[10] The temporary buffer is appended to it.

[11] The `\vsplit` command works by splitting a vbox at a permissible point. If the insertion material is made up of line boxes, it will be split *between lines*, not in the middle of a line. Penalties also control the split. Sometimes a box will be split at a point away from where we wish, because of a penalty that encouraged breaking the box at that point. However, the material split will be shrunk or stretched to bring it to the desired size.

[12] Although it cannot do the entire job.

[13] If the amount of marginal notes exceeds `\vsize`, some of it will be printed off the page, but will not be held over to the next page.

[14] Because of the narrow box width, there will be overfull boxes, but the thick vertical bars accompanying them can be eliminated by `\overfullrule=0pt`.

[15] Things like `\hsize=`*xxx*, `\raggedright`, and `\obeylines`.

[16] It is not returned to the MVL when the OTR says `\unvbox\midins`.

[17] However, outside the OTR it contains, not the number, but the sum of penalties, of all the heldover insertions [111].

◇ David Salomon
California State University,
Northridge
Computer Science Department
Northridge, CA 91330
dxs@mx.csun.edu

Macros

A New Editor

Victor Eijkhout

Starting this issue, I've joined the editorial committee as associate editor for macro affairs (see the reverse of the title page for the other members).

The fact that incoming articles about TEXnical affairs will undergo my scrutiny does not mean that there is suddenly a large chance that submitted articles will be returned, rubber-stamped 'rejected'. My job will be to assist authors in creating articles that are of maximum value to the *TUGboat* readership. Often this means that my main concern is 'how well does this article explain whatever it is telling', rather than 'is this all completely original'. Remember that TEX is not something you read about, it is something you actually *do*. The subject matter of the article is therefore a secondary concern: *TUGboat* is read by beginners and grand masters alike, so articles need not be very high-brow. In fact, we need more articles that help the beginners take the first steps to grand masterhood.

Let these few lines with which I have introduced myself then also be an invitation to prospective authors: if you have done something new, or if you have something interesting to say about something old, write it down, and send it to *TUGboat*. Should you have trouble with the finishing touch, send in what you have and we will discuss it.

◇ Victor Eijkhout
Center for Supercomputing
Research and Development
University of Illinois
305 Talbot Laboratory
104 South Wright Street
Urbana, Illinois 61801-2932, USA
eijkhout@csrd.uiuc.edu

Line Breaking in `\unhboxed` Text

Michael Downes

In the course of my work (macro writing and troubleshooting for TEX-based production at the American Mathematical Society) I recently had to investigate a line-breaking problem in the bibliography macros of the documentstyle `amsppt`, used with AMS-TEX. This is a report on the results of my investigations. Applications where this information might be useful include (1) implementation in TEX of SGML-style macros with omitted end tags as an option, and (2) using the width of a piece of text to choose between two formatting alternatives.

The `amsppt` bibliography macros

Although they're less sophisticated than BIBTEX, the `amsppt` bibliography macros are simple to use and provide a certain degree of style independence (which makes the `.tex` file more portable). They are designed to allow the individual parts of a

reference to be specified in any order, with punctuation between the parts and other formatting supplied automatically. In addition, an individual field within the reference does not have an ending delimiter: \paper (used for article titles) does not have a matching \endpaper, and so on. In SGML terminology, these would be called structures with 'omitted end tags'. A typical reference looks like this:

```
\ref\key C1
\by B. Coomes
\book Polynomial flows, symmetry
  groups, and conditions sufficient
  for injectivity of maps
\bookinfo Ph.D. thesis
\publ Univ. Nebraska--Lincoln
\yr 1988
\endref
```

In TEX, the combination of omitted end tags and randomly ordered elements (possibly with some elements absent) is not easy to provide. If it were required that all the tags had to be present, and in the right order, one way of obtaining at least the appearance of omitted end tags would be to define each beginning tag as a macro with an argument delimited by the next tag:

```
\def\key#1\by{⟨process the argument⟩\by}
\def\by#1\book{⟨process the argument⟩
  \book}
\def\book#1\bookinfo{⟨process the
  argument⟩\bookinfo}
```

and so on. Another approach would be to use ^^M (carriage return) as the ending delimiter; however, this would require the user to have each element on a separate line, and to add percent signs at the end of each line but the last if an element were more than one line long — not too impractical perhaps in the context of a bibliography situation, but trouble-prone in general.

