the last `\marginpar` as well as the recorded size of in-text floats as we are now on a new page.

```
\global\@colroom\@colht
\global \@mparbottom \z@
\global \@textfloatsheight \z@
```

Then we process deferred floats waiting for their chance to be placed on the next page.

```
\process@deferreds
\@whilesw\if@fcolmade\fi{\@outputpage
   \global\@colroom\@colht
   \process@deferreds}%
```

If the user is interested in statistics we inform him about the amount of space reserved for floats.

```
\mult@info\@ne
  {Colroom:\MessageBreak
   \the\@colht\space
           after float space removed
           = \the\@colroom \@gobble}%
```

Having done all this we must prepare to tackle the next page. Therefore we assign a new value to `\vsize`. New, because `\partial@page` is now empty and `\@colroom` might be reduced by the space reserved for floats.

```
 \set@mult@vsize \global
```

The `\footins` skip register will be adjusted when the output group is closed.

```
\fi}
```

[12] This will produce a lot of overhead since both output routines are held in memory. The correct solution would be to redesign the whole output routine used in LaTeX.

This macro is used to subtract the amount of space occupied by footnotes for the current space from the space available for the current column. The space current column is stored in `\dimen@`. See above for the description of the default action.

```
\def\leave@mult@footins{%
   \advance\dimen@-\skip\footins
   \advance\dimen@-\ht\footins
}
```

We left out two macros: `\process@deferreds` and `\speci@ls`.

```
\def\speci@ls{%
 \ifnum\outputpenalty <-\@Mi
```

If the document ends in the middle of a multicols environment, e.g., if the user forgot the `\end{multicols}`, TEX adds a very negative penalty to the end of the galley which is intended to signal the output routine that it is time to prepare for shipping out everything remaining. Since inside multicols the output routine of LATEX is disabled sometimes we better check for this case: if we find a very negative penalty we produce an error message and run the default output routine for this case.

```
  \ifnum \outputpenalty<-\@MM
   \PackageError{multicol}{Document end
           inside multicols environment}\@ehd
   \@specialoutput
  \else
```

For the next block of code see comments in section 7.2.

```
    \ifnum\outputpenalty = -\@Mv
        \mult@info\@ne{Forced column
                       break seen}%
        \global\advance\vsize-\pagetotal
        \global\setbox\colbreak@box
          \vbox{%
             \ifvoid\colbreak@box
             \else
               \unvbox\colbreak@box
               \penalty-\@Mv
             \fi
```

As this is the place of a forced break we now remove vertical white space just in front of it (or some of it at least) as it is quite likely that the break is not exactly in the right place, e.g., after a display environment (if LaTeX would break here by its own it would break before the space following the display).

Thus we rebox box 255 once (using `\@maxdepth` and calling `\remove@discardable@items` inside). The depth of 255 will then give us the depth the box would have had if it would have been a natural break. We then unbox 255 to get it into the `\colbreak@box` and then back up by this depth. This will position the bottom of the box at its natural baseline which is useful for balancing later on.

```
             \boxmaxdepth\@maxdepth
             \setbox\@cclv\vbox{%
                \unvbox\@cclv
                \remove@discardable@items}%
             \dimen@\dp\@cclv
             \unvbox\@cclv
             \kern-\dimen@
        }%
        \reinsert@footnotes
    \else
```

Another special case is reaching the end of the multicols environment which is signaled by `-\@Mvi`.

```
      \ifnum\outputpenalty = -\@Mvi
        \mult@info\@ne{End penalty of
                       multicols seen}%
```

If we are at this point then we have to run the balancing code (which was previously its own output routine). First we pretend that we had a normal forced breakpoint and then call `\balance@column@out`. The latter may be let to `\multi@column@out` if we are inside `multicols*` in which case we would get a loop if the `\outputpenalty` is not changed—this could be cleaned up in a better way; basically it is like this, because of the older code was using different ORs and I simply reused most of it.

```
        \outputpenalty\@M % pretend we had
                          % a natural
                          % forced break
        \balance@columns@out
      \else
```

If we encounter a float or a marginpar in the current implementation we simply warn the user that this is not allowed. Then we reinsert the page and its footnotes.

```
        \PackageWarningNoLine{multicol}%
           {Floats and marginpars not
            allowed inside 'multicols'
            environment!}%
        \unvbox\@cclv\reinsert@footnotes
```

Additionally we empty the `\@currlist` to avoid later error messages when the LATEX output routine is again in force. But first we have to place the boxes back onto the `\@freelist`. (`\@elts` default is `\relax` so this is possible with `\xdef`.)

```
        \xdef\@freelist{\@freelist
                        \@currlist}%
        \gdef\@currlist{}%
      \fi
    \fi
  \fi
```

If the penalty is −10001 it will come from a \clearpage and we will execute \@doclearpage to get rid of any deferred floats.

```
\else \@doclearpage \fi
}
```

\process@deferreds is a simplified version of LaTeX's \@startpage. We first call the macro \@floatplacement to save the current user parameters in internal registers. Then we start a new group and save the \@deferlist temporarily in the macro \@tempb.

```
\def\process@deferreds{%
   \@floatplacement
   \@tryfcolumn\@deferlist
   \if@fcolmade\else
     \begingroup
    \let\@tempb\@deferlist
```

Our next action is to (globally) empty \@deferlist and assign a new meaning to \@elt. Here \@scolelt is a macro that looks at the boxes in a list to decide whether they should be placed on the next page (i.e. on \@toplist or \@botlist) or should wait for further processing.

```
      \gdef\@deferlist{}%
      \let\@elt\@scolelt
```

Now we call \@tempb which has the form

\@elt⟨*box register*⟩\@elt⟨*box register*⟩...

So \@elt (i.e. \@scolelt) will distribute the boxes to the three lists.

```
        \@tempb \endgroup
   \fi}
```

The \raggedcolumns and \flushcolumns declarations are defined with the help of a new \if... macro.

```
\newif\ifshr@nking
```

The actual definitions are simple: we just switch to true or false depending on the desired action. To avoid extra spaces in the output we enclose these changes in \@bsphack...\@esphack.

```
\def\raggedcolumns{%
   \@bsphack\shr@nkingtrue\@esphack}
\def\flushcolumns{%
   \@bsphack\shr@nkingfalse\@esphack}
```

Now for the last part of the show: the column balancing output routine. Since this code is called with an explicit penalty (\eject) there is no need to check for something special (eg floats). We start by balancing the material gathered.

```
\def\balance@columns@out{%
```

For this we need to put the contents of box 255 into \mult@box. For the next block of code see also comments in section 7.2. All forced breaks except the last are inside \colbreak@box so all we have to do is to concatenate this box with box \@cclv and put a penalty in between. Here we test if \colbreak@box is void so that the message is only generated if we really add forced breaks and the penalty.

```
   \setbox\mult@box\vbox{%
      \ifvoid\colbreak@box\else
        \unvbox\colbreak@box
        \penalty-\@Mv
        \mult@info\@ne{Re-adding
          forced break(s) in balancing}%
      \fi
      \unvbox\@cclv
```

