The manyfoot package*

Alexander I. Rozhenko
rozhenko@oapmg.sscc.ru

2005/09/11

This package implements a command, `\newfootnote`, that adds footnote levels to the standard LaTeX's footnote mechanism. Footnotes of every additional level are automatically grouped together on a LaTeX 2_ε output page and are separated from another levels by the special vertical space and rule. The `\newfootnote` command allows customization of the way footnotes of additional level are represented in LaTeX 2_ε documents. Two customization styles are available now: the `plain` style is the ordinary LaTeX's style of footnote representation; the `para` style causes footnotes to be typeset as a run-in paragraph (derived from Donald Knuth's TeXbook and from another sources such as the package `fnpara` by Dominik Wujastyk and Chris Rowley and the package `footmisc` by Robin Fairbairns).

An additional `\DeclareNewFootnote` command is introduced since the version 1.5 of the package. It simplifies creation of new footnote levels with automatic enumeration. Thanks to Frank Mittelbach for this suggestion and for many other proposals for the package improvement.

Since version 1.5, a new footnote rule selection method is introduced (thanks to Christian Tapp `<chr.tapp@gmx.de>` for the idea of this improvement). It allows a customization of footnote rules to be inserted before footnote levels.

1 User Interface

`\extrafootnoterule` Footnotes of different levels are separated at the output page by the special footnote rule, `\extrafootnoterule`. By the default this command is empty. If you want to separate footnotes by a footnote rule you may redefine it or call the package with the `ruled` option:

```
\usepackage[ruled]{manyfoot}
```

In this case, the `\extrafootnoterule` receives a value of the default footnote rule command.

`\defaultfootnoterule` The default footnote rule is saved in the `\defaultfootnoterule` command at the beginning of document (we provide this for compatibility with the `splitrule` option of the **footmisc** package).

*This file has version number v1.10, last revised 2005/09/11.

The style `para` of footnotes typesetting needs many code. To save space, the support for this style is loaded with `para` or `para*` options. If you are going to use run-in paragraph footnotes indented as ordinary footnotes, use the package with the `para` option:

```
\usepackage[para]{manyfoot}
```

To suppress indentation, use it with the `para*` option:

```
\usepackage[para*]{manyfoot}
```

Note, that these options only *allow* you to generate additional footnote levels in `para` style. But what style you prefer for every footnote level is up to you. For example, let as generate two footnote levels: the first—in ordinary style and the second—in `para` style. To do this we have to write the following code in the preamble of the document:

```
\usepackage[para]{manyfoot}
\newfootnote{A}
\newfootnote[para]{B}
```

`\newfootnote` The mandatory parameter of `\newfootnote` is a *suffix* to be added to the end of command names generated by `\newfootnote` command. The optional parameter is a customization style for generated footnote level (two styles now implemented, named `plain` and `para`; the default is `plain`).

This example generates two commands, `\FootnotetextA` and `\FootnotetextB`, for insertion of a text into corresponding footnote levels. Their syntax is the following:

`\Footnotetext`⟨*suffix*⟩`{`⟨*marker*⟩`}{`⟨*inserted text*⟩`}`

They put the ⟨*inserted text*⟩ marked with ⟨*marker*⟩ into TEX's insert register `\footins`⟨*suffix*⟩ (this insert is also generated by `\newfootnote` command). We use *the hand* style of footnote marking, because the choice of how such footnotes have to be marked is user's one. Such a way simplifies the syntax of new commands and minimize a number of additional commands needed.

`\Footnotemark` `\Footnotetext` `\Footnote` It is clear that accompany to `\Footnotetext`⟨*suffix*⟩ must present something like *the hand footnote mark* command. Such commands are provided by `nccfoots` package which is automatically loaded in this package. Their syntax is the following:

```
\Footnotemark{⟨marker⟩}
\Footnotetext{⟨marker⟩}{⟨inserted text⟩}
\Footnote{⟨marker⟩}{⟨inserted text⟩}
```

The first command is useful for all footnote levels. Two last commands are the hand companions for LATEX's `\footnote...` commands.

The question is what these commands have to do when ⟨*marker*⟩ is empty? In such a case we leave the current marker unchanged. Therefore, `\Footnote` command is equal to

`\Footnotemark{`⟨*marker*⟩`}\Footnotetext{}{`⟨*inserted text*⟩`}`

This is useful for `\Footnotetext`⟨*suffix*⟩ commands in `plain` style. For `para` style the empty ⟨*marker*⟩ means the footnote without marker (this is the special case used for splitting long footnotes in `para` style; see below).

Finally, we explain[1] on the previous example[i] how to automate[2] enumeration of the additional[ii] footnote[3] levels[iii] (we have used in this sentence a number of level's `A` and `B` footnotes to show how this package works). Let us enumerate the footnotes of `A` level by arabic numbers and the footnotes of `B` level by roman numbers. We allocate two new counters named `footnoteA` and `footnoteB` and define the corresponding `\footnote...` commands with automatic enumeration. The corresponding code is the following

```
\newcounter{footnoteA}
\newcommand{\footnoteA}{%
   \stepcounter{footnoteA}%
   \Footnotemark\thefootnoteA \FootnotetextA{}}
\newcounter{footnoteB}
\newcommand{\footnoteB}{%
   \stepcounter{footnoteB}%
   \Footnotemark\thefootnoteB \FootnotetextB\thefootnoteB}
\renewcommand{\thefootnoteB}{\roman{footnoteB}}
```

To produce footnotes presented here, we have done the following

```
Finally, we explain\footnoteA{This is the first A-level
footnote.} on the previous example\footnoteB{This is the first
B-level footnote.} how to automate\footnoteA{The second
A-footnote.} enumeration of the additional\footnoteB{The second
B-footnote.} footnote\footnoteA{The third A-footnote.}
levels\footnoteB{The third very very very very very very very
very very long B-footnote.}
```

2 Declaring New Footnotes

`\DeclareNewFootnote`

To create a new footnote level with automate enumeration, you need to type a bulk of code: create a new footnote level (e.g. `\newfootnote{A}`), create a counter for automate enumeration (e.g. `\newcounter{footnoteA}`), and create a footnote insertion command (e.g. `\newcommand{\footnoteA}`). All these things can be automated with the `\DeclareNewFootnote` command used in the preamble only:

`\DeclareNewFootnote[`⟨*footnote style*⟩`]{`⟨*suffix*⟩`}[`⟨*enumeration style*⟩`]`

[1] This is the first A-level footnote.
[2] The second A-footnote.
[3] The third A-footnote.

