marginfix package documentation

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

v1.1 – 2013/09/08

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. The credit for the concept behind this algorithm goes to Prof. Andy Ruina, who employed me to work on some of his textbook macros in 2007–9.

2 Options

There are currently no options that do anything yet.

3 Commands

For the most part, this is a drop-in replacement. Simply include a call to `\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

\clearmargins 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 a call to \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 margin notes are in the wrong place, the callout positions can be adjusted globally 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.

\blockmargin
\unblockmargin As of version 1.0, we now support "margin phantoms": sections of the margin in which no notes will be placed, which can be useful for large figures that jut into the margin (note: margins already move out of the way of floats, regardless of whether or not they extend into the margin; this is mainly for in-place figures or equations). The easiest way to block off part of the margin is to call \blockmargin before the extended content and \unblockmargin afterwards. No margin notes will be placed between these two points (though one must be careful: if one of these is called in horizontal mode, the toggle will occur at the *top* of the current line). Each of these commands takes an optional argument: \blockmargin[⟨*pos*⟩] will begin the margin block at a position *pos* below the current position (or above if *pos* is negative), and \unblockmargin[⟨*pos*⟩] will likewise end the block at a position *pos* below the current position.

\marginphantom Margin phantoms may also be called out in place with a known size using \marginphantom[⟨*pos*⟩]{⟨*size*⟩}, which is essentially equivalent to \blockmargin[⟨*pos*⟩]\unblockmargin[
Either argument may be negative to refer upward rather than downward.

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.

- Use of pdfTeX's `\pdfsavepos` and `\pdflastypos` for more accurate margin placement.
- `\vadjust` to correct inconsistencies with `\@pageht`.
- Margin note placement is irrespective of vertical stretch. Previously we gobbled any vertical stretch, but now that we have fixed that bug, there's the possibility of wrong alignment since we don't know where the positions will ultimately end up. This may be fixed by `\pdfsavepos` as well.
- 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

Make the `@`-sign into a letter for use in macro names.

```
1 ⟨*package⟩
2 \makeatletter
```

`\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{\message{^^JMFX:}\message}
\errorcontextlines=20
\def\MFX@mac#1{\expandafter\MFX@@mac\meaning#1>>>}
\def\MFX@@mac#1->{<<<}
\def\MFX@htdp#1{\ht#1=\the\ht#1, \dp#1=\the\dp#1}
\showboxbreadth=100
\showboxdepth=10
⟨/debug⟩
```

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 (with clever usages of `\expandafter`), 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. There are no options yet.
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@inject` `\Mfx@inject@insert` These are used to hijack `\marginpar` to inject arbitrary code into the output routine, rather than actually set a note. To inject code, we unshift two copies of this dummy insert onto `\@freelist` for `\marginpar` to pull off. We append whatever code we want to inject into `\mfx@inject` and then call `\marginpar`. Then our custom `\@addmarginpar` will recognize the dummy inserts and run the code instead of setting a margin note.

```
\let\mfx@injected\@empty
\newinsert\Mfx@inject@insert
```

`\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@marginpos@min
\Mfx@marginpos@max
\Mfx@marginspace

While we build up the margin piece boxes, we need to keep track of the possible range of positions. The pair `\Mfx@marginpos@min` and `\Mfx@marginpos@max` are used to accumulate how much material has been added so far, with the difference that `\Mfx@marginpos@min` doesn't take into account compressible space, while `\Mfx@marginpos@max` does. Finally, `\Mfx@marginspace` is the amount of (incompressible) space since the last note, which allows skips to span margin phantoms.

```
\newdimen\Mfx@marginpos@min
\newdimen\Mfx@marginpos@max
\newdimen\Mfx@marginspace
```

\Mfx@marginheight

Because the margin height can be altered by, `\extendmargin`, we must maintain a dimension for the height of the current margin. This dimension is reused in several different ways in the shipout-time margin building routines, keeping track of how much much space is left in (for the global passes) and the end position of the end of (for the piecewise passes) the current piece.

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

\mfx@marginstart
\mfx@marginpieces
\Mfx@piece@content
\Mfx@piece@count
\ifmfx@in@phantom

These control sequences keep track of the margin phantoms. When the margin is unblocked then `\mfx@marginstart` is not `\relax`. When a phantom begins, the current `\mfx@marginstart` and page position are stored as a pair in `\mfx@marginpieces`, which is iterated over while building individual pieces of the margin. We also define a box to keep the content of each margin piece, and a counter to keep track of how many pieces we have. Finally, we define a switch for use in the second pass to indicate that we're inside a phantom.

```
\def\mfx@marginstart{0pt}
\let\mfx@marginpieces\@empty
\newbox\Mfx@piece@content
\newcount\Mfx@piece@count
\newif\ifmfx@in@phantom
```

\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 (or \@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. In addition to actually setting the margin note, we also use this routine to inject arbitrary code into the OR (see \MFX@inject).