The last column again is a forced break, so here we discard white space as well as that is normally unwanted.

```
      \remove@discardable@items
   }%
   \balance@columns
```

If during balancing the columns got too long the flag \iftoo@bad is set to true.

```
   \iftoo@bad
     \mult@info\@ne
        {Balancing failed ...
         cut a normal page}%
```

In that case we put the material back in box 255 so that we can cut a normal page. The curious set of \vskips we add is necessary to cancel out the \splittopskip that got added for balancing.

```
     \setbox\@cclv\vbox
        {\vskip\topskip
         \vskip-\splittopskip
         \unvbox\mult@box
```

We also have to re-add the end of environment penalty since after this page we may want balance the remaining material.

```
         \penalty-\@Mvi
        }%
```

We then call the standard multicol output routine which will produce a normal page for us (remember we are still within the OR so some part of the code in \multi@column@out is actually not doing anything—perhaps this should be cleaned up at some point). This also means that if there was an \enlargethispage present it will apply to this page as \multi@column@out will look at the status of \@kludgeins.

```
     \multi@column@out
```

Because balancing made the columns too long we are sure that there will be some material remaining which was put back onto the main vertical list by `\multi@column@out`. This will also put the explicit `\eject` penalty back so the current `\balance@columns@out` output routine will be called again (so we better do not add another penalty or else the OR will be called twice and we may get scrambled results).

```
\else
```

If the balancing went ok, we are in the position to apply `\page@sofar`. But first we have to set `\vsize` to a value suitable for one column output.

```
    \global\vsize\@colroom
    \global\advance\vsize\ht\partial@page
```

We also have to look at `\@kludgeins` and generate a new `\insert` in case there was one present due to an `\enlargethispage` command.

```
    \ifvbox\@kludgeins
       \insert\@kludgeins
              {\unvbox\@kludgeins}\fi
```

Then we `\unvbox` the `\partial@page` (which may be void if we are not processing the first page of this multicols environment.

```
    \unvbox\partial@page
```

We then handle mark structures of the columns, return the gathered material to the main vertical list and then also reinsert the first and last marks that have been found in the columns.

```
    \mc@handle@marks@and@reinserts
        {in multicol OR (balancing)}%
    \page@sofar
    \mc@reinsert@marks
```

We need to add a penalty at this point which allows to break at this point since calling the output routine may have removed the only permissible break point thereby "glueing" any following skip to the balanced box. In case there are any weird settings for `\multicolsep` etc. this could produce funny results.

```
    \penalty\z@
  \fi
}
```

As we already know, reinserting of footnotes will be done in the macro `\endmulticols`.

This macro now does the actual balancing.

```
\def\balance@columns{%
```

We start by adding a forced break point at the very beginning, so that we can split the box to height zero later on, thereby adding a known `\splittopskip` glue at the beginning.

```
  \setbox\mult@box\vbox{%
       \penalty-\@M
       \unvbox\mult@box
       }%
```

Then follow values assignments to get the `\vsplit`ting right. We use the natural part of `\topskip` as the natural part for `\splittopskip` and allow for a bit of undershoot and overshoot by adding some stretch and shrink.

```
  \@tempdima\topskip
  \splittopskip\@tempdima
      \@plus\multicolundershoot
      \@minus\multicolovershoot
  \splitmaxdepth\@maxdepth
```

We also have to set `\boxmaxdepth` which normally allows to build boxes with arbitrary depth, but as we are building text columns we really want to restrict the depth. This is necessary as we sometimes rebox the boxes generated by `\vsplit` and then the restriction posed by `\splitmaxdepth` gets lost.

```
  \boxmaxdepth\@maxdepth
```

The next step is a bit tricky: when TEX assembles material in a box, the first line isn't preceded by interline glue, i.e. there is no parameter like `\boxtopskip` in TEX. This means that the baseline of the first line in our box is at some unpredictable point depending on the height of the largest character in this line. But of course we want all columns to align properly at the baselines of their first lines. For this reason we have opened `\mult@box` with a `\penalty -10000`. This will now allow us to split off from `\mult@box` a tiny bit (in fact nothing since the first possible break-point is the first item in the box). The result is that `\splittopskip` is inserted at the top of `\mult@box` which is exactly what we like to achieve.

```
  \setbox\@tempboxa\vsplit\mult@box to\z@
```

Next we try to find a suitable starting point for the calculation of the column height. It should be less than the height finally chosen, but large enough to reach this final value in only a few iterations. The formula which is now implemented will try to start with the nearest value which is a multiple of `\baselineskip`. The coding is slightly tricky in TEX and there are perhaps better ways ...

```
  \@tempdima\ht\mult@box
  \advance\@tempdima\dp\mult@box
  \divide\@tempdima\col@number
```

The code above sets `\@tempdima` to the length of a column if we simply divide the whole box into equal pieces. To get to the next lower multiple of `\baselineskip` we convert this dimen to a number (the number of scaled points) then divide this

by \baselineskip (also in scaled points) and then multiply this result with \baselineskip assigning the result to \dimen@. This makes \dimen@ $\leq$ to \@tempdimena.

```
\count@\@tempdima
\divide\count@\baselineskip
\dimen@\count@\baselineskip
```

Next step is to correct our result by taking into account the difference between \topskip and \baselineskip. We start by adding \topskip; if this makes the result too large then we have to subtract one \baselineskip.

```
\advance\dimen@\topskip
\ifdim \dimen@ >\@tempdima
  \advance\dimen@-\baselineskip
\fi
```

As a further restriction we want to see a minimum number of rows in the balanced result based on the setting of the counter minrows. If the starting value is lower we adjust.

```
\@tempdima\dimexpr
   \topskip +\c@minrows\baselineskip
            -\baselineskip\relax
\ifnum\dimen@<\@tempdima
  \mult@info\@ne
    {Start value
       \the\dimen@  \space ->
       \the\@tempdima \space
       (corrected for minrows)}%
  \dimen@\@tempdima
\fi
```

At the user's request we start with a higher value (or lower, but this usually only increases the number of tries).

```
\advance\dimen@\c@unbalance\baselineskip
```

We type out statistics if we were asked to do so.

```
\mult@info\@ne
   {Balance columns\on@line:
     \ifnum\c@unbalance=\z@\else
    (off balance=\number\c@unbalance)\fi
    \@gobbletwo}%
```

But we don't allow nonsense values for a start.

```
\ifnum\dimen@<\topskip
  \mult@info\@ne
    {Start value
       \the\dimen@  \space ->
       \the\topskip \space (corrected)}%
  \dimen@\topskip
\fi
```

