marginfix package documentation

Stephen Hicks
sdh33@cornell.edu
http://shicks.github.com/marginfix

v0.9.1 – 2010/08/28

Usage

1 Overview

Authors using LaTeX to typeset books with significant margin material often run into the problem of long notes running off the bottom of the page. A typical workaround is to insert `\vshift`s by hand, but this is a tedious process that is invalidated when pagination changes. Another workaround is memoir's `\sidebar` function, but this can be unsatisfying for short textual notes, and standard marginpars cannot be mixed with sidebars. This package implements a solution to make marginpars "just work" by keeping a list of floating inserts and arranging them intelligently in the output routine.

2 Options

There are currently no options that do anything yet.

3 Commands

For the most part, this is a drop-in replacement. Simply add `\usepackage{marginfix}` to the preamble, use `\marginpar` normally and hope for the best. In the event, however, that it doesn't work exactly as hoped, there are a number of tweaks that the user can apply.

`\marginskip` Calling `\marginskip{`⟨*length*⟩`}` will insert an incompressible skip in the margin. These skips will force neighboring notes on the same page to be separated, but will disappear at the top or bottom of a margin.

`\clearmargin` `\softclearmargin` In an analog to `\clearpage`, `\clearmargin` prevents any further material from being added to the current margin. These calls are cumulative, so that two `\clearmargin`s in a row will produce a completely empty margin on the next page as well. If this is not the desired effect, use `\softclearmargin`, which is effectively idempotent: multiple calls have the same effect as one call to end the current margin.

`\extendmargin` If a page has too much margin material to fit and an important note is floating to the next page, `\extendmargin{`⟨*length*⟩`}` will extend the margin (for the current

page only) by the given length. If the length is negative, the margin will shrink. Multiple calls on the same page are cumulative.

`\mparshift` To adjust the position of a single note, use `\mparshift{`⟨*length*⟩`}` before `\marginpar`. Positive lengths move it down the page. This essentially shifts the call-out location, so the actual position of the note might not change if the margin is sufficiently crowded. Multiple calls before the same note are cumulative.

`\marginheightadjustment` If all the margins are the wrong size, the height of the margin on every page can be adjusted by assigning a non-zero value to the dimension register `\marginheightadjustment` (as in `\marginheightadjustment=`⟨*length*⟩). This is effectively the same as a call to `\extendmargin` on every page.

`\marginposadjustment` Similarly, if all the margins are in the wrong place, the callout positions can be adjusted by assigning a non-zero value to the dimension register `\marginposadjustment`. This is effectively the same as a call to `\mparshift` before every note. This is particularly useful at present because the height of the line on which the margin note is called is currently only estimated, and appears to be off by a point or two. This may get fixed in the future, but until then, the adjustment is possibly the easiest workaround.

4 Interaction with other packages

4.1 memoir

There are no known issues with memoir at present, provided that `\sidebar` is not used.

4.2 mparhack

mparhack was designed to deal with the problem of margin notes showing up in the wrong margin because the left/right was decided before it was known exactly which page the note would be on. Because we defer this decision to shipout time in this package, we are not susceptible to this problem, so mparhack is no longer needed and should not be included (though I'm unaware whether it causes any actual problems).

4.3 Multiple columns

There is currently no support for multiple columns.

5 Coming attractions and known issues

Here is a list of things to possibly look forward to in a future version. If any of them are particularly important, please let me know.

- Margin phantoms.
- Use of pdfTeX's `\pdfsavepos` and `\pdflastypos` for more accurate margin placement.
- `\vadjust` to correct inconsistencies with `\@pageht`.

- Better interaction with floats. (We can set a default one way or the other and then allow a macro to override it (presumably with a CS defined in terms of the box name/meaning, so as not to get in the way of LaTeX's use of the insert registers). We would then add or not add phantoms in the right spots. We'd also need to shift all the callout points by the size of the top figures (unless we're using `\pdfsavepos`).)

