An environment for multicolumn output*†

Frank Mittelbach
Email: see top of the source file

Printed September 27, 2001

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 the current implementation. At the moment only page-wide floats (i.e., star-forms) can be used within the scope of the environment.

Preface to version 1.5

This new 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 treshold.

At the same time multicols has been upgraded to run under LaTeX 2$_\varepsilon$.

I apologise for the state of the code documentation but the work on LaTeX 2$_\varepsilon$ kept me too busy to do a proper job. This will hopefully be corrected in the near future.

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.

---

*This file has version number v1.5z, last revised 2000/07/10.

†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.

[1] I started with the algorithm given in the TeXbook on page 417. Without this help a weekend would not have been enough.

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.

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. On the other hand the `\sloppy` declaration (which sets `\tolerance` to $10000 = \infty$) is too large, allowing really bad spacing.[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` parameter holding the value for `\pretolerance` within a multicols environment. Both parameters are usually used only by package 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

[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 `\sloppy` declaration.

be done when the end of the multicols environment is reached. In the first step TeX might consider more material whilst finding the final columns than it actually use 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.

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.

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 colunms 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 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 is allowed to be "too full" or "too short" without affecting the column badness. They are set to 2pt by default.

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

[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.

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 absense of a fixed value.

2.3 Manually breaking columns

Another request often voiced was: "How to 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 artifically forced into the second column (resulting in some white space between paragraphs in the first column).

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 then by `\dbl...`.

2.5 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 the badness values is below 10000 the result is probably acceptable.

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

If a column contains an 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 that 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 unstared version of the figure or table environment. Such floats will disappear!

2.6 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 'multicols' 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 trough 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 in 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!

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 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 misbehaviour 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

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@gfirstbox`, the next in register `\mult@firstbox` + 2, ..., only the last one as an exception in register `\mult@grightbox`. Furthermore it has to set up two the macros `\kept@firstmark` and `\kept@botmark` to hold the values for the first and bottom mark as found in the individual columns. There are some helper functions defined in section 5.1 which may be used for this. Getting the marks right "by hand" is non-trivial and it may pay off to first take a look at the documentation and implementation of `\balance@columns` below before trying anew.

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

`\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.

want to try: wait for TeX 3.0. It has a few features which will make life much easier in multicolumn 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.

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}
\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... multicolum formatting]
```

Next we declare options supported by multicols. Twocolumn 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@@}
\DeclareOption{debugshow}
    {\c@tracingmulticols5\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 ten columns, so we need to check that the user hasn't asked for more.

```
  \ifnum\col@number>10
     \PackageError{multicol}%
```

```
        {Too many columns}%
        {Current implementation doesn't
         support more than 10 columns.%
         \MessageBreak
         I therefore use 10 columns instead}%
       \col@number10 \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`.

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

Now we can safely look for the optional arguments.

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

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
```

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
```

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 of `\@totalleftmargin` directly then lists inside the multicols environment could cause a shift of the output.

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

We also set the register `\doublecol@number` for later use. This register should contain $2 \times$ `\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
  \if@boxedmulticols
    \let\l@kept@firstmark\kept@firstmark
    \let\l@kept@botmark\kept@botmark
    \global\let\kept@firstmark\@empty
    \global\let\kept@botmark\@empty
  \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

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

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
        }%
```

Finally we need to record the marks that are present within the `\partial@page` so that we can construct correct first and bottom marks later on. This is done by the following code.

```
         \prep@keptmarks
```

Finally we have to initialize `\kept@topmark` which should ideally be initialized with the mark that is current on "top" of this page. Unfortunately we can't use `\topmark` because this register will not always contain what its name promises because LaTeX sometimes calls the output routine for float management.[8] Therefore we use the second best solution by initializing it with `\firstmark`. In fact, for our purpose this doesn't matter as we use `\kept@topmark` only to initialize `\firstmark` and `\botmark` of a following page if we don't find any marks on the current one.

```
          \global\let\kept@topmark\firstmark
         }\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 calulate 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 pagewide 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 (pagewide 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

[8] During such a call the `\botmark` gets globally copied to `\topmark` by the TeX program.

appropriate amount since we have changed the magnification factor, etc. above.

```
      \reinsert@footnotes
```

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:

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

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 pagewide footnotes[9] this simple trick saved us from rewriting the `\footnote` macros. However, some applications refered 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
}
```

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` $\times$ `\@colroom` in order to collect enough material before entering the `\output` routine again. In fact we have to add another (`\col@number` $-$ 1) $\times$ (`\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
```

[9] I'm not sure that I really want pagewide 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.