Now we try to find the final column height. We start by setting \vbadness to infinity (i.e. 10000) to suppress underfull box reports while we are trying to find an acceptable solution. We do not need to do it in a group since at the end of the output routine everything will be restored. The setting of the final columns will nearly always produce underfull boxes with badness 10000 so there is no point in warning the user about it.

```
\vbadness\@M
```

We also allow for overfull boxes while we trying to split the columns. They can easily happen if we have objects with unusual depth.

```
\vfuzz \maxdimen
```

The variable \last@try will hold the dimension used in the previous trial splitting. We initialize it with a negative value.

```
\last@try-\p@
\loop
```

In order not to clutter up TEX's valuable main memory with things that are no longer needed, we empty all globally used box registers. This is necessary if we return to this point after an unsuccessful trial. We use \process@cols for this purpose, starting with \mult@grightbox. Note the extra braces around this macro call. They are needed since PLAIN TEX's \loop...\repeat mechanism cannot be nested on the same level of grouping.

```
  {\process@cols\mult@grightbox
        {\global\setbox\count@
                \box\voidb@x}}%
```

The contents of box \mult@box are now copied globally to box \mult@grightbox. (This will be the right-most column, as we shall see later.)

```
  \global\setbox\mult@grightbox
        \copy\mult@box
```

We start with the assumption that the trial will be successful. If we end up with a solution that is too bad we set too@bad to true. We also assume that all forced breaks (if any) will be used during balancing. If this is not the case we record this in forcedbreak@leftover.

```
⟨*badness⟩
  \too@badfalse
  \forcedbreak@leftoverfalse
⟨/badness⟩
```

Using \vsplit we extract the other columns from box register \mult@grightbox. This leaves box register \mult@box untouched so that we can start over again if this trial was unsuccessful.

```
  {\process@cols\mult@gfirstbox{%
        \global\setbox\count@
        \vsplit\mult@grightbox to\dimen@
```

After splitting we need to ensure that there isn't any space at the bottom, so we rebox once more.

```
        \global\setbox\count@
                \vbox to\dimen@
                 {\unvbox\count@}%
```

After every split we check the badness of the resulting column, normally the amount of extra white in the column.

```
⟨*badness⟩
        \ifnum\c@tracingmulticols>\@ne
          \@tempcnta\count@
          \advance\@tempcnta-\mult@grightbox
          \divide\@tempcnta \tw@
          \message{^^JColumn
             \number\@tempcnta\space
              badness: \the\badness\space}%
        \fi
```

If this badness is larger than the allowed column badness we reject this solution by setting `too@bad` to `true`.

```
        \ifnum\badness>\c@columnbadness
          \ifnum\c@tracingmulticols>\@ne
            \message{too bad
                 (>\the\c@columnbadness)}%
          \fi
          \too@badtrue
        \fi
⟨/badness⟩
                              }}%
```

There is one subtle point here: while all other constructed boxes have a depth that is determined by `\splitmaxdepth` and/or `\boxmaxdepth` the last box will get a natural depth disregarding the original setting and the value of `\splitmaxdepth` or `\boxmaxdepth`. This means that we may end up with a very large depth in box `\mult@grightbox` which would make the result of the testing incorrect. So we change the value by unboxing the box into itself.

```
    \global\setbox\mult@grightbox
       \vbox{\unvbox\mult@grightbox}%
```

We also save a copy `\mult@gfirstbox` at its "natural" size for later use.

```
    \setbox\mult@nat@firstbox
       \vbox{\unvcopy\mult@gfirstbox}%
```

After `\process@cols` has done its job we have the following situation:

$$\begin{aligned}
\text{box } \texttt{\mult@rightbox} &\longleftarrow \text{all material}\\
\text{box } \texttt{\mult@firstbox} &\longleftarrow \text{first column}\\
\text{box } \texttt{\mult@firstbox} + 2 &\longleftarrow \text{second column}\\
\vdots \qquad & \qquad \vdots\\
\text{box } \texttt{\mult@grightbox} &\longleftarrow \text{last column}
\end{aligned}$$

We report the height of the first column, in brackets the natural size is given.

```
    \ifnum\c@tracingmulticols>\@ne
      \message{^^JFirst column
          = \the\dimen@\space
          (\the\ht\mult@nat@firstbox)}\fi
```

If `\raggedcolumns` is in force older releases of this file also shrank the first column to its natural height at this point. This was done so that the first column doesn't run short compared to later columns but it is actually producing incorrect results (overprinting of text) in boundary cases, so since version v1.5q `\raggedcolumns` means allows for all columns to run slightly short.

```
%    \ifshr@nking
%      \global\setbox\mult@gfirstbox
%            \copy\mult@nat@firstbox
%    \fi
```

Then we give information about the last column.[13]

```
    \ifnum\c@tracingmulticols>\@ne
      \message{<> last column =
               \the\ht\mult@grightbox^^J}%
```

Some tracing code that we don't compile into the production version unless asked for. It will produce huge listings of the boxes involved in balancing in the transcript file.

```
⟨*debug⟩
      \ifnum\c@tracingmulticols>4
         {\showoutput
          \batchmode
          \process@cols\mult@grightbox
           {\showbox\count@}}%
          \errorstopmode
      \fi
⟨/debug⟩
    \fi
```

We check whether our trial was successful. The test used is very simple: we merely compare the first and the last column. Thus the intermediate columns may be longer than the first if `\raggedcolumns` is used. If the right-most column is longer than the first then we start over with a larger value for `\dimen@`.

```
    \ifdim\ht\mult@grightbox >\dimen@
```

If the height of the last box is too large we mark this trial as unsuccessful.

```
⟨*badness⟩
      \too@badtrue
      \ifnum\c@tracingmulticols>\@ne
         \typeout{Rejected: last
                  column too large!}%
      \fi
    \else
```

[13] With TEX version 3.141 it is now possible to use LATEX's `\newlinechar` in the `\message` command, but people with older TEX versions will now get `^^J` instead of a new line on the screen.