The straightforward approach of having \key, \by, etc., be macros with an argument enclosed in braces would work fine, but doesn't seem to be a case of true omitted end tags since the closing } is necessary. And it's a little more work for the user to type the braces.

The amsppt bibliography macros take a different approach, using \hbox\bgroup ... \egroup. The definition of \by, for example, ends with

```
\setbox\bybox=\hbox\bgroup
```

and the \egroup to end the box is supplied by \book, or whatever tag follows next. This stores the author name in the box \bybox. Each part of the reference is similarly stored away in a box instead of being put on the page immediately. The \endref macro then unboxes all the boxes, using \unhbox, and sets them in a paragraph in the proper order.

This method avoids reading the text as a macro argument, and makes omitted end tags possible, because during the \setbox operation TEX actually typesets the text, expanding macros along the way. This is essential if TEX is to find the \egroup to close each box.

The problem

Interestingly, it seems that the amsppt documentstyle was used for more than five years before the line-breaking problem, which had been present from the beginning, was identified (by Barbara Beeton's eagle eye). Most likely, the problem did manifest itself occasionally during that time but was dismissed without investigation because it could be resolved easily by adding a \linebreak. What Barbara noticed was that, in the example given above, the compound "Nebraska–Lincoln" wasn't breaking properly at the end of a line. The best line break was definitely after the en-dash, but instead "Lincoln" was hanging over the right margin. After she pointed the bad break out to me and we did some experiments, it became clear that although hyphenation between letters was working as normal, after explicit hyphens the possibility of a line break was disappearing.

Horizontal lists

If you have the complete Chapter 14 of *The TEXbook* stored in the "non-volatile memory" of your brain then you probably already know the cause of the phenomenon we were seeing. For those of you who don't, I'll review some terminology and ideas.

All characters typeset by TEX are put into what is called a "horizontal list". Characters and other elements of a math formula or subformula are first processed as a math list, but they end up being transformed into ordinary horizontal list material: characters, boxes, glue, penalties.

To be more specific, the components of a horizontal list are:

(1) characters;
(2) glue (usually interword spaces)
(3) kerns (usually adjustments between letters);
(4) discretionary breakpoints (usually discretionary hyphens);
(5) penalties (encouraging or discouraging line breaks);

(6) boxes (\vboxes or \hboxes containing subsidiary vertical or horizontal lists);

and (7) a few other miscellaneous kinds of things not important in the current discussion.

For line-breaking purposes TEX does not discriminate between single characters, boxes, or rules — each of these treated as a box with a particular width — except that automatic hyphenation occurs only between letters (more precisely, characters with \lccode $\neq 0$; nonletters normally have an \lccode of 0).

Horizontal lists are constructed either in "horizontal mode" or in "restricted horizontal mode". The former is the mode used in making ordinary paragraphs; the latter is the mode used inside an \hbox. Actually the material of a paragraph also ends up in \hboxes, because a finished paragraph is just a stack of \hboxes separated by \baselineskip glue; but the horizontal list for a paragraph is different in a few significant respects from a horizontal list constructed in restricted horizontal mode.

Recall that TEX optimizes line breaks over an entire paragraph; the horizontal list of a paragraph is not broken up into separate lines until the \par or other paragraph-ending command has been reached. At that point TEX goes through the entire horizontal list of the paragraph and chooses line breaks based on the current values of \hsize, \parfillskip, \rightskip, \leftskip, \parshape, \tolerance, \hyphenpenalty, and other parameters. During the initial construction of the horizontal list, TEX adds certain things to help in the line-breaking process.

These items added to a horizontal list by TEX are not explicitly present in the input file, but are inferred by TEX based on the context. Many of the seemingly magical effects of TEX are accomplished this way: paragraph indentation is obtained by inserting an \hbox of width \parindent in the horizontal list; and one step in the process of automatic hyphenation is the addition of \discretionary{-}{}{} in the horizontal list at all the hyphenation points determined by TEX's hyphenation patterns. These items aren't ephemeral, they're really there in the finished list, and can be seen using \showlists.

Because these items are really present in the finished list, they use up box memory (part of TEX's main memory). Therefore TEX avoids adding items unnecessarily. Primarily this means that in *restricted* horizontal mode — in the making of an \hbox — where line breaking is not a possibility, breakpoint items such as \discretionarys or \penaltys that would be added in *un*restricted horizontal mode are omitted.