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,' the number depending on the value of the 'multicols' counter. The final value is assigned globally if `#1` is `\global` because we want to use this macro later inside the output routine too.

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

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

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

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.

```
     \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`. But before we do this we reinsert any marks found in box `\mult@box`.

```
    \return@nonemptymark{first}%
                   \kept@firstmark
    \return@nonemptymark{bot}%
                   \kept@botmark
    \page@sofar
    \global\let\kept@firstmark
                 \l@kept@firstmark
    \global\let\kept@botmark
              \l@kept@botmark
⟨*marktrace⟩
    \mult@info\tw@
      {Restore kept marks to\MessageBreak
       first: \meaning\kept@firstmark
       \MessageBreak bot\space\space:
                         \meaning\kept@botmark }%
⟨/marktrace⟩
```

This finishes the code for the "boxed" case.

```
  \else
```

If we are in an unrestricted multicols environment we end the current paragraph with `\par` but this isn't sufficient since TeXs *page_builder* will not totally empty the contribution list.[10] 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 change the output routine in order to balance the columns.

```
     \output{\balance@columns@out}\eject
```

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 take of this by hand.

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

After the output routine has acted we restore the kept marks to their initial value.

```
     \global\let\kept@firstmark\@empty
     \global\let\kept@botmark\@empty
⟨*marktrace⟩
     \mult@info\tw@
       {Make kept marks empty}%
⟨/marktrace⟩
  \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

[10]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.

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.[11]

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

Now it's time to return any footnotes if we are in unrestricted mode:

```
\if@boxedmulticols\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 desaster. 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
```

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

```
\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
```

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 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

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

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.[12] 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.

```
    \mult@info\z@
     {Column spec:\MessageBreak
        (\the\multicol@leftmargin\space -->
         \the\full@width\space = \the\hsize
         \space x \the\col@number)%
      }%
```

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)

```
    \moveright\multicol@leftmargin
     \hbox to\full@width{%
```

and separating the columns with a rule if desired.

```
      \process@cols\mult@gfirstbox{\box\count@
        \hss{\normalcolor\vrule\@width\columnseprule}\hss}%
```

As you will have noticed, we started with box register `\mult@gfirstbox` (i.e. the left column). So this time `\count@` looped through 2, 4,… (plus the appropriate offset). Finally we add box 0 and close the `\hbox`.

```
      \box\mult@rightbox
```

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 at least the depth of a letter 'p'.

```
%      \strut
       \rlap{\phantom p}%
     }%
}
```

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}
```

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.

```
⟨*colbreak⟩
   \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
⟨/colbreak⟩
```

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

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
```

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
```

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@gfirstbox{%
         \setbox\count@
             \vsplit\@cclv to\dimen@
```

After splitting we update the kept marks.

```
             \set@keptmarks
```

If `\raggedcolumns` is in force we add a `vfill` at the bottom by unboxing the split box.

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

Then the last column follows.

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

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

```
    \ifvoid\@cclv \else
        \unvbox\@cclv
        \penalty\outputpenalty
```

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
```

To get a correct marks for the current page we have to (locally redefine `\firstmark` and `\botmark`. If `\kept@firstmark` is non-empty then `\kept@botmark` must be non-empty too so we can use their values. Otherwise we use the value of `\kept@topmark` which was first initialized when we gathered the `\partical@page` and later on was updated to the `\botmark` for the preceding page

```
    \ifx\@empty\kept@firstmark
       \let\firstmark\kept@topmark
       \let\botmark\kept@topmark
    \else
       \let\firstmark\kept@firstmark
       \let\botmark\kept@botmark
    \fi
```

We also initalize `\topmark` with `\kept@topmark`. This will make this mark okay for all middle pages of the multicols environment.

```
    \let\topmark\kept@topmark
⟨*marktrace⟩
    \mult@info\tw@
         {Use kept top mark:\MessageBreak
           \meaning\kept@topmark
          \MessageBreak
          Use kept first mark:\MessageBreak
           \meaning\kept@firstmark
         \MessageBreak
          Use kept bot mark:\MessageBreak
           \meaning\kept@botmark
         \MessageBreak
          Produce first mark:\MessageBreak
           \meaning\firstmark
         \MessageBreak
         Produce bot mark:\MessageBreak
           \meaning\botmark
          \@gobbletwo}%
⟨/marktrace⟩
```

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.[13] Admittedly this is a funny way to do it.

```
    \setbox\@cclv\vbox{\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
```

After the page is shipped out we have to prepare the kept marks for the following page. `\kept@firstmark` and `\kept@botmark` reinitilized by setting them to `\@empty`. The value of `\botmark` is then assigned to `\kept@topmark`.

