Output Routines: Examples and Techniques. Part I: Introduction and Examples.

David Salomon

A general note: Square brackets are used throughout this article to refer to the TEXbook. Thus [400] refers to page 400, and [Ch. 6], [Ex. 15.13] refer to chapter 6 and exercise 15.13, respectively, in the book. Advanced readers are referred to the actual `WEB` code by the notation [§1088]. Also, since the words "output routine" are commonly used in this article, they are replaced by the logo OTR.

"It would be possible to write an entire book about TEX output routines; but the present appendix is already too long..." This quotation, from [400], best describes this article. It is not a book, but it tries to do justice to OTRs; justice denied them in the TEXbook, because of lack of space.

This article is long and is divided into three parts. Part I is an introduction to OTRs and related concepts, followed by examples. The examples are mostly simple OTRs, useful for common applications. No advanced techniques are used. In the second part, various techniques are developed, for communicating with the OTR by means of marks, special penalty values, kerns, and special boxes inserted in the text. Some methods require several passes, saving information between the passes either on a file or in memory. Some techniques examine the contents of `\box255`, going as far as breaking it (or a copy) up into individual components. Whenever possible, the techniques are applied to practical cases. Part III treats insertions. OTRs with insertions introduce more complexities and deserve detailed treatment. Specifically, the `plain` format OTR is introduced in part III since it supports insertions.

The reader is encouraged to send the author problems related to OTRs, suggestions for new techniques, and errors found in this document.

Certain methods described here are not completely general and may fail in certain situations. Others are more general and seem to work always. However, none of them have been thoroughly tested, and new problems may be discovered at any time.

To make it easy to read and understand the macros described here, they were deliberately kept as simple as possible. As a result, they may not be general and may not handle every possible situation. The point is that the macros should not be copied and used verbatim. Rather, they should be studied and modified for specific problems.

Advanced TEX users hardly need be convinced that an understanding of OTRs is important, since they must be used whenever special output is desired. However, the entire topic of OTRs has traditionally been considered complex, and it is! The reasons are: (1) OTRs are asynchronous with the rest of TEX (this is explained later) and involve difficult concepts such as splitting boxes and insertions. (2) Certain features which could be very useful in OTRs are not supported by TEX. Specifically, there are no commands to identify marks, rules, and whatsits in a box, and to break up a line of text into individual characters. One can only hope that, with more interest in TEX and more demand from users, there will be a future version of TEX (4.0?) with the missing features included.

Introduction

A few introductory concepts are introduced in this section for the benefit of the inexperienced reader.

Boxes. Definition: A box is an indivisible unit of material, inside which TEX switches to one of the internal modes. Thus in an `\hbox`, TEX is in restricted horizontal mode, which means that items placed in such a box will be positioned side by side. If an `\hbox` is typeset, it is not broken into lines, but is typeset as one line. Similarly, items placed in a `\vbox` are stacked from top to bottom. When such a box is typeset, the entire box is placed on the same page. More information on the properties of boxes can be found in [Ch. 11–12].

Lists. A *list* is best thought of as a bunch of boxes (and other items) positioned either horizontally or vertically. Thus TEX supports horizontal and vertical lists. The specific items that may appear in a horizontal list can be found on [95]; those which may appear in a vertical list can be found on [110]. An example of a horizontal list is the TEX logo. Its components are the two letters 'T', 'X', a box with the letter 'E', and two pieces of `\kern`. An example of a vertical list is a page of text. Its components are the individual lines of text and the penalties, glue, and other items between them.

The Main Loop. This is also known as the *Inner Loop*, the *Chief Executive*, or the *Main Control* (see [§1029, §1030, §1035]). This is where TEX spends most of its time, preparing pages of text which eventually are sent to the OTR. This loop consists of reading characters from the source file, scanning them and converting them into tokens, using the tokens to build boxes, and combining the boxes into lists.

The Main Vertical List and the Page Builder

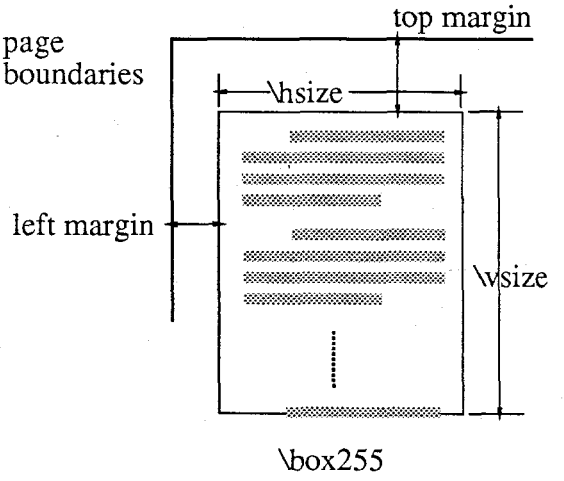
This section introduces a simple picture of a structure called the *Main Vertical List* (MVL). The precise way the MVL is organized and used is presented in a later section.

When TEX reads the first character of a paragraph, it switches to horizontal mode, where it reads the rest of the paragraph. It then switches back to vertical mode and invokes the line breaking algorithm. The resulting lines are saved in the MVL. Gradually, more and more lines of text are appended to the MVL and, at a certain point, when the MVL contains more than enough material for a page, TEX invokes (exercises) the *page builder*. The page builder decides on a good point to break the page, cuts a chunk of material off the MVL, places it in `\box255`, and invokes the OTR. The OTR usually adds things such as a page number, a header, or footnotes, and ships out the page (i.e. writes the page to the `.dvi` file).

The OTR does not have to ship the page out. It can ship out just part of it, and save or discard the rest. It can also return the rest to the MVL. The rule is that any material left over by the OTR (i.e. any material placed on the OTR's vertical list) is returned to the MVL. This is a very useful feature.

An important point, which should be emphasized at this early stage, is the *asynchronicity* of the OTR. `\box255` is *not* filled up gradually with text. It is the MVL that's filled up with lines of text until there is more than enough material for a page. The page builder then cuts a chunk, the size of a page, off the MVL, and places it in `\box255`. At that point, there is normally some material left in the MVL, for the next page. This means that, when the OTR is invoked, TEX has already read text from the source file past the end of the current page. Thus the asynchronous nature of the OTR. It is not synchronized with the main loop but is invoked from time to time, as the need arises. Another way to express this idea is to say that the OTR lags behind TEX's main loop.

Note that TEX does not know the size of the paper which eventually comes out of the printer. There are only four parameters that relate to the size and position of the page, namely: `\hsize`, `\vsize`, `\hoffset` and `\voffset`. The first two become the width and height of `\box255`. The two 'offset' parameters are the amounts by which the left and top margins differ from the default offsets of the printer which typesets the `.dvi` file. Many drivers default to offsets of `1in`, and `\hoffset` and `\voffset` give displacements from that point. Thus setting `\hoffset=0.5in`, `\voffset=-0.2in` creates a left margin of `1.5in` and a top margin of `0.8in`. The right and bottom margins are unknown to TEX, and are normally only used by the printer driver. In the rare cases where TEX needs to know those quantities, they have to be entered by the user or computed. This is demonstrated in one of the examples (for printer registration marks) shown later in this part.



Page Boundaries and the Printed Area
Figure 1

Figure 1 shows how those quantities are related. It also shows that the depth of `\box255` is the depth of the bottom line of text. As a result, the vertical size of the printed area is the height plus depth of `\box255`, and is slightly larger than `\vsize`.