Since each piece of an amsppt reference is typeset using \hbox, breakpoints are not added by TEX to the enclosed horizontal list. This is not a problem *unless* you \unhbox the box and reuse the contained horizontal list to make a paragraph. But that's exactly what the amsppt bibliography macros do.

Examples

Here are some examples of output from the \showlists command, to make it easier to picture the structure of horizontal lists.

Characters and glue. The word "in", along with surrounding word spaces:

```
\glue 3.33333 plus 1.66666 minus 1.11111
\tenrm i
\tenrm n
\glue 3.33333 plus 1.66666 minus 1.11111
```

Each line corresponds to one item in the horizontal list. In TEX's eyes \tenrm — the name of the current font — is not a separate piece of the horizontal list, but an attribute of the character "i" or "n". The font attribute is displayed with each character for informational purposes.

For the numbers given here the units are points; thus an interword space in this particular font has a natural width of about 3.33 pt, with stretchability of 1.67 pt and shrinkability of 1.11 pt. The em-width of the font is 10 points, so as you can see, the values correspond to 1/3 em, 1/6 em, and 1/9 em.

Kerns and ligatures. The kerns added by TEX in a horizontal list are related to ligatures in that both of them are dependent on the current font:

> If two or more [ordinary characters] occur in succession, TEX processes them all as a unit, converting pairs of characters into ligatures and/or inserting kerns as directed by the font information. (*TEXbook*, p. 286)

Take the word "mode", for example: it has a kern added between the o and the d, in the font \tenrm (cmr10).

```
\tenrm m
\tenrm o
\kern0.27779
\tenrm d
\tenrm e
```

And as an example of a ligature, consider the "ff" ligature in the word "off":

```
\tenrm o
\tenrm ^^K (ligature ff)
```

The ligature character resides in font position 13, which is the ASCII location of control-K. When TEX reads two consecutive "f"s, it replaces them with a single control-K character in the horizontal list, following the instructions in the ligature table for this particular font. Similarly, the ligature character for an en-dash in the same font resides in the ASCII position 123, so in the output of a `\showlists` command it's represented by a left brace:

```
\tenrm { (ligature --)
```

Discretionaries. The discretionary items added by TEX in a horizontal list are of two kinds. A plain `\discretionary` is added after every hyphen character or ligature formed from hyphen characters. So a more complete picture of an en-dash is as shown here (using the text "1–9"):

```
\tenrm 1
\tenrm { (ligature --)
\discretionary
\tenrm 9
```

As already mentioned, to accomplish automatic hyphenation, a discretionary hyphen (equivalent to `\discretionary{-}{}{}`) is added at every hyphenation point within words, according to TEX's internalized hyphenation patterns. But this second kind of `\discretionary` is not added at the same time as the first kind. We'll see the significance of this shortly.

Penalties and glue. Penalties and glue added behind the scenes in a horizontal list are mainly added in math formulas. Internally, the automatic spacing in math formulas is done by adding `\glue` items in the horizontal list in the amount of `\thinmuskip`, `\medmuskip`, or `\thickmuskip`. Penalties in the amount of `\relpenalty` and `\binoppenalty` are added after binary relations and binary operators to allow line breaks. They serve essentially the same purpose as `\discretionary`s, but unlike `\discretionary`s, which neither encourage nor discourage a break, `\relpenalty` and `\binoppenalty` are usually set to some positive value that discourages line breaking. The plain TEX values are 500 and 700, respectively, so that line breaks after operators are discouraged slightly more than after relations. The horizontal list representation of the formula $a = b + c$ looks like this:

```
\mathon
\teni a
\glue(\thickmuskip) 2.77771 plus 2.77771
\tenrm =
\penalty 500
\glue(\thickmuskip) 2.77771 plus 2.77771
\teni b
\glue(\medmuskip) 2.22217 plus 1.11108
                          minus 2.22217
\tenrm +
\penalty 700
\glue(\medmuskip) 2.22217 plus 1.11108
                          minus 2.22217
\teni c
\mathoff
```

If `\mathsurround` were, say, 3pt instead of 0pt, the `\mathon` and `\mathoff` items would be followed by the note

```
(surrounded 3.0)
```

Mathsurround spacing behaves more or less like a kern of the given amount; if a line break occurs at the end of a math formula, the spacing is discarded to keep the line from ending short of the margin. If you look at the horizontal list using `\showlists`, you'll see that the `\mathoff` item remains, but the `(surrounded 3.0)` note disappears.