Implementation

6 Initial Setup

`\@ight` `\f@ur` `\@C` Make the `@`-sign into a letter for use in macro names. We also define a few other (hopefully) obvious and unambiguous macros that other packages won't destructively clobber.

```
⟨*package⟩
\makeatletter
\chardef\@ight=8
\chardef\f@ur=4
\chardef\@c=100
```

`\MFX@debug` We have some optionally-included code for debugging. `\MFX@debug` prints a new line followed by "`MFX: `" and then the message. The newline can be suppressed with a `*`. We'll also ask for more error context in the debug mode.

```
⟨debug⟩\def\MFX@debug{\@ifstar\message{\message{^^JMFX:}\message}}
⟨debug⟩\errorcontextlines=20
```

The reader might begin to note at this point a convention we adopt throughout this package. While we strive to avoid introducing new names as much as possible (using the inlinedef package whenever we can), any new names we do introduce will be prefixed by `\MFX@`, `\Mfx@`, or `\mfx@`, depending on the type of name. The all-capitol `\MFX@` is used for fully-constant macros. The initial-caps `\Mfx@` is used for control sequences that are technically constant, but that refer to things that change, such as counters, token lists, dimension registers, etc. Finally, the lowercase `\mfx@` is used for control sequences whose meaning changes dynamically (i.e. variable macros).

7 Options

Here we define the various package options.

`\ifmfx@ypos` The ypos option signifies that we should use the pdfTeX primitives `\pdfsavepos` and `\pdflastypos` to improve positioning of margin notes relative to their callouts. This requires two passes to work, and the first time through, the margin notes will be positioned very naïvely.

```
\newif\ifmfx@ypos
\DeclareOption{ypos}{\mfx@ypostrue}
```

Now we actually process the options.

```
\ProcessOptions\relax
```

8 Variables

\mfx@marginlist We need a place to store our list of marginal material. We store material in this variable using insert registers and a variety of macros, to be explained later.

```
\let\mfx@marginlist\@empty
```

\Mfx@marginbox While we're building the margin, we need to put it in a box before we can attach it to the main columm.

```
\newbox\Mfx@marginbox
```

\Mfx@marginboxspace \Mfx@marginpos While we build up the margin box, we need to keep track of both where we are in the margin (\Mfx@marginpos) and how much of that space is incompressible (\Mfx@marginboxspace).

```
\newdimen\Mfx@marginboxspace
\newdimen\Mfx@marginpos
```

\Mfx@marginheight Because the margin height can be altered by, \extendmargin, we must maintain a dimension for the height of the current margin.

```
\newdimen\Mfx@marginheight
```

\Mfx@mparshift We store the current shift in a dimension register.

```
\newdimen\Mfx@mparshift
```

9 User-configurable dimensions

We export a few dimensions that the user can redefine to tweak behavior.

\marginheightadjustment This length will be added to the total margin height of each page (the default is zero).

```
\newdimen\marginheightadjustment
```

\marginposadjustment We will offset each margin note from its callout location by this length (the default is zero).

```
\newdimen\marginposadjustment
```

10 Plan of attack

10.1 \marginpar

The default sequence of events for a \marginpar is roughly the following (assuming no errors):

```
\marginpar:
  let \@floatpenalty := (horizontal ? -10002 : -10003)
  allocate inserts \@currbox and \@marbox from \@freelist
  let \count\@marbox := -1 % signifies marginpar (not float)
  if optional argument then \@xmpar else \@ympar
\@xmpar:
  \@savemarbox \@currbox := required argument
  \@savemarbox \@marbox := optional argument
  \@xympar
```

```
\@ympar:
  \@savemarbox \@currbox := required argument
  copy \@marbox := \@currbox
  \@xympar
\@xympar:
  append \@marbox to \@currlist
  \end@float
\end@float:
  append \@currbox to \@currlist
  if horizontal then following two lines are in \vadjust:
    \penalty -10004
    \penalty \@floatpenalty
```

To get the rest of the picture, we need to peek into the output routine. The pertinent parts are as follows (in vanilla LaTeX):

```
\output:
  if \outputpenalty < -10000 then
    \@specialoutput
  else
    do regular output...
  details for dealing with footnotes...
\@specialoutput:
  switch \outputpenalty:
   case -10001: \@doclearpage
   case -10004: set box \@holdpg := \vbox{\unvbox255}
   case -10002 or -10003:
    set box \@holdpg := \vbox{\unvbox\@holdpg \unvbox255}
    let \@pageht := \ht\@holdpg, \@pagedp := \dp\@holdpg
    \unvbox\@holdpg
    pop \@currbox off of \@currlist
    \@addmarginpar (assuming \count\@currbox <= 0)
\@addmarginpar:
  pop \@marbox off of \@currlist
  free \@currbox and \@marbox back to \@freelist
  if left-hand margin then let \@marbox := \@currbox
  let \@tempdima := \@mparbottom - \@pageht + \ht\@marbox
  if \@tempdima < 0 then let \@tempdima := 0
  let \@mparbottom := \@pageht + \@tempdima + \dp\@marbox + \marginparpush
  decrement \@tempdima := \@tempdima - \ht\@marbox
  prepend \vskip\@tempdima to \@marbox
  let \ht\@marbox := \dp\@marbox := 0
  \kern -\@pagedp, \nointerlineskip
  set an \hbox to \columnwidth (zero height/depth):
    attach \@marbox to correct margin
  set a \vbox with height 0 and depth \@pagedp
```

