```
            \loop %
               \expandafter %
               \ParseLine\Remainder\endParse %
               \expandafter\getMeaning %
                  \meaning\FirstLine\endget %
               \immediate\write#1{\Meaning}%
               \ifx\Remainder\empty\else%
            \repeat}%
         \endgroup % end \returnactive %(*)
      \fi %
   \fi %
```

Notice that `\returnactive` must be used not only when the lines are actually broken, but also for the definitions of `\Parseline` and `\Write` since they involve the active `^^M`. The definition to be used here for `\loop` is that in Ron Whitney's paper (note the `\else\repeat`). The switch `\ifexpmacros` is controlled by the following code in `prepare.tex` for yet another option, `\ExpandMacros`:

```
\newif\ifexpmacros \expmacrosfalse
\def\ExpandMacros{\expmacrostrue}
```

The swarm of %-signs is here to prevent the overactive `^^M`s from creating mischief. In a more advanced course, you may learn a simpler way to tame them (cf. the "sanitizing" paper in this issue).

Special challenge. Consider the endnotes in Edith Hamilton's *The Greek way* (W. W. Norton, 1942): No marks appear in the text itself, but each note indicates the page number and the line number on that page to which the note refers. Had TEX existed in 1942, how might this have been achieved?

In the next tutorial, we consider some questions related to the construction of indexes. Among other ideas, there will be more about parsing by context and some examples of other ways to use loops.

Note. A disk (`5.25in` DSDD) containing source text for the figures in this series of four tutorials, and the code files used to produce them, is available for MS DOS users who are members of the TEX Users Group. Send $6 (which includes a royalty for the TEX Users Group) to the address below. Outside North America, add $2 for air postage.

As usual, Ron Whitney has been generous with ideas and inspiration. He has taught me a lot about TEX — everything I know about it, except for all the things I learned from Barbara Beeton, over a period of several years, and things that I understood when I read about them for the first time in *The TEXbook*.

◇ Lincoln Durst
46 Walnut Road
Barrington, RI 02806

Output Routines: Examples and Techniques. Part III: Insertions

David Salomon

Note: Before reading this article, the reader should glance at parts I or II for disclaimers and remarks on notation.

Insertions are considered one of the most complex topics in TEX. Many users master topics such as tokens, file I/O, macros, and even OTRs before they dare tackle insertions. The reason is that insertions **are** complex, and *The TEXbook*, while covering all the relevant material, is somewhat cryptic regarding insertions, and lacks simple examples. The main discussion of insertions takes place on [115–125], where TEX's registers are also discussed. Examples of insertions are shown, mostly without explanations, on [363–364, 423–424]. There is, therefore, a need for an article like the present one. It tries to explain insertions in detail, and shows specific, simple examples. Concepts are developed gradually, and the ultimate truth revealed in steps.

Introduction

Definition: An *insertion* is a piece of a document that is generated at a certain point but should appear in the document at another point.

Common examples of insertions are footnotes, endnotes (Note 1), and floating insertions. These are important features, which explains why a general insertion mechanism has been incorporated into TEX. The following short quote (from [124]) says it all: "*This algorithm is admittedly complicated, but no simpler mechanism seems to do nearly as much.*" Using insertions, it is possible to accumulate material (text/pictures) in a box and typeset it anywhere in the document. The material can be inserted on the current page, it may be *held over* by TEX and inserted on the following page, it may be *split* between the current page and the next one, or it may wait for the end of the document. The `plain` format also provides very convenient macros, based on the general insertion mechanism, to handle footnotes and floating insertions.

A good example of insertions is the placement of index items in the right margin [423–424], an operation that is part of the *manmac* format [App. E]. See (Note 2) for an outline of the idea. A simple version is developed elsewhere in this article.

It is important to point out that, even though the insertion mechanism of TEX is general and complex, it cannot deal with every conceivable situation. Consider the case of *facing figures* (Note 3).

This is a problem that TEX's insertion mechanism cannot handle. It is easy to implement in other ways, though (Note 4).

A simple example

Before delving into the details of insertions, it is useful to develop a simple example from scratch, without using any of the built-in features for insertions. We will develop a simple mechanism for handling *floating insertions*. Suppose that diagrams should be pasted into our document (after it's been typeset) at certain points. We need to reserve room for each diagram, which is done by placing an empty \vbox at each insertion point.

Exercise: Why not simply say \vskip... or \kern... to reserve vertical space on the page? (Note 5).

We therefore define a macro \Pic by

```
\def\Pic#1 high{\par\vbox to#1{}}
```

and call it by, e.g., \Pic 3.5in high. The problem, of course, is that there may not be 3.5 inches of space left on the current page. In such a case, the insertion should be 'floated' to the top of the next page. We therefore have to generalize our macro such that it measures the space left on the current page before it creates the \vbox. To understand how this is done, the reader should first review the section on \pagetotal and \pagegoal in part I, where macro \pagespace was developed. This macro, whose definition is copied below, does just that.

```
\newdimen\spaceleft
\def\pagespace{%
  \ifdim\pagetotal=0pt
   \spaceleft=\vsize
  \else
   \spaceleft=\pagegoal
   \advance\spaceleft by -\pagetotal
  \fi}
```

We now generalize macro \Pic. It starts by setting \box0 to the desired, empty \vbox. It then compares the height of the picture to the available space on the page. If there is enough room, \box0 is simply typeset, which reserves room on the page for the diagram; otherwise, \box0 is appended to another box, called \fig.

```
\newbox\fig
\def\Pic#1 high{%
  \setbox0=\vbox to #1{}
  \pagespace
  \ifdim#1>\spaceleft
   \setbox\fig=\vbox{
    \unvbox\fig\nointerlineskip\box0}
  \else
   \box0
  \fi}
```

After several calls of \Pic, \box\fig is either void, or contains a bunch of vboxes with nothing in between. When the OTR is next invoked, it first ships out the current page, then checks \box\fig. If that box is nonvoid, the OTR empties it by simply saying \unvbox\fig, which places its contents on top of the MVL, to appear at the top of the next page.

```
\output={\shipout\box255 \advancepageno
  \ifvoid\fig\else \unvbox\fig\fi}
```

This way, enough space is reserved on top of the next page for as many diagrams as necessary. It is important to say \unvbox\fig, rather than \box\fig, since this places on the MVL, not the single \box\fig — which is indivisible — but its contents, as separate boxes. The contents may now be spread over more than one page, if they involve many elements.

This simple example should be studied carefully, since it provides a good starting point for a full understanding of insertions.

Insertions (introductory)

On the first reading of this section, the endnotes should be ignored.

The insertion mechanism used by TEX (see [122–125]) is based on box variables. A box variable is allocated, and the \insert command is then used to accumulate, in that box (Note 6), vertical material to be eventually typeset (on the same page or someplace else in the document). The OTR can typeset the box anywhere on the page, using standard features, as shown below (Note 7).

Example: The command \newinsert\fig allocates the box variable \box\fig. Each command of the form \insert\fig{⟨*vertical material*⟩} accumulates material in the box (Note 8), material which TEX assumes is to be eventually typeset, by the OTR, somewhere in the document. If the material is to be typeset on the current page, TEX is instructed (see discussion of \count\fig below) to decrement g (Note 9) by the vertical size of the material, in order to reserve room on the page.

Just before the OTR is invoked, the insertion box becomes available (Note 10). We quote from [254] "*...just before the output routine begins, insertions are put into their own boxes.*" The OTR can

typeset the material in \box\fig by constructions such as:

1. \shipout\vbox{\box255\unvbox\fig}, to typeset the insertion at the bottom of the page.
2. \shipout\vbox{\unvbox\fig\box255}, to typeset it at the top.
3. \shipout\vbox{\vsplit255 to 4in \box\fig \box255}, to typeset it 4 inches from the top of the page.
4. \shipout\vbox{\rlap{\kern\hsize \vbox to 0pt {\box\fig \vss}}\box255}, to place the insertion at the top right margin.

Insertions (intermediate)

The actual steps taken by TEX are more complicated. In response to the \insert\fig command, the material is accumulated, not in the insertion box but rather in a temporary buffer. Just before the OTR is invoked, as much of the material in the buffer as can fit on the page, is appended to the insertion box. Note that the user may, from time to time, append things to the insertion box explicitly, by means of

\setbox\fig=\vbox{\unvbox\fig ⟨*material*⟩...}

The accumulated material is eventually appended to those things. When the OTR typesets the box on the page, all the box contents go on the page; however, room on the page is reserved only for material handled through the \insert\fig command.