[i] This is the first B-level footnote. [ii] The second B-footnote. [iii] The third very very very very very very very very very long B-footnote.

Here ⟨*footnote style*⟩ is the customization style (`plain` is default) and ⟨*enumeration style*⟩ is a style of numbering. This macro also prepares `\footnotemark`⟨*suffix*⟩, `\footnotetext`⟨*suffix*⟩, `\Footnotemark`⟨*suffix*⟩, and `\Footnote`⟨*suffix*⟩ commands for completeness. For example, the command `\DeclareNewFootnote{A}` creates the following:

- The new plain footnote level with TEX's insert register `\footinsA`;
- The counter `footnoteA` with arabic numbering;
- The command `\FootnoteA{`⟨*marker*⟩`}{`⟨*inserted text*⟩`}`;
- The command `\FootnotemarkA{`⟨*marker*⟩`}`;
- The command `\FootnotetextA{`⟨*marker*⟩`}{`⟨*inserted text*⟩`}`;
- The command `\footnoteA[`⟨*number*⟩`]{`⟨*inserted text*⟩`}`;
- The command `\footnotemarkA[`⟨*number*⟩`]`; and
- The command `\footnotetextA[`⟨*number*⟩`]{`⟨*inserted text*⟩`}`.

The first three `\Footnote...` commands work as their analogues without suffix, and the behavior of three last commands is just the same as for ordinary `\footnote`, `\footnotemark`, and `\footnotetext`. Examples of footnote controls shown in the previous section can be easy specified in the following two lines:

```
\DeclareNewFootnote{A}
\DeclareNewFootnote[para]{B}[roman]
```

3 Custom Footnote Rules

`\SelectFootnoteRule` A custom footnote rule can be specified for every new footnote level. Just store the `\SelectFootnoteRule` command before a new footnote declaration. Its syntax is the following:

`\SelectFootnoteRule[`⟨*priority*⟩`]{`⟨*rule name*⟩`}[`⟨*action*⟩`]`

The ⟨*priority*⟩ is a nonnegative integer number specifying an importance of the rule. It controls the process of a footnote rule selection while typeset (see more detail description below). The default priority is 0. The ⟨*rule name*⟩ is a prefix of the footnote rule command to be used before the next footnote level. The command `\`⟨*rule name*⟩`footnoterule` is used as a footnote rule. The ⟨*action*⟩ parameter specifies an additional action to be applied just before the next footnote group if it is nonempty (for example, an action can produce a marginal mark near the footnote group). The default action is empty. A footnote rule and an action must insert a material of zero height in vertical mode.

Two footnote rule commands are predefined: `\extrafootnoterule` and `\defaultfootnoterule`. The `\extrafootnoterule` is selected with zero priority before a new footnote level if no other footnote rule was selected with the

\SelectFootnoteRule command. The \defaultfootnoterule is usually equal to the \footnoterule, but if the \footnoterule was redefined in the **footmisc** package with the splitrule option, the \defaultfootnoterule will save the original value of the footnote rule. So, if you want to select an ordinary footnote rule for the next footnote level, use the following command

```
\SelectFootnoteRule{default}
```

The following example create four footnote levels with rule inserted between footnotes A,B and C,D:

```
\DeclareNewFootnote{A}
\DeclareNewFootnote[para]{B}[alph]
\SelectFootnoteRule[1]{default}
\DeclareNewFootnote{C}[roman]
\DeclareNewFootnote{D}[Roman]
```

To use more custom footnote rules, you must create corresponding commands in some way.

The algorithm of footnote rule insertion while a page output is the following:

- At first, we set the \defaultfootnoterule to be the current rule. Then we test the insert register of the standard footnote group. If it is empty, we set the priority of current rule to 1, otherwise to -1 (-1 means that this rule is already played).
- After that, we do the following for every next footnote group. We compare the priority of current rule and the priority of rule linked with the next footnote group. If the current priority is less or equal to the next priority, the current rule is changed to the next rule and the current priority is set to the next priority. Then we test an insert register of the next footnote group. If this footnote group is nonempty, we insert the current rule before it and decrease the current priority to -1 (played rule).

\footnoterulepriority The priority for the rule of standard footnote group is specified in the \footnoterulepriority command. Its default value is 1 (this means that the standard footnote rule is more important then every next rule of 0th priority). You can redefine this priority with the \renewcommand.

4 Add Hooks at the Beginning of Footnotes

\SetFootnoteHook Since version 1.9, a new command \SetFootnoteHook{⟨*text*⟩} is introduced. This command is used before a new footnote declaration and specifies an action applied for such footnotes.

If the new footnote level is plain, the hook is applied at the beginning of every its footnote. For example, the following declaration specifies A-footnotes with hang numbering:

```
\SetFootnoteHook{\hangindent=1.8em\noindent}
\DeclareNewFootnote{A}
```

For para-footnotes, the hook is applied in the output routine after merging all para-footnotes together. The following declaration specified B-footnotes starting with the word "Cases:" typeset in boldface:

```
\SetFootnoteHook{\noindent\textbf{Cases:}\quad}
\DeclareNewFootnote[para]{B}[alph]
```

5 Per-page Footnotes Numbering

The per-page resetting of counters can be implemented with the `perpage` package by David Kastrup. For example, if you need to reset the `footnoteA` counter every page, just insert the following in the preamble of you document (after declaring the footnote level A of course):

```
\usepackage{perpage}
\MakePerPage{footnoteA}
```

If all new footnote levels declared with the `\DeclareNewFootnote` command must reset every page, use the `perpage` option:

```
\usepackage[perpage]{manyfoot}
```

In this case, the `perpage` package is loaded automatically and the `\MakePerPage` command is applied to every counter created with the `\DeclareNewFootnote`.

6 Splitting of Para-Footnotes

The algorithm proposed by Donald Knuth for processing run-in paragraph footnotes has some shortages. Namely, small overfulls of output page frequently arise and the automatical splitting of long footnotes is impossible. First bug is corrected here (look at the implementation section below), but the second one couldn't be easily eliminated.

`\SplitNote` To split a long footnote near the end of the output page we propose the following method. You should decide where the splitting have to be done. Then you split footnote "by hands" into two parts. You leave *the first part* at the same position in the text and complete its text by the command `\SplitNote`. You move *the second part* down in the source file and attach it to any text corresponding to the next page via the `\Footnotetext...` command with *the empty marker*. E.g. you source text will look as follows:

```
This text goes on the current
page\footnoteB{This is the beginning of the long
footnote ... the splitting must be here\SplitNote}
```

```
    ...
This text goes on the next
page\FootnotetextB{}{and the end of the split
footnote is here ...}
```

If both parts of split footnote get into the same output page, the splitting is ignored.