We see from here that `\@addmarginpar` is the place where LaTeX does the work of calculating the current page position and where the next note should go, and then actually puts it there. We will need to completely replace this routine, but can leave everything else as is.

10.2 \output

While LaTeX's margin routines end with \@addmarginpar, we must dig even deeper to apply our patch, since we need to insert some code to run during the *main* output routine that ships out each page. Thus, we'll expand "do regular output..." from the previous \output listing.

```
do regular output...:
  \@makecol
  do { \@opcol \@startcolumn } while @fcolmade
\@makecol:
  set box \@outputbox := box255 (plus any footnotes)
  let \@freelist := \@freelist + \@midlist, \@midlist := \@empty
  \@combinefloats
  add \@texttop and \@textbottom to \@outputbox (default no-op)
\@opcol:
  \@outputpage (of \@outputdblcol in twocolumn mode)
  let \@mparbottom := \@textfloatsheight := 0
  \@floatplacement
\@startcolumn:
  try to make a float column from \@deferlist, setting @fcolmade
  if !@fcolmade then add floats from \@deferlist to next column
\@combinefloats:
  aggregate \@toplist floats into a box and prepend to \@outputbox
  aggregate \@botlist floats into a box and append to \@coutputbox
  free inserts from \@toplist and \@botlist
\@outputpage:
  ship out the page
  reset a bunch of stuff
  let \@colht := \textheight (in \@outputpage)
```

We've seen two main times when action occurs: callout time and shipout time. We proceed chronologically with our patches.

11 Callout-time patches

\@addmarginpar

The first thing we must modify is that at callout time, we need to get the inserts into \mfx@marginlist. This should happen in the output routine so that we can get ahold of the current page position. Even if we have a better idea of the page position (e.g. from pdfTeX), we still might as well do this in the OR.

```
\def\@addmarginpar{%
  \@next\@marbox\@currlist{}\MFX@bug
⟨debug⟩\MFX@debug{addmarginpar (running insert) \@marbox/\@currbox at
⟨debug⟩    \the\c@page:\the\@pageht, marginlist=\meaning\mfx@marginlist}%
  \MFX@getypos
  \MFX@cons\mfx@marginlist{% TODO: later this will be a run@marginlist
    \noexpand\mfx@margin@note\@marbox\@currbox{\mfx@ypos}% (^i.e. for phantoms)
    \noexpand\mfx@margin@skip{\the\marginparpush}%
  }%
⟨debug⟩\MFX@debug{addmarginpar (exit): marginlist=\meaning\mfx@marginlist}%
}
```

\MFX@cons
\MFX@snoc

In passing we'll define the cons macro, which fully-expands its second argument, but makes sure to only expand the first one once, so that any fragile control sequences in it are corectly protected. We also define snoc, which prepends. Note that we could put the \temp@ definition into a group if it was really gonna matter...

```
\def\MFX@cons#1#2{%
  \edef\temp@{#2}%
  \expandafter\expandafter\expandafter\gdef
  \expandafter\expandafter\expandafter#1%
  \expandafter\expandafter\expandafter{\expandafter#1\temp@}%
}

\def\MFX@snoc#1#2{%
  \edef\temp@{#2}%
  \expandafter\expandafter\expandafter\gdef
  \expandafter\expandafter\expandafter#1%
  \expandafter\expandafter\expandafter{\expandafter\temp@#1}%
}
```