The \newinsert command mentioned above does more than just allocate a box. It allocates a *class of insertions*. The class includes \count, \dimen, and glue (\skip) variables, all of the same number, and all set to zero by default. So, for example, the \newinsert\fig above reserves variables \box\fig, \count\fig, \dimen\fig, and \skip\fig. They are considered class insertion \fig. If \fig happens to be 100, then the \newinsert\fig above allocates variables \box100, \count100, \dimen100, and \skip100.

Since \box255 is reserved for special OTR use, only insertion classes 0...254 can be allocated. Macro \newinsert computes a number (counting down from 254) and allocates a box, a count, a dimen, and a skip register with that number. The reason for allocating from 254 instead of 255 is that \box255 is reserved for special OTR use. The reason for allocating downwards is that registers \count0, \count1... are used for the page number, and that many people tend to use registers \box0, \box1... for temporary storage.

The \dimen\fig variable limits the size of the insertion material per page. In response to \dimen\fig=8in TEX will place at most 8 inches worth of insertion material from the temporary buffer in \box\fig per page. If the buffer contains more than 8in of material, the excess will be heldover for the next page. Placing 8 inches worth of material from the buffer in \box\fig may also mean that an insertion will have to be split by TEX. The splitting is done by \vsplit (Note 11), an operation which is also available for general use. If \dimen\fig is not set by the user, its value is zero, which means no room at all on the page for insertion material. The material simply accumulates in the buffer without being used, or until the value of \dimen\fig is changed.

The \count\fig variable specifies by how much g should be decremented. Setting \count\fig=250 causes g to be decremented by 25% of the height (plus depth) of each block of insertion material placed in \box\fig. Example 4 above should set \count\fig=0, since the insertion is done on the right margin and no room should be reserved for it on the page.

The \skip\fig variable specifies how much vertical skip the user wants to place, by means of the OTR, on the page above or below the insertion. TEX decrements g *once* by the amount of \skip\fig on those pages which have some insertion material of class \fig in order to reserve room on the page for the skip. The skip itself, however, is not done automatically, and the OTR should not forget to add vertical glue totalling \skip\fig to the page.

Tracing insertions (preliminary)

A good way to understand insertions (and many other aspects of TEX) is to trace the values of the various quantities involved. Such tracing is easily done by \message commands, which can display many internal quantities at run time. A test of the type shown below is simple and can reveal a lot about the inner workings of insertions.

```
\hsize=3in \vsize=100pt
\output={\shipout\vbox{
 \unvbox255 \vskip\skip\fig \unvbox\fig}
 \advancepageno}

\newinsert\fig
\count\fig=1000
\dimen\fig=\vsize
\skip\fig=6pt

\message{1:t=\the\pagetotal; g=\the\pagegoal}
Text for the first paragraph
```

```
\message{2:t=\the\pagetotal; g=\the\pagegoal}
\insert\fig{⟨Material⟩}
Text for the second paragraph

\message{3:t=\the\pagetotal; g=\the\pagegoal}
\insert\fig{⟨Material⟩}
...
\bye
```

This simple experiment should be repeated with `\tracingpages=1` to get even more information on how TEX (actually, the page builder) handles insertions (see detailed examples in a later section on tracing).

Example: Endnotes

Endnotes are used in this article as a simple example of insertions. They are implemented in three steps.

1. A new class of insertions is declared and initialized by:

```
\newinsert\notes
\count\notes=0
\dimen\notes=\maxdimen
\skip\notes=0pt
```

Since the notes will be typeset on the last page, no room should be reserved for them on the current page, which is the reason for setting `\count\notes=0`. Setting `\dimen\notes=\maxdimen` guarantees that any amount of endnotes, even more than a page worth, could be placed in `\box\notes`.

2. Macro `\endnote` can be expanded anywhere in the document. It accepts one parameter, the text of the endnote, and executes `\insert\notes{#1}`. It also computes the note number, and typesets the word 'Note' and the note number in parentheses.

```
\newcount\notenumber
\notenumber=0
\long\def\endnote#1{\advance\notenumber by 1
  (Note \the\notenumber)%
  \insert\notes{\noindent[\the\notenumber]
    #1.\medskip}}
```

3. The endnotes should be typeset at the end of the document, but how? Generally, a box, such as `\box0`, is typeset by saying `\box0` or `\unvbox0`. However, we cannot do that with an insertion box, since the contents is only placed in it before the OTR is invoked. The job, therefore, has to be done in the OTR, and one way of doing it is:

```
\output={\shipout\box255
  \ifnum\outputpenalty=-20000
  \unvbox\notes\penalty-20000\fi
  \advancepageno}
```

This method uses the special penalty value of −20000, and is explained later, in the section on `\supereject`.

Each `\insert\notes` command places the material in `\box\notes` as a paragraph or as several paragraphs. Commands that apply to paragraphs in general, may be used for this material. The `\noindent` above is one example. Without the `\noindent`, the insert becomes

```
\insert\notes{[\the\notenumber]
  #1. \medskip}
```

and the material will be placed in `\box\notes` with the first paragraph indented. Another possibility is

```
\insert\notes{\narrower[\the\notenumber]
  #1. \medskip}
```

which will place the material in `\box\notes`, broken into narrow lines.

It is also possible, of course, to say

```
\insert\notes{\vbox{[\the\notenumber]
  #1. \medskip}}
```

and this will place each endnote in `\box\notes` as a `\vbox`. Such endnotes cannot be split across pages, and the last page where they appear, may come out too long or too short.

(See Lincoln Durst's article beginning on p. 577 of this issue of *TUGboat* for a different treatment of endnotes.)

Example: Footnotes

The footnotes example shown here is similar to the one implemented in the `plain` format [363], but is much simpler.

1. An insertion class `\footins` is declared and initialized by:

```
\newinsert\footins
\skip\footins=12pt plus 4pt minus4pt
\count\footins=1000
\dimen\footins=8in
```

The last line limits the amount of footnote material per page to 8 inches. If there are more footnotes than that, the excess is held over to the next page. This is automatically done by TEX's insertion mechanism. Note that preparing 8 inches worth of footnotes may necessitate splitting one footnote.

2. A `\footnote` macro is defined, with two parameters: the footnote reference symbol, and the footnote text. It typesets its first parameter and appends both parameters (without a space in between) to the insertion box.

```
\def\footnote#1#2{#1\insert\footins{
  \noindent#1#2}}
```

The footnote text may be longer than one line, but, when placed in \box\footins, it will be broken into lines of size \hsize, and will not be indented.

If the footnotes should be typeset in a small size, we can say, e.g.,

```
\def\footnote#1#2{#1\insert\footins
  \noindent#1\sevenrm#2}}
```

which typesets the footnote text in seven-point roman. The footnote symbol will be set in the current font (the font that is current at the time of insertion).

Readers experimenting with these macros will notice that the two examples of \footnote above result in bad vertical spacing, both inside and between the footnotes. The reasons are (1) selecting a font does not automatically change the interline spacing. The value of \baselineskip in \box\footins is still 12pt, appropriate for cmr10, but not for cmr7; (2) there is no separation, in \box\footins, between the individual footnotes.

To correct the spacing, (1) the interline glue (\baselineskip) should be set, in \box\footins, to a value appropriate for a seven-point font; (2) the individual footnotes should be separated by placing a strut with the desired height and depth at the beginning and end of each of them (see also [Ex. 21.3]). Much better footnote spacing is obtained by:

```
\def\footnote#1#2{#1\insert\footins{
 \baselineskip=8pt\noindent\sevenrm
 \strut#1#2 \strut}}
```

Further improvement is obtained when TEX is discouraged from splitting a footnote between pages, whenever possible. This is done by (1) placing a penalty between the lines of each footnote; (2) placing a negative penalty between footnotes in \box\footins; (3) adding flexibility to the 4pt separating the footnotes. Some flexibility may also be added to the interline glue, but this results in nonuniform appearance of the pages.

```
\def\footnote#1#2{#1\insert\footins{
 \interlinepenalty=1000
 \baselineskip=8pt plus1pt\noindent
 \sevenrm#1#2\endgraf \penalty-1000
 \vskip4pt plus2pt minus2pt}}
```

The last point to consider is the two parameters \leftskip, \rightskip. They are inserted on the left and right of every line of text [100]. Normally they are zero, but the user may set them to any value at any time. If we don't want them to affect the horizontal size of our footnotes, they should be set to zero locally, when the footnote text is inserted into \footins. This is done by:

```
\def\footnote#1#2{#1\insert\footins{
 \leftskip=0pt\rightskip=0pt
 \interlinepenalty=1000
 \baselineskip=8pt plus1pt\noindent
 \sevenrm#1#2 \penalty-1000
 \vskip4pt plus2pt minus2pt}}
```

3. The OTR should ship out a page consisting of (1) the body of the text, in \box255; (2) a \vskip\skip\footins, with a rule *inside it*; (3) the footnotes for the page, in \unvbox\footins. Here is how it's done:

```
\output={\shipout\vbox to \vsize{\unvbox255
  \ifvoid\footins\else
    \vskip\skip\footins
    \kern-3pt\hrule width2in\kern2.6pt
    \unvbox\footins
  \fi}
  }
```

In practice, the OTR should do other things, such as typesetting and incrementing the page number, but those are ignored here. The reason for unboxing \box255 is so that its flexible glues could blend with the ones in the insertion box (see the last six lines on [125] for a similar comment).