Automatic hyphenation

To summarize what's been covered so far: when TEX is working in restricted horizontal mode, it omits all the items that are needed only for line-breaking purposes — discretionaries after explicit hyphens, discretionary hyphens, and the penalties after math relations and binary operators. If the resulting horizontal list is then `\unhboxed` and used to make a paragraph, certain line breaks will simply be impossible because the breakpoints aren't present.

But one question remains: Why was ordinary intraword hyphenation still working as normal in the `amsppt` bibliography macros? The answer is that the discretionary hyphens added by TEX to enable automatic hyphenation are not added at the same time as the other breakpoints. All the other kinds of breakpoints are inserted during the initial construction of the horizontal list, but, striving for more efficiency, TEX tries first to make a paragraph without resorting to automatic hyphenation; if and only if this first attempt fails — if line breaks cannot be found such that the badness of each line is less than `\pretolerance` — TEX goes back through the horizontal list of the paragraph and adds the discretionary hyphens indicated by its hyphenation patterns, and goes through another line-breaking pass. On this second pass it also uses `\tolerance` instead of `\pretolerance`.

Thus the `amsppt` bibliography macros first construct the pieces of a paragraph in restricted horizontal mode, so that no breakpoints are added; then the pieces are combined into one long horizontal list and sent to TEX for paragraphing; if the first attempt at paragraphing fails, TEX follows its usual

process of adding discretionary hyphens, and tries again to make a paragraph, whereupon hyphenation works as normal.

Using \vboxes instead of \hboxes

The TEXbook, Appendix D, pp. 398–400, has an example that uses the technique of \unhboxing to construct a paragraph out of many short footnotes. Finding no mention there of hyphenation peculiarities, I wrote to Knuth to suggest that a footnote about hyphenation might be useful to add in some future printing, and to ask if there was any way to provide normal line breaking after hyphens in unhboxed text; I couldn't think of any solution short of catcoding the hyphen to be active and having it do some laborious checking to handle the possibility of en-dashes and em-dashes (the need to consider \relpenalty and \binoppenalty hadn't even occurred to me). In response Knuth outlined an interesting alternative: If instead of an \hbox you use a \vbox with \hsize set to \maxdimen, the product will be a one-line paragraph, with all the necessary breakpoints present (because unrestricted, rather than restricted, horizontal mode will be used to construct the horizontal list). There is an extra level of boxing present, but an extra unpacking step will take care of that.

Here is a sketch of the TEXnical details, using a simplified bibliography scheme with three tags: reference label \key, author name \by, and article title \paper.

To start with, some box names need to be declared:

```
\newbox\keybox \newbox\bybox
\newbox\paperbox
```

At the very beginning of a reference, we need to provide a \bgroup to match the first upcoming \egroup. We do this by setting a \vbox that will simply be discarded.

```
\def\ref{\par\setbox0=\vbox\bgroup}
```

And here's the definition of \key:

```
\def\key{\par\egroup
 \setbox\keybox=\vbox\bgroup
 \hsize=\maxdimen \noindent\bf}
```

Without the \noindent we'd get a box of width \parindent because we're beginning a paragraph; this would interfere later when the pieces of the reference are combined.

Actually, since the macros \by and \paper are nearly identical, it's better to write a generalized macro that can be shared by all three:

```
\def\makerefbox#1#2{\par\egroup
 \setbox#1=\vbox\bgroup
 \hsize=\maxdimen \noindent#2}
```

Then the definitions are

```
\def\key{\makerefbox\keybox\bf}
\def\by{\makerefbox\bybox\rm}
\def\paper{\makerefbox\paperbox\it}
```

\endref performs the usual sequence \par\egroup to close the final data box, whatever it may be, and then unpacks \keybox, \bybox, and \paperbox, inserting punctuation and space as desired. Since each unpacking operation is the same, it's best done as a macro, say \unvxh ("unvbox, extract the last line, and unhbox it").

```
\def\endref{\par\egroup
% preliminary formatting
  \noindent\hangindent\parindent
% reference label
  {\bf[\unvxh\keybox]}\enspace
% author name(s)
  \unvxh\bybox,\space
% article title
  \unvxh\paperbox.\par
}
```

The \bf here is necessary if we want bold [ ] around the reference key. The contents of \keybox are already typeset, so we could not change them to bold at this point if they were not bold already.