7 Footnotes within Minipages and Multicolumns

If you attach an additional level footnote to some text inside a minipage, it will appear at the bottom of the page nor the minipage.

The package correctly works together with `multicol` package and gives good results when switching between one and two columns by LaTeX's commands `\twocolumn` and `\onecolumn`.

8 Compatibility with footmisc

The `footmisc` package also provides the `para` option. You cannot use this option with `footmisc` if you plan to use it (or `para*`) with `manyfoot`.

The footnote margins management from the `footmisc` package acts on the `manyfoot` also, but with some limitations concerning to footnotes in the `para` style:

- If the `manyfoot` package is loaded with the `para*` option, the margin management options of the `footmisc` package have no influence on additional footnotes of `manyfoot` in `para` style.
- If the `manyfoot` package is loaded with the `para` option, the margin management options `flushmargin` and `hang` of the `footmisc` package appropriately change the indentation of `para` footnotes created with the `manyfoot` package. More exactly, a nonnegative value of the `\footnotemargin` is taken into account in this case. But if a width of the starting footnote marker becomes greater than `0.8em`, the hang indentation of such footnote will exceed the value of `\footnotemargin`.

Footnotes having the `plain` style inherit all formatting specified in the `footmisc`, because they use the standard `\@makefntext` hook.

9 Add Extra Skip for Para-Footnotes

The algorithm used for calculation the vertical space occupied with para-footnotes has one serious disadvantage. It cannot exactly calculate how many vertical space the collected para-footnotes will occupy because the formatting of such footnotes in vertical box is applied in the output routine *after* TeX decides on page breaking. For example, if collected para-footnotes occupy 2.25 lines, the algorithm reserves

the vertical space of `2.25\normalbaselineskip` for them, but when such footnotes will be formatted in vbox, 3 lines will be necessary of course. This is the reason why the use of para-footnotes can lead to page overfull. To compensate this overfull we use a special trick: we add a special space to a value of skip for all insert registers taking part with para-inserts. When the insertions are formatted in the output routine, the adjusted space is turned back. So, the additional space appears and the overfull disappears.

The value of space to be adjusted is calculated as follows:

```
max(\footnotesep-\height\strut,0)+0.5\normalbaselineskip
```

We take into account here that the value of `\footnotesep` can be larger than the height of `\strut` and the total height of para-footnotes is less than the required height by `0.5\baselineskip` on the average.

`\ExtraParaSkip` Sometimes, the assumptions on the required extra space are wrong and footnotes can overlap on the text (this situation can occur in long tables). So, the new command, `\ExtraParaSkip{`⟨*space*⟩`}`, was introduced since version 1.7 (by the proposal of Uwe Lück `<ednotes.sty@web.de>` and Florian Kragl `<a9902976@unet.univie.ac.at>`) to adjust the default extra space. The command can be used in the preamble only. It can be used more than once. The later use of the command overrides the previous one.

10 The Implementation

First we load `nccfoots` package, containing hand footnote mark commands and the command `\NCC@makefnmark{`⟨*marker*⟩`}` which generates marker in `\@thefnmark` command.

```
⟨*package⟩
\RequirePackage{nccfoots}
```

`\extrafootnoterule` Then we define the empty `\extrafootnoterule` command and implement `ruled` option that sets the `\extrafootnoterule` to be equal to the `\defaultfootnoterule`. The `\defaulfootnoterule` is later defined at the beginning of document. It is set to the `\pagefootnoterule` if the last is specified or to the `\footnoterule` if not. This trick provides the compatibility with the `splitrule` option of the **footmisc** package in which the default `\footnoterule` is saved in `\pagefootnoterule` and then redefined.

```
\newcommand{\extrafootnoterule}{}
\DeclareOption{ruled}{\def\extrafootnoterule{\defaultfootnoterule}}
```

`\MFL@columnwidth` `\MFL@floathook` We use the dimen `\MFL@columnwidth` instead of `\columnwidth` while producing the footnote for insertion. We set this dimen to be equal to `\columnwidth` at the beginning of document and within the `\@floatplacement` command. The command `\MFL@floathook` does this job. Later, in `para` option, we'll add to this hook the resetting of the fudge factor.

```
\newdimen\MFL@columnwidth
\def\MFL@floathook{\MFL@columnwidth\columnwidth}
```

\MFL@insert The command \MFL@insert{⟨*insert register*⟩}{⟨*text*⟩} inserts the text to the insert register and sets the standard splitting parameters. We let this command to be equal \MFL@mpinsert when go into a minipage. To use this command after a minipage we save its value in \MFL@realinsert command. To support multiple option from footmisc we add the \FN@mf@prepare command from footmisc (suggested by Frank Mittelbach).

```
\long\def\MFL@insert#1#2{%
  \insert#1{\splittopskip\footnotesep \splitmaxdepth \dp\strutbox
    \floatingpenalty\@MM #2%
  }%
  \FN@mf@prepare
}
\providecommand\FN@mf@prepare{}
\let\MFL@realinsert\MFL@insert
```

\MFL@applyhook The command \MFL@applyhook{⟨*insert register*⟩} applies a hook corresponding to the given insert register.

```
\def\MFL@applyhook#1{\csname MFL@hook\string#1\endcsname}
```

10.1 Footnote Styles Support

\MFL@start... Every additional footnote level has deal with its own *insert register* which is allocated by the \newfootnote command. This insert register is automatically initialized with the same values as the \footins register. You can modify its parameters and do something more in the command

\MFL@start⟨*style*⟩{⟨*insert register*⟩}

You must do all modifications globally, because this command is called within the group. It is called at the beginning of the document for every footnote of such style and is needed in the preamble only.

\MFL@fnote... To put footnote into the insert register the command

\MFL@fnote⟨*style*⟩{⟨*insert register*⟩}{⟨*marker*⟩}{⟨*inserted text*⟩}

is used. Note that you have to define it with \long modifier if you allow footnotes consisting of a number of paragraphs. You have to use the macros \MFL@insert and \MFL@columnwidth instead of \insert and \columnwidth.

\MFL@process... And the last style customization command

\MFL@process⟨*style*⟩{⟨*insert register*⟩}

is called within the output routine to prepare the box of the ⟨*insert register*⟩ for joining it with another footnote inserts.

