

The **bigfoot** package

version 2.1

David Kastrup*

2015/08/30

Purpose of this package is to provide a one-stop solution to almost all problems related to footnotes. You can use it as a drop-in replacement of the **manyfoot** package, but without many of its shortcomings, and quite a few features of its own. It uses the existing document class layouts for footnotes, so you can usually use it without having to worry about the looks.

Features are:

- You can specify and use multiple footnote apparatus. Footnotes for an apparatus lower on the page^a can be anchored in an apparatus¹ that is higher on the page.
- The last footnote in each apparatus may be broken to the next page.² Any subordinate footnote anchors that get moved to the next page will take the corresponding footnote with them.
- The order of footnotes in an apparatus is ‘natural’: it starts with any footnote that may have been broken from the next page, followed by footnotes from the current page in the order of the appearance of their footnote marks.^b Where the order of appearance in the document differs from the order in the source code, you will usually want to use the **\MakeSorted** command from the **perpage** package to get the numbering fixed appropriately.

*dak@gnu.org ¹The plural of “apparatus” is actually “apparatus”^c

²This will probably be interesting for footnotes that contain stuff like math equations^d or lists^e.

^alike this one ^bThis footnote appears above notes on notes.

^cWell, actually “apparātūs” with a long “ū”, but that’s just obvious in spoken Latin.

^dLike

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} \tag{1}$$

^eLike

- This, or
- this.

- Footnotes can be formatted in separate paragraphs, or be run into a single paragraph. The choice is made per footnote apparatus, but can be overridden for single footnotes.³
- If footnotes are run into one paragraph, a variety of criteria makes sure that this formatting is only chosen when it saves noticeable space and delivers visually attractive results.
- Parameters for footnote formatting can be specified globally, or separately for each footnote.
- The material in footnotes can contain `\verb`-like material without problems.⁴
- You can use color in footnotes. If a footnote gets broken across pages, the color at the point of the break will get resumed on the next page. Actually, the whole color stack will get reinstated.

As an example of how simple the usage can be, here is the documentation driver for this document:

```
1 \*driver
2 \documentclass{ltxdoc}
3 \usepackage{bigfoot}
4 \usepackage{tabularx}
5 \usepackage{hyperref}
```

After loading the packages, we declare two footnote blocks. One is the default footnote block, another block is called `B` and is numbered with letters. The letters start new on each page. Both footnote blocks default to in-paragraph footnotes. Since the block `B` can get entries from both the main text as well as the default footnote block, the entries are not necessarily generated in page order. So we need to use a sorted counter to fix this (feel free to try what happens when using an unsorted counter).

```
6 \DeclareNewFootnote[para]{default}
7 \DeclareNewFootnote[para]{B}[alph]
8 \MakeSortedPerPage{footnoteB}
```

In addition, we add an alternate footnote sequence that can be interspersed with the normal footnotes by use of the `\footnote` command which we effectively define here.

```
9 \newcounter{footalt}
10 \def\thefootalt{\fnsymbol{footalt}}
11 \MakeSortedPerPage[2]{footalt}
12 \WithSuffix\def\footnotedefault'\{\refstepcounter{footalt}}%
13 \Footnotedefault{\thefootalt}}
```

³I.e., footnotes with display math^a or list environments^b have to be done in vertical mode.

⁴We wrote `\verb`-like above in the main text.^c

^aWe had this already, right? ^bAnd this looks familiar, too.

^cWell, this is not so impressive. But we wrote `\verb+|\verb|-like+` in the footnote then.

Actually, that already was all. We can now start the document. The following makes sure that we get the full documentation only by compiling the separate driver file:

```

14 \begin{document}
15 \OnlyDescription
16 <driver> \AlsoImplementation
17 \DocInput{bigfoot.dtx}
18 \end{document}
19 </driver>

```

In order to be useful without additional hassle, the normal footnote level will be called `default`. If no such style has been defined at the start of the document, it will get defined and used for ordinary footnotes, fixing quite a few problems of L^AT_EX's own footnote placement algorithms.

Apart from that, usage is very much like that of `manyfoot`, so for the customization possibilities of `bigfoot` with regards to multiple footnote blocks and rules between them, refer to `manyfoot`'s documentation.

`bigfoot` contains a lot of bells and whistles for defining your footnote formats and can use different formats for different footnote blocks. Those expert options are not documented separately yet: look through the code sections to see them explained.

1 The implementation

1.1 Startup code

We declare the package and several compatibility options supposed to make `bigfoot` a drop-in replacement for `manyfoot`.

```

20 <*style>
21 \NeedsTeXFormat{LaTeX2e}
22 \ProvidesPackage{bigfoot}[2015/08/30 2.1 makes footnotes work]
23
24 \DeclareOption{para}{\PackageInfo{bigfoot}{Compatibility option 'para'
25   has no effect:^^J%
26   Spacing will be guessed from '\string\@makefnhtext' unless^^J%
27   '\string\@preparefnhtext' is redefined}}
28
29 \DeclareOption{para*}{\PackageInfo{bigfoot}{Compatibility option
30   'para*':^^J%
31   Redefining '\string\@preparefnhtext'}%
32   \def\@preparefnhtext{\ifx\@thefnmark\@empty
33     \else\@makefnmark\nobreak\fi}}
34
35 \DeclareOption{ruled}{\PassOptionsToPackage{ruled}{manyfoot}}

```

The normal processing makes footnote text macros allow verbatim and similar. We call this `robust` processing though it is not totally accurate. This is the default. There is also an option `fragile` which will not allow this, but may be required

for some definitions of `\@makefn`text. It turns out that most document classes work with the ‘robust’ option, but there might be some that fail, and there might be some footnote-modifying packages that also can cause failure.

```
36 \DeclareOption{robust}{\def\FN@makefncall{\FN@makefnrobust}}
37 \DeclareOption{fragile}{\def\FN@makefncall{\FN@makefnfragile}}
38 \ExecuteOptions{robust}
```

The `verbose` option talks about changed labels at the end of a run. It is for debugging instable configurations that fail to converge after a number of $\text{\TeX}$ runs. The output is probably obscure.

```
39 \DeclareOption{verbose}{\AtBeginDocument{%
40   \def\@testdef #1#2#3{%
41     \def\reserved@a{#3}%
42     \expandafter \ifx \csname #1@#2\endcsname
43       \reserved@a
44     \else \@tempswattrue
45       \typeout{Changed label #1/#2:
46         \csname #1@#2\endcsname->#3}%
47     \fi}}}
```

The `trace` option is only available if you used `docstrip` while explicitly requesting `trace` functionality. If you set the `trace` option, the next option specifies a bit map of trace bits.

- 1 break decisions
- 2 horizontal box building
- 4 allocation stuff
- 8 output routine stuff
- 16 retained and kept boxes

The following bits can be set in tracing:

```
48 <trace>\DeclareOption{trace}{%
49 <trace>\DeclareOption*{\ftflags=\CurrentOption\relax
50 <trace> \DeclareOption*{\OptionNotUsed}}%
51 <trace> \AtEndOfPackage{\RequirePackage{trace}\relax
52 <trace> \errorcontextlines\maxdimen
53 <trace> \showboxdepth4
54 <trace> \showboxbreadth100
55 <trace> \tracingonline=\@ne}%
56 <trace>}
```

The `tracepage` option is followed by another option specifying the page to be traced. If you use it, tracing happens only on the specified page. Only a single page can be specified.

```
57 <trace>\def\FN@tracepage{\c@page}
58 <trace>\DeclareOption{tracepage}{%
59 <trace> \DeclareOption*{\edef\FN@tracepage{\CurrentOption}%
60 <trace> \DeclareOption*{\OptionNotUsed}}
61 <trace>\newcount\ftflags
62 <trace>\def\foottrace#1{\ifnum\numexpr(\ftflags+(#1))/(2*#1)*(2*#1)%
63 <trace> =\numexpr(\ftflags+3*#1/2)/#1*#1\relax
64 <trace> f\else t\fi\ifnum\FN@tracepage=\c@page t\else n\fi}
65 \ProcessOptions*
```

`hyperref`'s footnote support will just cause trouble. So if `hyperref` was already loaded or is going to be loaded, we turn off its footnote support. If you think you know what you are doing, you can use `\hypersetup` to turn it on again before the start of the document. Unfortunately, it appears like `\hypersetup` refuses to be called more than once, so this actually does not work unless you load `hyperref` last.

```
66 \ifx\hypersetup\@undefined
67   \PassOptionsToPackage{hyperfootnotes=false}{hyperref}
68 \else
69   \hypersetup{hyperfootnotes=false}
70 \fi
```

We require the `etex` package because

1. We need the facilities of the ε -TeX engine; and where they are not available, the error messages from not finding the `etex` package or from loading it into the wrong engine make much more sense than what would happen otherwise.
2. We allocate quite a few registers, and the danger of running out of them is smaller when the extra registers of ε -TeX are taken into account. Now unfortunately the LaTeX team has decided in 2015 to do its own extended allocation scheme incompatible with the `etex` package, so we need to guard against this load in case the new LaTeX allocation scheme is detected.

We need the `manyfoot` package to build on. The `suffix` and `perpage` package are needed for some small stuff.

```
71 \ifx\@alloc\@undefined
72 \RequirePackage{etex}
73 \fi
74 \RequirePackage{manyfoot}
75 \RequirePackage{suffix}
76 \RequirePackage{perpage}
```

1.2 Fixes to the `manyfoot` package

While those fixes have been submitted once to the author of `manyfoot`, they have not made it into its distribution at the current point of time. In the interest of stability, it would probably be best just to incorporate the parts from `manyfoot` that get used by `bigfoot`. This has not yet been done.