```
      \global\let\kept@topmark\botmark
      \global\let\kept@firstmark\@empty
      \global\let\kept@botmark\@empty
⟨*marktrace⟩
      \mult@info\tw@
         {(Re)Init top mark:\MessageBreak
          \meaning\kept@topmark
          \@gobbletwo}%
⟨/marktrace⟩
```

Now we reset `\@colroom` to `\@colht` which is LaTeX's saved value of `\textheight`.

```
    \global\@colroom\@colht
```

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}
```

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.

```
⟨*colbreak⟩
     \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
                        \unvbox\@cclv}
         \reinsert@footnotes
```

[13] 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.

```
      \else
⟨/colbreak⟩
```

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.

```
        \PackageWarning{multicol}%
           {Floats and marginpars not
            allowed inside ‘multicols’
            environment!
           \@gobble}%
        \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`. (`\@elt`s default is `\relax` so this is possible with `\xdef`.)

```
        \xdef\@freelist{\@freelist\@currlist}%
        \gdef\@currlist{}%
⟨*colbreak⟩
      \fi
⟨/colbreak⟩
  \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`.

```
⟨-colbreak⟩ \setbox\mult@box
⟨-colbreak⟩   \vbox{\unvbox\@cclv}%
```

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

```
⟨*colbreak⟩
   \setbox\mult@box\vbox{%
       \ifvoid\colbreak@box\else
         \unvbox\colbreak@box\break
         \mult@info\@ne{Re-adding
           forced break(s) in balancing}%
       \fi
       \unvbox\@cclv}%
⟨/colbreak⟩
   \balance@columns
```

This will bring us into 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
```

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

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

Then we return the first and bottom mark and the gathered material to the main vertical list.

```
   \return@nonemptymark{first}\kept@firstmark
   \return@nonemptymark{bot}\kept@botmark
   \page@sofar
```

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@
}
```

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 setting the kept marks by updating them with any marks from this box. This has to be done *before* we add a penalty of -10000 to the top of the box, otherwise only an empty box will be considered.

```
   \get@keptmarks\mult@box
```

We then contine by resetting trying to remove any discardable stuff at the end of `\mult@box`. This is rather experimental. We also add 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
        \remove@discardable@items
        }%
```

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
```

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
```

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.

```
    \vfuzz \col@number\baselineskip
```

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`.

```
⟨*badness⟩
    \global\too@badfalse
⟨/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@firstbox{%
          \global\setbox\count@
          \vsplit\mult@grightbox to\dimen@
```

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
            \global\too@badtrue
          \fi
⟨/badness⟩
                        }}%
```

There is one subtle point here: while all other constructed boxes have a depth that is determined by `\splitmaxdepth` 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.

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

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

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

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

$$\begin{array}{rcl}
\text{box }\texttt{\textbackslash mult@rightbox} & \longleftarrow & \text{all material} \\
\text{box }\texttt{\textbackslash mult@gfirstbox} & \longleftarrow & \text{first column} \\
\text{box }\texttt{\textbackslash mult@gfirstbox}+2 & \longleftarrow & \text{second column} \\
\vdots & & \vdots \\
\text{box }\texttt{\textbackslash mult@grightbox} & \longleftarrow & \text{last column}
\end{array}$$

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@firstbox
%              \copy\mult@nat@firstbox
%     \fi
```

Then we give information about the last column.[14]

```
    \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\@ne
           {\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
```

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.

```
⟨*colbreak⟩
      \setbox\@tempboxa
           \copy\mult@grightbox
      \setbox\z@\vsplit\@tempboxa to\maxdimen
      \ifvoid\@tempboxa
⟨/colbreak⟩
```

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\vfill}%
           \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.

```
⟨*colbreak⟩
      \else
        \too@badtrue
        \ifnum\c@tracingmulticols>\@ne
          \typeout{Rejected: unprocessed
            forced break(s) in last column!}%
        \fi
      \fi
    \fi
⟨/colbreak⟩
```

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 potientially better solution. (This could have happened if for some reason our first trial size was too high.) In

[14]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.

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
```

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

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

```
\newbox\mult@rightbox
\newbox\mult@grightbox
\newbox\mult@gfirstbox
\newbox\mult@firstbox
\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{%
```

[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.

```
    \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.

```
\def\set@floatcmds{%
 \let\@dblfloat\@dbflt
 \def\end@dblfloat{\par
   \vskip\z@
   \egroup
   \color@endbox
   \@largefloatcheck
   \outer@nobreak
```

This is cheap (defering 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
   \fi}}
```

5.1 Maintaining the mark registers

This section contains the routines that set the marks so that they will be handled correctly. They have been introduced with version 1.4.