`\pagetotal` and `\pagegoal`. Two `\dimen` variables are maintained by the page builder and are used in the page breaking algorithm. They are also used for insertions.

- `\pagetotal` [114], is the vertical height of the MVL. We will denote it by t. This variable starts at zero and is incremented by the page builder each time something with a height (i.e., a box or some glue) is appended to the MVL. Things like marks, penalties, and whatsits do not have any dimensions and do not affect t. Since some vertical glues have flexibility, this variable is generally flexible. Initially, when the MVL is empty, t is set to zero.
- `\pagegoal` [114], which we denote by g, is the desired height of the page. Generally, it is equal to `\vsize` but, when insertions are generated, g is decremented. Initially, when the MVL is empty, g is set to `\maxdimen`. When the first

("... TEX salts away the value of `\vsize` ..." [114]). When footnotes or other material are generated, to be inserted in the page, their heights are subtracted from g [123].

These two variables can be 'shown', either in the document itself (using `\the`) or in the log file (using either `\showthe` or `\message`). They can even be modified, but this should be done very carefully. It is even possible to get the two values displayed after each line by setting `\tracingpages=1` [112]. This option exists mainly to show the feasible points for page breaks, and how the page breaking algorithm works. It is a good idea, however, to try it once, on a short page which also has footnotes, and to follow the changes in the values of the two quantities.

As an example, the two variables will be used to determine how much space is left on the current page. If t is zero, the space left on the page is the entire page (`\vsize`). Otherwise, it is the difference $g - t$. Macro `\pagespace` calculates that difference.

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

The Current Page and the List of Recent Contributions

The discussion so far has been general, ignoring certain important details. This section presents a more accurate picture of the MVL, and the way the page builder operates.

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 are used to 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 an extremely 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 resonably looking page, b (and, as a result, c) start getting smaller. A while later, there may be too many lines of text in 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 at that point, and places it in `\box255`. 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, the end of a display equation within a paragraph, at the end of an `\halign`, and in a few other cases [122, 286]. The OTR is invoked only 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 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 earlier 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 the page builder and the MVL must include glue and penalties. When a paragraph is typeset, the lines of text are appended to the recent contributions, with glue and penalty items between them. These items are moved to the current page with the lines of text. If the current page is empty, all glues, kerns and penalties moved to it are discarded. This is how discardable items disappear at the top of a page. When the first box is moved to the current page, a special `\topskip` glue is placed above it, to keep its baseline 10pt (the value in `plain` format) below the top of the page. Following that, glue, kern, and penalties are moved, with the text, from the recent contributions to the current page.

When a penalty ≤ -10000 is encountered, the page builder breaks a page. If the resulting page does not have enough text lines, it may be underfull. Such penalty values can be used to eject a page (say, by `\vfill\penalty-10000`), or to communicate with the OTR (which knows the penalty associated with the point at which a page was broken).

It should be stressed, however, that a penalty of -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 only be broken at (i.e. just above) 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`. This variable can be used to communicate with the OTR. Also, if the OTR decides to return some material from `\box255` to the current page, it may want to place back the original penalty at the breakpoint, by saying `\penalty\outputpenalty`.

The precise rules of where a page can be broken are listed on [110]. One of them says "A page break may occur at glue, provided that this glue is immediately preceded by a non-discardable item." If a set of successive glues is moved to the current page, a page break can occur either before or after that set, but not inside it. If the page builder decides to break the page before the set, the entire set is returned to the recent contributions, to disappear at the top of the next page. If the page is broken after the set, it becomes glue at the bottom of the page. This information will be used in part II, when we try to communicate with the OTR by means of `\kern`.

The depth of the current page. The discussion so far has been kept simple (even though some readers may disagree) by ignoring certain features that have to do with the depths of the boxes involved. These features are important since, normally in a document, successive pages should have the same (or almost the same) vertical size. The height of a page is controlled by (actually, it is equal to) `\vsize`. The depth of a page should also be under the user's control since, in certain situations, it may spoil the uniform appearance of the document. This is why it is important to consider TEX features which have to do with the depth of a page, and we start by introducing certain quantities that have to do with the depth of vboxes in general.

Consider a large `\vbox` with lines of text, separated by glue and penalties. The depth of this `\vbox` [80] is the depth of the bottom component. If that component is a glue or penalty, the depth is zero. If it is a box, then its depth becomes the depth of the entire `\vbox`, except that it is limited to the value of parameter `\boxmaxdepth`. If `\boxmaxdepth=1pt` and the depth of the bottom box is `1.94444pt`, then the depth of the entire `\vbox` will be `1pt` and its height will be incremented by `.94444pt`. This is equivalent to lowering the reference point (or, equivalently, the baseline) of the `\vbox` by `.94444pt`. In the `plain` format, `\boxmaxdepth=\maxdimen` [348], so it has no effect on the depths of boxes. However, `\boxmaxdepth` can always be changed by the user.

The *current page* is that part of the MVL that contains the material for `\box255`. Its current height is t, its goal height is g, but what is its depth? It is, of course, the depth of the bottom line of text — normally a small dimension which may vary a little from page to page. This results in pages with slightly different vertical sizes (i.e. height + depth). However, if the bottom line of text contains a large symbol with a depth of, say, 35pt, the vertical size of the page will be `\vsize` + 35pt. The page will be much taller than its neighbors, spoiling the uniform appearance of the document. To avoid this, the page builder uses another parameter, `\maxdepth`, when it appends lines to the current page [125]. The `plain` format sets [348] `\maxdepth=4pt`. In our example, when the line with depth = 35pt is added to the current page, the depth of the current page is set to 4pt and the difference of 31pt is added to its height t. A good way to visualize this situation is to say that the baseline of the current page no longer coincides with the baseline of the bottom line, but is located 31pt below it.

The internal quantity `\pagedepth` (d) contains the depth of the current page, and is updated each time a line (or glue) is added to the current page. d is a 'relative' of t, the height of the current page. It should be noted that t has a few more 'relatives' [114], the most important of which are `\pagestretch` and `\pageshrink`, the amounts of

stretchability and shrinkability in the current page. t and its relatives are used by the page builder to determine a pagebreak (some of them are also used for insertions).

A simple test can show how those quantities are updated. First, change `\parskip` to some flexible value such as `1pt plus2pt minus1pt`, then typeset text in small pages and place a command such as

```
\message{(total: \the\pagetotal;
 depth: \the\pagedepth;
 shrink: \the\pageshrink;
 stretch: \the\pagestretch)}
```

at the start of each paragraph. It will show the values of the 4 parameters at the end of the preceding paragraph.

The `\message` commands will show the value of t growing from paragraph to paragraph, until a page is shipped out by the OTR. The value of d is usually 1.94444pt (the depth of many letters in cmr10) but is different when anything other than cmr10 is used. If the last line of a paragraph happens to contain letters without any depth, d will be zero at the end of that paragraph. Note that d can only be displayed while in vertical mode, between paragraphs (within paragraphs, `\showthe\pagedepth` only shows 0pt, not the depth of the last line of the previous paragraph). Each time another `\parskip` is inserted, between paragraphs, `\pageshrink` is incremented by `1pt`, and `\pagestretch`, by `2pt`.