`\MFL@reinsout` We need the appropriate splitting parameters set for the footnote again. `\MFL@realinsert` does that, but it has the disadvantage that it uses `\strutbox`, and that may be set to arbitrary values at the time the output routine is invoked. `manyfoot` already has this problem with minipages: the split sizes will be those of the font at the end of the minipage instead of those at the time the footnote body was set up. So we do this here, and see later for more info about how to do this right:

```
77 \def\MFL@reinsout#1#2{\ifvoid#2\else
78   \ifnum\count\@currbox>\z@
```

```

79      \advance\@pageht \ht#2%
80      \advance\@pageht \skip#2%
81      \advance\@pageht \dp#2%
82      \fi
83      \MFL@realinsert{#2}{\unvbox#2}%
84      \fi
85 }

```

`\MFL@reins` Actually, I don't get the purpose of the following line in the first place. But if we do need it for some reason, it is rather certain that we don't want this empty insert to float. Use `\MFL@realinsert`, or set the floatingpenalty the hard way.

```

86 \def\MFL@reins#1#2{\ifvoid#2\else\insert#2{\floatingpenalty\@MM}\fi}

```

`\MFL@mpinsert` The structure of the `\MFL@mpinsert` box is overly complicated, and it is a bad
`\MFL@minipage` idea to unpack the boxes put into it too early: the `\lastbox` command is pretty inefficient when the list before it is long due to unpacking. So we just leave everything packed in its own boxes, and unpack only at the moment when we are reinserting.

```

87 \long\def\MFL@mpinsert#1#2{%
88   \global\setbox#1\vbox{%
89     \unvbox#1%
90     \nointerlineskip
91     \vbox{#2}%
92   }%
93 }
94
95 \def\FN@divert{%
96   \let\MFL@mpinsertsave\MFL@insert
97   \MFL@reinsert \let\MFL@insert\MFL@mpinsert}
98 \def\FN@enddivert{\let\@elt\MFL@mpreinsert \MFL@list}}
99
100 \def\MFL@minipage{\ifinner\else \FN@divert\fi}
101 \def\MFL@endminipage{\ifinner\else \FN@enddivert\fi}

```

`\MFL@mpreinsert` When reinserting, we put all but the last insertion into one humongous blob. This is so that the last insertion can be split by $\text{\TeX}$'s paragraph splitting routines. The footnote types that `bigfoot` supports will never get split by $\text{\TeX}$, anyhow, but it is conceivable that other extension packages for `manyfoot` work differently. There is one difference, though: we let a slave mark escape into the main vertical list.

```

102 \def\MFL@mpreinsert#1#2{%
103   \ifvoid#2\else
104     \setbox\@tempboxa\vbox\bgroup\unvbox#2%
105     \global\setbox#2\lastbox
106     \setbox\z@\lastbox
107     \ifvoid\z@
108       \egroup
109       \setbox\z@\box#2%

```

```

110     \else
111         \MFL@removevboxes \unvbox\z@
112         \egroup
113         \setbox\z@\box#2%
114         \MFL@mpinsertsave#2{\unvbox\@tempboxa}%
115     \fi
116     \ifvoid\z@\else
117         \MFL@mpinsertsave#2{\unvbox\z@}%
118     \fi
119     \marks\FN@slave{\number\FN@id}%
120 \fi}

```

`\MFL@removevboxes` This trick works like `\removehboxes` in the `TEXbook`'s appendix D.

```

121 \def\MFL@removevboxes{{\setbox\z@\lastbox
122     \ifvbox\z@ \MFL@removevboxes \unvbox\z@\fi}}

```

`\NCC@makefnmark` This provides the command in case it is not present (some versions did not have it).

```

123 \ifx\NCC@makemark\@undefined
124 \ifx\NCC@makefnmark\@undefined \else
125     \def\NCC@makemark{\NCC@makefnmark}
126 \fi
127 \fi

```

While the above operations actually were fixes to `manyfoot`, now we actually patch it for our own purposes. When allocating a new footnote, we set its maximum dimension to `\maxdimen` (since no hard limit makes sense, given that we recalculate all respective sizes at output time) and allocate a cache box to go with it. We also add the insertion to the list of insertions in `\FN@nestlist`.

```

\MFL@startplain
\MFL@startpara 128 \def\MFL@startplain#1{\global\dimen#1\maxdimen
129     \@cons\FN@nestlist{{}\#1}%
130     \expandafter\expandafter\expandafter\newbox\FN@cache#1}
131
132 \let\MFL@startpara\MFL@startplain

```

`\RestyleFootnote` This macro gets two arguments: a footnote *<type>*, and the style to be used for it. It works by redefining the corresponding `Footnotetext<type>` macro.

```

133 \def\RestyleFootnote#1#2{\expandafter\xdef
134     \csname Footnotetext#1\endcsname{\expandafter
135         \noexpand\csname MFL@fnote#2\endcsname{\csname footins#1\endcsname}}}

```

`\FN@stripfootins` We need the same kind of functionality for a footnote specified by its footnote insertion. So we strip the footnote *<type>* from the insertion macro name. Kind of ugly.

```

136 \expandafter\def\expandafter\FN@stripfootins\string\footins{}
137
138 \def\FN@restylefootnote#1#2{{\edef\next{%

```

```

139     \noexpand\RestyleFootnote{\expandafter\FN@stripfootins
140       \string#1}{#2}}\next}}

```

1.3 Dealing with footnote-specific code

The formatting of footnotes is determined by macros such as `\@makefntext`. For several blocks of footnotes, we might want to have several different ways for formatting them. Whenever this is the case, we call them with

```
\FN@specific{<insert#>}{<macroname>}
```

This will use the default `<macroname>` unless a special macro has been defined with something like

```
\FootnoteSpecific\marg{type}...
```

A number of other defining commands and constructs are available: those are pretty much like the ones for the `\WithSuffix` command implemented by the `suffix` package.

`\FN@specific` We use `\romannumeral` here just for the purpose of sustaining expansion. It expands to nothing when followed by `\z@` eventually. Thus expanding the expansion of `\FN@specific` again delivers the (unexpanded) final token to use.

```

141 \def\FN@specific#1#2{\romannumeral
142   \ifcsname FN\string#2\number#1\endcsname
143     \expandafter
144       \z@\csname FN\string#2\number#1\expandafter\endcsname
145   \else\expandafter\z@
146     \expandafter#2\fi}

```

`\FootnoteSpecific` This is all a bit muddy, but quite similar to what the `suffix` package does, so you might want to look there for the explanation.

```

\FN@specific@ii
147 \def\FootnoteSpecific#1{\count@\csname footins#1\endcsname\toks@{}}%
148   \FN@specific@ii}
149
150 \long\def\FN@specific@ii#1#2{\toks@\expandafter{\the\toks@#1}%
151   \the\expandafter\toks@
152   \csname FN\string#2\number\count@\endcsname}
153
154 \WithSuffix\def\FN@specific@ii\long{\toks@\expandafter
155   {\the\toks@\long}\FN@specific@ii}
156
157 \WithSuffix\def\FN@specific@ii\global{\toks@\expandafter
158   {\the\toks@\global}\FN@specific@ii}
159
160 \WithSuffix\def\FN@specific@ii\expandafter{\expandafter
161   \FN@specific@ii\expandafter}

```

1.4 Putting footnotes into insertions

1.4.1 Dealing with Ids

Since we have to store additional information for each footnote as long as it is not yet typeset, we allocate and deallocate numeric ‘id’s on an as-needed base, since we do not want to store this sort of information indefinitely, with a large toll on hash space. So we work with indirect ids, where the unique ids are just referenced indirectly. We do this with ‘slots’.

`\FN@slotxdef` New slots are assigned values with `\FN@slotxdef`, which can be retrieved again with `\FN@slotget`.

```

162 \def\FN@slotxdef#1{%
163   \global\expandafter\xdef\csname FN@slot#1\endcsname}
164
165 \def\FN@slotget#1{\csname FN@slot#1\endcsname}
166 \trace\def\FN@slotget#1{%
167   \trace\expandafter\FN@slotgetii\expandafter
168   \FN@slotfreelist\expandafter
169   {\number\number#1}}
170 \trace\def\FN@slotgetii#1#2{%
171   \trace\ifx#1\@empty \csname FN@slot#2\endcsname\else
172   \trace\ifnum#1=#2 \errmessage{Use after freed: #1}\else
173   \trace\expandafter\FN@slotgetii
174   \trace\csname FN@slot#1\endcsname{#2}\fi\fi}

```

`\FN@slotfreelist` `\FN@slotfreelist` point to the first already allocated available id to be reused. If `\FN@nextslot` it is empty, none exist. In that case, `\N@nextslot` contains the next slot number to use.

```

175 \def\FN@slotfreelist{}
176 \def\FN@nextslot{1}

```

`\FN@newslot` This allocates a new slot by setting the given macro to a currently unused slot number in decimal form. If there is something left in the freelist, it is assigned, otherwise a new slot gets allocated.

```

177 \def\FN@newslot#1{%
178   \ifx\FN@slotfreelist\@empty
179     \edef#1{\FN@nextslot}%
180     \xdef\FN@nextslot{\number\numexpr \FN@nextslot+\@one}%
181   \else
182     \let#1\FN@slotfreelist
183     \xdef\FN@slotfreelist{\csname FN@slot\FN@slotfreelist\endcsname}%
184   \fi
185   \trace\if\foottrace4\message{^^JAllocated #1^^J}\fi
186 }