```
\def\@addmarginpar{%
  \@next\@marbox\@currlist{}\MFX@AssertionError
⟨debug⟩\MFX@debug{addmarginpar (running insert) \@marbox/ \@currbox at
⟨debug⟩    \the\c@page:\the\@pageht, marginlist=\MFX@mac\mfx@marginlist}%
⟨debug⟩\MFX@debug{addmarginpar outputpenalty=\the\outputpenalty}%
  \MFX@getypos
  \expandafter\ifx\@marbox\Mfx@inject@insert
    \mfx@injected\global\let\mfx@injected\@empty
  \else
    \MFX@cons\mfx@marginlist{%
      \noexpand\mfx@build@note\@currbox\@marbox{\mfx@ypos}%
      \noexpand\mfx@build@skip{\the\marginparpush}%
    }%
  \fi
⟨debug⟩\MFX@debug{addmarginpar (exit): marginlist=\MFX@mac\mfx@marginlist}%
}
```

\MFX@cons
\MFX@snoc
\MFX@run@clear

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}%
}
```

Finally, `\MFX@run@clear` is a quick trick to expand the contents of a macro and then clear it (to `\@empty`) before any of its tokens are consumed.

```
\def\MFX@run@clear#1{%
  \expandafter\global\expandafter\let\expandafter#1\expandafter\@empty#1%
}
```

`\MFX@inject` As mentioned earlier, `\@addmarginpar` is also a hook for injecting arbitrary code into the output routine, i.e. to get a vertical position for blocking the margin. We define `\MFX@inject` to facilitate this.

```
\def\MFX@inject#1{
  \expandafter\def\expandafter\@freelist\expandafter{%
    \expandafter\@elt\expandafter\Mfx@inject@insert
    \expandafter\@elt\expandafter\Mfx@inject@insert
    \@freelist}%
  \expandafter\def\expandafter\mfx@injected\expandafter{\mfx@injected#1}%
  \marginpar{}%
}
```

`\MFX@getypos` `\mfx@ypos` We now need to settle on a way to determine the vertical position. Someday this may 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 should ultimately be only a fallback for when `\pdflastypos` isn't available, it's good enough. (NOTE: we might be able to use a `\vadjust` instead here?)

```
\def\MFX@getypos{%
  \dimen@\dimexpr\@pageht+\@pagedp+\marginposadjustment+\Mfx@mparshift\relax
  \ifnum\outputpenalty=-10002\relax
    \advance\dimen@-\Mfx@strutheight
  \fi
  \edef\mfx@ypos{\the\dimen@}%
  \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 that box onto the correct side of \@outputbox. We also add any global \marginheightadjustment to \Mfx@marginheight before building the margin, and then reset it back to zero at the end. This allows any calls to \extendmargin during the page itself to work as expected.

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

\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.

```
⟨debug⟩\MFX@debug{attachmargin: \MFX@htdp\@outputbox, \MFX@htdp\Mfx@marginbox}%
 %<debug>\showbox\@outputbox
 %<debug>\showbox\Mfx@marginbox
  \setbox\Mfx@marginbox\vtop{%
    \vskip\z@\unvbox\Mfx@marginbox}%
  %% \ifdim\ht\@outputbox<\ht\Mfx@marginbox
  %%   \setbox\Mfx@marginbox\vbox to \ht\@outputbox{%
  %%     \unvbox\Mfx@marginbox
  %%     \vskip\z@ shrink 1filll\relax
  %%   }%
  %% \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. We now use \columnwidth instead of \wd\@outputbox to set the right-hand margins, since tufte-LaTeXsometimes makes too-wide output boxes. If this becomes a problem, we'll need to consider making this configurable elsewhere. We should also pay attention to whether adding a box at the top of \@outputbox

might have unintended consequences w.r.t. any glue being retained that should have been swallowed. This will require further investigation.

```
\setbox\@outputbox\vbox{% to \ht\@outputbox{%
  \begingroup
  \setbox\@tempboxa\vbox{% to \ht\@outputbox{%
    \hbox{% to \wd\@outputbox{%
      \if\MFX@leftmargin
        \llap{\box\Mfx@marginbox\hskip\marginparsep}%
      \else
        \hskip\columnwidth
        \rlap{\hskip\marginparsep\box\Mfx@marginbox}%
      \fi
    }}%
  \ht\@tempboxa\z@
  \dp\@tempboxa\z@
  \box\@tempboxa
  \endgroup
  \unskip
  \unvbox\@outputbox
 }%
 %<debug>\showbox\@outputbox
}
```