\MFX@getypos
\mfx@ypos

We now need to settle on a way to determine the vertical position. Ultimately this will be an option, and will depend on a variety of factors. But for starters, we define the simplest version. Note the subtraction of \Mfx@strutheight. Ideally we would simply grab a copy of \@holdpg from the middle of \@specialoutput and then discard the last box to figure out what height we're really at, since \@holdpg includes the box from the line we're currently on, and we want to be level with the *top* of that box, rather than the baseline. But since \@holdpg is accessible only deep within \@specialoutput, and it's not worth the risky job of performing surgery on it (which is unfortunately brittle if anyone else has a similar idea), we instead resort to this approximation. And since this will ultimately be only a fallback for when \pdflastypos isn't available, it should be good enough. NOTE: we might be able to use a \vadjust instead here?

```
\def\MFX@getypos{%
  \edef\mfx@ypos{%
    \the\dimexpr\@pageht - \Mfx@strutheight
       + \marginposadjustment + \Mfx@mparshift\relax}%
  \global\Mfx@mparshift\z@
}
```

\marginpar
\Mfx@strutheight

We need to make sure \Mfx@strutheight gets defined somewhere, and the best time is probably right before the \marginpar does its work, since that will most likely ensure we're using the right font for the line.

```
\newdimen\Mfx@strutheight
\edef\marginpar{%
  \unexpanded{\setbox\@tempboxa\hbox{\strut}\Mfx@strutheight\ht\@tempboxa}%
  \expandafter\unexpanded\expandafter{\marginpar}%
}
```

12 Shipout-time patches

\@combinefloats
\MFX@combinefloats@before

We need to patch in somewhere before \@combinefloats at the latest, so that any heights calculated from \@pageht are correct—otherwise the top figures will

confuse us. So we'll start by simply adding our own `\MFX@combinefloats@before` at the very beginning of `\@combinefloats`

```
\expandafter\def\expandafter\@combinefloats\expandafter{\expandafter
  \MFX@combinefloats@before\@combinefloats}
```

`\MFX@combinefloats@before` is then responsible for picking the needed notes from `\mfx@marginlist`, building them into a box, and attaching said box onto the correct side of `\@outputbox`. This is also a convenient place to reset `\Mfx@marginheight` to zero (since we reuse the register for one-time extensions).

```
\def\MFX@combinefloats@before{%
  \MFX@buildmargin
  \MFX@attachmargin
  \Mfx@marginheight\marginheightadjustment
}
```

`\MFX@attachmargin` We'll start with the second half of `\MFX@combinefloats@before`, since it's simpler. We need to do several things here.

```
\def\MFX@attachmargin{%
⟨debug⟩\MFX@debug{attachmargin}%
```

First, we need to make sure that the boxes we're combining are the same size.

```
  \ifdim\ht\@outputbox<\ht\Mfx@marginbox
    \setbox\@outputbox\vbox to \ht\Mfx@marginbox{%
      \box\@outputbox
      \vfill
    }%
  \else
    \setbox\Mfx@marginbox\vbox to \ht\@outputbox{%
      \unvbox\Mfx@marginbox
      \vfill
    }%
  \fi
```

Next we need to figure out which side of `\@outputbox` to attach the `\Mfx@marginbox` on.

```
  \setbox\@outputbox\vbox to \ht\@outputbox{%
    \hbox to \wd\@outputbox{%
      \if\MFX@leftmargin
        \llap{\box\Mfx@marginbox\hskip\marginparsep}%
        \box\@outputbox
      \else
        \box\@outputbox
        \rlap{\hskip\marginparsep\box\Mfx@marginbox}%
      \fi
  }}%
}
```

`\MFX@buildmargin` When `\MFX@buildmargin` is called, we have a list of tokens in `\mfx@marginlist` that need to be processed. After it's done working, `\mfx@marginlist` should have the first $n > 0$ of these removed, and leaving only notes that were deferred to the next page. Additionally, `\Mfx@marginbox` must contain a box the same height as `\@outputbox`. We do this in several steps. We start with a height of `\@colroom` rather than `\textheight` because `\@colroom` has already subtracted off the top/bottom floats that have been set. Eventually we will want to encroach

on these floats, particularly if they don't encroach on the margin, but that will be tricky...

```
\def\MFX@buildmargin{%
  \advance\Mfx@marginheight\@colroom
⟨debug⟩\MFX@debug{buildmargin: marginheight=\the\Mfx@marginheight}%
  \MFX@buildmargin@down
  \MFX@buildmargin@up
}
```