```

`\FN@freeslot` This frees a given slot (by number) again by adding it to the freelist.

```

187 \def\FN@freeslot#1{%
188   \trace\if\foottrace4\message{^^JFreeing #1^^J}\fi

```

```

189 \global\expandafter\let\csname FN@slot#1\endcsname=\FN@slotfreelist
190 \xdef\FN@slotfreelist{#1}}

```

1.4.2 Dealing with footnote stacks

Footnote stacks are used for paired footnotes that refer to a text range instead of a single text point. For example, you can use something like

```
Text \var<{was there}is here\var>
```

To have a text variant “was there” for the original passage “is here”, and mark it, say, as “^ais here^a” by employing the `suffix` package suitably. This would anchor the footnote at the start of the passage. It would also be imaginable to implement the syntax

```
Text \var<is here\var>{was there}
```

for anchoring it at the end of the given passage.

```
191 \global\let\FN@stacklist\@empty
```

\DefineFootnoteStack This command is used for defining a footnote stack. It gets a single argument which is the name of the stack and should consist just of ordinary character tokens.

```

192 \def\DefineFootnoteStack#1{%
193   \global\expandafter\let\csname FN@stack@#1\endcsname\@empty
194   \@cons\FN@stacklist{#1}}%
195 }

```

At the end of the document, all stacks are checked to make sure they have been used up completely.

```

196 \AtEndDocument{\FN@checkstacklist}
197
198 \def\FN@checkstacklist{{\let\@elt\FN@checkstack
199   \FN@stacklist}}
200
201 \def\FN@checkstack#1{{\let\@elt\FN@checkstackentry
202   \csname FN@stack#1\endcsname}}
203
204 \def\FN@checkstackentry#1#2#3{%
205   \PackageError{bigfoot}{Unfinished #1 #2 from line #3}%
206   {The specified footnote range is uncomplete}}

```

\PushFootnoteMark This gets one argument, the name of the footnote stack. It pushes the current footnote mark name stored in `\@thefnmark` onto the footnote stack.

```

207 \def\PushFootnoteMark#1{{\let\@elt\relax
208   \expandafter\unrestored@protected\xdef \csname FN@stack@#1\endcsname
209   {\@elt{#1}{\@thefnmark}{\number\inputlineno}\csname
210     FN@stack@#1\endcsname}}}

```

`\PopFootnoteMark` This gets one argument, the name of the footnote stack. It pops the value of `\@thefnmark` from the named footnote stack.

```

211 \def\PopFootnoteMark#1{\expandafter
212   \ifx\csname FN@stack@#1\endcsname\@empty
213     \PackageError{bigfoot}{Empty footnote stack #1}%
214     {The specified footnote type has no uncompleted range}%
215   \else
216     {\let\@elt\FN@firstpop
217      \iffalse{\fi\csname FN@stack@#1\endcsname}}\fi}

218 \def\FN@firstpop#1#2#3{\protected@xdef\@thefnmark{#2}%
219   \let\@elt\relax
220   \expandafter\protected@xdef\csname FN@stack@#1\endcsname{%
221     \iffalse}\fi}

```

1.4.3 Continuation marks

We add a possibility of adding continuation marks. While the box is assembled, immediately before the break, `\FN@beforebreak` gets called, and `\FN@afterbreak` is called at the top of the continuing box.

```

222 \ifx\FN@beforebreak\@undefined
223   \let\FN@beforebreak\@empty
224 \fi
225 \ifx\FN@afterbreak\@undefined
226   \let\FN@afterbreak\@empty
227 \fi

```

1.4.4 The works

`\FN@cache` Cacheboxes cache the typeset forms of the insertion boxes for a certain configuration of footnotes.

```

228 \def\FN@cache#1{\csname FN@cache\number#1\endcsname}

```

`\FN@sortlist` takes the current vertical list and sorts the contained boxes according to their width (which is supposed to contain the sort key).

The algorithm is a pretty straightforward insertion sort with $O(n^2)$ steps. This is the best one can hope for without comparisons across non-adjacent list elements. For presorted lists, the performance will be $O(n)$, and that's what we expect to see for simple cases (and when there are no sortkeys yet). Any negative width will certainly hang the algorithm.

It also happens that $\text{\TeX}$ has a hardwired limit for grouping levels that hits at 255. Oops. We better not have a few hundred footnotes in a single block on one page...

```

229 \def\FN@sortlist{%
230   \setbox\z@\lastbox
231   \ifvoid\z@ \else \FN@sortlist\FN@sortlistii \fi}
232
233 \def\FN@sortlistii{%

```

```

234 \setbox\tw@lastbox
235 \ifvoid\tw@else
236 \ifdim\wd\tw@<\wd\z@ {\FN@sortlistii}%
237 \fi\nointerlineskip\box\tw@\fi\nointerlineskip\box\z@}

```

\FN@sortinsert This function is an `\@elt` function that will sort the given insertion if it is non-empty and if there is no cache box present (which would imply that the insertion had already been sorted previously).

```

238 \def\FN@sortinsert#1#2{\ifvoid\FN@cache#2%
239 \ifvoid#2\else\global\setbox#2\vbox{\unvbox#2%
240 \FN@sortlist}\fi\fi}

```

\FN@maybeinvalidatecache This is called after pulling in additional material from the page. If the material added an insertion, the cache is junk and must be regenerated.

```

241 \def\FN@maybeinvalidatecache#1#2{%
242 \ifvoid#2\else\global\setbox\FN@cache#2=\box\voidb@x\fi}

```

\FN@regeneratecache This unconditionally regenerates one cache box. The structure of a cache box is basically a list of vertical boxes. All but the last such box are packed into a single vertical box which is then followed by the last vertical box.

```

243 \def\FN@regeneratecache#1#2{%
244 \global\setbox\FN@cache#2=%
245 \ifvoid#2%
246 \box\voidb@x
247 \else
248 \vbox{\vbox{\unvcopy#2%
249 \setbox\z@lastbox
250 \def\FN@masterinsert{#2}%
251 \FN@assembleboxes
252 \global\setbox\FN@cache#2\box\z@}%
253 \nointerlineskip \box\FN@cache#2}%
254 \fi}

```

\FN@mayberegeneratecache This regenerates the cache in case the cache box has been voided in order to mark it as invalid.

```

255 \def\FN@mayberegeneratecache#1#2{%
256 \ifvoid\FN@cache#2%
257 \FN@regeneratecache{#2}%
258 \fi}

```

\FN@cachesize This calculates the size impact of a cache box on the current page as a term to be added into a `\glueexpr`-type of expression.

```

259 \def\FN@cachesize#1#2{%
260 \ifvoid\FN@cache#2%
261 \else
262 +\skip#2+(\ht\FN@cache#2+\dp\FN@cache#2)*\count#2/\@m
263 \fi}

```

`\FN@clearcache` This just completely voids a cache register.

```
264 \def\FN@clearcache#1#2{%
265   \global\setbox\FN@cache#2=\box\voidb@x}
```

`\@makefnvtext` Ok, this is one of the parts putting together footnotes in para mode. The footnotes themselves have already been formatted into hboxes (placed there with `\@preparefnhtext` in order to cater for proper indentation). `\@makefnvtext` then formats a single footnote block from horizontal mode pieces (vertical mode pieces are kept as-is). This takes text and typesets it in a single block. To get correct indentation, it breaks before the first footnote and adjusts the clubpenalties to move them to one line lower effectively. `\@makefnvtext` is called in vertical mode, and its argument is typeset in horizontal mode right after a break, inside of `\@makefntext`.

```
266 \def\@makefnvtext#1{%
267   \FN@specific\FN@masterinsert\@makefntext{%
268     \clubpenalties\thr@@\@MM\clubpenalty\z@
269     \vadjust{\nobreak\vskip-\baselineskip}\nobreak\hfill\break#1}}
```

`\@preparefnhtext` This creates appropriate skips to be put before the horizontal material to make the indentation correct with a breakpoint before the footnote as well as when in run-in text. This is run once at the start of each horizontal mode footnote when it is first being typeset, in horizontal mode.

```
270 \ifx\@preparefnhtext\@undefined
271 \def\@preparefnhtext{%
272   \setbox\z@\vbox{\FN@specific\FN@masterinsert\@makefntext{%
273     \unskip\unpenalty\setbox\z@\lastbox
274     \dimen@
275     \ifnum\parshape>\z@
276       \dimexpr\parshapeindent\tw@-\parshapeindent\@ne\relax
277     \else \ifnum\hangafter=\@ne\hangindent \else
278       \ifnum\hangafter=\m@ne -\hangindent
279       \else \z@ \fi\fi\fi
280     \dimen@ii\dimen@
281     \ifhbox\z@ \advance\dimen@-\wd\z@
282     \setbox\z@\hbox{\unhbox\z@}%
283     \advance\dimen@\wd\z@
284     \fi
285     \xdef\FN@tempinfo{\hskip\the\dimen@
286       \vadjust{\nobreak\hskip-\the\dimen@ii\relax}}}%
287   \FN@tempinfo}
288 \fi
```