First thing we do is to reserve three macro names to hold the replacement text for TeX's primitives `\firstmark`, `\botmark` and `\topmark`. We initialize the first two to be empty and `\kept@topmark` to contain two empty pair of braces. This is necessary since `\kept@topmark` is supposed to contain the last mark from a preceding page and in LaTeX any "real" mark must contain two parts representing left and right mark information.

```
\def\kept@topmark{{}{}}
\let\kept@firstmark\@empty
\let\kept@botmark\@empty
```

Sometimes we want to return the value of a "kept" mark into a `\mark` node on the main vertical list. This is done by the function `\return@nonemptymark`. As the name suggests it only acts if the replacement text of the kept mark is non-empty. This is done to avoid adding an empty mark when no mark was actually present. If we would nevertheless add such a mark it would be regarded as a valid `\firstmark` later on.

```
\def\return@nonemptymark#1#2{%
  \ifx#2\@empty
  \else
```

For debugging purposes we take a look at the value of the kept mark that we are about to return. This code will get stripped out for production.

```
⟨*marktrace⟩
    \mult@info\tw@
      {Returned #1 mark:\MessageBreak
       \meaning#2}%
%      \nobreak
%    \fi
⟨/marktrace⟩
```

Since the contents of the mark may be arbitrary LaTeX code we better make sure that it doesn't get expanded any further. (Some expansion have been done already during the execution of `\markright` or `\markboth`.) We therefore use the usual mechanism of a toks register to prohibit expansion.[16]

```
    \toks@\expandafter{#2}%
    \mark{\the\toks@}%
```

We don't want any breakpoint between such a returned mark and the following material (which is usually just the box where the mark came from).

```
    \nobreak
  \fi}
```

If we have some material in a box register we may want to get the first and the last mark out of this box. This can be done with `\get@keptmarks` which takes one argument: the box register number or its nick name defined by `\newbox`.

```
\def\get@keptmarks#1{%
```

For debugging purposes we take a look at the current dimensions of the box since in earlier versions of the code I made some mistakes in this area.

```
⟨*debug⟩
      \typeout{Mark box #1 before:
                 ht \the\ht#1, dp \the\dp#1}%
⟨/debug⟩
```

[16] Due to the current definition of `\markright` etc. it wouldn't help to define the `\protect` command to prohibit expansion as any `\protect` has already vanished due to earlier expansions.

Now we open a new group an locally copy the box to itself. As a result any operation, i.e. `\vsplit`, will only have a local effect. Without this trick the box content would get lost up to the level where the last assignment to the box register was done.

```
  \begingroup
   \vbadness\@M
   \setbox#1\copy#1%
```

Now we split the box to the maximal possible dimension. This should split off the full contents of the box so that effectively everything is split off. As a result `\splitfirstmark` and `\splitbotmark` will contain the first and last mark in the box respectively.

```
   \setbox#1\vsplit#1to\maxdimen
```

Therefore we can now set the kept marks which is a global operation and afterwards close the group. This will restore the original box contents.

```
   \set@keptmarks
  \endgroup
```

For debugging we take again a look at the box dimension which shouldn't have changed.

```
⟨*debug⟩
    \typeout{Mark box #1 \space after:
             ht \the\ht#1, dp \the\dp#1}%
⟨/debug⟩
}
```

The macro `\set@keptmarks` is responsible for setting `\kept@firstmark` and `\kept@botmark`, by checking the current values for `\splitfirstmark` and `\splitbotmark`.

```
\def\set@keptmarks{%
```

If `\kept@firstmark` is empty we assume that it isn't set. This is strictly speaking not correct as we loose the ability to have marks that are explicitly empty, but for standard LaTeX application it is sufficient. If it is non-empty we don't change the value—within the output routines it will then be restored to `\@empty`.

```
   \ifx\kept@firstmark\@empty
```

We now put the contents of `\splitfirstmark` into `\kept@firstmark`. In the case that there wasn't any mark at all `\kept@firstmark` will not change by that operation.

```
     \expandafter\gdef\expandafter
        \kept@firstmark
        \expandafter{\splitfirstmark}%
```

When debugging we show the assignment but only when something actually happened.

```
⟨*marktrace⟩
     \ifx\kept@firstmark\@empty\else
       \mult@info\tw@
         {Set kept first mark:\MessageBreak
          \meaning\kept@firstmark%
          \@gobbletwo}%
     \fi
⟨/marktrace⟩
   \fi
```