To ensure that there isn't a forced break in the last column we try to split off a box of size `\maxdimen` from `\mult@grightbox` (or rather from a copy of it). This should result in a void box after the split, unless there was a forced break somewhere within the column in which case the material after the break would have stayed in the box.

```
\setbox\@tempboxa
     \copy\mult@grightbox
\setbox\z@\vsplit\@tempboxa to\maxdimen
\ifvoid\@tempboxa
```

Thus if `\@tempboxa` is void we have a valid solution. In this case we take a closer look at the last column to decide if this column should be made as long as all other columns or if it should be allowed to be shorter. For this we first have to rebox the column into a box of the appropriate height. If tracing is enabled we then display the badness for this box.

```
\global\setbox\mult@grightbox
   \vbox to\dimen@
      {\unvbox\mult@grightbox}%
\ifnum\c@tracingmulticols>\@ne
  \message{Final badness:
           \the\badness}%
\fi
```

We then compare this badness with the allowed badness for the final column. If it does not exceed this value we use the box, otherwise we rebox it once more and add some glue at the bottom.

```
\ifnum\badness>\c@finalcolumnbadness
  \global\setbox\mult@grightbox
   \vbox to\dimen@
      {\unvbox\mult@grightbox\vfil}%
   \ifnum\c@tracingmulticols>\@ne
     \message{ setting natural
      (> \the\c@finalcolumnbadness)}%
   \fi
\fi
```

If `\@tempboxa` above was not void our trial was unsuccessful and we report this fact and try again.

```
\else
```

If we have unprocessed forced breaks we normally reiterate with a larger column size to fit them in eventually. However, if there are simply too many of them (e.g., 3 forced breaks but only 2 columns to balance) then this will never succeed and we would continue growing the columns until we hit the largest possible column size. So in addition we check how big the column size is compared to available room and if we exceed this by `\maxbalancingoverflow` we give up and instead of balancing cut another normal page. To be indicate this case we set `forcedbreak@leftover` to true.

```
  \@tempdima\@colroom
  \advance\@tempdima \maxbalancingoverflow
  \ifdim \dimen@ < \@tempdima
   \too@badtrue
    \ifnum\c@tracingmulticols>\@ne
      \typeout{Rejected: unprocessed
        forced break(s) in last column!}%
    \fi
  \else
    \forcedbreak@leftovertrue
    \ifnum\c@tracingmulticols>\@ne
     \typeout{Failed: columns too large
       with unprocessed forced break(s)!}%
    \fi
  \fi
 \fi
\fi
```

If the natural height of the first box is smaller than the current trial size but is larger than the previous trial size it is likely that we have missed a potentially better solution. (This could have happened if for some reason our first trial size was too high.) In that case we dismiss this trial and restart using the natural height for the next trial.

```
\ifdim\ht\mult@nat@firstbox<\dimen@
  \ifdim\ht\mult@nat@firstbox>\last@try
    \too@badtrue
    \ifnum\c@tracingmulticols>\@ne
       \typeout{Retry: using natural
                height of first column!}%
    \fi
    \dimen@\ht\mult@nat@firstbox
    \last@try\dimen@
    \advance\dimen@-\p@
  \fi
\fi
```

Finally the switch `too@bad` is tested. If it was made true either earlier on or due to a rightmost column being too large we try again with a slightly larger value for `\dimen@`.

```
  \iftoo@bad
⟨/badness⟩
    \advance\dimen@\p@
  \repeat
```

If we come out of the loop with the switch `forcedbreak@leftover` set to true then balancing has failed and we should cut a normal page. We indicate this below with `\too@badtrue` when any of the columns get too high, so we set this flag here too in order to get the same processing logic.[14]

```
  \ifforcedbreak@leftover
     \too@badtrue
  \else
```

[14] Should get cleaned up as we now have two different routes to reach this part of the processing.

At that point `\dimen@` holds the height that was determined by the balancing loop. If that height for the columns turns out to be larger than the available space (which is `\@colroom`) we squeeze the columns into the space assuming that they will have enough shrinkability to allow this.[15] However, this squeezing should only be done if we are balancing columns on the main galley and *not* if we are building a boxed multicol (in the latter case the current `\@colroom` is irrelevant since the produced box might be moved anywhere at a later stage).

```
\if@boxedmulticols\else
  \ifdim\dimen@>\@colroom
    \dimen@\@colroom
  \fi
\fi
```

Then we move the contents of the odd-numbered box registers to the even-numbered ones, shrinking them if requested. We have to use `\vbox` not `\vtop` (as it was done in the first versions) since otherwise the resulting boxes will have no height (*TEXbook* page 81). This would mean that extra `\topskip` is added when the boxes are returned to the page-builder via `\page@sofar`.

```
\process@cols\mult@rightbox
    {\@tempcnta\count@
     \advance\@tempcnta\@ne
```

when putting the final column together we want overfull information:

```
      \vfuzz\z@
      \setbox\count@\vbox to\dimen@
         {%
          \vskip \z@
            \@plus-\multicolundershoot
            \@minus-\multicolovershoot
          \unvbox\@tempcnta
          \ifshr@nking\vfilmaxdepth\fi
         }%
```

If the resulting box is overfull there was too much material to fit into the available space. The question though is how much? If it wasn't more than `\maxbalancingoverflow` we accept it still to avoid getting very little material for the next page (which we would then have difficulties to balance).

```
      \ifnum\badness>\@M
        \vfuzz\maxdimen % no overfull warning
        \setbox\@tempboxa \vbox to\dimen@
             {\vskip-\maxbalancingoverflow
              \unvcopy\count@}%
        \ifnum\badness>\@M
          \mult@info\@ne
            {Balanced column more than
              \the\maxbalancingoverflow\space
              too large}%
```

Fail the balancing attempt:

```
          \too@badtrue
        \else
```

Otherwise report that there is a problem but within the accepted boundary.

```
          \mult@info\@ne
            {Balanced column
             too large, but less than
             \the\maxbalancingoverflow}%
        \fi
      \fi
    }%
```

Finally end the `\ifforcedbreak@leftover` conditional.

```
  \fi
}
```

Amount that balancing is allowed to overflow the available column space. We default to 12pt which means about one line in most layouts.

```
\newdimen\maxbalancingoverflow
\maxbalancingoverflow=12pt
```

4.5 The box allocations

Early releases of these macros used the first box registers 0, 2, 4,... for global boxes and 1, 3, 5,... for the corresponding local boxes. (You might still find some traces of this setup in the documentation, sigh.) This produced a problem at the moment we had more than 5 columns because then officially allocated boxes were overwritten by the algorithm. The new release now uses private box registers.

There was in fact a bug in the new implementation because at one point LaTeX started to use the extended registers and so jumped from below 255 to above omitting the boxes allocated for inserts and the output page box.