`\MFX@buildmargin` When `\MFX@buildmargin` is called, we have a list of tokens in `\mfx@marginlist` that need to be processed: combinations of `\mfx@build@note`, `\mfx@build@skip`, and `\mfx@build@clear`, with various parameters to indicate what material still needs to be set in the margin. This macro therefore must pull off the first $n > 0$ of these commands to set on the current page (n must be positive to prevent infinite loops), and leave the rest to be deferred. We do not currently support taking notes out of order, though that is a possible feature to allow in the future, on an opt-in basis. The typeset material will be left in `\Mfx@marginbox`, which must have the same height as `\@outputbox` (although because we `\unvbox\@outputbox` in `\MFX@attachmargin`, we can't guarantee that this will correctly line up the notes with their callouts). This procedure happens in four passes. But first, we initialize `\Mfx@marginheight` to `\@colroom`, which is the height of the page minus any floats that have been added to the top of bottom (these floats may extend into the margins: in the future we may look into detecting this and using the whole page, with overwide floats blocked off as phantoms). We add `\@colroom` rather than assigning it because any global or per-page adjustments have already been added to `\Mfx@marginheight`. We can then close out any still-open margin pieces (this is the typical case, where the margin is not blocked across a page boundary, so that `\mfx@marginstart` will hold a position, rather than `\relax`). After this, `\Mfx@marginheight` is no longer necessary, so we reuse it for keeping track of available space in individual margin pieces.

```
\def\MFX@buildmargin{%
  \advance\Mfx@marginheight\@colroom
  \ifx\mfx@marginstart\relax
  \else
```

```
    \MFX@cons\mfx@marginpieces{%
      \noexpand\@elt{\mfx@marginstart}{\the\Mfx@marginheight}}%
    \gdef\mfx@marginstart{0pt}%
    \global\advance\Mfx@piece@count\@ne
  \fi
⟨debug⟩\MFX@debug{buildmargin: marginheight=\the\Mfx@marginheight,
⟨debug⟩            marginlist=\MFX@mac\mfx@marginlist,
⟨debug⟩            marginpieces=\MFX@mac\mfx@marginpieces}%
```

We now execute the four passes. First is a global downward pass, whose purpose is to determine the maximum number of notes (and other material) that can fit in the margin, taking any phantoms into consideration. Every note identified by `\MFX@buildmargin@down` is guaranteed to show up on this page, so we free its inserts back to `\@freelist`. The second pass is the global upward pass, in which we determine the lowest possible margin piece each note may go into without causing lower notes to fall off the bottom. The third pass is the piecewise downward pass. For each piece, we figure out where in the piece each note will go by inserting compressible spaces between the notes. If a note is called out past the end of the piece and does not need to go into the piece (as determined by pass 2), it will be deferred to a later piece. The fourth pass is the piecewise upward pass, in which the compressible spaces are shrunk just enough to fit everything into the piece. The last two (piecewise) passes both occur in each piece before the next piece is addressed. The whole process is bypassed if there are no eligible margin pieces.

```
  \ifx\mfx@marginpieces\@empty\else
    \MFX@buildmargin@down
    \MFX@buildmargin@up
    \MFX@buildmargin@pieces
  \fi
}
```

12.1 First pass: global downward

`\MFX@buildmargin@down`
`\mfx@pieceheights`

The first step is the global "down" step, in which we move the notes that will fit on the current page into `\mfx@marginout` in reverse order (to prepare for the second, upward, pass), and anything that doesn't fit is deferred back into `\mfx@marginlist`. We do this by changing the meaning of `\mfx@build@note`, `\mfx@build@skip`, and `\mfx@build@clear`, which delimit the different types of material in `\mfx@marginlist`. Note that as we continue processing, these macros will change from time to time (i.e. changing `\mfx@build@skip` to actually doing something once we find a note, rather than gobbling so as to remove skips at page boundaries; or changing them to save material back onto `\mfx@marginlist` once the margin fills up). The first thing we need to do is iterate over the piece positions to get the list of heights.

```
\def\MFX@buildmargin@down{%
  \let\mfx@pieceheights\@empty
  \def\@elt##1##2{%
    \MFX@cons\mfx@pieceheights{\noexpand\@elt{\the\dimexpr##2-##1}}}%
  \mfx@marginpieces
```

```
  \MFX@popdimen\Mfx@marginheight\mfx@pieceheights
```

Now we run forwards over the `\mfx@marginlist` to actually operate on each thing in the margin.

```
  \let\mfx@build@note\MFX@margin@note@down
  \let\mfx@build@skip\@gobble
  \let\mfx@build@clear\MFX@build@clear@down
  \let\mfx@marginout\@empty
  \MFX@run@clear\mfx@marginlist
⟨debug⟩\MFX@debug{buildmargin@down: RETURN
⟨debug⟩            marginout=\MFX@mac\mfx@marginout,
⟨debug⟩            marginlist=\MFX@mac\mfx@marginlist}%
}
```

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

We now define the various `\MFX@margin@...@down` macros. At this stage in the game, the only difference between notes and skips is that we ignore skips before any notes by setting `\mfx@build@note` initially to `\@gobble`. Once we've seen the first note, skips are treated exactly the same: as fixed-height material. If there is room in the current piece for the given height, then we prepend it to `\mfx@marginout`, decrement the remaining height, and arrange for the boxes to be freed. If not, we unshift the next piece height from `\mfx@pieceheights` and try again, until `\mfx@pieceheights` is empty and we simply defer everything to later pages.