Now we have in `\FN@tempinfo` the excess width of the label we don't want to preserve when doing in-paragraph footnote setting. A sequence of glue before a label now has to consist of stuff that vanishes at a breakpoint, followed by stuff that remains. We have to have two behaviors for the contents: behavior one is justification at the start of a line, behavior two is justification in the line. When we are at the start of the line, preceding interword space disappears swallowed

and so the natural criterion for distinguishing those cases is this initial line break. This means that we can't avoid artificially adding a line break at the start of such a box. We will back up its height again. Some packages specify a `\hangindent` (I know of no examples where they would actually set `\hangafter` to a value different from its default of 1, or set `\hangindent` to a negative value which would affect the right margin): due to our artificial line at the top, the indent will actually be active for the first line already. We back it out of the actual labels happening at the start of the line. Two-line parshapes have the same effect: the first line is not actually used, and we put the relevant info for the first line into the label. Different right indentation for the first line is something we can't simulate, but again, it should occur rarely. When `\parshape` is active, `\hangindent` is ignored.

```

289 \def\@makefnstartbox{%
290   \ifdefined\setspace@singlespace
291     \def\baselinestretch{\setspace@singlespace}%
292   \fi
293   \reset@font\footnotesize
294   \hsize\MFL@columnwidth \@parboxrestore
295   \interlinepenalty\FN@specific\FN@masterinsert\interfootnotelinepenalty
296   \widowpenalty\FN@specific\FN@masterinsert\footnotewidowpenalty
297   \clubpenalty\FN@specific\FN@masterinsert\footnoteclubpenalty
298   \advance\linepenalty500\relax}
299
300 \def\@makefnendbox{%
301   \widowpenalty\FN@specific\FN@masterinsert\finalfootnotewidowpenalty}
302
303 \newcount\footnotewidowpenalty
304 \footnotewidowpenalty=250
305 \newcount\footnoteclubpenalty
306 \footnoteclubpenalty=250
307 \newcount\finalfootnotewidowpenalty
308 \finalfootnotewidowpenalty=4000

```

`\@makefnvbox` This is the formatting code for a vertical mode footnote box from already set horizontal material. It uses `\@makefnstartbox` for setting up the initial widow/club penalties, and `\@makefnendbox` for preparing the final end. It results in a vbox.

```

309 \ifx\@makefnvbox\@undefined
310   \def\@makefnvbox#1{\vbox{%
311     \@makefnstartbox
312     \clubpenalties\thr@@\MM\clubpenalty\z@
313     \let\@thefnmark\@empty
314     \FN@specific\FN@masterinsert\@makefntext{\rule\z@\footnotesep
315       \nobreak
316       #1\@finalstrut\strutbox
317     \@makefnendbox}}}}
318 \fi

```

`\hfootfraction` Those parameters govern when a footnote block is going to be set completely in vertical mode. If a footnote block does not shrink to less than `\hfootfraction`

its size when using in-paragraph notes or has at least `\vtypefraction` of forcedly vertical footnotes (specified as purely vertical, or vertical because of being large), it is set entirely in vertical mode.

```
319 \def\hfootfraction{0.9}
320 \def\vtypefraction{0.7}
```

`\FN@assembleboxes` This will produce the finished product, by generating all boxes and concatenating them except for the last vbox. It is assumed that have already set `\box\z@` to `\lastbox` before calling this routine (or, more likely, have already assembled and split the last box). The last, not yet unpacked `\vbox` is left in `\box\z@` on return. The original id of the last box of a block is properly transferred to it.

The last box might have come about by joining several horizontal boxes, so splitting it might separate footnotes. We deal with that problem at a different point of time by checking the respective Ids when breaking a vbox into pieces: if the split piece does not contain the last footnote beginning, we switch to a slow motion decomposal. `\FN@assembleboxes` is supposed to be entered and exited in vertical mode.

```
321 \def\FN@assembleboxes{%
322   \ifhmode \PackageError{bigfoot}{Unexpected hmode}{}\fi
323   \ifhbox\z@
324     \dimen@dp\z@
325   \ifdim\dimen@dp>\dimen@ii\dimen@dp\z@
326     \setbox\tw@\box\voidb@x
327     \loop \advance\dimen@ii\dimen@dp\z@
328     \setbox\tw@\hbox{\box\z@\unhbox\tw@}%
329     \setbox\z@\lastbox
330   \ifhbox\z@
331     \repeat
332   {\FN@assembleboxes\nointerlineskip\unvbox\z@}%
333 }
```

At this point of time, `\box\tw@` contains a plain hbox with nothing but the unadorned hboxes in horizontal mode to be joined into one footnote block. All preceding footnote blocks have been emptied into the current vertical list. We put the `\unvbox` operations in a group so that the paragraph shapes will not get reset over the break.

```
334   \global\setbox\FN@tempbox\copy\tw@
335   \setbox\z@\@makefnvbox{%
336     {\unhbox\FN@tempbox}%
337     \setbox\z@\lastbox\FN@joinhboxes}%
338   \ifcase
339     \ifdim\FN@vfound>\dimen@vtypefraction\p*\FN@found\relax \one\fi
340     \ifdim\dimen@vfound>\dimen@hfootfraction\dimen@ii \one\fi \z@
341   \or
342     \global\setbox\FN@tempbox\box\tw@
343     \setbox\z@\@makefnvbox{\let\@makefnbreak\FN@pseudofillbreak
344       {\unhbox\FN@tempbox}\setbox\z@\lastbox\FN@joinhboxes}%
345   \fi
```

```

346 \setbox\tw@\box\voidb@x
347 \ht\z@\dimexpr \ht\z@+\dp\z@-\dimen@ \relax
348 \dp\z@\dimen@
349 <trace> \MFL@checksinglebox\z@\z@{}{}%
350 \else
351 \ifvbox\z@
352 <trace> \MFL@checksinglebox\z@\z@{}{}%
353 {\setbox\z@\lastbox
354 \FN@assembleboxes\nointerlineskip\unvbox\z@}%
355 \fi
356 \fi}

```

Ok, now follow a lot of fuzzy calculation routines. When we are considering truth values, `\p@` (1pt) corresponds to a value of “true”, and `\z@` corresponds to “false”.

`\FN@fuzzyeval` This calculates a ratio, something with which you multiply. The first two arguments of the function define an interval, and the third argument is a value in that interval. If `#3` is equal to `#1`, the resulting ratio is 0, if the `#3` is equal to `#2`, the resulting ratio is 1. Values in between are linearly interpolated. Values outside of the interval are mapped to 0 and 1. If all three values are equal (hardly useful), 1 is returned.

```

357 \def\FN@fuzzyeval#1#2#3{%
358 \ifdim\dimexpr(#3)<\dimexpr(#2)\relax
359 \ifdim\dimexpr(#3)>\dimexpr(#1)\relax
360 *(\dimexpr(#3)-(#1))%
361 /(\dimexpr(#2)-(#1))%
362 \else *\z@
363 \fi
364 \fi}

```

`\FN@fuzzyor` This returns probabilistic OR:

$$(\#1 + \#2 - \#1 \cdot \#2)$$

```

365 \def\FN@fuzzyor#1#2{(\p@-(\p@-(#1))*(\dimexpr\p@-(#2))/\p@)}

```

`\FN@magicclue` Ok, so here is the magic glue calculator. `#1` and `#2` give the range over which the preceding line changes from ‘short’ to ‘long’. `#3` and `#4` give the range over which the current line changes from ‘short’ to ‘long’. Both are combined with a probabilistic or function, and then a penalty is chosen which ranges from `#5` to `#6` for short to long.

```

366 \def\FN@magicglue#1#2#3#4#5#6{%
367 <trace> \if\foottrace2\traceon\fi
368 \dimen@ \dimexpr\p@\FN@fuzzyeval{#1}{#2}\FN@lastsize \relax
369 \dimen@ii \dimexpr\p@\FN@fuzzyeval{#3}{#4}{\ht\z@+\dp\z@} \relax
370 \dimen@ \dimexpr\FN@fuzzyor\dimen@\dimen@ii
371 \count@ \numexpr((#6)-(#5))*\dimen@/\p@+(#5) \relax
372 \xdef\FN@vfound{\the\dimexpr\FN@vfound+\dimen@}%
373 \ifnum\count@>-\@M

```

```

374     \penalty\count@
375     \hskip\glueexpr -\parfillskip+1em minus 0.5em\relax
376   \else
377     \FN@pseudobreak
378   \fi
379   \xdef\FN@found{\number\numexpr\FN@found+\@ne}%
380 }

```

`\FN@pseudobreak` This ends a line, but without introducing `\par` and similar. It also ‘breaks in’ the next line to get proper indentation. The main difference with regard to `\break` is that this restarts the reckoning of line numbers for the sake of `\clubpenalty` calculation.

```

381 \def\FN@pseudobreak{%
382   {\parskip\z@skip\parfillskip\z@skip\parindent\z@\vadjust{}\par\noindent
383     \vadjust{\nobreak\vskip-\baselineskip}\nobreak\hfill\break}}

```

`\FN@pseudofillbreak` This is basically just for separating paragraphs by force.

```

384 \def\FN@pseudofillbreak{\nobreak\hskip\parfillskip\FN@pseudobreak}

```

`\@makefnbreak` This calculates the glue for the standard horizontal footnotes.

```

385 \def\@makefnbreak{\FN@magicglue {\footnotesep+\dp\strutbox}%
386   {\footnotesep+\dp\strutbox+\baselineskip}%
387   {\footnotesep+\dp\strutbox+0.5\baselineskip}%
388   {\footnotesep+\dp\strutbox+2\baselineskip}{-200}{-12000}}