Controlling the depth of the current page. It has already been mentioned that d is limited to `\maxdepth`. If a line of text with a depth>`\maxdepth` is moved to the current page, the depth of the current page (`\pagedepth`) is set to `\maxdepth`, and its height t is incremented by the difference (the baseline of the current page is lowered, and is now located below the baseline of the bottom line of text.)

Here is a simple experiment to clear up this point. First, set

```
\hsize=3.5in \vsize=2in
\tracingpages=1
```

and typeset some text in font cmr10. Most lines of text will have a height of $250/36 \approx 6.94444$pt and a depth of $70/36 \approx 1.94444$pt. The `\baselineskip` glue between the lines is thus set to 3.1112pt, to achieve a separation of 12pt between consecutive baselines. The last three `%` lines of TEX's tracing report should be:

```
% t=130.0 g=144.54 b=10000 p=0 c=100000#
% t=142.0 g=144.54 b=204 p=0 c=204#
% t=154.0 g=144.54 b=* p=0 c=*
```

This shows that t is incremented by 12pt between lines of text. The last line results in infinite cost, so the page is broken after the line with `c=204`.

Next, repeat the experiment with the depth of the second line increased to 7pt by placing an `\hbox{\vrule depth7pt}` in it. Typesetting the same material now results in different `%` lines:

```
% t=130.0 g=144.54 b=10000 p=0 c=100000#
% t=145.0 g=144.54 b=10 p=0 c=10#
% t=156.94444 g=144.54 b=* p=0 c=*
```

Fig. 2 shows the layout of the last three text lines in both cases. In 2*a*, the lines all have the same height and depth and are separated with the same size glue. In 2*b*, however, the situation is more complex. A `\baselineskip` of 3.1112pt is inserted between the first and second lines, so their baselines are separated, as usual, by 12pt. Since the second line has a depth of 7pt, the value of `\pagedepth` is set to `\maxdepth` (= 4pt), and the difference of `3pt` is added to the height t. The baseline of the entire page is thus lowered 3pt below the baseline of the second line. Normally, t would be set to $130 + 12 = 142$pt. Instead, its value now is 145pt.

Because of the large depth of the second line, it is separated from the third line by `\lineskip` [78], which has a `plain` format value of `1pt`. The baseline of the third line is set $4 + 1 + 6.94444 = 11.94444$pt below the baseline of the page, and t is incremented

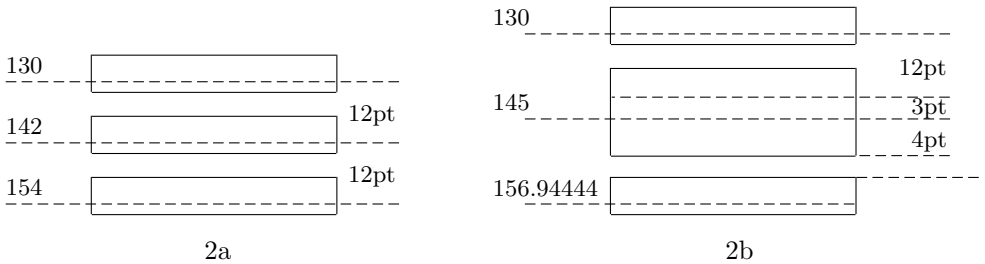


Figure 2

by that amount, to 156.94444pt. d is reset to the depth of the third line, namely 1.94444pt.

Again, appending the third line to the current page has resulted in infinite cost, and it is eventually returned to the recent contributions.

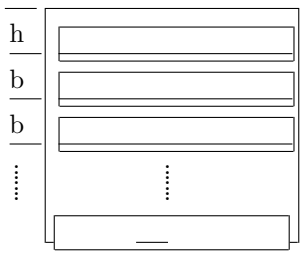
This is part of the overall task of the page builder while constructing the current page.

The height of a box of text. We denote the value of `\baselineskip` (normally 12pt) by b. A large `\vbox` with text consists mainly of lines of text, each an `\hbox`, separated by globs of glue, normally in the (varying) amounts necessary to separate baselines by exactly b, but sometimes just the amount `\lineskip`. We assume a simple case where no large characters or equations are used. In such a case, all lines of text are separated by b. The height of the box is thus

$$b(n-1) + \text{the height of the first line}$$

where n is the number of text lines.

The height of `\box255`. In the case of `\box255`, enough glue is placed above the first line of text to reach to `\topskip` from the first baseline. We denote the value of `\topskip` by h (10pt in `plain`). So if the baseline of the first line is now h below the top of the page, the height H of `\box255` should be $b(n-1)+h$ (Fig. 3). However, the height of `\box255` is always set, by the page builder, to `\vsize`. The difference between the two heights is usually supplied by the flexible glues on the page, the most common of which is `\parskip`.



The Height of a Page Box
Figure 3

Example: `\vsize=180pt` creates 15-line pages, since $12(15-1)+10 = 178$. The `\parskip` glues on the page are stretched by a total of `2pt`. Similarly, `\vsize=189pt` creates 15-line pages, but each page has to be stretched by `11pt`.

What if there isn't enough stretchability? In such a case, the bottom of the page remains empty. This is an unusual situation where the height of a box is greater than the sum of the vertical dimensions of its components. Normally such a case is considered an 'underfull box' but, in the case of the page builder and `\box255`, "underfull and overfull boxes are not reported when `\box255` is packaged for use by the OTR" [400].

A simple experiment is recommended, to clear up this point (note that this experiment uses `\output`, which hasn't been introduced yet, nevertheless it is a useful experiment to perform at this point.)

```
\vsize=189pt \parskip=0pt
\output={\setbox0=\vbox{\unvcopy255}
\message{[\the\ht0, \the\ht255;]}
\setbox1=\vbox to\vsize{\hrule width3in
 \vfil\hrule width3in} \wd1=0pt
\shipout\hbox{\box1 \box255}
 \advancepageno}

\vrule height10pt ⟨text to be typeset...⟩
```

The `\parskip` glue is now rigid, so there is no flexibility on the page at all. The messages will be `[178pt, 189pt;]`. `\box1` is set to two rules with a 'fil' in between, and is superimposed on `\box255`. A look at the typeset page will show that the top rule is placed exactly `10pt` above the baseline of the top line, and the bottom rule is well below the bottom line of text.

A better understanding of `\box255` is gained by trying

```
\setbox0=\vbox to\vsize{\unvbox255}
```

Even though both `\box255` and `\box0` have the same height namely, `\vsize`, they don't have the same status. `\box255` may be created underfull without an error message but, when transferred to `\box0`, the destination box becomes underfull, *with* an error message.

The most natural thing to do, in such a case, is to try to fill up `\box0` by saying `\setbox0=\vbox to\vsize{\unvbox255 \vfill}`. Surprise! This may, sometimes, cause an 'overfull box'. The explanation has to do with the depth of `\box0`. Without the `\vfill`, it is the depth of the bottom line. With the `\vfill`, it is zero, and the depth of the bottom line is added to the *height* of `\box0`, which may cause it to be overfull.

Examples of OTRs

The following sections show how to write an OTR, and illustrate typical OTRs for common applications. It should again be stressed that the examples are kept simple and, therefore, are not completely general. They should be read, understood and modified for specific needs, rather than copied and used verbatim.

An OTR is simply a sequence of TEX commands assigned to the token-register `\output`. Thus `\output={...}` would cause TEX to execute the commands `...` whenever it decides to invoke the OTR.

The simplest OTR is `\output={}`. When TEX sees this OTR, it substitutes the *default* OTR, which is: `\output={\shipout\box255}`. This default OTR is the simplest one which ships out a page.