12.1 First pass

\MFX@buildmargin@down

The first step is the "down" step, in which we move the notes that will go on the current page into `\mfx@marginout` in reverse order, and anything that will be deferred gets put back in `\mfx@marginlist`. This behavior is configured by changing the meaning of `\mfx@margin@note`, `\mfx@margin@skip`, and `\mfx@margin@clear`. Note that during the course of processing `\mfx@marginlist`, these meanings will continue to change.

```
\def\MFX@buildmargin@down{%
⟨debug⟩\MFX@debug{buildmargin@down: ENTRY}%
⟨debug⟩\MFX@debug{marginlist=\meaning\mfx@marginlist}%
  \let\mfx@margin@note\MFX@margin@note@down
  \let\mfx@margin@skip\@gobble
  \let\mfx@margin@clear\MFX@margin@clear@down
  \let\mfx@marginout\@empty
```

We do a little bit of `\expandafter` trickery here to first expand the current meaning of `\mfx@marginlist`, then clear it before actually executing anything.

```
  \expandafter\global\expandafter\let
  \expandafter\mfx@marginlist\expandafter\@empty
  \mfx@marginlist
⟨debug⟩\MFX@debug{buildmargin@down: RETURN}%
⟨debug⟩\MFX@debug{marginlist=\meaning\mfx@marginlist}%
}
```

We must now define the different meanings for the `\mfx@margin@...` macros.

\MFX@margin@note@down
\MFX@whichbox
\MFX@margin@skip@down
\MFX@margin@clear@down

We'll dive right into the notes first. When we see a note in the `\marginlist`, we need to do several things.

1. optionally add a *compressible* skip before it, of length `#3` − `\Mfx@marginpos`, to line it up with the callout location
2. figure out which box we need for the current page, not yet freeing the allocated boxes, since we may still end up deferring
3. add the height of the correct box to `\Mfx@marginboxspace` and `\Mfx@marginpos`
4. if `\Mfx@marginboxspace` exceeds `\Mfx@marginheight`, then we defer this note and change the meaning of `\mfx@margin@note` to defer all remaining notes
5. otherwise, we add tokens to construct the box to `\toks@` and free the allocated inserts.

6. redefine `\mfx@margin@skip` to `\MFX@margin@skip@down`, now that we've gotten a real note (it starts out as `\@gobble` since we don't want to apply skips at the very beginning).

```
\def\MFX@margin@note@down#1#2#3{%
⟨debug⟩\MFX@debug{note@down: ENTRY: #1/#2 at #3}%
  \ifdim#3>\Mfx@marginpos
    \dimen@\dimexpr#3-\Mfx@marginpos\relax
⟨debug⟩\MFX@debug{note@down: adding compressible \the\dimen@}%
    \MFX@snoc\mfx@marginout{\noexpand\mfx@margin@compressible{\the\dimen@}}%
    \advance\Mfx@marginpos\the\dimen@
  \fi
  \MFX@whichbox#1#2%
  \advance\Mfx@marginboxspace\dimexpr\ht\@marbox+\dp\@marbox\relax
  \ifdim\Mfx@marginboxspace>\Mfx@marginheight
```

We've run out of margin space, so we now defer every following box, which means appending them back to `\mfx@marginlist`, except with a callout position of zero, since we want them as high up as possible (also, the position should be monotonic).

```
⟨debug⟩\MFX@debug{note@down: out of space:
⟨debug⟩     \the\Mfx@marginboxspace>\the\Mfx@marginheight}%
    \advance\Mfx@marginboxspace\dimexpr-\ht\@marbox-\dp\@marbox\relax
    \mfx@margin@clear
    \mfx@margin@note#1#2{#3}%
  \else
```

At this point, the box is definitely going onto this page, so we can arrange for the boxes to be freed and then add the correct box to the output list.

```
⟨debug⟩\MFX@debug{note@down: adding \@marbox
⟨debug⟩     ht \the\ht\@marbox dp \the\dp\@marbox}%
    \advance\Mfx@marginpos\dimexpr\ht\@marbox+\dp\@marbox\relax
    \MFX@snoc\mfx@marginout{%
      \noexpand\mfx@margin@note\@marbox
      \noexpand\@cons\noexpand\@freelist#1%
      \noexpand\@cons\noexpand\@freelist#2%
    }%
  \fi
  \let\mfx@margin@skip\MFX@margin@skip@down
⟨debug⟩\MFX@debug{note@down: RETURN space=\the\Mfx@marginboxspace,
⟨debug⟩            pos=\the\Mfx@marginpos}%
}
```