```

`\FN@joinhboxes` is called with box 0 set to the next box to be appended to the current list (all preceding hboxes on the current vertical list will have to go in front). `\FN@joinhboxes` is entered in vertical mode, and will be exited in horizontal mode.

```

389 \def\FN@joinhboxes{%
390   \ifvmode \errmessage{Unexpected vertical mode.}\fi
391   \begingroup\setbox\z@\lastbox
392   \ifhbox\z@ \FN@joinhboxes
393   \ifvmode \errmessage{Unexpected vertical mode.}\fi
394   \endgroup
395   \nobreak\hskip\parfillskip
396   \@makefnbreak
397   \else
398   \ifvbox\z@ \errmessage{Unexpected vbox.}\fi
399   \endgroup
400   \vadjust{\nobreak\vskip-\baselineskip}\nobreak\hfill\break
401   \xdef\FN@vfound{\z@}%
402   \xdef\FN@found{\z@}%
403   \fi
404   \xdef\FN@lastsize{\the\dimexpr \ht\z@ +\dp\z@}%
405   \unhbox\z@}

```

`\FN@par` In-paragraph footnotes are collected in horizontal mode. So `\par`, `\noindent` and `\FN@noindent` simply don’t work. We replace them with something having the same `\FN@indent`

effect when the boxes get unboxed. Note that this does not admit the tracking of club/widow penalties: in a later version, it should get replaced by something that actually allows for separate paragraphs. One possibility would be to replace the current single hbox for an in-paragraph footnote by an hbox of hboxes and unbox all of them in separate paragraphs. But that glosses over the fact that a multi-paragraph footnote does not make sense in anything but vertical mode. So a saner way would probably be to close off the hbox altogether and reinsert it into a vbox, restarting the whole footnote in vertical mode. Both of those approaches would require that no groups have been opened since the start of the footnote by the time `\par` gets called. The below pseudosolution at least has the advantage of not depending on the grouping structure at all.

```

406 \def\FN@par{\unskip\nobreak\hskip\parfillskip
407   \vadjust{\vskip\parskip}\break\null\kern\parindent\ignorespaces}
408 \def\FN@noindent{\unkern}
409 \def\FN@indent{\unkern{\setbox\z@\null\wd\z@\parindent\box\z@}}

```

`\MFL@fnoteplain` We redefine manyfoot's basic footnote calls to use our own, versatile variant.

```

\MFL@fnotepara 410 \def\MFL@fnoteplain{\FN@fnotenested{plain}}
411 \def\MFL@fnotepara{\FN@fnotenested{para}}

```

`\FN@fnotenested` This is somewhat contorted: we want `\footnote+` to be in `plain` style and `\footnote-` in `para` style regardless of the current footnote style. Adding a second `+` or `-` after the first will actually restyle all footnotes coming afterwards appropriately. This should work for all footnote commands getting footnote text.

```

412 \def\FN@fnotenested#1#2#3{%
413   \edef\reserved@d{#1}%
414   \FN@checkvariant{\edef\reserved@d}{%
415     \FN@checkvariant{\FN@restylefootnote{#2}}}%
416     {\csname FN@fnote\reserved@d\endcsname{#2}{#3}}}}
417
418 \def\FN@checkvariant#1#2{\def\reserved@a{#1}%
419   \def\reserved@b{#2}%
420   \futurelet\reserved@c\FN@checkvariantii}
421
422 \def\FN@checkvariantii{%
423   \ifx\reserved@c+{%
424     \reserved@a{plain}\expandafter\@firstoftwo
425   \else\ifx\reserved@c-%
426     \reserved@a{para}\expandafter\expandafter\expandafter
427     \@firstoftwo
428   \fi\fi
429   \reserved@b}

```

In order to be able to sort footnotes according to the order of their reference points, we use a sorted counter.

```

430 \newcounter{FN@totalid}
431 \MakeSorted{FN@totalid}

```

`\FN@fnotplain` The actual commands are easy enough:

```

\FN@fnotepara 432 \def\FN@fnotplain{\FN@fnotecommon\vbox}
433 \def\FN@fnotepara{\FN@fnotecommon\hbox}

```

`\FN@masterinsert` This contains the insert number of the insert where the footnote mark appears. If it appears in the main text, 255 will be used.

```

434 \def\FN@masterinsert{\@ccclv}

```

`\FN@id` Here is the deal with master and slave ids: each footnote has a unique master id.

`\FN@master` This master id is larger by one than the last id of its subordinate footnotes. It is recorded in the mark `\FN@master` in the footnote box at the start itself, although with an indirection through the `\FN@news slot` mechanism since the actual id can only become known after all subfootnotes have been typeset. The same id is recorded in `\FN@slave` at the ultimate end of the footnote.

`\FN@slave`

At the point where a `\FN@master` mark is placed, a default `\FN@slave` mark is placed also with an id that is one less than the smallest id generated from a footnote that is a ‘descendent’ of the current one. This makes it possible to distinguish any split off subordinate footnotes. It must be noted that this sentinel slave id will be the valid id of a completely unrelated footnote! Since the value is only used for determining one end of an *open* interval of excluded ids, this is no problem. All subordinate footnotes are numbered sequentially in the order of *completion*, so that any subordinate footnotes have lower ids than their master.

```

435 \newcount\FN@id
436 \FN@id\@ne
437 \newmarks\FN@master
438 \newmarks\FN@slave

```

`\FN@errorstack` This records the history of nested footnotes in order to deliver more useful error messages.

```

439 \let\FN@errorstack\@empty

```

`\FN@fnotecommon` Well, this is the work horse if the footnote macro. Really bad thing. We start off by stepping our absolute counter and making a mark. `\leavevmode` is required so that the action of `perpage.sty` is done smoothly.

```

440 \def\FN@fnotecommon#1#2#3{%
441   \leavevmode
442   \stepcounter{FN@totalid}%
443   \NCC@makemark{#3}%

```

It is an error if the footnote insert number of the current footnote does not correspond to a block below the current insertion level.

```

444   \ifnum#2<\FN@masterinsert
445     \FN@colorstackbgroup\FN@divert
446     \FN@news slot\FN@masterslot
447     \count@\FN@id

```

`\dimen@` is here set to a sorting criterion. This is designed to make the conversion of footnote blocks as reliable as possible. If we could guarantee convergence, just

using `\c@FN@totalid` would be sufficient for sorting. It turns out that this is too sensitive to footnotes of different blocks changing pages, so the number of the superior footnote block is allowed to take precedence by multiplying it with 4194304 which is unlikely to get exceeded by `\c@FN@totalid`.

```

448 \dimen@=\dimexpr64\p* $\c@FN@masterinsert-\c@FN@totalid$  sp\relax
449 \def\FN@masterinsert{#2}%
450 \edef\FN@errorstack{\FN@errorstack^^J%
451 \FN@masterinsert\space entered in line \number\inputlineno}%
452 \let\FN@boxtype=#1%
453 \setbox\z@#1\bgroup

```

The following is for the likes of PDF_TE_X which has its own idea about how to restore a color stack.

```

454 \let\current@color\default@color
455 \FN@@color@begingroup
456 \let\MFL@minipage\relax
457 \let\MFL@endminipage\relax
458 \@makefnstartbox

```

We reset the list parameters in footnotes. Strictly speaking, this is interfering with L^AT_EX's standard operation, but the standard operation does not make sense.

```

459 \let\@listdepth\@mplistdepth \@mplistdepth\z@
460 \@itemdepth\z@ \@enumdepth\z@
461 \protected@edef\@currentlabel{\csname p@footnote%
462 \expandafter\FN@stripfootins\string#2\endcsname\@thefnmark}%
463 \ifx\FN@boxtype\vbox \normalcolor\nobreak
464 \else \FN@specific{#2}\@preparefnhtext \normalcolor
465 \fi

```

Ok, now we do the call to `\@makefnhtext` which may occur in one of several ways, depending on whether the ‘robust’ or the ‘fragile’ package option got used.

```

466 \expandafter \FN@makefnhtext
467 \else

```

We still needed to cater for the error of badly anchored footnotes:

```

468 \PackageError{bigfoot}{#2 forbidden in \FN@masterinsert.}%
469 {Higher-placed footnotes can't be anchored in inferior ones.^^J%
470 I am not putting this text in a footnote. History:%
471 \FN@errorstack}%
472 \rule{1em}{\ht\strutbox}%
473 \fi}

```

`\FN@makefnstart` This is called in the start of `\@makefnhtext`.

```

474 \providecommand{\FN@seitenobreak}{\nobreak}
475 \def\FN@makefnstart{%

```

Record the footnote specific dimensions. It is assumed that they don't change in the document, at least not before the footnote gets actually placed.

```

476 \expandafter\xdef\csname FN@ht\number\FN@masterinsert\endcsname
477 {\the\footnotesep}%
478 \expandafter\xdef\csname FN@dp\number\FN@masterinsert\endcsname

```

```

479     {\the\dp\strutbox}%
480     \expandafter\xdef\csname FN@wd\number\FN@masterinsert\endcsname
481     {\the\hsize}%

```

The footnote gets markers for identifying it and its starting block.

```

482     \marks\FN@master{\FN@masterslot}%
483     \marks\FN@slave{\number\FN@id}%
484     \nobreak

```

\FN@commonending will intervene before any tokens that are shifted in due to switching back the color stack. Those will only be executed once we completely relinquish control.

```

485     \ifx\FN@boxtype\vbox
486         \rule{z@}{footnotesep}
487     \else
488         \ifx\FN@par\par\else
489             \let\FN@@par\par
490             \let\FN@@noindent\noindent
491             \let\FN@@indent\indent
492         \fi
493         \everyvbox\expandafter{\expandafter\everyvbox
494             \expandafter{\the\everyvbox}}%
495             \let\par\FN@@par
496             \let\noindent\FN@@noindent
497             \let\indent\FN@@indent
498             \the\everyvbox}%
499         \let\par\FN@par
500         \let\noindent\FN@noindent
501         \let\indent\FN@indent
502     \fi
503     \FN@seitenobreak
504     \afterassignment\ignorespaces}

```

\FN@makefnrobust After preparation, we now do the big bad trick for making footnotes cooperate with `\verb` and other catcode changing things: we call `\@makefnstext` with an argument of `\iffalse`. This kills off its expansion right at the point where it would choose to place its argument.