The last line of a paragraph ends with three special items:

```
\penalty 10000
\glue(\parfillskip) 0.0 plus 1.0fil
\glue(\rightskip) 0.0
```

If we made sure \parfillskip and \rightskip are zero, by setting them to zero at the same time as we set \hsize to \maxdimen, these items could perhaps be left in place. On the other hand, if we remove them, the reassembled reference will more closely resemble a paragraph typeset naturally, and furthermore, it will use slightly less of TEX's main memory. So we remove them using \unskip and \unpenalty in the macro \unvxh.

```
\def\unvxh#1{%
  \setbox0=\vbox{\unvbox#1%
    \global\setbox1=\lastbox}%
  \unhbox1
% remove \rightskip, \parfillskip,
% and penalty
  \unskip\unskip\unpenalty
}
```

Now to try these macros out:

```
\ref \paper Title of the important
   work he wrote
\by Arthur Aja Desc
\key De \endref

\ref\key K\by Kustim Kunsla
\paper And to test line breaking
  after explicit hyphens: pneu-mono-ul-%
  tra-mi-cro-scop-ic-sil-i-co-%
  vol-ca-no-co-ni-o-sis \endref
```

[De] Arthur Aja Desc, *Title of the important work he wrote.*

[K] Kustim Kunsla, *And to test line breaking after explicit hyphens: pneu-mono-ul-tra-mi-cro-scop-ic-sil-i-co-vol-ca-no-co-ni-o-sis.*

Without the use of Knuth's idea there would be no legal breakpoints after any of the explicit hyphens in pneu-mono-ul-tra-mi-cro-scop-ic-sil-i-co-vol-ca-no-co-ni-o-sis and we'd have an overfull line.

Complications

A significant stumbling block was pointed out to me by Ron Whitney at TUG when I submitted this article (thank you, Ron): if we're typesetting a piece of a reference using `\vbox` instead of `\hbox`, explicit line breaks typed by the user will take effect as soon as a `\par` is read — that is, when the information is stored, rather than when it is combined with the rest of the reference. In addition to the `underfull \hbox` message that will result (because `\hsize = \maxdimen`), this means that the `\vbox` will contain more than one line of text, and unpacking it will not be so simple after all.

Let's suppose that, as in $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX, the user has a single command for forcing a line break, called `\linebreak`. We don't need to worry about "suggested" line breaks — anything with a `\penalty` greater than -10000 — because these will remain inactive during the initial typesetting, thanks to the large `\hsize`. The problem is to take the text that is split by a `\linebreak` and save it in such a form that it can later be joined seamlessly with the rest of the reference, but with the line break preserved. And we want to suppress the `underfull \hbox` message while we're at it.

There are various alternatives, and some readers may be able to devise a better solution than the one I chose. But first let me mention briefly a couple of the more tempting and less practical alternatives that I considered:

(1) Tell the users that they can only use pure bibliographic information inside the reference macros, unsullied by uncouth raw typesetting commands like `\linebreak`. This would mean only that users would grumble about their output and line breaking problems would be deferred to the attention of publishers' production troubleshooters, e.g., me.

(2) Redefine `\penalty` to check the penalty amount and make sure it's -9999 or greater; that is, convert forced line breaks to "emphatically suggested" line breaks. This would work reasonably well in a bibliography context (especially with a high setting of `\tolerance`), since penalties are only used for line breaking and page breaking, and within the scope of the `\penalty` redefinition there wouldn't be any embedded vboxes wherein line-breaking had to be restored to normal. However, this alternative seems dangerous; I was able to imagine at least one scenario (too complicated to be worth describing here) where changing forced breaks to nonforced breaks would cause a problem.