Upon seeing a note, we must do several things:

1. determine which box (left or right) is needed for the current page, by calling `\MFX@whichbox`
2. if the box fits, free both boxes, prepend `\mfx@marginout` with a call to `\mfx@build@note`, and re-enable skips
3. otherwise, defer the current note and all future notes

The latter two steps are taken care of by `\MFX@margin@fit`, which takes the height and two blocks of material: one to prepend to `\mfx@marginout` if it fits, the other to append to `\mfx@marginlist` if it doesn't.

```
\def\MFX@margin@note@down#1#2#3{%
⟨debug⟩\MFX@debug{margin@note@down: ENTRY: #1/ #2 at #3}%
  \MFX@whichbox\@marbox#1#2%
  \if\MFX@check@fit{}{\ht\@marbox+\dp\@marbox}%
    \MFX@snoc\mfx@marginout{%
      \noexpand\@cons\noexpand\@freelist#1%
      \noexpand\@cons\noexpand\@freelist#2%
      \noexpand\mfx@build@note\@marbox{#3}}%
    \let\mfx@build@skip\MFX@margin@skip@down
  \else
    \mfx@build@clear
    \mfx@build@note{#1}{#2}{#3}%
  \fi
}
```

Skips are similar. A skip needs only to save itself back into `\mfx@marginout`, provided it fits. If not, there is no need to defer it because it will just get gobbled at the top of the next page anyway.

```
\def\MFX@margin@skip@down#1{%
⟨debug⟩\MFX@debug{margin@skip@down #1}%
  \if\MFX@check@fit{}{#1}%
    \MFX@snoc\mfx@marginout{\noexpand\mfx@build@skip{#1}}%
  \else
    \mfx@build@clear
  \fi
}
```

Finally, `\MFX@margin@clear@down` is the only place we actually need to handle full-margin clears, since the downward pass does not ever push `\mfx@build@clear` onto `\mfx@marginout`. When we see this, we simply redefine all three commands to append themselves back to `\mfx@marginlist`.

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

`\MFX@check@fit` `\MFX@popdimen`

We factored out some of the common functionality between the note and skip routines, so that must now be defined. The `\MFX@check@fit` macro acts as a conditional and should be used as `\if\MFX@check@fit{`⟨*piece-hook*⟩`}{`⟨*size*⟩`}`. It takes care of iterating through the list of heights and accumulating the total size of material encountered so far. The *piece-hook* is executed each time a new piece height is popped.

```
\def\MFX@check@fit#1#2{%
  00\fi % close out the \if
⟨debug⟩\MFX@debug{check@fit{\unexpanded{#1}}{#2=\the\dimexpr#2} ENTRY:
⟨debug⟩           marginheight=\the\Mfx@marginheight}%
  \@tempswafalse
  \ifdim\dimexpr#2<\Mfx@marginheight % it fits
    \advance\Mfx@marginheight-\dimexpr#2\relax % deduct the size
    \@tempswatrue
  \else % didn't fit: check the next piece
⟨debug⟩\MFX@debug{check@fit overflow: pieceheights=\MFX@mac\mfx@pieceheights}%
    \ifx\mfx@pieceheights\@empty\else % make sure there's anything there
      #1%
      \MFX@popdimen\Mfx@marginheight\mfx@pieceheights
      \if\MFX@check@fit{#1}{#2}\fi
    \fi
  \fi
⟨debug⟩\MFX@debug{check@fit RETURN \meaning\if@tempswa:
```

```
⟨debug⟩            marginheight=\the\Mfx@marginheight,}%
  \if@tempswa % start a new \if
}
```

Here is a quick convenience routine. `\MFX@popdimen{`⟨*dimen*⟩`} {`⟨*list*⟩`}` removes the first dimension from *list* and stores it into *dimen*.

```
\def\MFX@popdimen#1#2{%
  \def\@elt##1{%
    #1##1\relax
    \def\@elt####1{%
      \MFX@cons#2{\noexpand\@elt{####1}}%
    }%
  }%
  \MFX@run@clear#2%
}
```

`\MFX@whichbox`
`\MFX@leftmargin`

We also need to determine which box should be used, since they may have different heights. The macro `\MFX@whichbox {`⟨*target-box*⟩`}{`⟨*left-box*⟩`}{`⟨*right-box*⟩`}` checks which margin we're setting and stores the correct box into *target-box*. Note that *target-box* must be a single control sequence.

```
\def\MFX@whichbox#1#2#3{%
  \if\MFX@leftmargin
    \def#1{#2}%
  \else
    \def#1{#3}%
  \fi
⟨debug⟩\MFX@debug{whichbox: \@marbox (\the\dimexpr\ht#1+\dp#1)}%
}
```

And here is the logic to figure out which margin were in, based on the page number and other flags. This is another conditional-like macro, and should be used after an `\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, this will have to do.

```
\def\MFX@leftmargin{%
  OO\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
}
```

\MFX@minus@inf

Finally, note that when deferring notes to the next page, we adjust their position to the top of the page, rather than the callout position. This is a large negative dimension (near TEX's maximum), but we may reconsider making this zero or even a small positive amount, since there seems to be a small amount of space before the first paragraph in normal text, though I'm not sure where that comes from.