So nowadays we really have to check if we get the full sequence of boxes allocated without holes (i.e., $2 \times \mathit{max\ cols} + 1$) and if not alter the allocation registers to start allocating after 255. This is all done quite low-level by looking directly at the values of the allocation counters.

```
\ifnum\numexpr
```

[15] This might be wrong, since the shrinkability that accounts for the amount of material might be present only in some columns. But it is better to try then to give up directly.

```
        \count20-\count14-1<40
        % this is = 2 * 20
  \count14=\@cclv
\fi

\newbox\mult@rightbox
\newbox\mult@grightbox
\newbox\mult@firstbox
\newbox\mult@gfirstbox
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\newbox\@tempa\newbox\@tempa
\let\@tempa\relax
```

5 New macros and hacks for version 1.2

If we don't use TEX 3.0 `\emergencystretch` is undefined so in this case we simply add it as an unused ⟨*dimen*⟩ register.

```
\@ifundefined{emergencystretch}
      {\newdimen\emergencystretch}{}
```

My tests showed that the following formula worked pretty well. Nevertheless the `\setemergencystretch` macro also gets `\hsize` as second argument to enable the user to try different formulae.

```
\def\setemergencystretch#1#2{%
   \emergencystretch 4pt
   \multiply\emergencystretch#1}
```

Even if this should be used as a hook we use a `@` in the name since it is more for experts. For now we test if the socket is already defined

```
\def\set@floatcmds{%
 \let\@dblfloat\@dbflt
 \def\end@dblfloat{\@endfloatbox
   \UseTaggingSocket{float/end}%
   \@largefloatcheck
   \outer@nobreak
```

This is cheap (deferring the floats until after the current page) but any other solution would go deep into LATEX's output routine and I don't like to work on it until I know which parts of the output routine have to be reimplemented anyway for LATEX3.

```
   \ifnum\@floatpenalty<\z@
```

We have to add the float to the `\@deferlist` because we assume that outside the multicols environment we are in one column mode. This is not entirely correct, I already used the multicols environment inside of LATEXs `\twocolumn` declaration but it will do for most applications.

```
     \@cons\@deferlist\@currbox
   \fi
   \ifnum\@floatpenalty=-\@Mii
     \@Esphack
     \UseTaggingSocket{float/hmode/end}%
   \fi}}
```

There are situations when we may have some space at the end of a column and this macro here will attempt to get rid of it. The typical LATEX sequence is a series of self-canceling glues so if we remove them recursively we are usually fine.

Special care is needed with handling `\vspace*` as that corresponds to `\penalty10000`, `\vskip <skip>`, followed by `\vskip 0pt`. If we see this sequence going backwards in the vertical list we assume that this is a "desired" space. We therefore stop the recursion and reinsert the spaces.

As the multicol code sometimes add an explicit penalty at the end of a column we first attempt to remove it in case it is there.

```
\skip0=0pt
\edef\the@zero@skip{\the\skip0}
\def\remove@discardable@items{%
     \unpenalty
```

Save a previous skip (if there) and then remove it, we can't really tell the difference between no skip an a skip of zero but that's life.

```
     \edef\@tempa{\the\lastskip}%
%\typeout{s1=\@tempa}%
     \unskip
```

If it was a zero skip (or none) we save the next previous skip (if any).

```
     \ifx\@tempa\the@zero@skip
       \edef\@tempb{\the\lastskip}%
%\typeout{s2=\@tempb}%
```

If this one again was zero (or more likely not there in the first place) we stop.

```
       \ifx\@tempb\the@zero@skip
       \else
```

Otherwise we remove this "real" skip. Then we look if it was preceded by a penalty of 10000 (i.e., a \nobreak)

```
          \unskip
%\typeout{p=\lastpenalty}%
          \ifnum \lastpenalty=\@M
```

If so this was a \vspace* or something equivalent to it. Therefore we reintroduce the skips and stop. Otherwise we recurse.

```
            \vskip\@tempb\vskip\@tempa\relax
          \else
            \remove@discardable@items
          \fi
        \fi
      \else
```

If the first skip was a non-zero skip we recurse as well.

```
        \remove@discardable@items
      \fi
}
```

```
⟨*badness⟩
\newif\iftoo@bad
\def\too@badtrue{\global\let\iftoo@bad\iftrue}
\def\too@badfalse{\global\let\iftoo@bad\iffalse}

\newif\ifforcedbreak@leftover
\newcount\c@minrows
\c@minrows=1



\newcount\c@columnbadness
\c@columnbadness=10000
\newcount\c@finalcolumnbadness
\c@finalcolumnbadness=9999

\newdimen\last@try


\newdimen\multicolovershoot
\newdimen\multicolundershoot
\multicolovershoot=0pt
\multicolundershoot=2pt
\newbox\mult@nat@firstbox
⟨/badness⟩
```

A helper for producing info messages

```
\def\mult@info#1#2{%
  \ifnum\c@tracingmulticols>#1%
   \GenericWarning
       {(multicol)\@spaces\@spaces}%
       {Package multicol: #2}%
  \fi
}
```

6 Fixing the \columnwidth

If we store the current column width in \columnwidth we have to redefine the internal \@footnotetext macro to use \textwidth for the width of the footnotes rather than using the original definition.

Starting with version v1.5r this is now done in a way that the original definition is still used, except that locally \columnwidth is set to \textwidth.

This solves two problems: first redefinitions of \@footnotetext done by a class will correctly survive and second if multicols is used inside a minipage environment the special definition of \@footnotetext in that environment will be picked up and not the one for the main galley (the latter would result in all footnotes getting lost in that case).

See the definition of the \multicols command further up for the exact code.

7 Further extensions

This section contains code for extensions added to this package over time. Not all of them may be active, some might sit dormant and wait for being activated in some later release.

7.1 Not balancing the columns

This is fairly trivial to implement. we just have to disable the balancing output routine and replace it by the one that ships out the other pages.

The code for this environment was suggested by Matthias Clasen.

```
⟨*nobalance⟩
 \@namedef{multicols*}{%
```

If we are not on the main galley, i.e., inside a box of some sort, that approach will not work since we don't have a vertical size for the box so we better warn that we balance anyway.

```
\ifinner
  \PackageWarning{multicol}%
    {multicols* inside a box does
     not make sense.\MessageBreak
     Going to balance anyway}%
\else
```

If we aren't balancing we change the `\balance@columns@out` to work like the normal output routine that cuts normal pages. However, there is a catch: In case the last page we cut (after seeing the end of the environment) is actually larger than a page (for example, if it contains more `\columnbreak` commands than columns) we end up with some leftover material that is returned to the main galley, but now the environment end penalty is missing. So we add another one here too. Of course that shouldn't be done if there is really only a single final page, but fortunately in that case we have just finished a page and any penalty on the recent contributions will be discarded, thus the extra one is harmless—puh.

```
  \def\balance@columns@out
      {\multi@column@out \penalty-\@Mvi }%
\fi
\begin{multicols}
}
```

When ending the environment we simply end the inner `multicols` environment, except that we better also stick in some stretchable vertical glue so that the last column still containing text is not vertically stretched out.

We do this as follows: first we ensure that we are back in vertical mode and then we cancel out `\lastskip` if it was positive (in case of a negative glue we assume that it was deliberate, for a deliberate positive glue one needs to use `\vspace*`). We can't simply use `\remove@discardable@items` here as this only works inside boxes but we are here on the main vertical list.