Furthermore, this swallows the opening brace of the footnote text and then lets the footnote text progress. The closing group will then trigger the processing via `\aftergroup`.

```

505 \def\FN@makefnrobust#{%
506     \FN@specific\FN@masterinsert\@makefnstext
507     \iffalse\fi
508     \bgroup
509         \aftergroup\FN@robustending
510         \FN@makefnstart
511         \let\next}

```

\FN@robustending Here we put in the missing part of `\@makefnstext`.

```

512 \def\FN@robustending{%
513   \expandafter\expandafter\expandafter
514   \expandafter\expandafter\expandafter\expandafter
515   \iffalse \FN@specific\FN@masterinsert\@makefntext\fi
516   \FN@commonending}

```

`\FN@makefnfragile` This is the escape route when the robust variant does not work. In that case, `\verb` and similar won't work in footnotes.

```

517 \long\def\FN@makefnfragile#1{%
518   \FN@specific\FN@masterinsert\@makefntext
519   {\FN@makefnstart#1\FN@commonending}}

```

Ok, color handling is a nuisance, to say the least. Split footnotes need to close their color stack on the old page, and reopen it on the new one. So we record the color stack state at each time it changes in a marks register.

```

520 \newmarks\FN@color
521 \def\FN@colorstackbgroup{\let\FN@savecolorstack\FN@colorstack
522   \global\let\FN@colorstack\@empty
523   \bgroup
524   \ifdefined\FN@savecolorstack\else
525     \let\FN@@set@color\set@color
526     \let\FN@@reset@color\reset@color
527     \let\FN@@color@begingroup\color@begingroup
528   \fi
529   \let\set@color\FN@set@color
530   \let\reset@color\FN@reset@color
531   \let\color@begingroup\FN@color@begingroup}
532
533 \def\FN@colorstackegroup{\egroup
534   \global\let\FN@colorstack\FN@savecolorstack}
535
536 \def\FN@colorstackfinish{\def\@elt##1##2{\FN@@reset@color##2}%
537   \FN@colorstack
538   \def\@elt##1##2{\noexpand\@elt-}{##2}}%
539 \xdef\FN@colorstack{\FN@colorstack}%
540 \let\@elt\relax
541 \marks\FN@color{}}
542
543 \def\FN@reset@color{%
544   \bgroup\def\@elt##1##2{\def\FN@next{##1}{\gdef\FN@colorstack{##2}}}%
545   \let\FN@next\@empty
546   \FN@colorstack
547   \ifx\FN@next\@empty
548     \FN@colorstackegroup
549   \else \egroup
550     \FN@@reset@color
551     \marks\FN@color{\FN@colorstack}%
552   \fi}
553

```

```

554 \def\FN@color@begingroup{%
555   \let\reset@color\FN@@reset@color
556   \let\color@begingroup\FN@@color@begingroup
557   \let\set@color\FN@@set@color
558   \color@begingroup}
559
560 \def\FN@set@color{\FN@@set@color
561   \xdef\FN@colorstack{\@elt{\current@color}{\FN@colorstack}}%
562   \marks\FN@color{\FN@colorstack}}
563
564 \def\FN@coloraftersplit#1{%
565   \def\@elt##1##2{##2\def\current@color{##1}\set@color}%
566   #1%
567   \let\@elt\relax}

```

\FN@commonending We'll eventually arrive here at the end of the footnote. Now we again call \@makefntext, but this time pass it \fi as its argument, and place \iffalse before its expansion. This cuts away the start of the macro. If this start changes the tail of the macro when executed, the whole trickery will not work. It turns out that a large sampling of document classes (including the standard ones) happens to work.

```

568 \def\FN@commonending{%
569   \@makefnendbox
570   \ifx\FN@boxtype\vbox\@finalstrut\strutbox \else \unskip \fi
571   \FN@colorstackfinish
572   \color@endgroup
573   \egroup
574   \global\advance\FN@id\@ne
575   \FN@slotxdef\FN@masterslot{\number\FN@id}%

```

Now we want to get an upper estimate of the size. In case of a horizontal box, we do this by creating a vertical box of it all alone, and measuring that. Measuring the hbox itself is plain out: T_EX's maximal dimension of something like 5 m is already busted with about two pages of material. We put the master slot identification into the depth of the box, and arrange for the total of depth and height of the box to still give the total depth and height of its size on the page.

```

576 \ifhbox\z@
577   \global\setbox\FN@tempbox\copy\z@
578   \setbox\tw@\@makefnvbox{\unhbox\FN@tempbox}%
579   \ht\z@\dimexpr\ht\tw@+\dp\tw@-\FN@masterslot sp\relax
580 \else
581   \ht\z@\dimexpr\ht\z@+\dp\z@-\FN@masterslot sp\relax
582 \fi

```

Now we put the sorting criterion into the width of the box, and then put the masterslot id into the depth.

```

583 \wd\z@\dimen@
584 \dp\z@\FN@masterslot sp\relax
585 \trace \ifnum\z@<0\FN@slotget{\FN@masterslot} %

```

```

586 <trace>      \else \errmessage{Inconsistent
587 <trace>      \string\FN@masterslot=\FN@masterslot}\fi

```

Now we just need to place the stuff into an insertion and record the possibly changed slave id in order to know what subordinate footnotes belong to this one.

```

588 \MFL@insert\FN@masterinsert{\nointerlineskip\box\z@}%
589 \ifdim\lastkern=\z@ \let\FN@next\@empty\else
590 \edef\FN@next{\kern\the\lastkern\relax}\unkern
591 \fi
592 \marks\FN@slave{\number\FN@id}%
593 \expandafter\FN@enddivert\expandafter\FN@colorstackgroup
594 \FN@next
595 }

```

A lot of stuff follows. This should really be cleaned up and documented.

```

596 \dimen\footins\maxdimen
597 \gdef\FN@nestlist{}
598
599 \newdimen\FN@outervsize
600 \newskip\FN@vsize
601
602 \newbox\FN@insertions
603
604 <trace>\def\MFL@showone#1#2{\message{Box #2:}\showbox#2%
605 <trace> \MFL@checkconsistency{#2}%
606 <trace> \message{Cachebox #2:}\showbox\FN@cache#2}
607 <trace>
608 <trace>\def\MFL@checkconsistency#1{#{%
609 <trace> \setbox\z@\vbox{\unvcopy#1%
610 <trace> \MFL@checkconsistencyi{#1}}}}

```

`\MFL@checksinglebox` Check box #1 for consistency. If it is bad, output box #2. Execute #3 if it was good, #4 if it was bad.

```

611 <trace>\def\MFL@checksinglebox#1#2#3#4{%
612 <trace> \ifvoid#1\else
613 <trace> \ifnum\z@<0\FN@slotget{\number\dp#1} %
614 <trace> #3%
615 <trace> \else \errmessage{Inconsistent box #2}%
616 <trace> \showboxdepth4\showboxbreadth100
617 <trace> \showbox#2\relax
618 <trace> #4%
619 <trace> \fi\fi}

620 <trace>\def\MFL@checkconsistencyi#1{%
621 <trace> \unpenalty\unskip\unkern
622 <trace> \setbox\z@\lastbox
623 <trace> \MFL@checksinglebox\z@{#1}{\MFL@checkconsistencyi{#1}}{}}
624 <trace>
625 <trace>\def\MFL@showall{#{%
626 <trace> \showboxbreadth=\maxdimen

```

```

627 <trace> \showboxdepth=4
628 <trace> \tracingonline=\@ne
629 <trace> \FN@nest@iterate\MFL@showone}}

```

\FN@retaindelayed This is a complex macro that removes all boxes from the current list that are not to be kept for the next page. It works on the material from the original insertions, not the cache boxes. The slot specified by `\count@` is not freed when encountered, all others are freed upon removing the box. The last box is returned in box 0 if any is retained. The vertical list might have an unchecked part locked off in front by placing a `\nobreak` penalty there. This penalty is removed, and the list before it not touched.

```

630 \def\FN@retaindelayed{%
631   \setbox\z@\lastbox
632   \ifcase
633     \ifvoid\z@\m@ne\fi \FN@config\z@
634 <trace> \if\foottrace8\message{^^J\string\FN@retaindelayed:
635 <trace>   dropping Id \FN@slotget{\number\dp\z@}\fi
636 <trace> \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
637 <trace>   \tracingonline=\@ne\showbox\z@}\fi
638   \ifnum\dp\z@=\count@\else \FN@freeslot{\number\dp\z@}\fi
639 <trace> \ifnum\dp\z@<\@ne \errmessage{Unidentified box}\fi
640   \expandafter\FN@retaindelayed
641   \or
642 <trace> \if\foottrace8\message{^^J\string\FN@retaindelayed:
643 <trace>   retaining Id \FN@slotget{\number\dp\z@}\fi
644 <trace> \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
645 <trace>   \tracingonline=\@ne\showbox\z@}\fi
646   {\FN@retaindelayed \nointerlineskip \box\z@}%
647   \else \unpenalty \setbox\z@\lastbox
648 <trace> \ifnum\lastnodetype>\m@ne
649 <trace>   \errmessage{Unexpected node \number\lastnodetype}\fi
650 <trace> \ifvoid\z@ \else
651 <trace>   \if\foottrace8\message{^^J\string\FN@retaindelayed:
652 <trace>     carrying split box \FN@slotget{\number\dp\z@}\fi
653 <trace>   \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
654 <trace>     \tracingonline=\@ne\showbox\z@}\fi\fi
655 \fi}

```

\MFL@processplain This gets called for actually inserting the processed material into the footnote box. The current state of affairs is that `\FN@config` contains all footnotes that should get transferred to the next page completely. The cache boxes contain the collected and typeset footnotes for typesetting on the current page.