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

12.2 Second pass: global upward

\MFX@buildmargin@up
\mfx@phantomheights

The next step is the global "up" step, in which we figure out the lowest piece a note can possibly occupy (without pushing later notes off the bottom) and add this information to the \mfx@build@note in \mfx@marginout. We start similar to \MFX@buildmargin@down, except we need the list of heights to be backwards. We also need a list of phantom heights, in order to handle skips properly, which we intersperse as negative heights.

```
\def\MFX@buildmargin@up{%
⟨debug⟩\MFX@debug{buildmargin@up: ENTRY
⟨debug⟩           marginpieces=\MFX@mac\mfx@marginpieces,
⟨debug⟩           marginout=\MFX@mac\mfx@marginout}%
  \let\mfx@pieceheights\@empty
  \let\mfx@phantomheights\@empty
  \let\temp@@\relax
  \def\@elt##1##2{%
⟨debug⟩\MFX@debug{  -> piece (##1,##2), temp@@=\meaning\temp@@}%
    \MFX@snoc\mfx@pieceheights{\noexpand\@elt{\the\dimexpr##2-##1}}%
    \ifx\temp@@\relax\else
⟨debug⟩\MFX@debug{  -> phantom (\temp@@,##1)}%
      \MFX@snoc\mfx@phantomheights{\noexpand\@elt{\the\dimexpr##1-\temp@@}}%
    \fi
    \def\temp@@{##2}%
  }%
  \mfx@in@phantomfalse
  \mfx@marginpieces
```

The piece counter, \Mfx@piece@count has not been touched yet, so now we will start decrementing it for each piece to keep track of where we are. Note that \mfx@marginout will never contain \mfx@build@clear so we don't need to reassign it. Since we don't do any deferrals here, we don't need to empty out a new target list. Instead, we operate "in-place" in \mfx@marginout, after popping off the first height.

```
  \MFX@popdimen\Mfx@marginheight\mfx@pieceheights
  \let\mfx@build@note\MFX@margin@note@up
  \let\mfx@build@skip\@gobble
  \MFX@run@clear\mfx@marginout
```

```
⟨debug⟩\MFX@debug{buildmargin@up: RETURN marginout=\MFX@mac\mfx@marginout}%
}
```

`\MFX@margin@note@up`
`\MFX@margin@skip@up`

We must again define the specific behavior of each build command. These macros simply reuse `\MFX@check@fit`, but ask it to decrement the piece counter when a piece runs out of space. Aside from that, the only thing that actually happens here is that we append the current piece to each note, and also end up reversing the contents by again prepending everything. We are guaranteed that `\MFX@check@fit` will never fail. Since we cannot put notes in a phantom, we start by ensuring we're not in one.

```
\def\MFX@margin@note@up#1#2{%
⟨debug⟩\MFX@debug{margin@note@up: #1at #2, marginheight=\the\Mfx@marginheight}%
  \ifmfx@in@phantom
    \MFX@popdimen\Mfx@marginheight\mfx@pieceheights
    \advance\Mfx@piece@count\m@ne
    \mfx@in@phantomfalse
  \fi
```

Now we're guaranteed to be in a piece, rather than a phantom, we look for a piece that can fit this note, making sure to decrement the piece count and pop off a phantom for each new piece we check. Once it's found, we add the note back to `\mfx@marginout` with the corect piece.

```
  \if\MFX@check@fit{\advance\Mfx@piece@count\m@ne
                    \MFX@popdimen\dimen@\mfx@phantomheights}{\ht#1+\dp#1}%
    \MFX@snoc\mfx@marginout{%
      \noexpand\mfx@build@note{#1}{#2}{\the\Mfx@piece@count}}%
    \let\mfx@build@skip\MFX@margin@skip@up
  \else\MFX@AssertionError\fi
}
```

Skips are similar, but we have the added complication of handling margin phantoms. When we cross between phantom and piece, we split the skip so that we can use the simplest recursion possible.

```
\def\MFX@margin@skip@up#1{%
⟨debug⟩\MFX@debug{margin@skip@up: #1}%
  \dimen@#1\relax
  \advance\Mfx@marginheight-\dimen@
  \ifdim\Mfx@marginheight<\z@
```

This skip was bigger than the piece, so we need to split this skip, adding the overflow back, and try again. Since we're done with this piece, we'll pop the next one and recurse on whatever's left. This looks slightly different depending on whether or not we're in a phantom.

```
    \advance\dimen@\Mfx@marginheight
    \MFX@snoc\mfx@marginout{%
      \noexpand\mfx@build@skip{\the\dimen@}{\the\Mfx@piece@count}}%
    \dimen@-\Mfx@marginheight
    \ifmfx@in@phantom
      \MFX@popdimen\Mfx@marginheight\mfx@pieceheights
      \advance\Mfx@piece@count\m@ne
```

```
      \mfx@in@phantomfalse
    \else
      \MFX@popdimen\Mfx@marginheight\mfx@phantomheights
      \mfx@in@phantomtrue
    \fi
    \mfx@build@skip\dimen@
  \else
```

This skip fit entirely within the phantom, so we simply emit it.

```
    \MFX@snoc\mfx@marginout{%
      \noexpand\mfx@build@skip{\the\dimen@}{\the\Mfx@piece@count}}%
  \fi
}
```

12.3 Margin pieces

`\MFX@buildmargin@pieces` `\MFX@buildmargin@piece`

Before we can start the third and fourth passes, we need to set up a loop over the pieces so that each piece can do these passes at one time. In case we didn't use up all the pieces in the second phase, we'll reset `\Mfx@piece@count` to zero. We also reset `\Mfx@marginspace` and `\Mfx@marginbox`.

```
\def\MFX@buildmargin@pieces{%
  \Mfx@piece@count\z@
  \Mfx@marginspace\z@
  \setbox\Mfx@marginbox\vbox{\vskip\z@}%  TODO - do we need this?
```