Then we back up by `\prevdepth` but not more than `\boxmaxdepth` so that a baseline of the last box is now at the bottom. This way the material will align properly in case something like `\vfill` spreads it out after all. Finally we append `\vfil` to put white space at the bottom of the column, but we only do this if we aren't anyway doing `\raggedcolumns`.

```
\@namedef{endmulticols*}{%
  \par
  \ifdim\lastskip>\z@ \vskip-\lastskip \fi
  \ifdim \prevdepth>\z@
    \vskip-\ifdim\prevdepth>\boxmaxdepth
                  \boxmaxdepth
           \else \prevdepth \fi
  \fi
  \ifshr@nking\else
    \vfil
  \fi
  \end{multicols}}
```

⟨/nobalance⟩

7.2 Manual column breaking

The problem with manual page breaks within multicols is the fact that during collection of material for all columns a page-forcing penalty (i.e. -10000 or higher) would stop the collecting pass which is not quite what is desired. On the other hand, using a penalty like -9999 would mean that there would be occasions where the `\vsplit`ing operations within multicols would ignore that penalty and still choose a different break point.

For this reason the current implementation uses a completely different approach. In a nutshell it extends the LaTeX output routine handling by introducing an additional penalty flag (i.e., a penalty which is forcing but higher than -10000 so that the output routine can look at this value and thus knows why it has been called).

Inside the output routine we test for this value and if it appears we do two things: save the galley up to this point in a special box for later use and reduce the `\vsize` by the height of the material seen. This way the forcing penalty is now hidden in that box and we can restart the collection process for the remaining columns. (This is done in `\speci@ls` above.)

In the output routines that do the `\vsplit`ting either for balancing or for a full page we simply combine box 255 with the saved box thus getting a single box for splitting which now contains forcing breaks in the right positions.

`\columnbreak` is modeled after `\pagebreak` except that we generate a penalty -10005.

```
\mathchardef\@Mv=10005
\newcommand\columnbreak[1][4]{%
```

We have to ensure that it is only used within a multicols environment since if that penalty would be seen by the unmodified LaTeX output routine strange things would happen.

```
   \ifnum\col@number<\tw@
   \PackageError{multicol}%
    {\noexpand\columnbreak outside multicols}%
    {This command can only be used within
     a multicols or multicols* environment.}%
    \else
```

Increasingly lower penalty based on argument value. This is like `\pagebreak` but we use other penalty values as the LaTeX defaults are rather useless for pagination.

```
    \edef\mc@break@pen
 {-\ifcase#1\@m\or 3333\or 6666\or 9999\else\@Mv\fi\relax}%
    \ifvmode
     \penalty \mc@break@pen
   \else
     \@bsphack
     \vadjust{\penalty \mc@break@pen}%
     \@esphack
   \fi
  \fi}
```

This is modeled after `\newpage` but for column breaks.

```
\newcommand\newcolumn{%
   \ifnum\col@number<\tw@
   \PackageError{multicol}%
    {\noexpand\newcolumn outside multicols}%
    {This command can only be used within
     a multicols or multicols* environment.}%
  \else
    \ifvmode
```

We need to guard the `\vfill` from disappearing.

```
     \nobreak\vfill\kern\z@\penalty -\@Mv\relax
   \else
     \@bsphack
     \vadjust{\nobreak\vfill\kern\z@\penalty -\@Mv\relax}%
     \@esphack
   \fi
  \fi}
```

Need a box to collect the galley up to the column break.

```
\newbox\colbreak@box
```

⟨/package⟩

7.3 Supporting right-to-left languages

`\LR@column@boxes` is called when we are assembling the columns for left to right typesetting. When we start we are inside an `\hbox` of full width. Left to right typesetting is fairly easy, we basically output each column box intermixed with vertical rules and proper spacing. As this happens inside a box of a defined width the rules and the columns automatically get into the right positions.

```
\def\LR@column@boxes{%
```

We loop through the columns with `\process@cols`

```
    \process@cols\mult@firstbox{%
```

If the depth of the current box is larger than the maximum found so far in `\dimen2` we update that register for later use.

```
      \ifdim\dp\count@>\dimen\tw@
        \global\dimen\tw@\dp\count@ \fi
```

If the `colaction` option is given we write out status information about the current column, otherwise the next command does nothing.

```
      \mc@col@status@write
```

The typeset box followed by the column rule material

```
      \box\count@
      \hss{\columnseprulecolor\vrule
            \@width\columnseprule}\hss}%
```

As you will have noticed, we started with box register `\mult@firstbox` (i.e. the left column). So this time `\count@` looped through 2, 4,... (plus the appropriate offset). Finally we add box `\mult@rightbox` and we are done. Again we may have to update `\dimen\tw@`.

```
    \ifdim\dp\mult@rightbox>\dimen\tw@
      \global\dimen\tw@\dp\mult@rightbox \fi
```

If the `colaction` option is given we write out status information about the last column, otherwise the next command does nothing.

```
    \mc@lastcol@status@write
    \box\mult@rightbox
}
```

Assembling the boxes for right to left typesetting is far more complicated. When I first tried to build a solution for this my thinking was that all that is necessary to do is to reverse the order of the columns. But such an approach produces a subtle bug: If we work this way then the first column put on the page will be the last column of the text to read. and this means that the order in which TEX executes write statements or assembles mark material will not happen in the order of the textual flow. So if, for example each column contains a section command then these sections will appear in reverse order in the table of content.

For this reason some amount of gymnastics is needed to add the columns in their natural flow.

```
\def\RL@column@boxes{%
```

First step is to put all rules in the right place (without adding the comes which are instead represented by a space of `\hsize`.

```
    \process@cols\mult@firstbox{%
      \hskip\hsize
      \hss{\columnseprulecolor\vrule
            \@width\columnseprule}\hss
    }%
    \hskip\hsize
```

At this point in the code our typesetting reference point is at the right end of the rightmost column (or rather where that column should appear).

We are now typesetting all columns by first backing up by their width (which is `\hsize`) then typesetting the box and then backing up again, but this time further, i.e., also across the column separation. That will

then enable us to typeset the next column using the same approach until we are done with all but the final column.

```
    \process@cols\mult@firstbox{%
      \ifdim\dp\count@>\dimen\tw@
        \global\dimen\tw@\dp\count@ \fi
      \hskip-\hsize

      \mc@col@status@write
      \box\count@
      \hskip-\hsize
      \hskip-\columnsep
    }%
```

The approach for the final column is similar only that we do not have to back up over any column gap.

```
    \ifdim\dp\mult@rightbox>\dimen\tw@
      \global\dimen\tw@\dp\mult@rightbox \fi
    \hskip-\hsize

    \mc@lastcol@status@write
    \box\mult@rightbox
    \hskip-\hsize
```

However we do have to move the reference point to its right place: to make the rules appear at the expected places, we should get the typesetting position to the far right again. As we at the moment at the far left we skip to the far right like this:

```
    \hskip\full@width
}
```

Macros to switch between left-right and right-left typesetting. In LR typesetting the `\LR@column@boxes` is used to combine the columns. When typesetting right to left the `\RL@column@boxes` is used instead.

```
\newcommand\RLmulticolcolumns
    {\let\mc@align@columns
         \RL@column@boxes}
\newcommand\LRmulticolcolumns
    {\let\mc@align@columns
      \LR@column@boxes}
```

The default is left-to-right:

```
\LRmulticolcolumns
```

7.4 Supporting `\docolaction`

Whenever we want to do something that depends on the current column we execute `\docolaction`. This command takes one optional and three mandatory arguments. The mandatory ones denote what to do if this is a "left", "middle", or "right" column and the optional one is simply there to say what to do if we don't know (default is to use the "left" column action in that case).