Do some comments on joining algorithm. It joins together all nonempty footnote insert boxes and puts the result into \footins box. Special vertical space and footnote rule are added between every two neighboring nonempty inserts. This space is defined by the skip value from the second neighbor. In other words the skip of the ⟨*insert register*⟩ is the vertical space to be added between this insert and

any nonempty footnote insert coming before it in the *list of footnote inserts*. Note that this skip can be modified while processing of the document (the `multicol` package multiplies `\skip` and `\count` of `\footins` to the number of columns when goes into multicolumns mode; we do the same with these parameters of all another footnote inserts to provide the compatibility with `multicol` package).

`\MFL@skip` This command adds the vertical space just before the `\MFL@process...` command. The value of this space is also calculated in `\@tempskipa`. Note that this command is inserted *between* vertical boxes in joining procedure. If processed box is the first nonempty footnote box (`\footins` register and all registers going in footins list before the processed insert are empty), the command `\MFL@skip` is ignored.

10.2 Plain Footnote Style

`\MFL@startplain` `\MFL@fnoteplain` `\MFL@processplain` It is very simple. The commands `\MFL@startplain` and `\MFL@processplain` do nothing

```
\let\MFL@startplain\@gobble
\@onlypreamble\MFL@startplain
\let\MFL@processplain\@gobble
```

and the command `\MFL@fnoteplain` does near the same as the usual footnotetext command.

```
\long\def\MFL@fnoteplain#1#2#3{\NCC@makefnmark{#2}%
  \MFL@insert#1{\reset@font\footnotesize
    \interlinepenalty\interfootnotelinepenalty
    \hsize\MFL@columnwidth \@parboxrestore
    \protected@edef\@currentlabel{\@thefnmark}%
    \color@begingroup
      \MFL@applyhook{#1}%
      \@makefntext{%
        \rule\z@\footnotesep\ignorespaces#3\@finalstrut\strutbox}%
    \color@endgroup
  }%
}
```

10.3 Para Footnote Style

`\ifMFL@paraindent` This style is too complicated. We load its commands optionally. First we define `\ifMFL@paraindent` command to switch between indented and non-indented versions of `para` footnotes.

```
\newif\ifMFL@paraindent \MFL@paraindenttrue
```

Now we implement the `para` option.

```
\DeclareOption{para}{%
```

`\footglue` The `\footglue` skip is the horizontal space between footnotes in run-in paragraph. It's name goes from TEXbook (Appendix D. Dirty Tricks) and we don't rename

this register. This gives us an additional protection from the usage this package in the document where another package also provides footnotes in `para` style.

```
\newskip\footglue
```

Contrarily to `footmisc` package we initialize this skip in terms of the footnote size (nor the normal size).

```
{\footnotesize \global\footglue=1em plus.3em minus.3em }
```

`\SplitNote` `\ifMFL@split` The switch `MFL@split` provides footnote splitting and the command `\SplitNote` simply sets this switch to *true*.

```
\newif\ifMFL@split  \MFL@splitfalse
\newcommand\SplitNote{\MFL@splittrue}
```

`\MFL@startpara` Now we prepare `para` support routines. The first is the starting routine.

```
\def\MFL@startpara#1{%
```

It adds to insert's skip additional space `\MFL@paraskip` (it is non-stretchable and is calculated at the beginning of the document).

```
  \global\advance\skip#1\MFL@paraskip
```

Then we define the command named `\MFL@split\footins`⟨*suffix*⟩ which saves splitting information for the corresponding footnote level. While processing the document this command will have the `\MFL@applyhook{`⟨*insert register*⟩`}` value (without splitting) or `\noindent` value (with splitting).

```
  \MFL@setsplit{#1}{\MFL@applyhook{#1}}%
}
\@onlypreamble\MFL@startpara
```

`\MFL@fnotepara` The next command inserts footnote into ⟨*insert register*⟩. We use Knuth's trick to inform the output procedure how many horizontal space occupy the footnote by modifying its vertical size as *fudgefactor* × *footnotebaselineskip* where *fudgefactor* is the ratio of *footnotebaselineskip* to `\columnwidth`. The real vertical size of such footnote is not needed because the footnote is save in `\hbox` nor `\vbox` and while processing to the run-in paragraph will be unboxed.

This footnote command is not `\long` because the `\par` command cannot be used in `\hbox`. At the beginning we make footnote mark and set current label only if footnote mark was nonempty. We use below the temporary switch `@tempswa` to select the case of nonempty footnote mark.

```
\def\MFL@fnotepara#1#2#3{\let\@thefnmark\@empty
  \NCC@makefnmark{#2}%
  \MFL@insert#1{\reset@font\footnotesize
    \ifx\@thefnmark\@empty \@tempswafalse \else
      \@tempswatrue
      \protected@edef\@currentlabel{\@thefnmark}%
    \fi
    \color@begingroup
```

Now we test the width of the footnote mark and if it less than `0.8em` we calculate in `\@tempdima` the difference between `0.8em` and the natural width of the marker.

This horizontal space is inserted before the footnote mark. Why it is needed? While the processing of `para` insert we set `\parindent` to `1em`. And taking into account that the marker width is at least `0.8em` we obtain the distance at least `1.8em` between the footnote text and the left margin. It is exactly the same distance as for footnotes in `plain` style. Why we add this space using `\hskip`? It should be removed when line will be broken at this point. We add this space when the switch `MFL@paraindent` is true.

```
      \if@tempswa
        \setbox\@tempboxa\hbox{\@makefnmark}%
        \ifMFL@paraindent
          \@tempdima.8em \advance\@tempdima-\wd\@tempboxa
          \ifdim \@tempdima<\z@ \@tempdima\z@ \fi
        \else
          \@tempdima\z@
        \fi
      \fi
      \setbox\@tempboxa\hbox{%
        \if@tempswa
          \hskip\@tempdima\unhbox\@tempboxa\nobreak
        \fi
```

Well. Now we insert the footnote text into hbox and test the `MFL@split` switch. If it is true (splitting needed) we add at the end of text the special small penalty `-1`. It will indicate us where the splitting needed. In false case we set penalty `-10` and insert `\footglue` space.

```
        \ignorespaces#3\unskip\strut
        \ifMFL@split \penalty\m@ne\space \else
          \penalty-10 \hskip\footglue
        \fi
      }%
```

And finally we use Knuth's trick.

```
      \dp\@tempboxa\z@ \ht\@tempboxa\MFL@fudgefactor\wd\@tempboxa
      \box\@tempboxa
    \color@endgroup
  }%
}
```

`\MFL@processpara` This is the last procedure of `para` style which is called in the output routine. We must reorganize the box of the ⟨*insert register*⟩ which is the "vbox of hboxes".

```
\def\MFL@processpara#1{%
```