Now we run over the individual margin pieces, clearing it out as we build up the contents of `\Mfx@marginbox`. Once we're done with that, we need to do a bit of clean-up before finishing.

```
  \let\@elt\MFX@buildmargin@piece
  \MFX@run@clear\mfx@marginpieces
  \let\@elt\relax
  \Mfx@piece@count\z@
}
```

The `\MFX@buildmargin@piece` macro is called for each piece of the margin, and is passed the top and bottom positions of the piece. Here we need to do a few things. First, if the output so far is smaller than the top of the piece (this is generally true, except sometimes for the first piece) then we need to insert padding into `\Mfx@marginbox` before continuing. We accumulate this padding into `\Mfx@marginspace`, which allows skips to do double-duty across phantoms.

```
\def\MFX@buildmargin@piece#1#2{%
⟨debug⟩\MFX@debug{buildmargin@piece: ENTRY (#1, #2)}%
  \ifdim\ht\Mfx@marginbox<#1\relax
    \dimen@\dimexpr#1-\ht\Mfx@marginbox\relax
⟨debug⟩\MFX@debug{buildmargin@piece: padding \the\dimen@}%
    \setbox\Mfx@marginbox\vbox{%
      \unvbox\Mfx@marginbox
      \vskip\dimen@
    }%
    \advance\Mfx@marginspace\dimen@
```

```
  \fi
```

Now that `\Mfx@marginbox` has been padded, we proceed to set things up for our down and up passes, and then run them to build `\Mfx@piece@content`.

```
  \Mfx@marginpos@min#1\relax
  \Mfx@marginpos@max#1\relax
  \Mfx@marginheight#2\relax
  \advance\Mfx@piece@count\@ne
  \MFX@buildpiece@down
  \MFX@buildpiece@up
```

Once `\Mfx@piece@content` has been built, we append it to the `\Mfx@marginbox`.

```
  \setbox\Mfx@marginbox\vbox{%
    \unvbox\Mfx@marginbox
    \box\Mfx@piece@content
    \vskip\z@
  }%
}
```

12.4 Third pass: piecewise downward

`\MFX@buildpiece@down` `\mfx@pieceout`

The next pass is another downward pass. This is very similar to the first (global downward) pass, except we're now moving the first several notes from `\mfx@marginout` into `\mfx@pieceout` (again, inverting the order) and deferring the rest back into `\mfx@marginout`.

```
\def\MFX@buildpiece@down{%
⟨debug⟩\MFX@debug{buildpiece@down: ENTRY piece=\the\Mfx@piece@count,
⟨debug⟩    marginpos=(\the\Mfx@marginpos@min, \the\Mfx@marginpos@max),
⟨debug⟩    marginspace=\the\Mfx@marginspace,
⟨debug⟩    marginout=\MFX@mac\mfx@marginout}%
  \let\mfx@build@note\MFX@piece@note@down
  \let\mfx@build@skip\MFX@piece@skip@down
  \let\mfx@pieceout\@empty
  \MFX@run@clear\mfx@marginout
⟨debug⟩\MFX@debug{buildpiece@down: RETURN pieceout=\MFX@mac\mfx@pieceout,
⟨debug⟩    marginout=\MFX@mac\mfx@marginout}%
}
```

`\MFX@piece@note@down` `\MFX@piece@skip@down` `\MFX@piece@clear`

We again define each of our building macros. First, the note builder. When we encounter a note, we first zero out `\Mfx@marginspace`. Then we need to decide whether to put it in the current piece, or whether to defer it to the next piece, based on a few signals. A note will only be deferred for one of two reasons. In either case, we store the decision to defer in `@tempswa`, so we'll start by clearing it.

```
\def\MFX@piece@note@down#1#2#3{%
⟨debug⟩\MFX@debug{piece@note@down: ENTRY: #1at #2, lowest=#3,
⟨debug⟩           marginpos=(\the\Mfx@marginpos@min,\the\Mfx@marginpos@max,
⟨debug⟩           marginspace=\the\Mfx@marginspace}%
  \Mfx@marginspace\z@
  \@tempswafalse
```

The first reason to defer is if its callout position (#2) is beneath the end of the current piece (\Mfx@marginheight) *and* if its lowest-possible-piece (#3, as computed in the second pass) is *after* the current one (\Mfx@piece@count).

```
\ifdim#2>\Mfx@marginheight
  \ifnum#3>\Mfx@piece@count
    \@tempswatrue
  \fi
\fi
```

The second possibility is that we've run out of room in this piece. In this case, there's no need to check #3, since that number was computed assuming everything that fit was as low as possible.

```
\ifdim\dimexpr\ht#1+\dp#1+\Mfx@marginpos@min>\Mfx@marginheight
  \@tempswatrue
\fi
```

If a note is deferred, then we push it and everything after it back onto \mfx@marginout.

```
\if@tempswa
⟨debug⟩\MFX@debug{piece@note@down: clearing margin}%
  \MFX@piece@clear
  \mfx@build@note{#1}{#2}{#3}%
\else
```