We always try to set the bottom mark to the `\splitbotmark` but of course only when there has been a `\splitbotmark` at all. Again, we assume that an empty `\splitbotmark` means that the split off box part didn't contain any marks at all.

```
   \expandafter\def\expandafter\@tempa
      \expandafter{\splitbotmark}%
   \ifx\@tempa\@empty\else
      \global\let\kept@botmark\@tempa
⟨*marktrace⟩
      \mult@info\tw@
        {Set kept bot mark:\MessageBreak
         \meaning\kept@botmark%
         \@gobbletwo}%
⟨/marktrace⟩
   \fi}%
```

The `\prep@keptmarks` function is used to initialize the kept marks from the contents of `\partial@page`, i.e. the box that holds everything from the top of the current page prior to starting the multicols environment. However, such a box is only available if we are not producing a boxed multicols.

```
\def\prep@keptmarks{%
   \if@boxedmulticols \else
     \get@keptmarks\partial@page
   \fi}
```

```
\def\remove@discardable@items{%
⟨*debug⟩
   \edef\@tempa{s=\the\lastskip,
                p=\the\lastpenalty,
                k=\the\lastkern}%
   \typeout\@tempa
⟨/debug⟩
        \unskip\unpenalty\unkern
⟨*debug⟩
   \edef\@tempa{s=\the\lastskip,
                p=\the\lastpenalty,
                k=\the\lastkern}%
   \typeout\@tempa
⟨/debug⟩
        \unskip\unpenalty\unkern
⟨*debug⟩
   \edef\@tempa{s=\the\lastskip,
                p=\the\lastpenalty,
                k=\the\lastkern}%
   \typeout\@tempa
⟨/debug⟩
        \unskip\unpenalty\unkern
```

```
⟨*debug⟩
    \edef\@tempa{s=\the\lastskip,
                 p=\the\lastpenalty,
                 k=\the\lastkern}%
    \typeout\@tempa
⟨/debug⟩
        \unskip\unpenalty\unkern
}
```

```
⟨*badness⟩
\newif\iftoo@bad
```

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

\newdimen\last@try

\newdimen\multicolovershoot
\multicolovershoot=2pt
\newdimen\multicolundershoot
\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 then using the original definition.

Starting with version v1.5r this is now done in a way that the original definition is still used, execpt 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 does contain 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
     \let\balance@columns@out
         \multi@column@out
   \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.

```
 \@namedef{endmulticols*}{\vfill
   \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 `\vspliting` 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 modelled after `\pagebreak` except that we generate a penalty -10005.

```
⟨*colbreak⟩
\mathchardef\@Mv=10005
\def\columnbreak{%
```

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
  \ifvmode
    \penalty -\@Mv\relax
  \else
    \@bsphack
    \vadjust{\penalty -\@Mv\relax}%
    \@esphack
  \fi
\fi}
```

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

```
\newbox\colbreak@box
⟨/colbreak⟩
⟨/package⟩
```

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

v1.0c
- \enough@room: Penalty 0 added to empty the contribution list. 9

v1.0d
- General: All lines shortened to 72 or less. 1

v1.0e
- General: Redefinition of description env. to use \descriptionmargin=5pt in documentation. 1
- \prepare@multicols: \textwidth changed to \linewidth. 11
  - Setting of \columnwidth removed. 11
  - So this file will work with the ‘twocolumn’ command. 11

v1.0f
- General: Changed \z@ to 0pt in redefinition of description. 1

v1.1a
- General: \multicolssep changed to \multicolsep. 1
- \flushcolumns: \flushedcolumns renamed to \flushcolumns. 17

v1.2a
- \balance@columns: Group around main loop removed. 19
- \prepare@multicols: \pretolerance -1 because it nearly never succeeds. 11
  - \set@floatcmds added. 11
  - \setemergencystretch added. 11
  - \vbadness 10001 now. 11
- \set@floatcmds: Macro added. 22
- \setemergencystretch: Macro added. 21
- \speci@ls: Float boxes freed. 17

v1.3a
- \balance@columns: Changed \vtop to \vbox. . 21

v1.3b
- \endmulticols: Do \penalty with \addpenalty 12
- \enough@room: Do \penalty with \addpenalty . 9
- \multicols: Minimum of two columns 7

v1.3c
- \balance@columns: \global\advance left over from older code 20
  - Limit column height to \@colroom 21
- \endmulticols: Check closing env. 13
- \multi@column@out: \unboxing avoided. 15
  - Check if footnotes are actually present before issuing a warning. 15
  - Unnecessary code removed 16