Firstly we redefine `\MFL@skip` command decreasing its skip by `\MFL@paraskip`.

```
  \advance\@tempskipa -\MFL@paraskip
  \edef\MFL@skip{\vskip\the\@tempskipa\relax}%
  \setbox#1\vbox{%
```

Now we execute the first step of Knuth's algorithm: convert the "vbox of hboxes" to "hbox of hboxes".

```
    \unvbox#1\setbox\@tempboxa\hbox{}\MFL@makehhbox
```

The second step is unhboxing of all first level hboxes. After that we have the normal hbox which can be easily converted to paragraph vbox.

```
      \setbox\@tempboxa\hbox{\unhbox\@tempboxa\MFL@removehboxes}%
```

Now we set all needed parameters to prepare run-in paragraph. When we set a `\parindent`, we do test on the compatibility with `footmisc` package. If this package is in use, the `\footnotemargin` register is specified. To provide just the same indent of the first line for para footnotes as for ordinary footnotes, we calculate the par indent as `\footnotemargin-0.8em` (`0.8em` is the width of marker going after). This calculation is executed in the only case of nonnegative `\footnotemargin` value.

```
      \footnotesize
      \hsize\MFL@columnwidth \@parboxrestore
      \ifMFL@paraindent
        \@ifundefined{footnotemargin}%
          {\parindent\footglue}%
          {\parindent\footnotemargin\relax
           \ifdim\parindent<\z@ \parindent\footglue
           \else \advance\parindent -0.8em \fi}%
      \fi
```

Then we call `\MFL@split\footins`⟨*suffix*⟩ macro to set `\noindent` if it is needed (this case occurs when the footnote was splitted at the previous page) or apply its hook otherwise.

```
      \csname MFL@split\string#1\endcsname
```

Here is the right place where `\footnotesep` rule have to be inserted.

```
      \rule\z@\footnotesep
```

Finally, we convert prepared hbox to vbox and test the last penalty (it is the penalty of the last `para` footnote inserted into this vbox). This penalty is `-10` or `-1`. The case `-1` means that the last footnote continues onto the next page (splitting case; see the command `\MFL@fnotepara`). In this case we adjust the last line of paragraph to the right margin and set `\MFL@split\footins`⟨*suffix*⟩ macro to `\noindent`. Otherwise, we apply the hook as its default value.

```
      \unhbox\@tempboxa\unskip
      \ifnum\lastpenalty=\m@ne \parfillskip\z@
        \MFL@setsplit{#1}{\noindent}%
      \else
        \MFL@setsplit{#1}{\MFL@applyhook{#1}}%
      \fi
    }%
  }
```

`\MFL@makehhbox` This procedure converts "vbox of hboxes" to "hbox of hboxes". Its implementation has minimal distinctions from the original code described in TEXbook. We removed from it the initialization of the accumulating box (`\@tempboxa`) and added a possibility some boxes in the list to be vboxes nor hboxes. Such vboxes arises because while processing a minipage we put all internal footnotes into vbox to prevent their splitting. We use the box 0 as the temporary box here.

```
\def\MFL@makehhbox{%
  \loop\setbox\z@\lastbox \ifhbox\z@
    \setbox\@tempboxa\hbox{\box\z@\unhbox\@tempboxa}%
  \repeat
  \ifvbox\z@ \unvbox\z@ \MFL@makehhbox \fi
}
```

`\MFL@removehboxes` This is an internal procedure described in TEXbook to unboxing "hbox of hboxes".

```
\def\MFL@removehboxes{\setbox\@tempboxa\lastbox
  \ifhbox\@tempboxa{\MFL@removehboxes}\unhbox\@tempboxa\fi
}
```

`\MFL@setsplit` This macro sets the value of `\MFL@split\footins`⟨*suffix*⟩ macro.

```
\def\MFL@setsplit#1#2{%
  \expandafter\gdef\csname MFL@split\string#1\endcsname{#2}%
}
```

Finally, we have to prepare something for work. We add to `\MFL@floathook` the calculation of the fudge factor. Such a calculation is needed when we switch between one and two columns by the standard LATEX commands. The original Knuth's version of the algorithm produces an overflow if `\normalbaselineskip` greater or equal to `16pt`. In version 1.10, the calculation was improved to remove overflow. Now overflow appears when `\normalbaselineskip` is `64pt`. I think it will be enough for all applications. By the way, in ordinary cases the new version calculates the fudge factor two times more accurate than the Knuth's one.

```
\g@addto@macro\MFL@floathook{%
  \begingroup
    \footnotesize \@tempdima\normalbaselineskip
    \multiply \@tempdima \@cclvi
    \@tempdimb \columnwidth
    \divide \@tempdimb \@cclvi
    \divide \@tempdima \@tempdimb
    \xdef\MFL@fudgefactor{\strip@pt\@tempdima}%
  \endgroup
}
```

`\MFL@paraskip` The last trick is the calculation of `\MFL@paraskip` — the skip which has to be added to the skip of all `para` style footnote inserts and then "turned back" while preparing of run-in paragraph. Why it is needed? Two reasons. The first, the value of `\footnotesep` may be larger than the height of `\strut`. And we nowhere take into account this exceeding. The second, the total height of `para` footnotes is less then the real height of the prepared run-in paragraph by `0.5\baselineskip` on the average. So we have to add it to the `\skip` register for the compensation.

`\ExtraParaSkip` I new command, `\ExtraParaSkip{`⟨*space*⟩`}`, is introduced here by the proposal of Uwe Lück `<ednotes.sty@web.de>`. Using it, everyone can adjust a value of skip to be added to all `\skip` registers of para-inserts.

```
\newcommand*\ExtraParaSkip[1]{%
```

```
    \def\MFL@xparaskip{\advance\@tempdima#1\relax}%
  }
  \let\MFL@xparaskip\relax
  \@onlypreamble\ExtraParaSkip
  \@onlypreamble\MFL@xparaskip
```

We do calculations of the `\MFL@paraskip` at the beginning of the document.

```
  \AtBeginDocument{%
    \begingroup
      \footnotesize
      \@tempdima\footnotesep
      \advance\@tempdima -\ht\strutbox
      \ifdim\@tempdima<\z@ \@tempdima\z@ \fi
      \advance\@tempdima.5\normalbaselineskip
      \MFL@xparaskip % Add extra para skip
      \xdef\MFL@paraskip{\the\@tempdima\relax}%
    \endgroup
  }
}
```