Otherwise, we have decided the note is going into this piece. In this case, we now need to check if any compressible space is needed above it. In order to get better alignment for deferred notes, we check for the case that the current position is zero and the note's callout position is \MFX@minus@inf. In this case, we change the callout position to instead be the greater of 0 or \topskip − \ht#1.

```
  \dimen@#2\relax
  \ifdim\dimen@=\MFX@minus@inf
    \ifdim\Mfx@marginpos@max=\z@
      \dimen@\topskip
      \advance\dimen@-\ht#1\relax
      \ifdim\dimen@<\z@ \dimen@\z@ \fi
    \fi
  \fi
  \advance\dimen@-\Mfx@marginpos@max
  \ifdim\dimen@>\z@
⟨debug⟩\MFX@debug{piece@note@down: adding compressible \the\dimen@}%
    \MFX@snoc\mfx@pieceout{\noexpand\mfx@build@compressible{\the\dimen@}}%
    \advance\Mfx@marginpos@max\dimen@
  \fi
```

After (maybe) adding the compressible space, we now add the (incompressible) box itself, and accumulate its height in both position registers. We no longer need the piece index or the position, so we only store the box itself here.

```
  \MFX@snoc\mfx@pieceout{\noexpand\mfx@build@note{#1}}%
  \advance\Mfx@marginpos@min\dimexpr\ht#1+\dp#1\relax
  \advance\Mfx@marginpos@max\dimexpr\ht#1+\dp#1\relax
```

```
  \fi
}
```

Skips are a bit more complicated now. We no longer gobble the initial skips (since the skips at the top and bottom of the page have already been eaten). Instead, we need to look at `\Mfx@marginspace`: if it's nonzero, we substract it from the skip length before adding it incompressibly (if there's any left). While we have piece number information here, we just pass it on to the upward pass, which can choose to "late-defer" initial (i.e. the bottom-most) skips in a piece.

```
\def\MFX@piece@skip@down#1#2{%
  \dimen@#1\relax
  \ifdim\Mfx@marginspace>\z@
    \advance\dimen@-\Mfx@marginspace
    \ifdim\dimen@<\z@ \dimen@\z@ \fi
    \advance\Mfx@marginspace-\dimen@
  \fi
```

At this point, `\dimen@` now stores any further incompressible skip we need to add, which is now relative to the top of this piece. In that case (note that `\Mfx@marginspace` is now necessarily zero), we need to check that it actually fits in the current piece, and if not, defer it.

```
  \ifdim\dimen@>\z@
    \ifdim\dimexpr#1+\Mfx@marginpos@min>\Mfx@marginheight
      \MFX@piece@clear
      \mfx@build@skip{\the\dimen@}{#2}%
    \else
```

If the skip *does* fit, we need to add it to `\mfx@pieceout` and advance the position registers.

```
      \MFX@snoc\mfx@pieceout{\noexpand\mfx@build@skip{\the\dimen@}{#2}}%
      \advance\Mfx@marginpos@min\dimen@
      \advance\Mfx@marginpos@max\dimen@
    \fi
  \fi
}
```

Finally, we need to handle the case of deferring material. By analogy with the previous two passes, we'll continue to refer to this as clearing. In this case, we need to redefine the note and skip macros to save themselves back to `\mfx@marginout`.

```
\def\MFX@piece@clear{%
⟨debug⟩ \MFX@debug{piece@clear}%
  \def\mfx@build@note##1##2##3{%
    \MFX@cons\mfx@marginout{\noexpand\mfx@build@note##1{##2}{##3}}}%
  \def\mfx@build@skip##1##2{%
    \MFX@cons\mfx@marginout{\noexpand\mfx@build@skip{##1}{##2}}}%
}
```

12.5 Fourth pass: piecewise upward

`\MFX@buildpiece@up`

We are now ready for the final pass, where we take the reversed list of notes for this piece and stack them up, compressing as much compressible space as

necessary to make them all fit. Since this is the last pass, we can finally prepend all the boxes into `\Mfx@piece@content`. Because it is still possible for pieces to have skips at the bottom, we need to worry about which piece these skips belong in. Since we've passed forward the lowest-piece information from pass 2, we can choose at the last possible moment to defer the bottom-most skips of a piece to the next piece (by prepending it to `\mfx@marginout`. Note that compressible spaces will *never* be at the beginning of `\mfx@pieceout`. We store the excess space in `\Mfx@marginheight`.

```
\def\MFX@buildpiece@up{%
  \Mfx@marginheight\dimexpr\Mfx@marginpos@max-\Mfx@marginheight\relax
  \ifdim\Mfx@marginheight<\z@\Mfx@marginheight\z@\fi
⟨debug⟩\MFX@debug{buildpiece@up: excess=\the\Mfx@marginheight}%
  \let\mfx@build@note\MFX@piece@note@up
  \let\mfx@build@compressible\MFX@piece@compressible@up
  \let\mfx@build@skip\MFX@piece@skip@maybedefer
  \MFX@run@clear\mfx@pieceout\relax
}
```

`\MFX@piece@skip@maybedefer` As mentioned earlier, the initial skips in a piece may be deferred to later pieces, provided they can possibly go there. Here we check the `#2` argument against `\Mfx@piece@count` and defer if it is larger. Otherwise, we switch back to the standard behavior defined in `\MFX@piece@skip@up`. Deferred skips do not need to be subtracted from the excess because they were never compressible in the first place.

```
\def\MFX@piece@skip@maybedefer#1#2{%
  \ifnum#2>\Mfx@piece@count
⟨debug⟩\MFX@debug{piece@skip deferring: #1, #2 (\the\Mfx@piece@count)}%
    \MFX@snoc\mfx@marginout{\noexpand\mfx@build@skip{#1}{#2}}%
  \else
    \let\mfx@build@skip\MFX@piece@skip@up
    \mfx@build@skip{#1}{#2}%
  \fi
}
```

`\MFX@piece@note@up` `\MFX@piece@skip@up` Now that we've taken care of late deferrals, we can define the standard behavior without worrying as much about that. These macros finally set their contents into `\Mfx@piece@content`, as well as reset `\mfx@build@skip` back to its standard value.