`\shipout` is a the TEX primitive which creates a page in the dvi file. That page reflects the contents of whatever box follows the call to `\shipout`. An interesting feature is that `\shipout` can be invoked anytime, not just from an OTR. This is demonstrated in one of the examples below. Another interesting point is that `\shipout` can be redefined (which, of course, is true for any control sequence, whether a macro or a primitive.) One of the methods shown later, for double-column pages, does just that.

The OTR itself can be redefined during a TEX run. The new OTR will be used when the page builder next invokes the OTR. It is possible to define a macro such as `\def\newotr{...}` and assign `\output={\newotr}` at any time. This will redefine the OTR. It is even possible to write an OTR which redefines itself! Here is an example:

```
\output={
 \shipout\box255 \advancepageno
 \global\output={
   \shipout\vbox{
     \box255
     \bigskip
     \centerline{\folio}}
   \advancepageno}
 }
```

This OTR typesets the first page without a page number. It then (globally) redefines itself to typeset the rest of the document with the page number centered below the text. The reason for the `\global` is the local nature of the OTR, which is explained below.

It should be noted that the OTR is expected to empty `\box255`; it can ship it out, move it to some other box, or return it to the MVL. The latter is done simply by saying `\box255` or `\unvbox255` inside the OTR. An OTR which does not do anything with `\box255` will cause the error message "`unvoid \box255`."

The following OTR just empties `\box255`. `\output={\setbox0=\box255}`. This does not cause an immediate error message but is probably not what you want to do. It amounts to tossing away the entire document, page by page.

The next example is `\output={\uvbox255}`. This OTR always returns the page to the MVL, which will cause the page builder to immediately find a new page break and invoke the OTR again. The new page break, by the way, may not be the same as the original one, because of the penalty at the breakpoint. When the breakpoint is chosen, the page builder places the penalty found at the point in variable `\outputpenalty`, not in `\box255`. The OTR can return the penalty to its original place by saying `\unvbox255 \penalty\outputpenalty`. This guarantees that the page builder will find the same breakpoint.

An execution of the OTR which does not ship out anything is called a *dead cycle*. Dead cycles have their uses and are illustrated by some of the examples shown later. However, many consecutive dead cycles normally indicate an error. This is why TEX counts the number of consecutive dead cycles (in register `\deadcycles`) and stops the run if `\deadcycles` $\geq$ `\maxdeadcycles`. The `plain` format value of `\maxdeadcycles` is 25, and it can be changed at any time. Each time `\shipout` is invoked, it clears `\deadcycles`.

The Page Number. The page number can come from any source. Here is an example where the OTR typesets a page number, from a `\count` variable, centered below the printed area:

```
\newcount\pageNum
\output={
  \shipout\vbox{
    \box255\smallskip
    \centerline{\tenrm\the\pageNum}}
  \global\advance\pageNum by1}
```

Note the `\tenrm` in the preceding example. It is necessary because of the asynchronous nature of the OTR. When the OTR is invoked, TEX can be anywhere on the next page. Specifically, it could be inside a group where a different font is used. Without the `\tenrm`, that font (the current font) would be used in the OTR.

In the `plain` format, the `\count0` variable serves as the page number, and the following two macros are especially useful.

- `\folio` typesets `\count0` as the page number. However, if `\count0` is negative, `\folio` typesets a roman numeral.
- `\advancepageno` advances the page number by one. This is done by either incrementing or decrementing `\count0`, according to its sign.

Any \count variable can be used to typeset the page number. However, the main advantage of using \count0 is that TEX writes its value on the dvi file, so a page preview program can easily display the page number with the page. Stated more negatively, driver programs may *only* understand the values of \count0 written into the dvi file as page numbers and may not be able to selectively print pages whose numbers correspond to some other counter.

Actually, the ten variables \count0...\count9 are used as a (composite) page number. Any non-zero variable in this group is written on the dvi file. Typesetting and advancing any of them must be done explicitly by the user. Macros \folio, \advancepageno only handle \count0.

Grouping and the OTR. The OTR, as mentioned earlier, is a list of tokens. It is also implicitly surrounded by a pair of braces, making it into a group. This means that anything done inside the OTR is *local*, unless preceded by \global. This is a useful feature since, normally, operations within the OTR should be local (i.e. hidden from TEX's usual operations of making paragraphs and putting together math formulas).

Example: A 'Boxed' Page

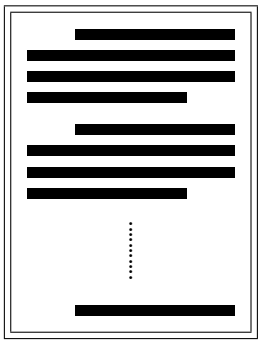
Here is an OTR for a 'boxed' page. It surrounds the page with double rules on all sides, and centers the page number below the double box. Note that the page shipped out is wider and taller than \box255. The value of \hsize in this case is, therefore, not the width of the final page shipped out, but the width of the text lines in \box255.

Macro \boxit typesets text and surrounds it with 4 rules (see [Ex. 21.3]). Parameter #2 is the space between the rules and the text. #1 is a box containing the text.

```
\def\boxit#1#2{%
  \vbox{\hrule
    \hbox{%
      \vrule \kern#2pt
      \vbox{\kern#2pt #1
        \kern#2pt}%
      \kern#2pt\vrule}
    \hrule}}

\output={
  \shipout\vbox{
    \boxit{\boxit{\box255}9}3
    \medskip
    \centerline{\tenrm\folio}}
  \advancepageno}
```

Figure 4 is an example of a small, doubly-boxed page.



A Boxed Page
Figure 4

Example: Header and Footer

This OTR typesets a header and a footer, both token lists supplied by the user. Typically one of them contains the page number.

```
\output={
  \shipout\vbox{
    \offinterlineskip
    \vbox to3pc{
      \line{\the\headline}
      \vss}
    \box255
    \vbox to3pc{
      \vss
      \line{\the\footline}}}
  \advancepageno}
```

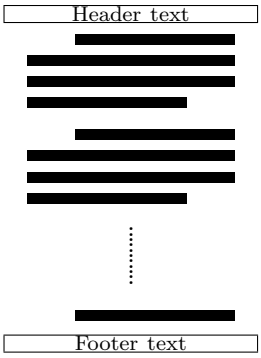
The headline and footline occupy 3pc each, enough for about 4 lines of text. If the user places more than 3pc worth of material in any of them, there will be no error message (because of the \vss), and the extra material will be typeset on top of the main body of the page. Both \headline and \footline are token lists, and may contain \if commands. Examples are

```
\headline={%
  \ifodd\pageno
    \line{\hfil\bf Header\hfil
      \llap{\tenrm\folio}}%
  \else
    \line{\rlap{\tenrm\folio}\hfil
      \bf Header\hfil}%
  \fi}

\footline={\it footer text\hfil}
```

The vertical size of the box shipped out (the printed page) is `6pc` plus `\ht255` (which is `\vsize`) plus `\dp255` (which is limited to `\maxdepth` and thus can be kept small).

Figure 5 is an example of such a page.