It should be mentioned here that these footnotes may appear on a page different from the one on which they are referenced (see [Ex. 15.13] for other cases where this may happen). This happens when there are many footnotes but we limit the amount of space on the page where footnotes can be typeset by assigning a small value to \dimen\footins. A value such as 0.4in is generally enough for 4 footnote lines and, if there is more footnote text for the page, it would be typeset on the following page. This sometimes requires splitting a footnote into two parts, which is why footnotes should be inserted into \footins as individual lines, not as a \vbox (which is indivisible). Thus we should avoid something like:

```
\def\footnote#1#2{#1\insert\footins{
  \vbox{#1#2 \vskip4pt}}}
```

Example: Right margin insertions

Another useful example of insertions has to do with index items. Preparing an index for a textbook can be no small task, and TEX can help a lot in this (Note 12). Typically, macros should be defined to identify parts of the text as index items, and write them on a file for future sorting and processing. However, it is very useful, while writing and modifying the document, to typeset all the index items of a page on the right margin of the page. When the document is ready, the final run omits the notes on the margin. Such an example is

shown on [415, 423–424] and is described here in a simplified form.

The main steps are:

1. A boolean variable \proofmode is declared and set to true. A new class of insertions, called \margin, is declared and initialized.

```
1. \newif\ifproofmode
2. \proofmodetrue
3. \newinsert\margin
4. \dimen\margin=\maxdimen
5. \count\margin=0
6. \skip\margin=0pt
```

Line 4 allows any amount of marginal notes per page (Note 13). Line 5 guarantees that no space will be reserved on the page for the notes, and line 6 says not to skip vertically before the notes are typeset.

2. The index macro is defined. It has one parameter, the index item. The macro writes it on a file, with the page number, and inserts it in \insert\margin. The latter part is done by:

```
\ifproofmode\insert\margin{
 \hbox{\sevenrm #1}}\fi
```

Each index item is placed in an \hbox, and so becomes one line. If it is too long to fit on the margin, part of it will fall off the page. If it is important to see the entire text of the note, it can be placed in a narrow \vbox, where it will be broken into lines. Assuming a 1 inch wide margin, we can write:

```
\ifproofmode
 \insert\margin{\vbox{\hsize=1in
   \baselineskip=8pt\tolerance=2000
   \sevenrm\noindent#1}\smallskip}
\fi
```

Note the vertical spacing of the notes, which is similar to the case of footnotes. (Note 14)

3. The OTR should typeset \box\margin on the right margin of the page during \shipout. Here are the basic steps:

```
\output={\shipout\vbox to \vsize{
  \ifvoid\margin \else
    \rlap{\kern\hsize\kern4pt
      \vbox to0pt{\box\margin\vss}}
  \fi
  \unvbox255}}
```

The \rlap leaps over to the right margin with the \kern\hsize, then moves another 4pt to the right, to separate the marginal notes from the body of the text. The OTR should, of course, do other things, such as advancing the page number, and appending a header, a footer, and footnotes.

The main differences between the marginal notes and the footnotes discussed earlier are (1) no marginal notes should be held over to the next page (even if they don't all fit on the current page); (2) no room should be reserved on the page for the marginal notes; (3) overfull boxes are okay since the marginal notes will be omitted anyway on the final run.

Example: Floating insertions

We describe a mechanism for floating insertions, similar to the \midinsert of the plain format. \midinsert is explained on [116] and its definition shown on [363]. Our example is simpler and does not do as much as \midinsert, but it works, and it serves to illustrate the principles involved.

An insertion class \midins is declared, and a macro pair \midinsert, \endinsert is defined and used to delimit the material to be inserted. It is used as follows:

```
\midinsert
⟨material to be inserted⟩
\endinsert
```

The material to be inserted may contain commands and specifications that should be kept local to the insertion (Note 15). This is achieved by the \bgroup, \egroup pair (see below), which acts as a quarantine. The main task of this pair, however, is to collect all the material appearing between \midinsert and \endinsert, and either typeset it, or place it in \midins. The \begingroup, \endgroup pair serves to localize the settings of \box0 and \dimen0 which the user never sees.

Most of the work is done by \endinsert. It closes the insertion material into \box0, and measures — with the help of our old friend, macro \pagespace — the amount of space left on the current page. If there is enough space, it typesets \box0 immediately, otherwise, it inserts it in \midins.

```
\newinsert\midins
\dimen\midins=\vsize
\newdimen\spaceleft

\def\pagespace{...}

\def\midinsert{\par\begingroup
 \setbox0=\vbox\bgroup}
\def\endinsert{\egroup % finish the \vbox
 \pagespace
 \dimen0=\ht0 \advance\dimen0 by\dp0
 \ifdim\dimen0>\spaceleft
   \insert\midins{\unvbox0}
 \else
   \box0 \bigbreak
 \fi
\endgroup}
```

```
\output={\shipout\box255
 \ifvoid\midins\else\unvbox\midins\fi
 \advancepageno}
```

Note that \unvbox0, not \box0, gets inserted in \midins. This way the insertion material is the contents of \box0 and, if it is too large, TEX will be able to split it (unless it is itself a box). Also, the \unvbox\midins has the effect of placing the contents of \midins as a top insert at the head of an otherwise empty MVL.

The maximum size of inserted \midins material per page is the value of \dimen\midins which, in our case, is \vsize. This means that the entire page can be devoted to \midins insertions. However, if we set \dimen\midins=2in then each page will contain at most 2 inches worth of material from \box\midins. If \box\midins contains more than \dimen\midins of material, some of it will be held over to the next page (requiring, perhaps, splitting one block of insertion material).

If the contents of \box0 is another box, then it is indivisible, and TEX will not split it. In such a case, more than \dimen\midins worth of material may appear on a page. In fact, the resulting page may even be larger than \vsize, and no error message would be issued. Thus when using unsplittable insertions, the user should make sure that they are not too big. A detailed discussion of insertion splitting appears later.

The appearance of the text can be improved if we automatically add some glue, such as a \bigskip, after each insertion. If a page is broken between the insertion and the glue, the glue will, as usual, be discarded at the top of the new page. Also, the natural size of the \bigskip, 12pt, should be included in the test for space left. Only \endinsert needs to be modified.

```
\def\endinsert{\egroup % finish the \vbox
 \pagespace
 \dimen0=\ht0 \advance\dimen0 by\dp0
  \advance\dimen0 by \bigskipamount
 \ifdim\dimen0>\spaceleft
   \insert\midins{\unvbox0 \bigskip}
 \else
   \box0 \bigskip
 \fi
\endgroup}
```

Readers experimenting with these macros will discover very quickly that insertions are sometimes typeset in reverse order. This may occur when a large insertion appears close to the bottom of a page. Imagine a situation where 3 inches are left on the page and the user calls \midinsert to insert a 4-inch-tall figure. \endinsert will save the figure in \midins and it will eventually appear at the top of the next page. Imagine now that the user immediately calls \midinsert to insert another figure, only 2 inches tall. Since there is room on the current page for the second figure, it will be inserted in place, with the result that the two figures are now inserted in reverse order.

A simple (but, unfortunately, incomplete) solution is: A new boolean variable, \ifSaved, is declared. When an insertion is placed in the insertion box, \endinsert invokes the OTR temporarily, using a penalty of −10001. The OTR sets \ifSaved to true, and returns without shipping out anything. When the OTR is invoked normally, it sets \ifSaved to false. \ifSaved therefore indicates whether an insertion has been saved on the current page.