Finally, we implement `para*` option which suppresses indentation of `para` footnotes.

```
\DeclareOption{para*}{%
  \@ifundefined{MFL@startpara}{\ExecuteOptions{para}}{}%
  \MFL@paraindentfalse
}
```

10.4 Perpage Option

The `perpage` option just sets the true value for the `MFL@perpage`:

```
\newif\ifMFL@perpage \MFL@perpagefalse
\DeclareOption{perpage}{\MFL@perpagetrue}
```

Now we can process the package options:

```
\ProcessOptions\relax
```

After that, we test the `MFL@perpage` and load the `perpage` package on demand:

```
\ifMFL@perpage \RequirePackage{perpage}\fi
```

10.5 Additional Footnotes Support

`\MFL@list` We initialize the list of all additional footnote levels to be empty.

```
\def\MFL@list{}
```

Its items will have the form `\@elt{`⟨*style*⟩`}`⟨*insert register*⟩

`\SelectFootnoteRule` Next we implement the footnote rule selection command. It defines the `\MFL@rule` command that is later used in the `\MFL@newinsert` command to specify accompany footnote level rule.

```
\newcommand*{\SelectFootnoteRule}[2][0]{%
  \edef\@tempa{\noexpand\MFL@selectrule{#1}{%
    \expandafter\noexpand\csname #2footnoterule\endcsname}}%
  \@ifnextchar[{\@tempa}{\@tempa[]}%
}
\def\MFL@selectrule#1#2[#3]{\def\MFL@rule{\MFL@joinrule{#1}{#2}{#3}}}
\SelectFootnoteRule{extra}% Set the default footnote rule
\@onlypreamble\SelectFootnoteRule
\@onlypreamble\MFL@selectrule
\@onlypreamble\MFL@rule
```

`\SetFootnoteHook` `\SetFootnoteHook{`⟨*hook*⟩`}` saves a hook in the internal command. When a new footnote is created, this hook is applied to it.

```
\newcommand{\SetFootnoteHook}[1]{\def\MFL@footnotehook{\MFL@fhook{#1}}}
\@onlypreamble\SetFootnoteHook
\@onlypreamble\MFL@footnotehook
\SetFootnoteHook{}% Empty hook by default
```

`\MFL@fhook` `\MFL@fhook{`⟨*hook*⟩`}{`⟨*insert register*⟩`}` associates a hook with the given insert register and resets the current hook.

```
\long\def\MFL@fhook#1#2{%
  \expandafter\def\csname MFL@hook\string#2\endcsname{#1}%
  \SetFootnoteHook{}%
}
\@onlypreamble\MFL@fhook
```

`\newfootnote` Then we implement the basic command which generates additional footnote levels.

```
\newcommand*{\newfootnote}[2][plain]{%
```

Firstly, we test the ⟨*style*⟩ to be valid.

```
  \@ifundefined{MFL@fnote#1}{%
    \PackageError{manyfoot}{Unknown footnote style #1}%
      {Known styles are `plain' and `para'\MessageBreak
       (if the package was loaded with the para or para* option)}}{}%
```

Then we allocate and initialize a new insert

```
  \expandafter\MFL@newinsert\csname footins#2\endcsname
```

generate `\Footnotetext`⟨*suffix*⟩ and its hook

```
  \edef\@tempa{\noexpand\newcommand
    \expandafter\noexpand\csname Footnotetext#2\endcsname
      {\expandafter\noexpand\csname MFL@fnote#1\endcsname{%
        \expandafter\noexpand\csname footins#2\endcsname}}%
    \noexpand\MFL@footnotehook{%
      \expandafter\noexpand\csname footins#2\endcsname}%
  }%
  \@tempa
```

and finally add the description of this insert to the list of additional footnote inserts.

```
  \@cons\MFL@list{{#1}\csname footins#2\endcsname}%
```

```
}
\@onlypreamble\newfootnote
```

`\MFL@newinsert` The initialization of a new insert. A current rule selection command is linked with the new insert by the insert count number.

```
\def\MFL@newinsert#1{\newinsert#1%
  \expandafter\let\csname MFL@join\number #1\endcsname \MFL@rule
  \SelectFootnoteRule{extra}% Reset to default rule again
  \skip#1\skip\footins \dimen#1\dimen\footins \count#1\count\footins
}
\@onlypreamble\MFL@newinsert
```

`\MFL@makemark` All additional footnote mark commands with automatic numbering are based on the following command:

`\MFL@makemark{`⟨*counter*⟩`}{`⟨*stepcounter*⟩`}{`⟨*command*⟩`}[`⟨*number*⟩`]`

This command tests an optional parameter and, if it exists, prepares the marker using the specified number. Otherwise, it at first executes the ⟨*stepcounter*⟩`{`⟨*counter*⟩`}` command (it is either `\stepcounter` or `\@gobble`) and then makes a mark. Finally, it executes the ⟨*command*⟩ parameter. Check for `multiple` footnotes added as suggested by Frank Mittelbach.

```
\def\MFL@makemark#1#2#3{%
  \FN@mf@check
  \@ifnextchar[{\MFL@xmkmark{#1}{#3}}{#2{#1}\MFL@mkmark{#1}{#3}}%
}
\providecommand\FN@mf@check{}
\def\MFL@xmkmark#1#2[#3]{%
  \begingroup
    \csname c@#1\endcsname #3\relax
    \unrestored@protected@xdef\@thefnmark{\csname the#1\endcsname}%
  \endgroup
  #2%
}
\def\MFL@mkmark#1#2{\protected@xdef\@thefnmark{\csname the#1\endcsname}%
  #2%
}
```

`\DeclareNewFootnote` Now we define the service command simplifying creation of footnotes in almost all cases.

```
\newcommand*{\DeclareNewFootnote}[2][plain]{%
  \@ifnextchar[{\MFL@declare{#1}{#2}}{\MFL@declare{#1}{#2}[arabic]}%
}
\def\MFL@declare#1#2[#3]{%
```

We start from creation a new footnote level:

```
  \newfootnote[#1]{#2}%
```

Now we prepare the `\@tempa` command which will create other commands at the end of macro. A counter creation command is prepared at first:

```
  \edef\@tempa{\noexpand\newcounter{footnote#2}%
```

After that we prepare the redefinition of the enumeration style

```
    \noexpand\renewcommand
    \expandafter\noexpand\csname thefootnote#2\endcsname{%
      \expandafter\noexpand\csname @#3\endcsname
      \expandafter\noexpand\csname c@footnote#2\endcsname
    }%
```

and specify the per-page resetting if necessary:

```
    \ifMFL@perpage \noexpand\MakePerPage{footnote#2}\fi
```