Header and Footer
Figure 5

Example: A Title Page for a Chapter

Sometimes, the book designer specifies a separate title page at the start of each chapter. Here is an example of a `\chapter` macro which typesets such a page *outside* the OTR. It starts with an `\eject`, to eject the last page of the previous chapter, then invokes `\shipout` to ship out a page with the chapter number and name. Note that, even though the page number isn't typeset, `\chapter` still has to advance it.

```
\def\chapter#1 #2;{%
  \vfill\eject
  \shipout\vbox to\vsize{
    \line{\bf Chapter\hfil#1}
    \vfil
    \vbox{\raggedcenter\bf#2}}
  \advancepageno}
```

There seem to be two problems with our simple macro:

1. If the first chapter starts on the first page of the document, our macro will eject a blank page before any text has been typeset.
2. If a page was ejected just before the new chapter started, our macro will eject a blank page.

It turns out that neither of these is a problem. An `\eject` is essentially a `\penalty-10000` and, if the very first thing in the document is a penalty, it gets discarded. Also, if the first thing on a new page is such a penalty, it gets discarded. Here is a relevant quote (from [114]) "If you say `\eject\eject`, the second `\eject` is ignored, because it is equivalent to `\penalty-10000` and penalties are discarded after a page break."

Example: Printer Registration Marks

We now turn to an OTR that optionally typesets the registration marks, also known as *crop marks*, which are used to align the pages for photography prior to printing and to indicate the size of the final page. The registration marks should be positioned at the 4 corners of the page, not at the corners of the printed area. The top left mark, e.g., should be located `1in+\voffset` above the top of the printed area, and `1in+\hoffset`, to the left. Similarly, the top right mark should be placed up and to the right, but by how much?

The problem is that TEX has no idea how wide and tall the paper is. All it knows is the left and top offsets, and the dimensions of the printed area (`\hsize` and `\vsize`). To place the registration marks properly, the user should specify the dimensions of the paper.

The document should thus start by specifying:

```
\newdimen\paperheight
\newdimen\paperwidth
\paperheight=..in \paperwidth=..in
```

It is also possible, although less desirable, to prompt the user to enter the two dimensions.

```
\newdimen\paperheight
\newdimen\paperwidth
\message{Enter paper height }
 \read-1to\tmp \paperheight=\tmp
\message{Enter paper width }
 \read-1to\tmp \paperwidth=\tmp
```

The next step is to create a `\vbox` of these dimensions, with the marks at the corners.

```
\newif\iffinalrun \finalruntrue
\newdimen\ruleht \ruleht=.5pt
\newdimen\gap \gap=2pt
\def\verrules{%
  \hbox to\paperwidth{%
    \vrule height1pc width\ruleht depth0pt
    \hfil \vrule width\ruleht depth0pt}}
\def\horrules{%
  \hbox to\paperwidth{%
    \llap{\vrule width1pc height\ruleht
      \kern\gap}
    \hfil
    \rlap{\kern\gap
      \vrule width1pc height\ruleht}}}
```

```
\newbox\rmarks \setbox\rmarks=
  \vbox to\paperheight{
    \offinterlineskip
    \vbox to0pt{\vss
      \verrules
      \kern\gap
      \horrules}
    \vfil
    \vbox to0pt{
      \horrules
      \kern\gap
      \verrules\vss}}
```

The dimensions of the box are then set to zero, so it will be superimposed on the printed page

```
\ht\rmarks=0pt \wd\rmarks=0pt
```

and the OTR typesets the box (which does not move the reference point), followed by the printed page. This causes the top left corner of the printed page to coincide with the top left registration mark. To center the printed page, it should be lowered and moved to the right.

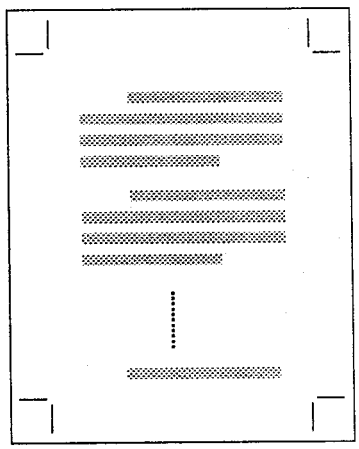
```
\newdimen\Mdown \Mdown=\paperheight
\advance\Mdown by-\vsize
\advance\Mdown by-6pc \divide\Mdown by 2
\newdimen\Mright \Mright=\paperwidth
\advance\Mright by-\hsize
\divide\Mright by 2
\output={
  \shipout\vbox{
    \offinterlineskip
    \iffinalrun
      \copy\rmarks
      \kern\Mdown
      \moveright\Mright
    \fi
    \vbox{
      \vbox to3pc{
        \line{\the\headline}\vss}
      \box255
      \vbox to3pc{
        \vss\line{\the\footline}}}}
  \advancepageno}
```

Sometimes the book designer specifies off-center pages. Even-numbered pages should be moved to the right and odd-numbered ones, to the left. This brings facing pages closer together, and leaves extra room on the outside margins.

```
...
\divide\Mright by 2
\newdimen\Mleft \Mleft=\Mright
\advance\Mright.5in \advance\Mleft-.5in
\output={
      ...
      \moveright
        \ifodd\pageno\Mleft
        \else\Mright\fi
        ...
```

This off-centering is only done on the final run.

Figure 6 shows a small page with registration marks.



Printer Registration Marks
Figure 6

The reader should now realize that three sets of dimensions should be specified when a book is designed and published. (1) The size of the sheet of paper which actually goes into the printer (specified by the human printer). After coming out of the printer, this sheet of paper is trimmed, at the crop marks, to (2) the size of the final page (`\paperheight` and `\paperwidth`). Finally, there is (3) the size of the printed area (`\hsize` and `\vsize`) on the page.

Example: A Border Around the Page

Here is an output routine which takes a 5" × 3" page and creates a border of size 5.5" × 3.5" around it. The border is done with `\leaders` (see [223] and [Ex. 21.8]). This example shows how easy it is to create a border out of a few characters. Ideally, 8 characters should be especially designed (Ref. 1), 4 for the four sides of the border, and 4 for the corner points. Their sizes should be chosen such that, e.g., the width of the border will be an integral multiple of the width of the top character. The reference point of each of the 8 should be placed such that they will align properly at the corners

1. \box2 is set to a \vbox containing the border.

```
\font\bord=cmsy10
\def\topp{\bord\char'176}
\def\bott{\bord\char'020}
\def\lft{\bord\char'032}
\def\rt{\bord\char'033}
\setbox2=\vbox to3.5in{
  \offinterlineskip
  \hbox to5.5in{\leaders\hbox{\topp}\hfill}
  \hbox to5.5in{%
    \leaders
    \vbox to3.4in{
      \leaders
      \hbox to5.5in{\lft\hfil\rt}
      \vfil}%
    \hfil}
  \hbox to5.5in{\leaders\hbox{\bott}\hfill}
  \vss}
```

2. All the dimensions of \box2 are set to 0 by

```
\wd2=0pt \ht2=0pt \dp2=0pt
```

3. The output routine now ships out \copy2 (which does not move the reference point), followed by \box255.

```
\output={
  \shipout\vbox{
    \copy2 \vskip.25in
    \moveright.25in\box255}
  \advancepageno}
```

Example: Mailing Labels

This example introduces the concepts of logical and physical pages. The material placed in \box255 constitutes a *logical page*. The material actually shipped out by the OTR is a *physical page*. Usually one physical page is shipped for each logical page generated. In this example, since the mailing labels are small, several mailing labels are combined and shipped together as one physical page.