When \endinsert finds that there is room on the current page for the current insertion, it typesets it only if \ifSaved is false.

```
\newif\ifSaved \Savedfalse

\def\midinsert{\par\begingroup
 \setbox0=\vbox\bgroup}
\def\endinsert{\egroup % finish the \vbox
 \pagespace
 \dimen0=\ht0 \advance\dimen0 by\dp0
 \ifdim\dimen0>\spaceleft
   \insert\midins{\unvbox0}\penalty-10001
 \else
   \ifSaved
     \insert\midins{\unvbox0}
   \else
     \box0 \bigbreak
   \fi
 \fi
\endgroup}

\output={%
  \ifnum\outputpenalty=-10001
   \global\Savedtrue
   \unvbox255
  \else
   \global\Savedfalse
   \shipout\box255 \advancepageno
   \ifvoid\midins\else\unvbox\midins\fi
  \fi}
```

This is a good solution that works almost always. It may fail in some rare cases, however. The reason is that \penalty-10001 does not invoke the OTR immediately. The penalty is stored in the MVL, and is only noticed by TEX when it starts looking for a good point to break the page. This process is explained in detail in part II, but here is an example.

Imagine a case where there is an insertion, with `\penalty-10001`, on line 60, and page 7 should be broken around that line. When TEX invokes the page break algorithm, it notices the special penalty, breaks the page at that point, and invokes the OTR. The OTR also senses the special penalty and assumes that there is an insertion on page 7. The OTR then returns the material to the MVL, which causes TEX to immediately start looking for a page break. Since the special penalty is no longer there (Note 16), TEX may select a different breakpoint, such as line 59. Line 60 is now the first line of the next page, page 8, but the OTR has already assumed that there is an insertion on page 7.

`\topinsert` and `\pageinsert`. These macros are part of the `plain` format, in addition to `\midinsert` [115–116]. Material appearing between `\topinsert` and `\endinsert` is considered a *floating top insertion.* TEX will try to place it at the top of the current page but, if there is not enough room on the current page, the material will be placed at the top of the next one. Similarly, material appearing between `\pageinsert` and `\endinsert` is stretched to the size of a page, and becomes the next page.

Readers who have read the preceding text and examples are urged to look at [363] and try to understand the definitions of the three macros.

Example: Two insertion classes

It is possible, of course, to declare several insertion classes and limit the amount of insertions placed on a page from each class. Following are the outlines of a case where two insertion classes, `\midins` and `\footins` are declared and limited to `2.5in` and `1in` per page, respectively.

```
\newinsert\midins \newinsert\footins
\dimen\midins=2.5in
\skip\footins=12pt plus4pt minus4pt
\count\footins=1000
\dimen\footins=1in

\def\midinsert{...}
\def\endinsert{...}
\def\footnote#1#2{...}

\output={\shipout\vbox{\box255
  \ifvoid\footins\else
   \vskip\skip\footins
   \kern-3pt\hrule width2in\kern2.6pt
   \box\footins
  \fi}
  \ifvoid\midins\else\unvbox\midins\fi
  \advancepageno}
```

The OTR ships out `\box255` followed by the footnotes, and TEX's insertion mechanism guarantees that the total amount of footnotes will not exceed `1in` per page. Also, if there are `\midins` insertions, they will not exceed `2.5in` per page.

It is now clear why material is not inserted directly into the insertion box but is saved in a temporary buffer. This is how insertion material can be held over for the next page. Right before the OTR is invoked, the right amount of material is moved from the buffer and is placed in the insertion box.

The `plain` format OTR

Short and elegant, this OTR makes a good example, since it supports both footnotes and floating insertions. It is described on [255–256] and, therefore, only a few short remarks are necessary here. The first step is to define a macro `\plainoutput`

```
\def\plainoutput{\shipout\vbox
    {\makeheadline\pagebody\makefootline}
  \advancepageno
  \ifnum\outputpenalty>-20000
    \else\dosupereject\fi}
```

following which, the OTR is defined by

```
\output={\plainoutput}
```

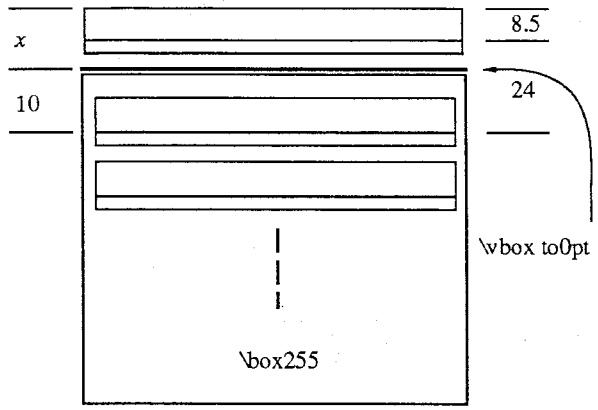
This way, the OTR can be redefined and then reset back to its original definition.

```
\def\makeheadline{\vbox to0pt{\vskip-22.5pt
  \line{\vbox to8.5pt{}\the\headline}\vss}
  \nointerlineskip}
```

Macro `\makeheadline` is the first item shipped. It suppresses the normal interline glue, so it is placed right on top of the second item (which is supplied by `\pagecontents`, see below). To achieve a uniform appearance of the document, the headline should have the same position, relative to the main body of the text, on all the pages. Its baseline is positioned, by `\makeheadline`, exactly 24pt above the baseline of the top line of `\box255`. This is achieved by placing the headline in a `\vbox to0pt`, moving up 22.5pt in the box, and typesetting the headline. The quantity 22.5pt (see diagram on following page) is the value that x should have in order that $x + 10$ should be equal to $24 + 8.5$.

The quantity `\headline` is declared as a `\toks` variable by `\newtoks\headline` and is set to an empty line `\headline={\hfil}`. It can be reset by the user to any token string.

Macro `\pagebody` limits the depth of the page to the value of parameter `\maxdepth`, whose `plain` format value is 4pt [348]. (See discussion of `\boxmaxdepth` in part I. See also the section, later in this part, on the depth of the current page.)



Position of Headline

```
\def\pagebody{\vbox to\vsize
 {\boxmaxdepth=\maxdepth \pagecontents}}
```

The `\pagecontents` macro starts by preparing the floating insertions, if any. It then opens `\box255` and, finally, prepares the footnotes, if any.

```
\def\pagecontents
 {\ifvoid\topins\else\unvbox\topins\fi
 \dimen0=\dp255 \unvbox255
 \ifvoid\footins\else
   \vskip\skip\footins
   \footnoterule
   \unvbox\footins
 \fi
 \ifraggedbottom \kern-\dimen0 \vfil \fi}
```

The two insertion boxes and `\box255` are opened, exposing their glues. The glues are now flexed to help `\pagebody` prepare a `\vbox to\vsize`. If the user wishes a ragged bottom, a `\vfil` glue is placed at the bottom of the page. This glue is flexed together with the other flexible glues on the page, leaving a glob of glue of non-zero size at the bottom of the page. The result is pages in which the bottom lines are not all at the bottom of the page. It should be noted that the definition of `\raggedbottom` (on [363]) also makes the `\topskip` glue stretchable, and that there is a `\normalbottom` macro (defined on the same page) that cancels the ragged bottom effect.

Macro `\footnoterule` creates the rule separating the footnotes from the main body of the text. The rule is placed 3pt above the top footnote.

```
\def\footnoterule{\kern-3pt
  \hrule width 2truein \kern 2.6pt}
% the \hrule is .4pt high
```

Finally, macro `\makefootline` places the footline 24pt below the main body of the page.

```
\def\makefootline{\baselineskip=24pt
 \line{\the\footline}}
```

The footline itself is a `\toks` variable declared by `\newtoks\footline`, and is set to

```
\footline={\hss\tenrm\folio\hss}
```

The page number. Some of the information in this section has already appeared in part I, and is repeated here for the sake of completeness.

In book publishing, both roman and arabic numerals are used for page numbers. Variable `\count0` is reserved by the `plain` format for the page number (`\countdef\pageno=0`) and, consequently, should not be used for anything else. It is initialized to one (`\pageno=1`), and is handled by several useful macros:

```
\def\folio{\ifnum\pageno<0
 \romannumeral-\pageno
 \else\number\pageno \fi}
\def\nopagenumbers{\footline{\hfil}}
\def\advancepageno{\ifnum\pageno<0
 \global\advance\pageno by -1
 \else\global\advance\pageno by 1 \fi}
```

Macro `\folio` typesets the page number either in arabic numerals or, if it is negative, in roman numerals.

The `\nopagenumbers` macro suppresses page numbers by eliminating them from the `\footline`.