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

Skips are also straightforward.

```
\def\MFX@piece@skip@up#1#2{%
```

```
⟨debug⟩\MFX@debug{skip@up: #1=\the\dimexpr#1\relax (#2)}%
  \setbox\Mfx@piece@content\vbox{%
    \vskip#1\relax
    \unvbox\Mfx@piece@content}%
}
```

`\MFX@piece@compressible@up` Finally we come to the compressible space. Here, as long as there's excess space (`\Mfx@marginheight`) we drop the compressible space. Once the excess is exhausted, we insert them as normal skips.

```
\def\MFX@piece@compressible@up#1{%
⟨debug⟩\MFX@debug{compressible@up: #1, excess=\the\Mfx@marginheight}%
  \advance\Mfx@marginheight-#1\relax
  \ifdim\Mfx@marginheight<\z@
    \MFX@piece@skip@up{-\Mfx@marginheight}\relax
    \Mfx@marginheight\z@
  \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
⟨debug⟩\MFX@debug{dumpmargins: marginlist=\MFX@mac\mfx@marginlist}%
    \let\temp@\mfx@marginlist
    \vbox{}\clearpage
    \ifx\temp@\mfx@marginlist
      \PackageError{marginfix}{lost some margin notes%
      \ifx\mfx@marginstart\relax\ (missing \noexpand\unblockmargin)\fi
⟨debug⟩: \MFX@mac\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@build@skip` to `\mfx@marginlist`.

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

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

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

`\blockmargin` `\MFX@blockmargin` We need two macros to process the optional bracket argument. Essentially, `\blockmargin` just checks that `\mfx@marginstart` is defined and then appends a new item to the `\mfx@marginpieces` list to indicate a piece that starts at `\mfx@marginstart` and ends at the current position, potentially modified by the argument. It then resets `\mfx@marginstart` to `\relax` and optionally adds a `\softclearmargin` if there was no star, to keep margin material separated on either side of the phantom.

```
\def\blockmargin{%
  \@ifnextchar[%]
    \MFX@blockmargin
    {\MFX@blockmargin[0\p@]}%
}
\def\MFX@blockmargin[#1]{%
  \MFX@inject{%
    \ifx\mfx@marginstart\relax
      \PackageError{marginfix}{two \\blockmargin with no \\unblockmargin}\@eha
```

```
    \else
      \MFX@cons\mfx@marginpieces{\noexpand
        \@elt{\mfx@marginstart}{\expandafter\dimexpr\mfx@ypos+#1\relax}}%
      \global\let\mfx@marginstart\relax
      \global\advance\Mfx@piece@count\@ne
    \fi
  }%
}
```

`\unblockmargin` `\MFX@unblockmargin`

This is the other half of `\blockmargin`. It ensures that the margin is currently blocked, and if so, sets `\mfx@marginstart` back to a real dimension (the current page position, plus the optional modifier).

```
\def\unblockmargin{%
  \@ifnextchar[%]
    \MFX@unblockmargin
    {\MFX@unblockmargin[0\p@]}%
}
\def\MFX@unblockmargin[#1]{%
  \MFX@inject{%
    \ifx\mfx@marginstart\relax
      \xdef\mfx@marginstart{\dimexpr\mfx@ypos+#1\relax}%
    \else
      \PackageError{marginfix}{\\unblockmargin with no \\blockmargin}\@eha
    \fi
  }%
}
```

`\marginphantom` `\MFX@marginphantom`

The `\marginphantom` command is basically just a concatenation of `\blockmargin` and `\unblockmargin`. We reimplement it from the ground up mainly so that the error messages make more sense.

```
\def\marginphantom{%
  \@ifnextchar[%]
    \MFX@marginphantom
    {\MFX@marginphantom[0\p@]}%
}
\def\MFX@marginphantom[#1]#2{%
  \ifdim#2<\z@\MFX@marginphantom[#1+#2]{-#2}\else
  \MFX@inject{%
    \ifx\mfx@marginstart\relax
      \PackageError{marginfix}{\\marginphantom while margin blocked}\@eha
    \else
      \MFX@cons\mfx@marginpieces{\noexpand
        \@elt{\mfx@marginstart}{\expandafter\dimexpr\mfx@ypos+#1\relax}}%
      \xdef\mfx@marginstart{\dimexpr\mfx@ypos+#1+#2\relax}%
      \global\advance\Mfx@piece@count\@ne
    \fi
  }%
  \fi
}
```

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.

```
534 \makeatother
535 ⟨/package⟩
```