The structure of a cachebox is currently as follows: it is filled with vboxes containing the arranged material, optionally followed by another box to be carried over to the next page flagged with a `\nobreak` penalty.

```

656 \def\MFL@processplain#1{%
657 <trace> \MFL@checkconsistency#1%
658   \ifvoid\FN@cache#1%

```

Now if the cache box is void, nothing gets typeset on the current page. What we do, however, is to collect all boxes from the original insertion that did not make it on this page and reinsert them. `\count@` is cleared to zero to retain nothing special.

```

659 \global\setbox\FN@tempbox\vbox\bgroup
660 \unvbox#1%
661 \count@\z@
662 \let\@elt\FN@removecheck\FN@retaineddelayed
663 \ifvoid\z@ \egroup
664 \else \nointerlineskip \box\z@ \egroup
665 \MFL@realinsert{#1}{\unvbox\FN@tempbox}%
666 \fi

```

The following stops in the insertion process within the `manyfoot` package.

```

667 \expandafter\expandafter
668 \fi\iffalse\fi

```

Ok, this is the case when we have a nonvoid cache box.

```

669 \global\setbox#1\vbox\bgroup%
670 \unvbox\FN@cache#1%
671 \ifnum\lastpenalty>\z@
672 \unpenalty
673 \setbox\z@\lastbox
674 \else
675 \setbox\z@\box\voidb@x
676 \fi

```

Ok, now box zero contains carryover material (if any). We initialize `\count@` to this so that we will keep this carryover material just once.

```

677 \count@\dp\z@
678 \global\setbox\FN@tempbox\vbox\bgroup
679 \box\z@
680 \nobreak
681 \unvbox#1%
682 \let\@elt\FN@removecheck\FN@retaineddelayed
683 \ifvoid\z@ \egroup \MFL@removeboxes\egroup
684 \else \nointerlineskip \box\z@ \egroup
685 \MFL@removeboxes \egroup
686 \MFL@realinsert{#1}{\unvbox\FN@tempbox}%
687 \fi}
688
689 \let\MFL@processpara\MFL@processplain

```

Ok, here is the bit about the caches: whenever we encounter a new configuration, we have to first update the caches since we don't know the sizes we are dealing with regarding the new configuration until we do so. The caches are kept up to date globally. When we are working at several levels in the recursion, we have a bottom active level where we may be looking for a way to find a best break and configuration. We will return at most one configuration once we are finished. While we are working with a returned configuration, adding more material on the

current list will not require another recursion as long as the totals stay underfull: the penalty difference between underfull configurations becomes smaller while the underfullness decreases, which means that smaller breaks that have not been chosen before might become eligible if the penalties allow for that. Only when the badness of underfullness remains infinite can't we have any improvement.

Ok, after we recurse for removing an underfull condition, the resulting configuration can't actually be used further for breaks with less remaining space. It is, however, clear that if less space remains, there is no better break with the same configuration leaving *more* space: if there were, it would already have been taken. That means that our goal height for the next break will be chosen in order to reach the exact size met on the last recursion. No break before that can be chosen on the next try, but a break after it might then be taken.

available, or an overfull one. If a deeper level at any point of time returns an overfull configuration, we are finished. The best configuration to be returned is the least underfull. If there is none, the least overfull. The case of no underfull at all can only happen if even splitting this and every subordinate level to minimal height and recursing does not yield an underfull. At every level, we need to maintain just a current split, and the previous best split at most.

When we change a configuration on recursing, we have to remember the configurations for the previous best split. We can manage that by sweeping the current cache values into a local box register before recursing with a different configuration: we have to rebuild the box registers for a different configuration, anyway. We don't save the configuration from an overfull setting: when we rework the list in slow motion mode, we can't help stopping the recursion by reaching an overfull setting that is at least as good as the initial one.

When we return to a caller, we leave the cache in the configuration of the best choice up to now: either we are returning an overfull configuration and if it is not the best so far, the caller can restore his better choice from his copy, or we are returning an underfull configuration in which case the caller might still want to improve upon it before returning to its caller in turn. New: If we return an underfull configuration, we also return an "optimal penalty estimate" that gives the best break point penalty under the assumption that additional stretchability is present on the page.

The purpose of this is to offer the possibility of avoiding widows and similar by moving more material in some footnotes to the next page in exchange for other material.

At the current grouping level we empty out our current cache and keep it for working purposes on the vertical list until we return (nobody references it while we are working on it). We always enter with an overfull configuration, meaning that `\FN@vsize` is negative. It is calculated with the current cache/config setting.

There is a danger of overflow involved with that: if we keep a swept complete configuration at each level of recursion, we need $O(n^2)$ of space here. The alternative would be to keep the history of how the configuration came about. Since that might involve some slow-motion splitting, this is also a speed issue. Since deep recursion with pending best data at each level is not really likely, and since we are not going to have that many footnote levels to go around, anyway, we just

rely on L^AT_EX having been started with sufficient memory.

A workable compromise would be to just store the split boxes from a configuration together with the configuration data for reconstructing the rest. After all, we don't need to reconsider such a configuration before actually typesetting anything. And whenever we find an acceptable fit (neither underfull/overfull), we could cut through all the hierarchy without having to restore anything. This has not been implemented yet: at the moment we go for the less complicated variation.

The algorithm we use here is a bit complicated. Whenever we recurse, we have one of the following situations:

1. An overfull/underfull dilemma: including a minimal amount of material at the current level will cause the page to become overfull. This can be the case in connection with zero (in case of interline penalties for larger blocks), one or more subordinate footnotes and related footnotes.
2. A pure overfull dilemma: the page was overfull to start with, we need to reduce it.
3. an underfull dilemma: some operation in the next level made the page become underfull, only too much so. We can't make it fuller on the current level, but we can make it even emptier, and let the next level fill it up again.

In the current implementation, we just ignore the slight probability that the optimum choice might lie with case 3. We don't recurse for making the page fuller again. If we have an overfull/underfull dilemma, the recursion will either give us a less awful overfull box, or an underfull one. An overfull box that occurs at the highest level of recursion can't be improved on any lower level. So we never need to locally return an overfull box: we can compare it to the best overfull box seen before, and if we turn out better than that, we overwrite the global best overfull value and return the best local underfull if there is such a one. The best local underfull will then be refilled as much as possible on the next level without changing the configuration. Actually, if we need to change the configuration, this would also be fine as long as we arrive at a better underfull eventually. But since a change of configuration renders our previous split completely useless, as the broken paragraph could look disastrously different under a changed configuration, we would need to recurse again. We repeat this recursing operation until we don't get an underfull solution returned anymore. We then return the best underfull, if any. The best overfull is stored globally, as mention before.

Does this sound complicated? Unfortunately, it does. It also sounds somewhat slow. For that reason, we do a few assumptions that will facilitate a good average-case behavior. The first assumption is that we will usually do fine by just splitting in the current level (if at all) and not at all in subordinate levels.

We do this assumption on the first pass used for gathering the size information and collect the corresponding boxes in nested lists. When the recursion tops out, it does so either with an overfull page, or an underfull page. If it does with an underfull page, we cache the current configuration for the next pass through the output routine, so that we won't need to retypeset and measure assembled boxes

that have not gathered any new material. If we top out with an overfull page, the previous underfull configuration is still worth keeping as well, as it might become the material actually chosen to be typeset.

Ok, the current best configuration of the next recursion level is gathered on the current vertical list, in a separate box. We use box 2 for this purpose. A saved configuration consists of the complete contents for the current cache box without the trailing penalty indicating material from a single split box carried over to the next page (boxes that are carried over completely to the next page are not maintained here but rather reinserted by `\MFL@processnested`). This penalty is added in case the box is actually disassembled and returned: there is no possibility for confusion since we only save such a configuration if indeed there is a split present.

```

690 \newtoks\FN@output
691 \FN@output\output
692
693 \newbox\FN@tempbox
694 \newinsert\FN@savebox
695 \count\FN@savebox\@m
696 \dimen\FN@savebox\maxdimen
697 \skip\FN@savebox\z@skip
698 \global\setbox\FN@savebox\box\voidb@x
699
700 \expandafter\expandafter\expandafter\let\FN@cache\FN@savebox=\cclv
701
702 \expandafter\def\csname FN@ht\number\FN@savebox\endcsname{\z@skip}%
703 \expandafter\def\csname FN@dpt\number\FN@savebox\endcsname{\maxdepth}
704 \expandafter\def\csname FN@wd\number\FN@savebox\endcsname{\columnwidth}
705
706 \def\FN@list{\MFL@list\@elt{}\footins}
707
708 \def\FN@sweepbox#1#2{\ifvoid#2\else
709   \nointerlineskip\box#2\penalty#2\fi}
710
711 \def\FN@sweepcachebox#1#2{\nointerlineskip
712   \box\FN@cache#2%
713   \penalty\FN@cache#2}
714
715 \def\FN@copycachebox#1#2{\nointerlineskip
716   \copy\FN@cache#2%
717   \penalty\FN@cache#2}
718
719 \def\FN@restoreboxes{\count@\lastpenalty \unpenalty
720   \ifnum\count@>\z@
721     \global\setbox\count@\lastbox
722     \expandafter\FN@restoreboxes
723   \fi}