One way of handling line breaks. Assume that, as in $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX, the user has a single command, `\linebreak`, to force a line break, and that its normal definition is essentially `\penalty-10000` (ignoring some frills like error messages if in vertical mode). We don't have to worry about penalties greater than -10000, as mentioned earlier. Inside the `\vbox` that is being typeset by `\makerefbox`, we can change the definition of `\linebreak`:

```
\def\linebreak{\par
  \setbox0=\lastbox
  \setbox\holdoverbox=
    \hbox{\unhbox\holdoverbox
```

(in case more than one `\linebreak` occurs within a single piece of the reference)

```
        \unhbox0
        \unskip\unskip\unpenalty
        \penalty-10000}%
  \noindent}

% Can't forget this
\newbox\holdoverbox
```

Thus `\linebreak` takes the text so far and saves it in `\holdoverbox`, along with a break penalty, inactive here because we're in an `\hbox` instead of a `\vbox`. This saved part will then be combined with the remainder of the current reference field, by way of some extra processing in `\makerefbox` and `\endref`.

```
\def\makerefbox#1#2{\par
  \checkholdoverbox \egroup
  \setbox#1=\vbox\bgroup
  \hsize=\maxdimen \noindent#2}
```

```
\def\endref{\par
  \checkholdoverbox\egroup
  \noindent\hangindent\parindent
  {\bf[\unvxh\keybox]}\enspace
  \unvxh\bybox,\space
  \unvxh\paperbox.\par}

\def\checkholdoverbox{%
 \ifvoid\holdoverbox
  \else \setbox0=\lastbox
 \hbox{\unhbox\holdoverbox
   \unhbox0}\fi}
```

After `\checkholdoverbox`, we have reduced the contents of the current `\vbox` to a single `\hbox`, just as if no `\linebreak` had been present.

There are extra complications if the user inserts a `\linebreak` at the end of a field, because that means the break will be taken between the text and ensuing automatic punctuation if we don't do something about it. Some nice checking and rearranging to handle this case was present in Spivak's original version of the `amsppt` reference macros and will not be discussed here.

Text measurement applications

To understand more clearly the second application mentioned at this article's beginning, consider the usual method for measuring a piece of text and using the width as a selector:

```
\def\caption#1{%
  \setbox0=\hbox{#1}%
  \ifdim\wd0<\hsize
   \centerline{\box0}% centered line
  \else
   \noindent#1\par % paragraph
  \fi}
```

Using `\unhbox0` in the ⟨*paragraph*⟩ branch would be slightly more efficient — it avoids typesetting the text twice. But that would bring in the line-breaking problems described above. We can have our cake and speed up, too, if we use Knuth's idea and start by setting the caption in a `\vbox`, rather than an `\hbox`.

Handling vertical mode material in a caption. After setting material in a `\vbox`, simply extracting `\lastbox` as in the `\unvxh` example may not be enough. What if someone wants two paragraphs in a caption, or maybe even a displayed equation? A `\par` or display in the `\vbox` will mean that after extracting `\lastbox` some material will be left behind. I had a chance to experiment in a recent assignment, which was to create a LaTeX documentstyle for electronic submissions to American Mathematical Society journals. One of the macros I had to modify was `\@makecaption`, whose original definition from `article.sty` was roughly the same as the `\caption` example above.

This is what `\@makecaption` had to be modified to do: If the total width of the caption material is greater than `\columnwidth` (29pc), break the caption into lines using a line width of 23pc, and center the resulting block between the margins. Otherwise set the caption as a single line, centered between the margins.

To do this I decided to set the caption as a `\vbox` with line width 23pc, but allow the last line (which may be the only line) to be up to 29pc long by adding a kern of −6pc. The last line is put into box register 1 using `\lastbox`. After extracting the last line, if there is anything left in the `\vbox`, that means the caption was (most likely) more than one line long and some extra processing is needed.

The first argument of `\@makecaption` is the name of the figure or table, e.g., "Figure 1", generated automatically by LaTeX. The second argument is the caption text typed by the user.

```
\long\def\@makecaption#1#2{%
```

We begin by setting the text in a `\vbox`.

```
  \setbox\@tempboxa\vbox{%
% hsize := 29 - 6 = 23 pc
    \advance\hsize-6pc\noindent
```

Ordinarily the `\unskip` here would be done automatically by `\par`, but here the `\kern` gets in the way so we must do the `\unskip` explicitly.

```
      {\sc#1}\enspace#2\unskip
        \kern-6pc\par
      \global\setbox\@ne\lastbox}%
```

Now box 1 holds either the entire caption, or the last line of a multiline caption. In either case we want to remove `\parfillskip`, `\rightskip`, and the `\kern` of −6pc. Before we get to the `\kern` we also have to remove the penalty of 10000 that is inserted by TeX at the end of every paragraph.

```
  \setbox\@ne\hbox{\unhbox\@ne
    \unskip\unskip\unpenalty\unkern}%
```