Macro `\advancepageno` increments the page number by either 1 or -1, depending on its sign.

In certain documents, composite page numbers are used, which consist of more than one number. A page number such as 12–52 is common and usually refers to page 52 of chapter 12. The best way to implement such numbers in TEX is to use some of the ten counters `\count0` through `\count9` [119, 254]. They should be declared, initialized, incremented and typeset by the user. TEX, however, helps in two ways:

- It writes the values of the ten counters on the dvi file with each page. This helps the preview program and the printer driver identify the pages previewed or printed. In fact, those programs do not know what page number actually appears on the page, and they consider the ten values on the dvi file as *the* page number. The user should thus refer to those ten numbers when communicating with any program that handles the dvi file.
- TEX also displays the ten counters on the user's terminal, with trailing zeros omitted, when a page is shipped out. This is how things such as `[1]`, `[12.0.52]` are displayed at typeset time.

`\supereject`

The `\bye` control sequence, which is the recommended way to stop, is a macro defined by

`\par\vfill\supereject\end`. Why `\supereject` and not just `\eject`? And what is `\supereject`?

If many insertions are used throughout a document, there is a good chance that, after the last page is shipped out, some insertions will be left in their buffers, waiting to be typeset. This should be done as part of the 'end game' of TEX, which is initiated by the `\supereject` macro [116].

It is defined on [353] as `\par\penalty-20000`. The `plain` format output routine tests (on [255]) for this value and, if `\outputpenalty=-20000`, expands macro `\dosupereject`. This macro, defined on [256], tests the parameter `\insertpenalties` (see below) to see if any insertions remain heldover in their buffers. If there are any, `\dosupereject` makes sure that the output routine will be invoked again, giving it a chance to shipout those insertions. To make sure that the OTR is invoked again, `\dosupereject` prepares a blank page in the MVL by executing `\line{}\vfill\supereject`. This generates vertical material with a blank line at the top and a penalty of -20000 at the bottom. The material is simply left in the OTR(more precisely, put on the vertical list constructed by the OTR), which means it will be returned to the MVL, causing TEX to invoke the OTR again.

> When the OTR is invoked again, it will output another page and, as usual, place `\topskip` worth of glue on top of it. To cancel that glue, `\dosupereject` really generates:
>
> ```
> line{}\kern-\topskip\nobreak
> \vfill\supereject
> ```
>
> [256], but this is a minor point.

If there are any insertions left, they will be placed in their boxes each time the OTR is invoked for an empty page. The amount of inserted material per page is controlled, as usual, by the `\dimen` variable associated with the insertion.

A simple example is the endnotes described earlier. In that example, notes are accumulated in a temporary buffer, and should be typeset at the end of the document. This has to be done from the OTR, and the best way to do it is to use the special penalty generated by the `\bye`.

`\insertpenalties`

This is one of many *internal quantities* that TEX uses (see the complete list on [271]). During an OTR, it is equal to the total number of heldover insertions [254] (Note 17). A heldover insertion is an insertion (a parameter of an `\insert` command) that should have been typeset on the current page but, because of lack of space on that page did not make it, and will be made available to the OTR in the next page. Such a heldover insertion is sometimes split and only part of it appears on the current page.

Insertions (advanced)

This is advanced material, potentially useful to users who are heavily involved with OTRs and insertions, or to people who want a deeper understanding of TEX. For most users, however, the following quote (from [123]) may apply: "*On the other hand, maybe you don't really want to read the rest of this chapter at all, ever.*"

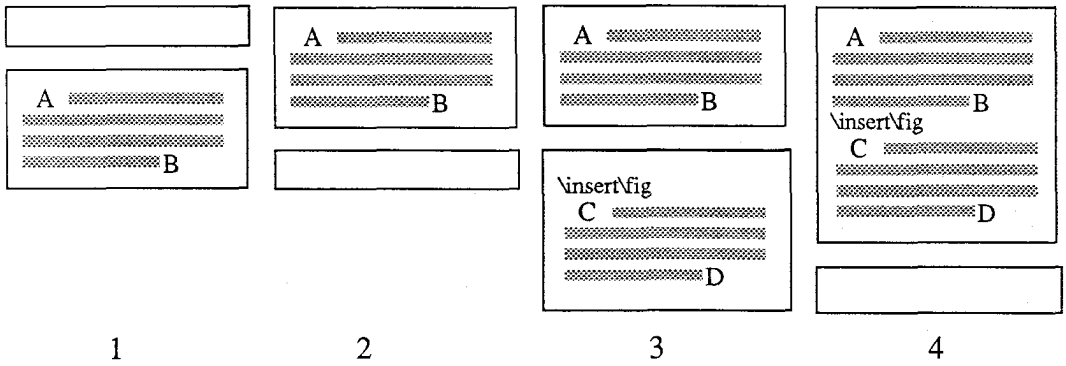
The current page and the list of recent contributions. As mentioned in part II, the MVL consists of two parts, the *current page* and, below it, the *list of recent contributions*. The current page holds the material that will become `\box255`. The recent contributions temporarily hold recently read material. After an entire paragraph has been read, it is typeset, and the lines of text appended to the recent contributions. At that point, the *page builder* is invoked (exercised). Its job is to move lines, one by one, from the recent contributions to the current page. For each line, the page builder calculates the cost of breaking the page after that line. For the first couple of lines the cost is very high because breaking there would result in a stretched page. Thus, for those lines, the badness b becomes 10000 and the cost c, 100000 (see formula on [111]).

At a certain point—when there are enough lines in the current page, for a normal page—b (and, as a result, c) starts getting smaller. A while later, there may be too many lines of text on the current page, and it has to be shrunk, increasing b and c again. The entire process can be seen, in real time, by setting `\tracingpages=1` [112]. If the page has to be shrunk more than its maximum shrinkability, both b and c become infinite. When c becomes infinite (or when a penalty ≤ -10000 is found, see below) the page builder goes back to the line of text where the cost was lowest, breaks the top of the current page and places it in `\box255` [§1017]. The bottom part of the current page is then returned to the recent contributions, and the page builder invokes the OTR.

The page builder is exercised at the end of a paragraph, at the end of a display equation within a paragraph, at the end of an `\halign`, and in a few other cases (see [122, 286]). The OTR can only be invoked by the page builder [§1025], which is why it is never invoked in the middle of a paragraph (unless the paragraph contains display math material).

The advanced reader might want to glance at [§980–1028] for the actual code of the page builder.

Since the page builder is exercised quite often, the list of recent contributions is usually small or



Figures. 1–4.

empty, and the current page gets larger and larger. When the OTR is invoked, the current page is empty. The `\showlists` command can always be used to display the two parts of the MVL in the log file.

The quantity t (`\pagetotal`) mentioned before as the height of the MVL is, actually, the height of the current page. It is updated by the page builder each time a line (or glue) is added to the current page.

A better understanding of this process must include glue and penalties. They are appended to the recent contributions, with the lines of text, when a paragraph is typeset, and are eventually moved to the current page. If the current page is empty, all glues, kerns and penalties moved to it are discarded. When the first box is moved to the current page, glue is added above it to keep its baseline `\topskip` below the top of the page. Following that, all glue, kern, and penalties are moved, with the text, from the recent contributions to the current page.

When a penalty ≤ -10000 is encountered, TEX breaks a page. The resulting page may be underfull. Such penalty values can be used to eject a page (by `\vfill\penalty-10000`), or to communicate with the OTR.

It should be stressed again, however, that `\penalty-10000` does not invoke the OTR *immediately.* If such a penalty is created inside a paragraph, between lines of text, it is saved in the recent contributions with the lines, and is only recognized as special when it is moved, by the page builder, to the current page. As a result, if a paragraph contains:

```
...\dimen0=2pt...\vadjust{\penalty-10000}
...\dimen0=1pt...\par
```

the OTR will be invoked after the entire paragraph has been read and broken into lines, and will find `\dimen0` to be `1pt`.

A page can be broken only at a glue, kern or penalty. If a page is broken at a glue or kern, the glue stays in the recent contributions (to be discarded when moved to the top of the next page). If the page is broken at a penalty, the penalty is saved in variable `\outputpenalty` and removed from the vertical list. This variable can be used to communicate with the OTR. Also, if the user wants to return some material from `\box255` to the current page, he may want to reinsert the penalty, by saying `\penalty\outputpenalty`.