The data for the labels is assumed to reside on an external file, named labels, which contains information with format:

\name ⟨*the name*⟩\\
\address ⟨*several lines of address*⟩\\

The mailing labels are 3.5in wide and 1.5in tall each. There are 4 labels arranged vertically on a 6in tall page, without any gaps in between. The data file is \input and macros \name, \address invoked automatically. Three approaches are described:

Approach 1. Macros \name and \address typeset the name and address in the desired format. The value of \vsize is set to the size of 4 labels. The OTR is thus invoked with a logical page consisting of 4 labels, and it simply ships it out, as one physical page.

```
\nopagenumbers \vsize=6in \hsize=3.5in
\def\name#1\\{\nointerlineskip
  \vbox to.25in{#1 \vfil}}
\def\address#1\\{\nointerlineskip
 \vbox to1.25in{\kern.1in#1 \vfil}\penalty0}
\output={\shipout\box255}
\obeylines\parindent=0pt
\input labels
\bye
```

The \obeylines guarantees that each line in file labels will become a typeset line on the final page. The \nointerlineskip suppresses the normal interline glue that would otherwise be inserted between the boxes in the MVL. The \penalty0 is necessary to supply a valid page breakpoint. The reader will recall that a page can only be broken at a glue, a kern, or a penalty [110]. Without the \penalty0, the page builder would have no place to break the page, and would place the entire document, as one page, in \box255 at the end of the job.

Approach 2. Macro \name typesets the name in a \vbox. Macro \address typesets the address in another \vbox. \vsize is set to 1.5in, the size of one label. The OTR is thus invoked for each label and receives, in \box255, a logical page consisting of one label. It collects 4 such pages and ships them out as one physical page. This approach distinguishes between a logical and a physical page. Note also that, 3 out of 4 times, this OTR goes through a dead cycle.

```
\nopagenumbers \vsize=1.5in \hsize=3.5in
\def\name#1\\{\vbox to.25in{#1 \vfil}}
\def\address#1\\{\nointerlineskip
 \vbox to1.25in{\kern.1in#1 \vfil}}

\newcount\four \newbox\physpage
\output={
  \global\setbox\physpage=
    \vbox{
      \unvbox\physpage
      \box255}
  \global\advance\four by1
  \ifnum\four=4
    \shipout\box\physpage
    \global\four=0
  \fi}

\obeylines\parindent=0pt
```

```
\input labels
\bye
```

The serious reader should try to understand what happens at the end of the document. Suppose we have 6 labels. The first 4 will be printed on the first page, and the last 2 will be accumulated, in `\box\physpage`, for the second page. When `\bye` is found, TEX finds out that `\deadcycles`$\neq 0$. It goes into its 'endgame' [264], where it prepares an empty page and invokes the OTR. This is repeated twice, accumulating two empty logical pages in `\box\physpage`, and then the OTR executes a `\shipout`, which clears `\deadcycles`, thereby stopping the 'endgame'.

Approach 3. Macro `\name` typesets the name in a `\vbox`. Macro `\address` appends the address to the same box, sets its height to `1.5in`, and invokes the OTR. The OTR again collects 4 such boxes and ships them out as one physical page. This approach is interesting since it makes minimum use of the MVL, the page builder, and `\box255`. It does not use `\vsize` which, consequently, can be set to any value.

```
\nopagenumbers \vsize=0in \hsize=3.5in
\def\name#1\\{\strut\setbox0=\vbox{#1}}
\def\address#1\\{%
  \setbox0=\vbox to1.5in{
    \unvbox0
    \kern.1in
    #1 \vfil}
  \eject}
\newcount\four \newbox\physpage \newbox\toss
\output={
  \setbox\toss=\box255
  \global\setbox\physpage=\vbox{
    \unvbox\physpage\box0}
  \global\advance\four by1
  \ifnum\four=4
    \shipout\box\physpage
    \global\four=0
  \fi}
\obeylines\parindent=0pt
\input labels
\bye
```

We want macro `\address` to invoke the OTR. However, the only part of TEX which invokes the OTR is the page builder. It does that when it calculates a page break with infinite cost, or when it moves a penalty≤ -10000 to the current page. Macro `\address` thus says `\eject`, which is essentially a `\penalty-10000`. The penalty is placed in the recent contributions and the page builder is exercised. The page builder tries to move the penalty to the current page but, since the current page is empty, the penalty is discarded [112]. The result is that the OTR is never invoked, and the job terminates without shipping out any pages.

To avoid this situation, macro `\name` typesets a strut. The strut is moved to the current page and, since the current page is no longer empty, the page builder agrees to move the `\penalty-10000` to the current page. This causes the page builder to break the page, place the current page (just the strut) in `\box255`, and invoke the OTR. The OTR does not need `\box255` and simply empties it.

It is possible, of course, to use any text instead of the strut.

The `\vsplit` Operation. It is important, when working with OTRs, to fully understand the `\vsplit` operation. Its syntax is: `\vsplit`⟨*box number*⟩ `to` ⟨*dimen*⟩, and the result is a box. Most often it appears in an assignment such as: `\setbox1=\vsplit0 to2.6in`. This sets `\box1` to a height of `2.6in`, moves material from the top of `\box0` to `\box1`, and keeps the remainder in `\box0`.

TEX assumes that the new `\box1` may have to be shipped out as part of the page. It therefore places a glue similar to h at the top of `\box1`. This glue is called `\splittopskip` and has a `plain` format value of `10pt` [348].

The most important thing to keep in mind is that a box can only be split *between* lines of text. If we perform a `\vsplit` to an 'inconvenient' size, `\box1` will come out underfull.

Example: `\vsize=375pt`. This creates, in `\box255`, a page of 31 lines (since $12(31-1)+10 = 370$). The assignment `\setbox0=\vsplit255 to 184pt` will set `\box0` to a height of `184pt` with 15 lines of text. Since 15 lines of text occupy `178pt`, the remaining `6pt` should be filled with some flexible glue. If there is not enough flexible glue, `\box0` will come out underfull. If there are, e.g., two paragraphs in `\box0`, then its total stretchability is `2pt`. TEX will stretch it by `6pt` and will report an underfull box with a glue set ratio of 3 (300%). The remaining 16 lines in `\box255` will now occupy $12(16-1)+10 = 190$`pt`. Any flexible glues in `\box255` will return to their natural size.

Here is an OTR which splits the page, ships out the top part and returns the rest to the MVL (actually, to the recent contributions):

```
output={\setbox0=\vsplit255 to1in
 \shipout\box0 \unvbox255}
```

Splitting a Box in Two

Imagine a box with n lines of text, and with no flexible vertical glues. How can it be split in two equal parts? If n is even, this is easy. However, if n is odd, it is impossible. If H is the height of the box then $H = b(n-1)+h$. Our approach is to split the box such that the top part will have $m = \lceil n/2 \rceil$ lines, and its height will thus be $H' = b(m-1)+h$. To calculate H' we start with:

$$m = \lceil n/2 \rceil = \begin{cases} n/2, & n \text{ even;} \\ \lfloor n/2 \rfloor + 1, & n \text{ odd;} \end{cases}.$$

Since TEX always performs an integer by integer division, the case where n is even is simple. It satisfies $m = n/2$ or $m - 1 = \frac{n-2}{2}$ and, therefore,

$$\begin{aligned} H' = b(m-1)+h &= \frac{b(n-2)}{2} + h \\ &= \frac{b(n-1)+h+h-b}{2} \\ &= \frac{H+h-b}{2}. \end{aligned}$$

We therefore split to $H + h - b$. For even n, this is ideal. For odd n, the top part will contain one less line than the bottom. Since we want the top part, in such a case, to be larger, we loop, increasing the split size slightly, until the top part becomes larger than the bottom one.

```
\halfsize=\ht0
\advance\halfsize by\topskip
\advance\halfsize by-\baselineskip
\divide\halfsize by 2
\splittopskip=\topskip
{\vbadness=10000
 \loop
   \global\setbox3=\copy0
   \global\setbox1=\vsplit3 to\halfsize
 \ifdim\ht3>\halfsize
   \global\advance\halfsize by1pt
 \repeat}
```

This method is used on [417] to implement double-column pages (see below). It can also serve to split a box into k equal parts [397]*. The principle is to first split the box to size $\lceil n/k \rceil$, then to split the rest recursively.