If `\@tempboxa` is not empty at this point it means the caption was more than one line long. In that case we reset the caption using the contents of `\@tempboxa` and `\unhboxing` box 1 (because the contents of box 1 may need to be made into two lines instead of one, if its length is greater than 23 picas). Otherwise the caption material is made into a single centered line. Note: A box register containing an empty box is not the same as a void

box register; a box register that contains \vbox{} will not return true if tested with the \ifvoid test. So to decide whether \@tempboxa is empty we cannot use \ifvoid. Instead we employ the simple strategy of measuring the width of the box. This will not be 100% failsafe but the failure cases that I've been able to imagine are all rather exotic.

```
\ifdim\wd\@tempboxa=\z@
  \setbox\@ne\hbox to\columnwidth{%
     \hss\kern-6pc\box\@ne\hss}%
\else % more than one line
   \setbox\@ne\vbox{\unvbox\@tempboxa
      \noindent\unhbox\@ne
      \advance\hsize-6pc\par}%
\fi
```

The \kern-6pc in the first branch is to offset the \moveright that is about to be done next. (If tortured, I would be forced to admit that it took me several attempts before I figured out the right amount for this kern and the proper place to put it.) Finally, we put the caption on the page, with a \vskip to separate it from the preceding or following material.

```
  \ifnum\@tempcnta<64 %if it's a figure
    \vskip 1pc%
    \moveright 3pc\box\@ne
  \else % if the float IS NOT a figure
    \moveright 3pc\box\@ne
    \vskip 1pc%
  \fi
}
```

By testing \@tempcnta we can tell whether the caption is being used in a figure environment or not; if so, we assume that the caption is placed below the artwork and hence put the \vskip above the caption; otherwise we assume the caption is at the top of the floating insertion and we put the \vskip below it.

\@makecaption presents a few extra complications that have been omitted for the sake of simplicity; as given here, the caption will not be quite centered if the figure caption has no text, and so on.

◇ Michael Downes
American Mathematical Society
201 Charles Street
Providence, RI 02904
mjd@Math.AMS.com

Looking Ahead for a ⟨box⟩

Sonja Maus

TEX's primitive \afterassignment can be used for macros which first assign a value to a parameter, and then perform some actions using that value. For instance the plain TEX macros \magnification and \hglue (see *The TEXbook*, p. 364 and 352), assign a ⟨number⟩ or ⟨glue⟩ value to a variable and then use this value. They provide a user-friendly "syntax mimicry": \magnification looks like an integer parameter in an assignment, and \hglue looks like the primitive command \hskip. There is another advantage to this method over the use of arguments with #1: At the moment when TEX looks at the tokens of the value, it already knows what kind of value it is looking for. This would be very useful when the value to be read is a ⟨box⟩, because an explicit \hbox or \vbox may contain \catcode changes and all tokens should not be read ahead.

There are seven ways to write a ⟨box⟩ (*The TEXbook*, p. 278). The \afterassignment command behaves differently with the first four and the last three of these ⟨box⟩es:

```
\afterassignment\t \setbox0=\box1
```

results in \setbox0=\box1 \t, whereas

```
\afterassignment\t \setbox0=\hbox{h}
```

results in \setbox0=\hbox{\t h}.

The macro \afterbox gives a substitute which is equally valid for all ⟨box⟩es. Its syntax is

```
\afterbox<argument><box>
```

where ⟨argument⟩ is an argument for an undelimited macro parameter (see *The TEXbook*, p. 204), i.e. a single token or several tokens in explicit braces. \afterbox puts the ⟨argument⟩ aside (without the braces, if any), assigns the ⟨box⟩ to the register \box\afbox, and then reads the ⟨argument⟩ again.

The definition must be read when @ is a letter:

```
\newbox\afbox
\def\afterbox#1{\def\afb@xarg{#1}%
  \afterassignment\afb@x
  \chardef\next`.}
\def\afb@x{\futurelet\next\afb@xtest}
\def\afb@xtest
 {\ifcase\ifx\next\hbox\tw@\fi
         \ifx\next\vbox\tw@\fi
         \ifx\next\vtop\tw@\fi
         \ifx\next\box\@ne\fi
         \ifx\next\copy\@ne\fi
         \ifx\next\vsplit\@ne\fi
         \ifx\next\lastbox\@ne\fi
         0\errmessage{No <box>}%
  \or\afterassignment\afb@xarg
```