Insertions and the page builder. We are now familiar with how the MVL is maintained in cases that don't involve insertions. In this section we see how insertions are handled in the MVL by the line break algorithm and the page builder. Let's assume that an insertion class n has been defined. When an `\insert`n is read from the source file, both the command and its insertion material are placed in the recent contributions. The next time the page builder is exercised, it finds the command, followed by the insertion material. The material should not be moved to the current page, since it is an insertion (review the definition of insertions). Instead, it should be moved to `\box`n, so the OTR should be able to typeset it anywhere on the page. However, material is only moved to `\box`n just before the OTR is invoked (see below). Therefore, when the page builder discovers the command, it (1) moves the command (and the insertion material), to the current page, but as a special item, not as a regular part of the current page (the material will later be moved to `\box`n from the current page); (2) decrements g by the size (height plus depth) of the insertion material.

Figures 1–2 show a paragraph (A–B) read into the recent contributions and moved to the current page. Figures 3–4 show how an `\insert\fig` command, followed by insertion material (C–D), is also read into the recent contributions and moved, as a special item, to the current page.

Splitting insertions. Before the page builder decrements g, it executes the rules on [123–124] to determine how much of the insertion material

can appear on the page. If there is no room for the entire insertion — either because it is large, or because \dimen n has been assigned a small value — the rules tell how to determine a good point to split the insertion material so the remainder can be *held over* for the next page. The result obtained by the rules is used to decrement g, to reserve room on the page for the insertion.

Again, it should be emphasized that the split itself does not occur at this point. It takes place just before the OTR is invoked (see below). At that time, the top part of the split insertion is placed in \box n, and the bottom part is saved as a heldover insertion.

The rules for splitting insertions, in simplified form, are:

1. The first \insert n for the page decrements g by (the natural size of) \skip n, and again by the height plus depth of \box n. Note that g is not decremented by the size of the present insertion (this is done in rule 3.)

What can \box n contain at this point?

1a. It may be empty.

1b. It may contain material from the previous page. Typically, such material should have been typeset, by the OTR, on the previous page, and \box n emptied. However, if the OTR did not empty the box, room is now reserved for its contents on the present page.

1c. It may contain material placed there by the user explicitly, not through the \insert n command. In such a case, room is now reserved on the page for this material. If anything is placed explicitly in the box after this point, no room will be reserved for it on the page [§1009].

2. If a previous \insert n on the current page has been split (because it didn't fit on the page), the present insertion will certainly not fit on the page, and has to be held over. The only thing done at this point is to increment \insertpenalties by the parameter \floatingpenalty. This increases the cost of breaking the page at this point. See [124–125] for examples of values of \floatingpenalty.

3. Determine if the insertion will fit on the page without being split. If it will, decrement g by the size x (height plus depth) of the insertion material. Otherwise go to step 4 to calculate the split size.

We denote the quantity 0.001\count n by f. The value of g should be decremented by the scaled size xf of the insertion material.

An insertion will fit on the page if its scaled size xf is zero (or negative), or if

$$xf \le g - t \qquad (1)$$

or if \count $n = 0$. The actual test also includes the \pagedepth, \pageshrink parameters, which are ignored here for simplicity. They are introduced in a later section.

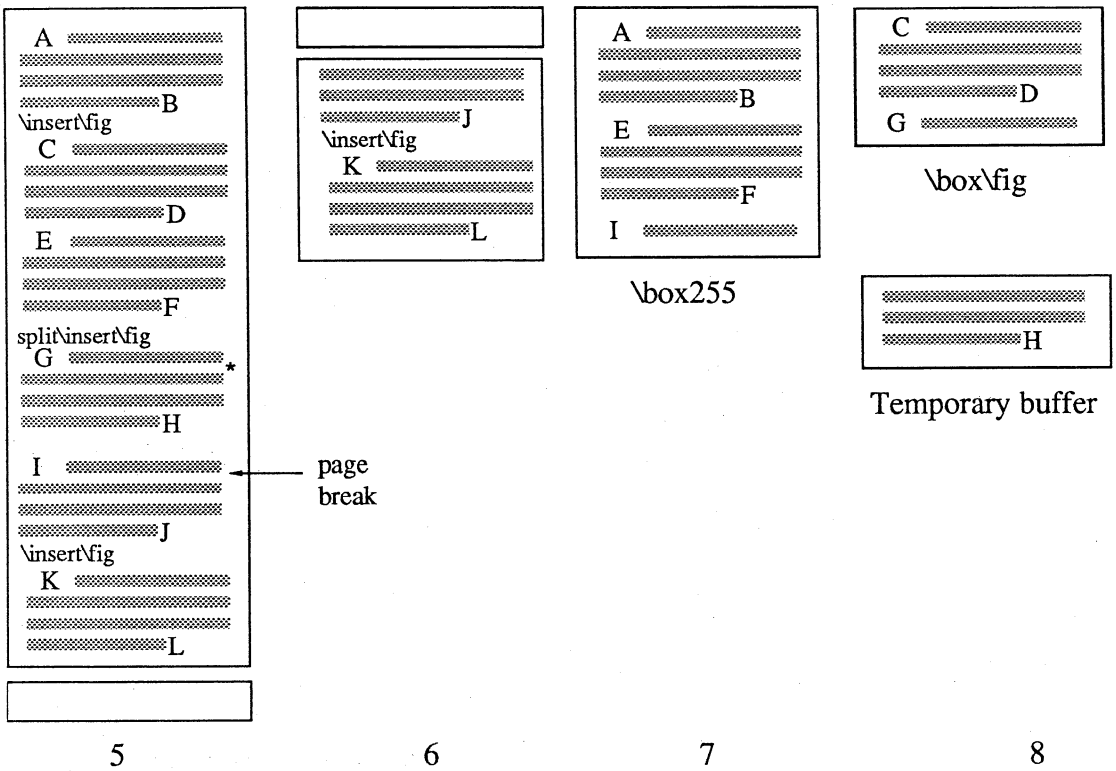
4. Determine where to split the insertion. Let's assume that we end up splitting \insert n at a distance v from its top. What determines v? After the material is split and is placed in \box n, the box's vertical size increases to $x + v$. The value of v should, therefore, be the largest number that satisfies (a) the new size, $x + v$, of \box n should be $\le$ \dimen n; (b) v should also be $\le g - t$ (the available space on the page). Relation (b) will also be modified later.

Since a split must occur between lines of text, it may be impossible to split \insert n to v. TEX therefore uses an algorithm, similar to the page builder but without insertions, to determine a value u close to v.

g is now decremented by u and the parameter \insertpenalties is incremented by the penalty value (if any) found at the split point. The page builder marks this insertion, in the current page, as a split insertion. Note that the split itself does not take place at this point. It is done after the page breakpoint is determined, and before the OTR is invoked.

All this happens when an \insert command is discovered by the page builder on the recent contributions, and is moved to the current page [§1000, §1008]. The page builder continues its operations and, finally, decides on a good breakpoint for the page. (Note: The value of \insertpenalties is used to help make the decision and, once it is made, \insertpenalties is free to be used for something else.) Fig. 5 shows an example of a current page with 3 paragraphs (A–B, E–F, and I–J) and 3 insertions (C–D, G–H and K–L) the second of which is stored in the current page as a split insertion (the '*' marks the split point.) The recent contributions list is empty.

The page builder then (see [125]) removes the bottom of the current page (everything below the breakpoint) and returns it to the recent contributions (Fig. 6). The next step is to place all the insertion material of class n in \box n. The page builder scans the current page and, for each \insert n found, appends the insertion material to \box n. When it finds a split insertion, it performs the actual split, appends the top part of the split material to \box n, and saves the bottom, as an independent insertion, in a separate place. All class n insertions found on the current page following this point, are saved in the same way, to be held over (Fig. 8).



Figures. 5–8.

For each of the heldover insertions saved, `\insertpenalties` is incremented by 1. This is why, in the OTR, this variable holds the number of heldover insertions.

The current page (without the insertion items) is now moved [§1017] into `\box255` (Fig. 7), which is set to height g, and the OTR is invoked [§1025]. It may return material to the MVL (to the recent contributions). When the OTR is finished, all the heldover insertions are moved from their saving place to the top of the recent contributions. At that point, if there is enough material in the recent contributions, TEX may exercise the page builder again.

If the OTR does not use the material in `\box`n, it stays in the box, and can be used in the following page. When the page builder builds the next page, g is decremented by the vertical size of `\box`n, just before the first block of insertion material is moved to the current page.