We need to figure out which box to set. This macro calls `\MFX@leftmargin` and then sets the correct box into `\@marbox` for use by `\MFX@margin@box@down`.

```
\def\MFX@whichbox#1#2{%
  \if\MFX@leftmargin
    \def\@marbox{#1}%
  \else
    \def\@marbox{#2}%
  \fi
⟨debug⟩\MFX@debug{whichbox: \@marbox}%
}
```

The next macro we'll write is the skip. This one just has to save itself onto whichever list we're working on (output or deferred) and in the case of output, update `\@Mfx@marginpos`.

```
\def\MFX@margin@skip@down#1{%
⟨debug⟩\MFX@debug{skip@down #1}%
  \advance\Mfx@marginpos#1\relax
  \advance\Mfx@marginboxspace#1\relax
  \MFX@snoc\mfx@marginout{\noexpand\mfx@margin@skip{#1}}%
}
```

Finally, `\MFX@margin@clear@down` simply signals the end of this margin. All further material will be deferred, so we redefine `\mfx@margin@note`, `\mfx@margin@skip`, and `\mfx@margin@clear` to defer.

```
\def\MFX@margin@clear@down{%
⟨debug⟩\MFX@debug{clear@down}%
  \def\mfx@margin@note##1##2##3{%
    \MFX@cons\mfx@marginlist{\noexpand\mfx@margin@note##1##2{\MFX@minus@inf}}}%
  \def\mfx@margin@skip##1{%
    \MFX@cons\mfx@marginlist{\noexpand\mfx@margin@skip{##1}}}%
  \def\mfx@margin@clear{%
    \MFX@cons\mfx@marginlist{\noexpand\mfx@margin@clear}}%
}
```

`\MFX@minus@inf` Note that when we added deferred boxes to the list, we put them at `\MFX@minus@inf`. We'll define that to be a large negative dimension.

```
\def\MFX@minus@inf{-4000pt}
```

`\MFX@leftmargin` Here we figure out which box to use based on the page number and other flags. This is a conditional that should be used after `\if`, as in `\if\MFX@leftmargin...\else...\fi`. This is different from the corresponding code in the LaTeX routines because we don't support double columns. In addition, we would ideally allow `\if@reversemargin` to work on a per-note basis (i.e. at callout time) but we also need something working at shipout time so we can figure out which margin to use. Thus, until we figure out how to use multiple margins.

```
\def\MFX@leftmargin{%
  00\fi % close out the \if
  \@tempcnta\@ne
  \if@mparswitch
    \unless\ifodd\c@page
      \@tempcnta\m@ne
    \fi
  \fi
  \if@reversemargin
    \@tempcnta-\@tempcnta
  \fi
⟨debug⟩\MFX@debug{margin on \ifnum\@tempcnta<\z@ left\else right\fi}%
  \ifnum\@tempcnta<\z@ % start a new \if
}
```

12.2 Second pass

`\MFX@buildmargin@up` Next is the "up" step. Here we simply take the reversed list in `\mfx@marginout` and prepend each item in turn to `\Mfx@marginbox`. We start by automatically discarding any skips, though we can't do this with a simple `\@gobble` anymore since we need to deduct them from `\Mfx@marginpos`. Once we hit a note, we'll

change the skips to be their normal meanings. Note that there are no clears in \mfx@marginout.

```
\def\MFX@buildmargin@up{%
⟨debug⟩\MFX@debug{buildmargin@up: excess=\the\dimexpr
⟨debug⟩            \Mfx@marginpos-\Mfx@marginheight\relax}%
⟨debug⟩\MFX@debug{marginout=\meaning\mfx@marginout}
  \let\mfx@margin@note\MFX@margin@note@up
  \let\mfx@margin@compressible\MFX@margin@skip@gobble@up
  \let\mfx@margin@skip\MFX@margin@skip@gobble@up
  \mfx@marginout
}
```

\MFX@margin@skip@gobble@up In case we have any skips at the beginning of \mfx@marginout, we'll gobble them and deduct their lenghts from \Mfx@marginpos.

```
\def\MFX@margin@skip@gobble@up#1{%
⟨debug⟩\MFX@debug{skip@gobble@up: #1}%
  \advance\Mfx@marginpos-#1\relax
}
```