Next we prepare the `\footnote`⟨*suffix*⟩ command:

```
    \noexpand\newcommand
    \expandafter\noexpand\csname footnote#2\endcsname{%
      \noexpand\MFL@makemark{footnote#2}{\noexpand\stepcounter}{%
        \noexpand\@footnotemark
        \noexpand\let\noexpand\@tempb\noexpand\@thefnmark
        \expandafter\noexpand\csname Footnotetext#2\endcsname{%
          \noexpand\@tempb
        }%
      }%
    }%
```

After that we prepare the `\footnotemark`⟨*suffix*⟩ command:

```
    \noexpand\newcommand
    \expandafter\noexpand\csname footnotemark#2\endcsname{%
      \noexpand\MFL@makemark{footnote#2}{\noexpand\stepcounter}{%
        \noexpand\@footnotemark
      }%
    }%
```

Then we prepare the `\footnotetext`⟨*suffix*⟩ command:

```
    \noexpand\newcommand
    \expandafter\noexpand\csname footnotetext#2\endcsname{%
      \noexpand\MFL@makemark{footnote#2}{\noexpand\@gobble}{%
        \noexpand\let\noexpand\@tempb\noexpand\@thefnmark
        \expandafter\noexpand\csname Footnotetext#2\endcsname{%
          \noexpand\@tempb
        }%
      }%
    }%
```

Finally, we provide suffixed equivalents for `\Footnotemark` and `\Footnote` commands:

```
    \noexpand\newcommand
    \expandafter\noexpand\csname Footnotemark#2\endcsname{%
      \noexpand\Footnotemark
    }%
    \noexpand\newcommand
    \expandafter\noexpand\csname Footnote#2\endcsname[1]{%
      \noexpand\Footnotemark{####1}%
      \expandafter\noexpand\csname Footnotetext#2\endcsname{####1}%
```

```
    }%
  }%
```

And now all prepared commands are created:

```
  \@tempa
}
\@onlypreamble\DeclareNewFootnote
\@onlypreamble\MFL@declare
```

`\MFL@start` This command executes the `\MFL@start`⟨*suffix*⟩ command. It works at the preamble of the document only once for every additional footnote level.

```
\def\MFL@start#1{\csname MFL@start#1\endcsname}
\@onlypreamble\MFL@start
```

10.6 The Basic Implementation Part

Now we have to build into LaTeX 2_ε the support for additional footnote levels. There are a number of points where the modifications should be done. We do all real modifications at the beginning of the document. Here we prepare macros needed.

10.6.1 Modifications of Output Routine

`\MFL@joinnotes` First point of modifications is the output routine. We choose the strategy of joining of the additional footnotes with `\footins` at the points where it is really needed. The procedure `\MFL@joinnotes` implements this job. It will be added later to the beginning of `\@makecol` and `\@doclearpage` macros.[1]

Since version 1.5, the `\MFL@joinnotes` has a parameter — a command the joining procedure is called before. We add a protection from double attempt for joining and restore the `\footnoterule` after execution of the parameter command. The `MFL@joined` conditional command is used for this purpose. Another improvement is concerned with the footnote rule customization procedure introduced. We now manage the footnote rule to be inserted before levels using the priorities. The default priority for the standard `\footnoterule` is defined with the `\footnoterulepriority` command.

```
\newcommand{\footnoterulepriority}{1}
\newif\ifMFL@joined \MFL@joinedfalse
\def\MFL@joinnotes#1{%
  \ifMFL@joined #1%
  \else
    \let\MFL@savedrule \footnoterule
```

After saving a footnote rule we test the standard footnote insert, and prepare the current footnote rule for additional levels.

[1] The version 1.2 of this package added this procedure to the beginning of `\@specialoutput` instead of `\@doclearpage`. It was incorrect because the special output routine is often called to add the next float to the output page without ejecting it. As a result the additional footnotes disappeared before floats. Thanks to François Patte who found this bug.

```
    \let\MFL@currule \defaultfootnoterule
    \ifvoid \footins
      \let\MFL@curpriority \footnoterulepriority
    \else
      \let\MFL@curpriority \m@ne
    \fi
```

Now we join inserts.

```
    \let\MFL@elt\@elt
    \let\@elt\MFL@join \MFL@list
    \let\@elt\MFL@elt
```

And finally, we process the parameter command and restore the `\footnoterule`.

```
    \MFL@joinedtrue #1\MFL@joinedfalse
    \let\footnoterule \MFL@savedrule
  \fi
}
```

The insert joining procedure simply calls the numbered join command linked with the insert.

```
\def\MFL@join#1#2{\csname MFL@join\number #2\endcsname{#1}{#2}}
```

The last one calls the `\MFL@joinrule` command with 5 parameters:

`\MFL@joinrule{`⟨*priority*⟩`}{`⟨*rule*⟩`}{`⟨*action*⟩`}{`⟨*style*⟩`}{`⟨*register*⟩`}`

First we select the rule comparing the priority of current rule and the new one.

```
\def\MFL@joinrule#1#2#3#4#5{%
  \ifnum #1<\MFL@curpriority \else
    \let\MFL@currule#2%
    \def\MFL@curpriority{#1}%
  \fi
  \ifvoid#5\else
```

Well. The current insert is nonempty. At first, we calculate the skip of insert. Within `multicols` environment it has the size multiplied in `\col@number` times matching to its natural size. Therefore, we have to divide it by `\col@number` in such a case.

```
    \@tempskipa\skip#5%
    \MFL@ifmcol{\divide\@tempskipa\col@number}{}%
    \edef\MFL@skip{\vskip\the\@tempskipa\relax}%
```

Now we process the current insert by the style processing driver

```
    \csname MFL@process#4\endcsname #5%
```

and finally we join it with `\footins` insert and decrease the current priority to `-1`.

```
    \ifvoid\footins
      \let\footnoterule\MFL@currule
      \setbox\footins\vbox{#3\unvbox#5}%
    \else
      \setbox\footins\vbox{%
```

```
        \unvbox\footins\MFL@skip\MFL@currule#3\unvbox#5%
      }%
    \fi
    \let\MFL@curpriority \m@ne
  \fi
}
```

`\MFL@reinsout` When the special output is called to process float insertion, all accumulated footnotes should be reinserted after the output box. This job carries out `\@reinserts` command. We will add to it the reinsertion of all additional footnotes with the help of `\MFL@reinsout` macro. Note that `\@reinserts` command is called at two points: when the float is the marginal note (`\count\@currbox` = 0) or when it is the real float. At the second case we must take into account the height of all additional footnotes by adding it to the `\@pageht` value.

```
\def\MFL@reinsout#1#2{\ifvoid#2\else
    \ifnum\count\@currbox>\z@
      \advance\@pageht \ht#2%
      \advance\@pageht \skip#2%
      \advance\@pageht \dp#2%
    \fi
    \insert#2{\unvbox#2}%
  \fi
}
```