The d and z parameters in insertions. The two parameters `\pagedepth` and `\pageshrink` have already been mentioned in part I, in connection with the depth of boxes. They are denoted d and z, respectively, and are included in the tests for insertion split. The reader will recall Eq. 1: $xf \le g - t$, which means that an insertion of size x will fit on the page if its scaled size xf is less than or equal to the available room on the current page, $g - t$. Now, that we know about d and z, it is clear that we should include $t + d$, instead of just t, in this equation. The actual test performed by TEX [§1008] is

$$\Delta = g - t - d + z;$$
if $xf \le 0$ **or** $xf \le \Delta$
then $g \leftarrow g - xf$
else go to step 4.

Δ is the room left on the current page after it is shrunk as much as possible. f is the value of `\count`n (divided by 1000).

The last thing to be updated is rule 4*b* above. It states that a block of insertion material should be split at a distance v from its top, where v is the largest number that satisfies $v \le g - t$. The actual expression used [§1010] also takes d and f into account. It is $vf \le g - t - d$.

`\holdinginserts`. Sometimes the OTR is invoked temporarily, before the current page is completed, just so that it can examine the page so far and do something special. Such a special invocation is easy to do by saying `\penalty-10001`. The OTR can perform tests on `\box255` and return it to the MVL. If the OTR is invoked in such a way, it may be desirable to leave all insertion material in place and not put it in the insertion boxes. Starting with TEX version 3.0, this can be achieved by setting

the new parameter \holdinginserts to a positive value.

This feature will be mentioned on [125] starting with the seventeenth printing of *The TEXbook*.

Tracing (in detail)

As mentioned before, a good way to learn about insertions is to trace the internal operations of TEX while it handles this 'sensitive' material. Fortunately, several tracing commands [303] are available, to bring out and print the values of many internal quantities. The most useful to us are \message, \tracingpages and \showlists. The following examples illustrate tracing, and should be studied, performed, and modified by the serious reader. This is an excellent way to understand the operations discussed in the previous section.

We start with a simple example involving 5 short paragraphs, and 4 unsplittable insertions.

```
\hsize=3in \vsize=100pt
\tracingpages=1
\showboxbreadth=1000 \showboxdepth=1

\newinsert\trace
\count\trace=1000
\skip\trace=12pt
\dimen\trace=100pt

\output={\message{%
    R: \the\ht255, \the\insertpenalties;}
  \shipout\vbox{\unvbox255
    \vskip\skip\trace \unvbox\trace
    \smallskip
    \centerline{\tenrm---\folio---}}
  \advancepageno}

\def\mes#1{\message{#1: \the\pagetotal,
 \the\pagegoal, \the\insertpenalties;}}

\mes1
Tracing insertions. Both message \&
tracingpages are used to keep track
of the values of certain quantities
involved with insertions. This helps
to understand the operations of the
page builder. \par\mes2

\insert\trace{\vbox to30pt{%
 A 30pt insertion\vfil\hrule}}
Paragraph 2 \par\mes3

\insert\trace{\vbox to25pt{%
 A 25pt insertion\vfil\hrule}}
Paragraph 3 \par\mes4

\insert\trace{\vbox to20pt{%
 A 20pt insertion\vfil\hrule}}
Paragraph 4 \par\mes5

\insert\trace{\vbox to15pt{%
 A 15pt insertion\vfil\hrule}}
Paragraph 5 \par\mes6

\bye
```

Typesetting the material above creates three small typeset pages (only the first two of which are shown here.)

Tracing insertions. Both message & tracingpages are used to keep track of the values of certain quantities involved with insertions. This helps to understand the operations of the page builder.

A 30pt *insertion*

—1—

Paragraph 2
Paragraph 3

A 25pt insertion

A 20pt insertion

—2—

It also generates the following log file.

```
1. \trace=\insert252
2. 1: 0.0pt, 16383.99998pt, 0;
3. %% goal height=100.0, max depth=4.0
4. % t=10.0 g=100.0 b=10000 p=250 c=100000#
5. % t=22.0 g=100.0 b=10000 p=0 c=100000#
6. % t=34.0 g=100.0 b=10000 p=150 c=100000#
7. 2: 46.0pt, 100.0pt, 0;
8. % t=46.0 g=58.0 b=10000 p=0 c=100000#
9. 3: 58.0pt, 58.0pt, 0;
10. % split252 to -1.94444,25.0 p=-10000
11. % t=58.0 plus 1.0 g=33.0 b=* p=0 c=*
12. R: 58.0pt, 0; [1]
13. %% goal height=100.0, max depth=4.0
14. % t=10.0 g=63.0 b=10000 p=0 c=100000#
15. 4: 22.0pt, 63.0pt, 0;
16. % t=22.0 plus 1.0 g=43.0 b=10000 p=0
17.   c=100000#
18. 5: 34.0pt, 43.0pt, 0;
19. % split252 to 7.05556,15.0 p=-10000
```

```
20. % t=34.0 plus 2.0 g=28.0 b=* p=0 c=*
21. R: 43.0pt, 0; [2]
22. %% goal height=100.0, max depth=4.0
23. % t=10.0 g=73.0 b=10000 p=0 c=100000#
24. 6: 22.0pt, 73.0pt, 0;
25. % t=22.0 plus 1.0 g=73.0 b=10000 p=0
26.   c=100000#
27. % t=23.94444 plus 1.0 plus 1.0fill g=73.0
28.   b=0 p=-20000 c=-20000#
29. R: 73.0pt, 0; [3]
```

Message 1 (line 2) shows the values of t and g before TEX encounters any text. Line 3 (with %%) shows the goal height, which is still `\vsize`. Line 4 is generated when the first text line is moved to the current page. It shows $t = 10\,\text{pt}$, the height of the first line of text (plus the `\topskip` glue above it). Line 5 shows $t = 22\,\text{pt}$, which is the height of the first text line, plus the `\baselineskip` following it, plus the height of the second line of text (the depth of the last line is the depth of the page, and is therefore not included in t). Lines 6–8 show t growing in steps of 12 pt until it reaches 46 pt, the total height of the 4 lines of the first paragraph. Message 2 (line 7) shows $t = 46\,\text{pt}$ and $g = 100\,\text{pt}$, still equal to `\vsize`. However, line 8 shows that g was decremented, as a result of the first `\insert`, from 100 to 58, a difference of 42 pt. This equals the size (30 pt) of the material inserted, plus the natural size (12 pt) of `\skip\trace`.

Message 3 (line 9) shows $t = 58\,\text{pt}$, because the second paragraph (a single line) was read, typeset, and moved to the current page. At this point both t and g equal 58 pt (but for different reasons!). It would seem like an ideal point to break the page, but the page builder starts looking for a page break only when $c = \infty$ or when the current penalty ≤ -10000 [§1005]. So it reads the next item from the source file, which happens to be the next insertion (25 pt). The page builder tries to move it to the current page, and it executes the 4 steps on [123–124]. Steps 1, 2, don't apply. The test in step 3 is not passed, so the page builder goes to step 4 and calculates a good splitting point for the insertion. The test on the second line of [124] results in $v = -d$ (since $t = g$ and $f = 1$). This means that the ideal split is at a point 1.9444 pt *above* its top. This is why line 10 shows that the page builder has tried to `split252 to -1.94444`. This is a strange split but, in any case, it cannot be done since the insertion is a box. The page builder thus moves the entire insertion to the current page, and decrements g to 33 pt.

However, the 58 pt of material cannot be shrunk to 33 pt, resulting in line 11 with `b=* p=0 c=*`, infinite badness and cost. This is the time to start looking for a page break, so the page builder goes back to the point, in the current page, with the least cost, and breaks the page there. What is that point? The current page contains 5 lines. Each of the first 4 lines is associated with a cost of 100000, and the last line has infinite cost. The most logical point for a page break is, therefore, following the fourth line.

The part of the current page below the breakpoint (consisting of the line "Paragraph 2" and the 25 pt insertion) is returned to the list of recent contributions. The insertion material from the current page is moved to `\box\trace`, the rest of the current page is moved to `\box255` (actually, the rest of the current page *becomes* `\box255`), and the page builder invokes the OTR.

A `\showlists` command placed in the OTR would show no current page, and recent contributions consisting of the line "Paragraph 2" and the 25 pt insertion.

The `R` message (line 12) shows `\ht255` = 58 pt, so the total height of the page shipped out is $58+12+30 = 100\,\text{pt}$. This is a successful case since, with many unsplittable insertions, some pages must be stretched a lot.