We use one counter `\mc@col@check@num` to generate us unique label names. Each time we execute `\docolaction` we increment this counter to get a new name.

```
\newcount\mc@col@check@num
```

The generated "labels" are named

```
\mc@col-\the\mc@col@check@num
```

and they hold as values the numbers 1, 2, or 3 denoting the current column type.

The `\docolaction` scans for a star and optional argument and 3 mandatory ones, but we do this in chunks (not having xparse available).[16]

```
\newcommand\docolaction{%
```

First check is the support is enabled.

```
\ifx\mc@col@status@write\relax
    \PackageError{multicol}%
     {Option 'colaction' not selected}%
     {\string\docolaction\space
      requires the use of the 'colaction'
      option on the package}%
\fi
```

Then prepare `\mc@col@type`.

```
\global\advance\mc@col@check@num\@ne
\edef\mc@col@type{\expandafter\ifx
  \csname mc@col-\the\mc@col@check@num
  \endcsname\relax
                  0\else
  \csname mc@col-\the\mc@col@check@num
  \endcsname
  \fi}%
```

Finally check for a star, record this information and then call `\@docolaction` to do the rest.

```
\@ifstar
  {\@docolactionstartrue \@docolaction}%
  {\@docolactionstarfalse\@docolaction}%
}
```

```
\newcommand\@docolaction[4][1]{%
```

How does the column number get associated with our label? We do this by writing another line into the aux file. Here are the preparations.

```
\edef\@docolactioncheck{\write\@auxout
  {\string\mc@set@col@status
    {mc@col-\the\mc@col@check@num}%
    {\mc@col@type}}}%
```

Where we do the actual `\write` depends on the whether or not we gave seen a `*`. If yes, we do it first and then execute the code argument, otherwise we execute that code first and check at the point after that.

```
\if@docolactionstar \@docolactioncheck \fi
```

We prefix with 0 so that an unknown label (that returns `\relax`) will result in case 0

```
\ifcase \mc@col@type\relax
```

If column is unknown we use the default action or the action denoted by the optional argument (so that arg can take the value 1, 2, 3).

```
    \ifcase #1\or #2\or#3\or#4\fi
  \or
```

Otherwise we know (or think we know) that this is a first, middle, or last column:

```
    #2%  % 1 First col
  \or
    #3%  % 2 any middle col
  \or
    #4%  % 3 last col
  \else
    \ERRORwrongdefaultgiven
  \fi
```

[16] We can do better now, as `\NewDocumentCommand` is part of the kernel. So this should be cleaned up one day.

```
\if@docolactionstar \else \@docolactioncheck \fi
}
```

Here is the if used above:

```
\newif\if@docolactionstar
```

Because of extra data writing to the aux file the aux file will now contain something like the following after the document is processed the first time:

```
\relax
\mc@col@status{1}
\mc@set@col@status{lcol-1}{0}
\mc@col@status{2}
\mc@set@col@status{lcol-2}{0}
\mc@col@status{3}
\mc@set@col@status{lcol-3}{0}
\mc@col@status{1}
\mc@col@status{2}
\mc@col@status{3}
\mc@set@col@status{lcol-4}{0}
```

The `\mc@col@status` line denotes the column type and has been written out just before corresponding the column box was placed onto the page. The`\mc@set@col@status` lines have been written out as part of shipping the column boxes out, e.g., `\mc@set@col@status{lcol-1}{0}` was therefore somewhere within the first column as it appears between `\mc@col@status{1}` and `\mc@col@status{2}` The second argument in that line is the value used in the previous run (or zero if there was no previous run. We can use this to determine if a rerun is necessary.

Thus with this knowledge we can set things up to get the labels working.

When the aux file is read in `\mc@col@status` is used to set `\mc@curr@col@status`:

```
\def\mc@col@status#1{%
    \gdef\mc@curr@col@status{#1}}
```

And when `\mc@set@col@status` is executed we can simply set up the label by associating it with the `\mc@curr@col@status` and ignore the second argument:

```
\def\mc@set@col@status#1#2{%
  \global\expandafter\let\csname #1\endcsname
                          \mc@curr@col@status}
```

The above definition is being used when the `.aux` file is read in at the beginning. At the end we need a different definition to test if another typesetting run is needed. There we compare the value used in the current run (stored in the second argument) with the value used on the next run. If those two values differ we set `@tempswa` to false which will trigger the "Label(s) may have changed" warning.

```
\AtEndDocument{\def\mc@set@col@status#1#2{%
     \ifnum #2=\mc@curr@col@status\else
       \@tempswatrue
     \fi}%
}
```

Finally, as part of determining in which column we are, we used a switch inside `\mc@col@status@write` to determine if we are in the first column or not.

```
\newif\ifmc@firstcol
\mc@firstcoltrue
```

7.5 Using the new mark mechanism

7.5.1 Helpers

```
\ExplSyntaxOn
```

⟨@@=mc⟩

Counter for tracking the current column number. At the start of a multicols environment is holds the number of columns for which there is currently `mcol-...` data (i.e., the number of columns in the last multicols environment).

```
\int_new:N \g_@@_curr_col_int
```

For now we reuse the internal debugging interface of ltmarks, this will probably change

```
\cs_new:Npn \@@_debug_marks:n #1 { \__mark_debug:n {#1} }
```

Helper function to update the `mcol-...` regions when we finish a page while within a multicols environment or when we finish the multicols and return the balanced columns back to the galley.

```
\cs_new_protected:Npn \@@_update_mcol_structures: {
  \@@_debug_marks:n
     { \typeout{Marks:~ update~ mcol~ structures~ (multicol)} }
```

It might be possible that there was a previous multicols (either before the current one, or a boxed one inside) and that one might have had more columns than the current one. If so, we should make these column structures an error as they are no longer valid (or at least empty them, not sure what is better).

```
  \int_step_inline:nnn {\col@number + 1} { \g_@@_curr_col_int }
              { \mark_set_structure_to_err:n { mcol - ##1 } }
```

There is no need to do anything to `mcol-1` up to `mcol-`⟨*\col@number*⟩ because those regions get new data in a second.