10.6.2 Minipages Support

`\MFL@reinsert` The command reinserts all additional footins by adding if necessary the empty insertion (such a way is used in `multicol`). It is used in `minipage` and `multicols` environments. The point of using it at a minipage is the beginning of the minipage. Using this command we release the additional footnote boxes to accumulate footnotes inside the minipage.

```
\def\MFL@reinsert{{\let\@elt\MFL@reins \MFL@list}}
\def\MFL@reins#1#2{\ifvoid#2\else\insert#2{}\fi}
```

`\MFL@mpinsert` Here we define the minipage insertion command which manually adds vbox to the insertion box. The last footnote within minipage can split. So, we specially enclose it into vbox and unvbox the previous last footnote.

```
\long\def\MFL@mpinsert#1#2{%
  \global\setbox#1\vbox{%
    \unvbox#1\setbox\@tempboxa\lastbox
    \ifvbox\@tempboxa \unvbox\@tempboxa \fi
    \vbox{#2}%
  }%
}
```

`\MFL@mpreinsert` This macro is useful when we really insert footnotes at the end of the minipage. We suppress splitting of all minipage insertions except the last one. To do this we

extract the last box from the insertion box, then put another footnotes into the insert enclosing them into vbox, and then put the last unvboxed footnote.

```
\def\MFL@mpreinsert#1#2{%
  \ifvoid#2\else
    \setbox\@tempboxa\vbox{\unvbox#2\global\setbox#2\lastbox}%
    \setbox\z@\box#2%
    \ifdim\ht\@tempboxa>\z@ \MFL@realinsert#2{\box\@tempboxa}\fi
    \MFL@realinsert#2{\unvbox\z@}%
  \fi
}
```

Then we define two hooks which will be added to the beginning and to the end of a minipage. We do them in not inner mode only (for the first level minipages).

`\MFL@minipage` We release all box registers of the additional inserts at the beginning of minipage to use them inside the minipage to accumulate inner inserts.

```
\def\MFL@minipage{%
  \ifinner\else
    \MFL@reinsert \let\MFL@insert\MFL@mpinsert
  \fi
}
```

`\MFL@endminipage` We simply reinsert all footnotes at the end of the first level minipage.

```
\def\MFL@endminipage{%
  \ifinner\else
    {\let\@elt\MFL@mpreinsert \MFL@list}%
  \fi
}
```

10.6.3 Multicol Package Support

`\MFL@mult` The command modifies parameters of the insert register. It is useful in the scope of `multicol` package only.

```
\def\MFL@mult#1#2{%
  \multiply\count#2\col@number
  \multiply\skip#2\col@number
}
```

`\MFL@ifmcol` The next macro tests the multicolumn mode. There are two conditions which have to be satisfied if we are in the multicolumn mode: the value of `\col@number` should be greater then 1 and the value of `\footins` count should be at least 2000.

```
\def\MFL@ifmcol#1#2{\@tempswafalse
  \ifnum\col@number>\@ne
    \ifnum\count\footins>1999 \@tempswatrue \fi
  \fi
  \if@tempswa #1\else #2\fi
}
```

10.7 What Do We Do at the Beginning of Document?

```
\AtBeginDocument{%
```

Firstly, we process starting commands for every level

```
  {\let\@elt\MFL@start \MFL@list}
```

Then we define the `\defaultfootnoterule` to provide compatibility with **footmisc**. We set it equal to `\pagefootnoterule` or `\footnoterule`.

```
  \@ifundefined{defaultfootnoterule}{%
    \@ifundefined{pagefootnoterule}%
      {\let\defaultfootnoterule\footnoterule}%
      {\let\defaultfootnoterule\pagefootnoterule}%
  }{}%
```

Then we modify `\@doclearpage` and `\@makecol` commands by added the joining algorithm at their beginning.

```
  \let\MFL@doclearpage\@doclearpage
  \def\@doclearpage{\MFL@joinnotes\MFL@doclearpage}
  \let\MFL@makecol\@makecol
  \def\@makecol{\MFL@joinnotes\MFL@makecol}
```

Then we modify `\@reinserts` command of the output routine to process reinsertion of all additional footnotes.

```
  \g@addto@macro\@reinserts{%
    \let\MFL@elt\@elt
    \let\@elt\MFL@reinsout \MFL@list
    \let\@elt\MFL@elt
  }
```

Then we execute `\MFL@floathook` and add it into `\@floatplacement` command which is called when the column mode is changed. One important note: in the multicolumn mode of `multicol` package the width of footnotes is unchanged. So, we test this case by the command `\MFL@ifmcol`.

```
  \MFL@floathook
  \g@addto@macro\@floatplacement{\MFL@ifmcol{}{\MFL@floathook}}
```

The next is the `minipage` environment. We modify `\@iiiminipage` and `\endminipage` adding to them hooks describe earlier.

```
  \let\MFL@iminipage\@iiiminipage
  \def\@iiiminipage{\MFL@minipage\MFL@iminipage}
  \g@addto@macro\endminipage\MFL@endminipage
```

Finally, we do some tricks to provide the compatibility with `multicol` package. If this package is loaded, the command `\multi@column@out` should be defined

```
  \@ifundefined{multi@column@out}
```

If it is undefined, the `multicol` specific commands are not useful. So, we delete `\MFL@mult` command and modify `\MFL@ifmcol` command to choose the second case every time.

```
    {\@onlypreamble\MFL@mult \let\MFL@ifmcol\@secondoftwo}
```

If `multicol` package presents, we add the joining algorithm to the beginning of `\multi@column@out` command

```
    {\let\MFL@mcolout\multi@column@out
```

```
      \def\multi@column@out{\MFL@joinnotes\MFL@mcolout}
```

and add the multiplication of additional footins parameters by the number of columns to the end of `\init@mult@footins` command. We can't do this globally. So, we save the previous value of `\@elt` command and then restore it after the calculation. We also modify `\reinsert@footnotes` command.

```
      \g@addto@macro\init@mult@footins{%
        \let\MFL@elt\@elt
        \let\@elt\MFL@mult \MFL@list
        \let\@elt\MFL@elt
      }
      \g@addto@macro\reinsert@footnotes{\MFL@reinsert}
    }
}
```

⟨/package⟩