The next page starts with (line 14) $t = 10\,\text{pt}$ (one line of text, "Paragraph 2"), and $g = 63\,\text{pt}$ $(= 100-12-25)$. On lines 16–18 t is incremented to 34 pt, which means that 3 lines of text (paragraphs 2, 3 and 4) are tentatively considered). Message 5 (line 18) shows $g = 43\,\text{pt}$ which means that the 20 pt insertion has been read. It also shows $t = 34\,\text{pt}$ which means that there is still room on the page for 9 pt worth of material (typically 7 pt high and 2 pt deep).

The next item is read from the source file. It is the 15 pt insertion. the page builder calculates (line 19) a split point (`split252 to 7.05556`) but, since it is an (indivisible) box, it cannot be split. It is moved to the current page, causing an infinite cost (line 20). A page break point is determined as before, and it is following the second line ("Paragraph 3"). Paragraph 4 and the 15 pt insertion are returned to the list of recent contributions, and the current page becomes `\box255`.

The `R` message (line 21) shows `\ht255` = 43 pt. The box contains just two lines of text (a height of 22 pt) and was stretched to 43 pt at the paragraph break.

The rest of the log file, pertaining to the third page, is easy to read and is left as an exercise.

Exercise: Add flexibility to `\skip\trace` (such as `12pt plus6pt minus4pt`) and typeset the example. Make sure that you see how the flexibility is reflected in the values for t.

Exercise: Change \vsize to 90 pt and repeat the experiment. The main changes should be in the splitting. The page builder will try to split the insertions at different points. Since the insertions are indivisible, they will not be split.

Exercise: Add \showlists commands after each \message, and in the OTR. You may have to fiddle with the values of \showboxbreadth and \showboxdepth in order to get the right amount of output.

The next experiment deals with splittable insertions. We modify the source file to:

```
\hsize=3in \vsize=100pt
\tracingpages=1
\showboxbreadth=1000 \showboxdepth=1

\newinsert\trace
\count\trace=1000
\skip\trace=12pt
\dimen\trace=100pt

\output={\message{R:
    \the\ht255, \the\insertpenalties;}
  \shipout\vbox{\unvbox255
    \vskip\skip\trace \unvbox\trace
    \smallskip
    \centerline{\tenrm---\folio---}}
  \advancepageno}

\def\mes#1{\message{#1:
 \the\pagetotal, \the\pagegoal,
 \the\insertpenalties;}}
\mes1

Tracing insertions. Both message \&
tracingpages are used to keep track
of the values of certain quantities
involved with insertions. This helps
to understand the operations of the
page builder. \par\mes2

\insert\trace{\noindent* This is the
first insertion, about four lines worth
of text. This would make it possible
for \TeX\ to split the insertion,
if necessary. Up until now our insertions
were unsplittable}
Paragraph 2 \par\mes3

\insert\trace{\noindent* This is the
second insertion, three lines worth
of text. This would make it possible
for \TeX\ to split the insertion, if
necessary.}
Paragraph 3 \par\mes4

\insert\trace{\noindent* The third
insertion, four lines worth of text,
to illustrate the insertion splitting
rules on [123]. Note how this is split,
and how the split part is typeset
following the text on this page.}
Paragraph 4 \par\mes5

\insert\trace{\noindent*
 Insertion 4, one line.}
Paragraph 5 \par\mes6
\bye
```

This produces 3 typeset pages, only the first 2 of which are shown here.

Tracing insertions. Both message & tracingpages are used to keep track of the values of certain quantities involved with insertions. This helps to understand the operations of the page builder.

* This is the first insertion, about four lines worth of text. This would make it possible for TEX to split the insertion, if necessary. Up until now our

—1—

Paragraph 2
Paragraph 3

insertions were unsplittable
* This is the second insertion, three lines worth of text. This would make it possible for TEX to split the insertion, if necessary.
* The third insertion, four lines worth of text, to il-

—2—

It also generates the following log file:

```
\trace=\insert252
1: 0.0pt, 16383.99998pt, 0;
%% goal height=100.0, max depth=4.0
% t=10.0 g=100.0 b=10000 p=250 c=100000#
% t=22.0 g=100.0 b=10000 p=0 c=100000#
% t=34.0 g=100.0 b=10000 p=150 c=100000#
2: 46.0pt, 100.0pt, 0;
% split252 to 40.05556,33.44444 p=150
% t=46.0 g=54.55556 b=10000 p=0 c=100000#
3: 58.0pt, 54.55556pt, 150;
% t=58.0 plus 1.0 g=54.55556 b=* p=0 c=*
R: 54.55556pt, 1; [1]
%% goal height=100.0, max depth=4.0
% t=0.0 g=76.05556 b=10000 p=0 c=100000#
% t=10.0 g=42.61111 b=10000 p=0 c=100000#
4: 22.0pt, 42.61111pt, 0;
```

```
% split252 to 18.66667,9.44444 p=250
% t=22.0 plus 1.0 g=33.16667 b=10000 p=0
  c=100000#
5: 34.0pt, 33.16667pt, 250;
% t=34.0 plus 2.0 g=33.16667 b=* p=0 c=*
R: 33.16667pt, 1; [2]
%% goal height=100.0, max depth=4.0
% t=0.0 g=52.05556 b=10000 p=0 c=100000#
% t=10.0 g=42.61111 b=10000 p=0 c=100000#
6: 22.0pt, 42.61111pt, 0;
% t=22.0 plus 1.0 g=42.61111 b=10000 p=0
  c=100000#
% t=23.94444 plus 1.0 plus 1.0fill
  g=42.61111 b=0 p=-20000 c=-20000#
R: 42.61111pt, 0; [3]
```

The main differences between this experiment and the previous one are:

1. Insertions can now be split. The message `split252 to 40.0555,33.4444 p=150` shows that the first insertion should, ideally, have been split at a distance of 40 pt from the top. Such a point, however, is between two lines of text, so the insertion ended up being split at 33.4 pt, after the third line of text. Note the widowpenalty of 150 found there.

The split operation is similar to a page break, a fact which shows us how to control insertion splitting. We can, e.g., place a penalty of -10000 in the first insertion.

```
\insert\trace{\noindent* This is the first
insertion, about four lines worth of text.
\vadjust{\penalty-10000} This would make
it possible ... were unsplittable}
```

This will force a split of the insertion after the second line. The log file will now contain the line:

```
split252 to 39.5,21.6527 p=-10000
```

showing that the split occurred 21.6 pt from the top (a height of two lines) because of the large negative penalty found.

2. The displayed values of `\insertpenalties` show the dual nature of this parameter. Several messages display the value 150 (= `\widowpenalty`). In the OTR, however, the value of `\insertpenalties` is not a penalty but the number of heldover insertions. When the first page is shipped out, the second insertion has already been read, and is being held over, together with the split part of the first insertion. As a result, the value of `\insertpenalties` in the OTR is 2.

Exercise: Place `\showlists` commands after each `\insert\trace{...}` and in the OTR. This will show how inserted material is stored in the recent contributions and in the current page.

Summary

This is a tutorial, not a cookbook. It does not contain any canned macros that can be directly copied and used. Instead, it tries to develop a better understanding of insertions, so that the reader will be able to implement insertions for specific applications.

All the material presented here (except, perhaps, some examples) can be found in *The TEXbook*, although in a somewhat cryptic language. The serious reader should, therefore, after reading this tutorial and doing the exercises, go back to the book to get a different perspective on the topics discussed here.

Endnotes

[1] This is an endnote. Look at the endnotes example to see how it works.

[2] The idea is that, when a textbook is written, items that should appear in the index of the book should be flagged by the author and written by TEX on a file, for the future preparation of an index. While the book is being written and proofread, it is also handy to have all the index items for a page printed on the right margin of that page. On the final printing of the page, those items are suppressed.

[3] Given two large figures that are textually related, they should be inserted into the document close to each other. If they don't both fit on one page, they should be inserted on facing pages, which means that the first figure should be inserted on the next even-numbered page, and the second figure, on the page following.

[4] All that the user has to do is save the figures in boxes and check, in the OTR, for the next even-numbered page.

[5] Answer: Because glue and kern are discardable items and disappear at a page break.

[6] Actually, in a temporary place.

[7] Actually, just before the OTR is invoked, the material is brought in from temporary storage and is appended to the box. Note that the allocated box may contain other material, placed there by the user not through the `\insert` command. Such material remains in the box and is eventually typeset on the page by the OTR. However, no room is reserved on the page for such material, and it may cause a page overflow.

[8] Actually, in a temporary buffer.

[9] We use t to denote `\pagetotal`, and g to denote `\pagegoal`.

[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