Next, consider the case where the original box contains flexible vertical glues. This considerably simplifies the problem, since the box can now be split to parts of almost any size and the glues will be flexed to fill up all the individual parts, eliminating any over/underfull boxes.

Example: Double-Column Pages

Multi-column pages are common in newspapers and technical publications. Note that, when using narrow columns, the tolerance usually has to be increased, leading to low quality results (see the experiments in [Ch. 6]).

Three approaches are described. The first one treats the two columns as separate logical pages, which are combined, in the OTR, into one physical page. This approach is described on [257], and is shown here for the sake of completeness. See also [Ex. 23.4] for a generalization of this case to three columns.

The second approach [417] creates one long and narrow column, and breaks it into two equal parts, which are typeset side by side. The problem of splitting a box into two equal parts has been discussed earlier.

The third approach uses the technique of the second approach to make it possible to switch between one- and two-columns at will.

Approach 1. The quantity `\lr` is a new control sequence, which is defined and redefined by the OTR depending on the current column (left or right). When the OTR is invoked for the first time (or the third, fifth,... times) it saves `\box255` (containing the left column) in another box. These are dead cycles. When it is invoked for the second time (or the fourth,...times), it ships out both columns.

```
\hsize=3.2in
\newdimen\fullhsize \fullhsize=6.5in
\let\lr=L \newbox\leftcolumn
\output={
  \if L\lr
    \global\setbox\leftcolumn=\box255
    \global\let\lr=R
  \else
    \shipout\hbox to\fullhsize{%
      \box\leftcolumn\hfil\box255}
    \global\let\lr=L
    \advancepageno
  \fi}
```

Again, it is important to understand what happens with the last page. If the last page ends somewhere in the left column, the `\bye` invokes the OTR, which saves the left column and returns without shipping out anything (a dead cycle). Since `\deadcycles` is now non-zero, TEX places an empty page in `\box255` and invokes the OTR again. This

* It seems that the inequality shown there is wrong.

time it ships a full page, so `\deadcycles` is reset. The last page comes out with unbalanced columns.

It is easy to add a headline and footline to this example:

```
\vsize=2in \hsize=3.2in
\newdimen\fullhsize \fullhsize=6.5in
\let\lr=L \newbox\leftcolumn
\def\fulline{\hbox to\fullhsize}
\output={
  \if L\lr
    \global\setbox\leftcolumn=\box255
    \global\let\lr=R
  \else
    \shipout\vbox{
      \vbox to3pc{%
        \fulline{\the\headline}\vss}
        \fulline{\box\leftcolumn\hfil\box255}
        \vbox to3pc{
          \vss\fulline{\the\footline}}}
    \global\let\lr=L
    \advancepageno
  \fi}

\headline{\hfil\bf A Header\hfil}
\footline{\hfil\tenrm\folio\hfil}
```

Approach 2. The OTR is invoked with a long and narrow column. It splits it into 2 equal parts (see earlier discussion) and ships out a page consisting of the two parts, laid side by side. The original value of `\hsize` is saved in `\ohsize`. `\hsize` is then set to the width of a single column. `\vsize` should be set to twice its original value.

```
1. \newdimen\ohsize \ohsize=\hsize
2. \hsize=0.5\hsize \advance\hsize -.1in
3. \newdimen\halfsize
4. \output={
5.   \setbox0=\vbox{\unvbox255}
6.   \halfsize=\ht0
7.   \advance\halfsize by\topskip
8.   \advance\halfsize by-\baselineskip
9.   \divide\halfsize by 2
10.  \splittopskip=\topskip
11.  {\vbadness=10000
12.   \loop
13.     \global\setbox3=\copy0
14.     \global\setbox1=\vsplit3 to\halfsize
15.   \ifdim\ht3>\halfsize
16.     \global\advance\halfsize by1pt
17.   \repeat}
18.  \shipout\hbox to\ohsize{\box1 \hfil\box3}
19.  \advancepageno}
```

On line 5, `\box255` is unveiled. This is necessary since `\ht255` equals `\vsize` but we want to start with a box of height $b(n-1)+h$. Also, for the last page, `\box255` may have a large `\vfil` at the bottom, which should be removed before the split. The right value for the split is calculated on 6–9. If the number of lines, n, is even, the split produces two equal parts. For odd n, the loop on lines 12–17 keeps incrementing the left column until it becomes one line larger than the right one. Both halves are shipped out, on line 18, side by side.

If the document contains just text, without equations or figures, the following can be used to calculate the best value of `\vsize`.

```
\newcount\col
\message{Enter number of lines per page: }
\read-1to\ent \col=\ent
\advance\col by-1 \multiply\col by12
\advance\col by10
\vsize=\col pt
```

It is easy to add a header and footer to the final page.

Exercise: Do it!

Approach 3. Switching between single- and double-columns. The principles of this method are described here, along with macros taken from the `manmac` [417]. They are the macros used to typeset the index of the TEXbook, which is why they use values such as `14pc` and `89pc`. The macros are easy to modify for different formats. The reader should also consult reference 2 for two corrections of the macros.

```
\newdimen\pagewidth  \pagewidth=\hsize
\output{\shipout\box255}

\newbox\partialpage
\def\begindoublecolumns{\begingroup
  \output={\global\setbox\partialpage=
    \vbox{\unvbox255\bigskip}}
  \eject
  \output={\doublecolumnout}
  \hsize=14pc \vsize=89pc}
\def\enddoublecolumns{%
  \output={\balancecolumns}\eject
  \endgroup \pagegoal=\vsize}

\def\doublecolumnout
  {\splittopskip=\topskip
  \splitmaxdepth=\maxdepth
  \dimen0=44pc
  \advance\dimen0 by-\ht\partialpage
  \setbox0=\vsplit255 to\dimen0
```

```
  \setbox2=\vsplit255 to\dimen0
  \onepageout\pagesofar
  \unvbox255 \penalty\outputpenalty}
\def\pagesofar{\unvbox\partialpage
  \wd0=\hsize \wd2=\hsize
  \hbox to\pagewidth{\box0\hfil\box2}}
\def\balancecolumns
 {\setbox0=\vbox{\unvbox255}
  \dimen0=\ht0
  \advance\dimen0 by\topskip
  \advance\dimen0 by-\baselineskip
  \divide\dimen0 by2 \splittopskip=\topskip
  {\vbadness=10000
   \loop
     \global\setbox3=\copy0
     \global\setbox1=\vsplit3 to\dimen0
   \ifdim\ht3>\dimen0
     \global\advance\dimen0 by1pt
   \repeat}
  \setbox0=\vbox to\dimen0{\unvbox1}
  \setbox2=\vbox to\dimen0{\unvbox3}
  \pagesofar}
```

Macro `\begindoublecolumns` starts by defining an OTR which saves the page so far (single-column) in `\box\partialpage`. The `\eject` invokes this OTR. The OTR is then redefined to do double-column by splitting a long, narrow, `\box255`.