\MFX@margin@note@up Once we actually do hit a note, we need to set it in \Mfx@marginbox. We also redefine \mfx@margin@skip and \mfx@margin@compressible here. Hopefully any skips aren't being dropped by our \unvboxing.

```
\def\MFX@margin@note@up#1{%
⟨debug⟩\MFX@debug{note@up: #1}%
  \setbox\Mfx@marginbox\vbox{\box#1\unvbox\Mfx@marginbox}%
  \let\mfx@margin@skip\MFX@margin@skip@up
  \let\mfx@margin@compressible\MFX@margin@compressible@up
}
```

\MFX@margin@skip@up This one is even easier—all we have to do is add a skip to the margin output box. We also need to make sure it's not getting dropped, which might entail adding some \vbox{}s.

```
\def\MFX@margin@skip@up#1{%
⟨debug⟩\MFX@debug{skip@up: #1}%
  \setbox\Mfx@marginbox\vbox{\vskip#1\relax\unvbox\Mfx@marginbox}%
}
```

\MFX@margin@compressible@up This token gets put into \mfx@marginout during the first pass. When it executes, it inserts a \vskip depending on how much extra margin space (stored in \Mfx@marginpos) we need to excise. If also updates \Mfx@marginpos.

```
\def\MFX@margin@compressible@up#1{%
⟨debug⟩\MFX@debug{compressible@up: #1, excess=\the\dimexpr
⟨debug⟩            \Mfx@marginpos-\Mfx@marginheight\relax}%
  \dimen@#1\relax
  \ifdim\Mfx@marginpos>\Mfx@marginheight
    \advance\dimen@\dimexpr\Mfx@marginheight-\Mfx@marginpos\relax
    \ifdim\dimen@<\z@
      \dimen@\z@
    \fi
    \advance\Mfx@marginpos\dimexpr\dimen@-#1\relax
  \fi
  \ifdim\dimen@>\z@
    \MFX@margin@skip@up\dimen@
```

```
  \fi
}
```

13 Cleaning up

We need to worry about a few more things. First, what happens if we reach the end of the document and there are still deferred margin notes? We need to be able to dump all the margin notes whenever the user wants (i.e. before a new chapter), so we'll make a macro `\dumpmargins` to do this, and then make sure it gets called `\AtEndDocument`. Since we're looping to do this, we need to make darned sure that every `\newpage` shrinks the marginlist.

`\dumpmargins`

```
\def\dumpmargins{%
⟨debug⟩\MFX@debug{dumpmargins}%
  \loop
  \unless\ifx\mfx@marginlist\@empty
    \let\temp@\mfx@marginlist
    \vbox{}\clearpage
    \ifx\temp@\mfx@marginlist
      \PackageError{marginfix}{lost some margin notes%
⟨debug⟩: \meaning\mfx@marginlist
      }\@eha
      \let\mfx@marginlist\@empty % be nicer by just dropping one?
      % TODO: also, set an emergency mode to allow oversized notes
    \fi
  \repeat
}
\AtEndDocument{\dumpmargins}
```

14 User macros

`\marginskip` Inserting a skip in the margin list is simple. We need only append `\mfx@margin@skip` to `\mfx@marginlist`.

```
\def\marginskip#1{%
  \MFX@cons\mfx@marginlist{\noexpand\mfx@margin@skip{#1}}%
}
```

`\clearmargin` Likewise, `\clearmargin` is easy too.

`\softclearmargin`

```
\def\clearmargin{%
  \MFX@cons\mfx@marginlist{\noexpand\mfx@margin@clear}%
}
```

While we call `\softclearmargin` a "clear margin", it's actually just a big `\marginskip`. This allows us to stack multiple copies without backing them all up.

```
\def\softclearmargin{%
  \marginskip{\the\textheight}%
}
```

`\extendmargin` We overload `\Mfx@marginheight` to be the amount of extension at all times except shipout-time.

```
\def\extendmargin#1{%
  \advance\Mfx@marginheight#1\relax
}
```

`\mparshift` This is as simple as setting the dimen register. We advance so that the shifts are cumulative, but there's not really any point either way.

```
\def\mparshift#1{%
  \advance\Mfx@mparshift#1\relax
}
```

15 Random scribbles

Later we'll get fancier with putting notes next to top/bottom figures but for now, not so much.
In the future we will support the use of `\pdfsavepos` and `\pdflastypos` for more accurately determining where the callouts actually were, which will end up going right around here. But in order to work with older versions of LaTeX, we still need to support the old style of using `\@pageht` to figure that out, so for now that's all we'll do.

16 Parting words

Finish it up

```
\makeatother
⟨/package⟩
```