```

`\FN@removecheck` This returns `\@one` if and only if the current slot master is strictly inside of the

specified open interval. In this case it is not to appear on the current page.

```

724 \def\FN@removecheck#1#2{%
725   \ifnum#1<\FN@slotget{\number\dp\z@} %
726   \ifnum#2>\FN@slotget{\number\dp\z@} %
727     \one\fi\fi}

```

Parameter recording merely records the relevant value of the skip register and sets it to zero. The purpose is to avoid changes of the reserved page space when we collect additional material from a page where an insertion of the appropriate kind had already been encountered. This is used for filling up underfull pages.

```

728 \def\FN@recordinsertparam#1#2{\ifvoid#2\else
729   \global\skip\number#2=\the\skip#2\relax\fi}
730
731 \def\FN@clearinsertparam#1#2{\ifvoid#2\else
732   \global\skip#2=\z@skip\fi}

```

\FN@insertouterspace will sum the size of the inserts manually.

```

733 \def\FN@insertouterspace#1#2{\ifvoid#2\else
734   +\skip#2+(\ht#2+\dp#2)*\count#2/\@m\fi}
735
736 \def\FN@list@iterate#1{\let\FN@eltsave\@elt
737   \let\@elt#1%
738   \FN@list
739   \let\@elt\FN@eltsave}
740
741 \def\FN@nest@iterate#1{\let\FN@eltsave\@elt
742   \let\@elt#1%
743   \FN@nestlist
744   \let\@elt\FN@eltsave}

```

1.5 The output routine stuff

Marks This is used for sweeping all marks up for reinsertion.

\FN@allmarks

```

745 \def\FN@allmarks#1{\@elt{#1}%
746   \ifnum#1<\count266
747     \expandafter\FN@allmarks\expandafter{\number\numexpr#1+\@ne}%
748   \fi}

```

\FN@sweeptopmarks This sweeps the current topmarks and places them into the global box \FN@topmarkbox.

\FN@topmarkbox

```

749 \def\FN@sweeptopmarks{\global\setbox\FN@topmarkbox\vbox{%
750   \def\@elt##1{\marks##1{\unexpanded\expandafter{\topmarks##1}}}%
751   \FN@allmarks0}}
752 \newbox\FN@topmarkbox

```

\FN@establishmarks

This sets marks from a marks sweep. The first argument is the mark number, the second is from the first mark on the first scan, the third argument from the bottom mark on the first scan, and the fourth argument from the bottom mark

on the second scan (with additional mark entries). If second and third arguments don't match, no mark gets placed.

```
753 \long\def\FN@establishmarks#1#2{\edef\reserved@a{\unexpanded{#2}}%  
754 \edef\reserved@b{\unexpanded\expandafter{\splitbotmarks#1}}%  
755 \ifx\reserved@a\reserved@b  
756 \marks#1{\unexpanded\expandafter{\splitfirstmarks#1}}%  
757 \marks#1{\unexpanded\expandafter{\reserved@b}}%  
758 \fi}
```

\FN@markspassone This constitutes the first pass for mark collection. We do this just to check whether there are any marks in the list.

```
759 \def\FN@markspassone#1{\noexpand\FN@establishmarks{#1}}%  
760 {\unexpanded\expandafter{\splitbotmarks#1}}}
```

\FN@insertmarks This routine transfers first and bottom marks from the current `\box255` to the vertical list in order to get the marks right. This is quite a bother, since we must detect the special case where there are no marks at all in the list, and since we might require the use of several `\vsplit` commands in a row, since infinite stretch


```
1175 <trace>   Config=\unexpanded\expandafter{\FN@config}^^J}\fi
1176   \fi}
```

```
\FN@newleveli
```

```
1177 \def\FN@newleveli{%
```


by first copying the insertion into it.

```
1536 \global\setbox\count@\vbox\bgroup\vbox\bgroup\unvcopy#2%
1537 \let\@elt\FN@removecheck
1538 \FN@retainkept
```

Now if nothing was retained, we void the cachebox.

```
1539 \ifvoid\z@ \egroup\egroup \global\setbox\count@ \box\voidb@x
```

Otherwise, we combine all the boxes that remain on the page.

```
1540 \else \def\FN@masterinsert{#2}%
1541 \FN@assembleboxes\global\setbox\count@\box\z@\egroup
1542 \nointerlineskip\box\count@\egroup
```

Note that now all footnote boxes are collected into a single vbox, followed by the last footnote box as another vbox. Now we just need to reduce the available size on the page by the height of the assembled material:

```
1543 \global\advance\FN@vsize
1544 -\glueexpr(\ht\count@+\dp\count@)*\count#2/\@m+\skip#2\relax
1545 \fi\fi}
```

\FN@rejoin This glues together cache boxes that have been split, without regenerating them.

This saves a lot of time as compared to **\FN@reconfig**.

```
1546 \def\FN@rejoin#1#2{%
1547 <trace> \if\foottrace1\message{^^JRejoining #2}\fi
1548 \count@\FN@cache#2%
1549 \ifvoid\count@\else
1550 \global\advance\FN@vsize\dimexpr
1551 (\ht\count@+\dp\count@)*\count#2/\@m\relax
1552 \global\setbox\count@\vbox{%
1553 \unvbox\count@
1554 \ifnum\lastpenalty>\z@
1555 \unpenalty
1556 \setbox\tw@\lastbox
1557 \setbox\z@\lastbox
1558 \dimen@\dp\z@
1559 \setbox\z@\vbox{%
1560 \unvbox\z@
1561 \setbox\z@\lastbox
1562 \unvbox\z@
1563 \unvbox\tw@}%
1564 \ht\z@=\dimexpr\ht\z@+\dp\z@-\dimen@\relax
1565 \dp\z@=\dimen@
1566 \nointerlineskip
1567 \box\z@
1568 \fi}%
1569 \global\advance\FN@vsize-\dimexpr
1570 (\ht\count@+\dp\count@)*\count#2/\@m\relax
1571 \fi}}
```

\FN@retainkept This relies on **\@elt** being set to **\FN@removecheck** which expands to **\@ne** if **\box0** is strictly between the two values from an entry of **\FN@config**, which means that

it is material that should get moved to the next page. In that case, we recurse while dropping the box in question. Otherwise we keep it. Recursion bottoms out when there are no boxes left. The function leaves the last retained box in box 0; if there are no boxes to be retained, this will be void.

```

1572 \def\FN@retainkept{%
1573   \setbox\z@\lastbox
1574   \ifcase
1575     \ifvoid\z@\m@ne\fi \FN@config\z@
1576   <trace> \if\foottrace8\message{^^J\string\FN@retainkept:
1577     retaining Id \FN@slotget{\number\dp\z@}}\fi
1578   <trace> \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
1579     \tracingonline=\@ne\showbox\z@}\fi
1580   {\FN@retainkept \nointerlineskip \box\z@}%
1581   \or
1582   <trace> \if\foottrace8\message{^^J\string\FN@retainkept:
1583     dropping Id \FN@slotget{\number\dp\z@}}\fi
1584   <trace> \if\foottrace{16}{\showboxdepth4 \showboxbreadth400
1585     \tracingonline=\@ne\showbox\z@}\fi
1586   \FN@retainkept
1587   \else
1588   <trace> \ifnum\lastnodetype>\m@ne
1589   <trace> \errmessage{Unexpected node \number\lastnodetype}\fi
1590   \fi}

```

Well, as the last measure, we change the output routine to our new routine.

```

1591 \let\output\FN@output

```

Ok, here is debugging code intercepting all calls of the regular output routine and reporting its entry and exit states. a a a a

```

1585 \if\foottrace{16}{\showboxdepth4 \showboxbreadth400

```

```

1606 \def\FN@maybestart#1#2#3{\ifx#3\relax
1607   \csname MFL@start#1\endcsname{#2}\fi#3}
1608 \onlypreamble\FN@maybestart
1609 \AtBeginDocument{\@ifundefined{footinsdefault}%
1610   {\newfootnote[plain]{default}}%
1611   {\let\@elt\FN@maybestart
1612     \MFL@list\relax}}%
1613 }{}%

```

And since LaTeX's macros are inferior to our own (and would probably not match too well), we reroot them to the default footnote style.

```

1614 \def\@footnotetext{\Footnotetextdefault{}}%
1615 \def\p@footnotedefault{\p@footnote}%
1616 }
1617 </style>

```

2 Various driver files

The installer, in case it is missing. If it is to be used via `make`, we don't specify an installation path, since

```
make install
```

is supposed to cater for the installation itself.

```

1618 <installer> \input docstrip
1619 <installer & make> \askforoverwritefalse \nopreamble
1620 <installer> \generate{
1621   <installer>   \file{bigfoot.drv}{\from{bigfoot.dtx}{driver}}
1622   <installer>   \file{perpage.drv}{\from{perpage.dtx}{driver}}
1623   <installer>   \file{suffix.drv}{\from{suffix.dtx}{driver}}
1624   <installer&!make> \usedir{tex/latex/bigfoot}
1625   <installer>   \file{bigfoot.sty}{\from{bigfoot.dtx}{style}}
1626   <installer>   \file{perpage.sty}{\from{perpage.dtx}{style}}
1627   <installer>   \file{suffix.sty}{\from{suffix.dtx}{style}}
1628 <installer> }
1629 <installer> \endbatchfile

```