Macro `\enddoublecolumns` again redefines the OTR to split the page-so-far in two, and typeset the two halves, side by side, below the single-column in `\box\partialpage`.

Example: Facing figures

When two figures are textually related, the user may want them typeset on facing pages. The first figure should be typeset on top of the next even-numbered page and the second one, on top of the following page.

A macro `\facefig#1#2` is defined, with 2 parameters, the heights of the 2 figures. It saves the two values in `\dimen` variables `\figa` and `\figb`. The OTR checks the two variables. If the current page number is even and `\figa` > 0, room is reserved on top of the current page for the first figure by placing an empty box on the MVL, followed by `\box255`. If the current page number is odd, `\figa` $= 0$ and `\figb` > 0, the OTR reserves room for the second figure in a similar way. In either case the OTR goes through a deadcycle.

```
\newdimen\figa \newdimen\figb \newif\ifdead
\def\facefig#1#2{\figa=#1 \figb=#2}
\output={
  \deadfalse
  \ifodd\pageno
    \ifdim\figa=0pt \ifdim\figb>0pt \message{b}
      \vbox to\figb{}\unvbox255
      \penalty\outputpenalty
      \global\figb=0pt \deadcycles=0 \deadtrue
    \fi\fi
  \else
    \ifdim\figa>0pt \message{a}
      \vbox to\figa{}\unvbox255
      \penalty\outputpenalty
      \global\figa=0pt \deadcycles=0 \deadtrue
    \fi
  \fi
  \ifdead\else
    \shipout\box255
    \advancepageno
  \fi}
```

The simple macros above only use one pair of variables to save the heights of the figures. In practice, the user may expand `\facefig` before the OTR has handled the previous pair of figures. This will place new values in our variables before the old ones have been used. Our macros should, therefore, be extended so that any number of pairs of heights can be saved. The macros below save such pairs, as `\kern` values, in a `\vbox`.

```
\newbox\save
\newdimen\figa \newdimen\figb
\newif\ifdone \donetrue \newif\ifdead
\def\facefig#1#2{%
  \setbox\save=\vbox
    {\kern#1 \kern#2 \unvbox\save}}
\output={
  \deadfalse
  \ifvoid\save\else
    \ifdone
    \global\setbox\save=
      \vbox{\unvbox\save
        \global\figb=\lastkern \unkern
        \global\figa=\lastkern \unkern}
    \global\donefalse
  \fi\fi
  \ifdone\else
    \ifodd\pageno
      \ifdim\figa=0pt \ifdim\figb>0pt
        \message{b=\the\figb;}
        \vbox to\figb{}
        \unvbox255
        \penalty\outputpenalty
        \global\figb=0pt \deadcycles=0
        \deadtrue \global\donetrue
       \fi\fi
    \else
```

```
      \ifdim\figa>0pt \message{a=\the\figa;}
        \vbox to\figa{}
        \unvbox255
        \penalty\outputpenalty
        \global\figa=0pt
        \deadcycles=0 \deadtrue
      \fi
  \fi\fi
  \ifdead\else
    \shipout\box255
    \advancepageno
  \fi}
```

Another boolean variable, `\ifdone`, is declared. It is set to 'false' when two values are extracted from `\box\save`, and to 'true', when the two figures have been typeset. As long as it is 'false', we are in the process of typesetting two facing figures, and no new values are extracted.

The following extensions are left as an **Exercise:**

1. Macro `\facefig` should check to make sure none of its parameters exceeds `\vsize`.

2. `\facefig` should also accept the captions of the two figures, as additional parameters, and save them. The OTR should later retrieve and typeset the captions below (or above) the reserved areas.

Note! Our macros do not use insertions and are therefore incompatible with `\midinsert` and its relatives. Using both `\midinsert` and `\facefig`, the figures would be inserted in an unpredictable order.

Example: Shipping-Out Pages Selectively

The following code, part of the `manmac` format, can be used to produce only a subset of pages. The numbers of the desired pages should be placed on separate lines in a file called `pages.tex`.

The first line saves `\shipout`, which is a primitive, in macro `\Shipout`. Later, `\shipout` is redefined as either `\Shipout` or `\Tosspage`.

```
\let\Shipout=\shipout
\newread\pages \newcount\nextpage
\openin\pages=pages
\def\getnextpage{%
  \ifeof\pages\else
    {\endlinechar=-1\read\pages to\next
     \ifx\next\empty % we should have eof now
     \else\global\nextpage=\next\fi}%
  \fi}
\ifeof\pages\else\message
 {OK, I'll ship only the requested pages!}
 \getnextpage\fi
\def\shipout{%
  \ifeof\pages\let\next=\Shipout
  \else\ifnum\pageno=\nextpage
    \getnextpage
    \let\next=\Shipout
  \else\let\next=\Tosspage\fi\fi
  \next}

\newbox\garbage
\def\Tosspage{\deadcycles=0\setbox\garbage=}
```

Detecting the End of the Document

How can the OTR find out if the page it has been given is the last one? The easiest way is to detect the `\vfill` at the bottom of that page. This can be done by:

```
\def\vfill{\vskip 1sp plus 1fill}
\output={
  \setbox0=\vbox to\vsize{
    \unvcopy255
    \ifdim\lastskip>0pt \message{last page}\fi}
  \shipout\box255
  \advancepageno}
```

(The `\lastskip` command is explained in part II.) This usually works but may fail in cases where the last page is full, or almost full. An example is `\vsize=1in`, which leaves room for about six lines of text on the page. Let's assume that the document has text for 6 lines, and the last line contains a deep character, such as a ']', whose depth is `2.5pt`. Setting `\tracingpages=1` generates the following in the log file:

```
%% goal height=72.26999, max depth=4.0
% t=10.0 g=72.26999 b=10000 p=0 c=100000#
% t=22.0 plus 1.0 g=72.26999 b=10000 p=150 c=100000#
% t=34.0 plus 1.0 g=72.26999 b=10000 p=100 c=100000#
% t=46.0 plus 1.0 g=72.26999 b=10000 p=100 c=100000#
% t=58.0 plus 1.0 g=72.26999 b=10000 p=150 c=100000#
% t=70.0 plus 1.0 g=72.26999 b=1168 p=0 c=1168#
% t=72.5 plus 1.0 plus 1.0fill g=72.26999 b=* p=-20000 c=*
```

The cost of breaking the page after the first six lines is 1168, very low. The page builder, however, waits for a point with infinite cost before it decides on a page break. It continues reading the source and finds the `\bye`. The definition of `\bye` is [357] `\par\vfill\penalty-20000\end`. The page builder adds the `\vfill` to the current page, causing the depth of the current page to become zero. The depth of the bottom line (`2.5pt`) is now added to the height of the current page, with the result that it is too high (`72.5pt`). The `\vfill` is therefore removed, and the last page is shipped out without any fill at the bottom.

References

1. Knuth, D. E., *A Course on* METAFONT *Programming*, TUGboat 5(2)105–118, Nov. 1984.
2. Platt, C., *Macros for Two-Column Format*, TUGboat (6)1,29–30, March, 1985.)

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