Once we have done this we reset the column counter for further processing.

```
  \int_gset:Nn \g_@@_curr_col_int {1}
```

Now we loop through all the assembled column material, using the `column` and `previous-column` regions as an intermediate holding area.

```
  \process@cols\mult@firstbox
      {
        \mark_update_structure_from_material:nn
%fmi             {mcol}
             {column}
             {\unvcopy\count@}
```

Once the `column` region got updated we copy it to `mcol-`⟨*\g_@@_curr_col_int*⟩ and then increment the counter.

```
        \mark_copy_structure:nn
             {mcol - \int_use:N\g_@@_curr_col_int }
%fmi             {mcol}
             {column}
        \int_gincr:N \g_@@_curr_col_int
      }
```

The above loop takes care of all columns, except for the last one (which is stored in `\mult@rightbox`. So we have to do that separately.

```
  \mark_update_structure_from_material:nn
%fmi     {mcol}
     {column}
     {\unvcopy\mult@rightbox}
  \mark_copy_structure:nn
     { mcol - \int_use:N\g_@@_curr_col_int }
%fmi     {mcol}
     {column}
```

Two more aliases to take care of: `first-column` and `last-column` and we are done:

```
  \mark_copy_structure:nn{first-column}{mcol-1}
%fmi  \mark_copy_structure:nn{last-column} {mcol}
  \mark_copy_structure:nn{last-column} {column}
}
```

If we are making a page while inside a multicols environment, we also have to take care of the page region.

```
\cs_new_protected:Npn \@@_update_page_structures: {
  \@@_debug_marks:n
    {
      \typeout{Marks:~ update~ page~ structure~ (multicol)}
    }
```

Since for the page region we are only interested in the top, first, and last marks on the whole page regardless in which column they appear, we simply string together all columns in a big box and update the structure from that.

```
  \mark_update_structure_from_material:nn
     {page}
     {
```

And we better not forget the `\partial@page` in case this is the first page of the multicols (on later pages this box will be void).

```
       \unvcopy\partial@page
       \process@cols \mult@firstbox { \unvcopy\count@ }
       \unvcopy\mult@rightbox
     }
}
```

When finishing a multicols environment, we have to return marks (in then boxed columns) to the current page so that they are available when that page is produced. This is what this helper does.

```
\cs_new_protected:Npn \@@_prepare_mark_reinserts: {
  \@@_debug_marks:n
    { \typeout{Marks:~ prepare~ for~ reinserting~ marks~ (multicol)} }
```

We are only interested in the top, first, and last marks of each class regardless in which column they appeared, so we can copy all column boxes together and process them in one go. The result is returned in `\l_@@_first_marks_tl` and `\l_@@_last_marks_tl` in form of `\mark_insert:nn` statements so we can later execute such token lists directly (and if there were no marks they will be empty).

```
  \mark_get_marks_for_reinsertion:nNN
    {
      \process@cols \mult@firstbox { \unvcopy\count@ }
      \unvcopy\mult@rightbox
    }
    \l_@@_first_marks_tl
    \l_@@_last_marks_tl
}
```

And here are the token lists used above.

```
\tl_new:N \l_@@_first_marks_tl
\tl_new:N \l_@@_last_marks_tl
```

So reinserting it just means executing the token lists (with some surrounding debugging statements).

```
\cs_new_protected:Npn \mc@reinsert@marks{
  \@@_debug_marks:n
    { \typeout{Marks:~ --~ reinsert~ marks~ (multicol)} }
    \l_@@_first_marks_tl \l_@@_last_marks_tl
  \@@_debug_marks:n
    { \typeout{Marks:~ --~ finished~ reinserting~ marks~ (multicol)} }
}
```

7.5.2 Interfaces used in the multicols code and its output routines

When a multicols environment starts we need to clear the column region as it may contain quite old data. Otherwise, that data would be used to generate the top marks and that would be obviously wrong for the first column.

```
\cs_new_protected:Npn \mc@prepare@mark@regions {
```

However, before we do this we need to save away the current column data in case this is a boxed multicol, because when that finishes we need to restore the previous state — this is not necessary for a normal multicols environment, because there the column region is overwritten in the standard output routine by copying the page region.

```
\legacy_if:nT { @boxedmulticols }
  { \mark_copy_structure:nn{saved-column}{column} }
\@@_debug_marks:n
   { \typeout{Marks:~ empty~ mcol~ regions~ (multicol)} }
\mark_clear_structure:n {column}
}
```

If a multicols ends the columns are balanced and then returned to the galley. in that situation we do have to prepare the mcol-... regions and also do this reinsertion.

```
\cs_new_protected:Npn \mc@handle@marks@and@reinserts #1 {
  \@@_update_mcol_structures:
```

Show the current region status when debugging:

```
  \@@_debug_marks:n { \__mark_status:nn {#1} {\the\col@number} }
```

Then prepare for reinserting the marks:

```
  \@@_prepare_mark_reinserts:
```

Finally, we restore the column region in case this was a boxed multicols environment.

```
  \legacy_if:nT { @boxedmulticols }
    { \mark_copy_structure:nn {column}{saved-column} }
}

```

If a page is generated while we are processing a multicols then we have to update the mcol-... regions but also the page region.

```
\cs_new_protected:Npn \mc@handle@col@andpage@marks #1 {
  \@@_update_mcol_structures:
  \@@_update_page_structures:
```

Once done we display the current status of the marks.

```
  \@@_debug_marks:n { \__mark_status:nn {#1} {\the\col@number} }
}
```

⟨@@=⟩

7.6 Tagging support

Here we collect adjustments necessary for tagging support, e.g., before the 2024-11-01 release it was necessary to allocate two float sockets. By now they are part of the kernel and their plugs are currently defined by tagpdf.

```
%\IfSocketExistsF {tagsupport/float/end}
%  {
%    \NewSocket{tagsupport/float/end}{0}
%    \NewSocket{tagsupport/float/hmode/end}{0}
%  }
```

This one is only relevant for multicol so declared here.

```
\NewSocket{tagsupport/page@sofar}{0}
```

The plug definition for now just uses the definition from tagpdf. There has to be a decision where such plugs should be implemented: in the package (like now) then they functions used from tagpdf should become public, in tagpdf, or in lttagging in the kernel.

```
\NewSocketPlug {tagsupport/page@sofar}{default}
  {
```

```
%    \__tag_check_typeout_v:n {====>~In~\string\page@sofar} % some similar debug message if wanted.
    \process@cols\mult@firstbox
        {  \tag_mc_add_missing_to_stream:Nn \count@ {multicol} }
    \tag_mc_add_missing_to_stream:Nn \mult@rightbox {multicol}
  }
```

In the LuaTEX engine there is no need to do anything special in this socket.

```
\sys_if_engine_luatex:TF
  { \AssignSocketPlug{tagsupport/page@sofar}{noop} }
  { \AssignSocketPlug{tagsupport/page@sofar}{default} }

\ExplSyntaxOff
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Change History