Over the multi-column

Péter Huszár

With a good multi-column environment you can handle *almost* every problem you are faced with. But sometimes you may want to produce more complicated pages: text around a picture or real newspaper pages which are sets of articles (boxes) rather than sequences of columns. The environment presented in this article gives you a convenient way to describe such pages.

The idea

When you use plain TEX, you needn't care about page-setting. TEX has a powerful algorithm for making lines into a single-column page. This format is so simple (even if it depends on a dozen or so parameters) that such a page can be produced by a constant output routine provided in `plain`. Thus you only have to type your text and TEX builds up this format. However, if you want to produce a more complex page, first you should describe its structure, i.e., how many parts does a page contain and how are they connected logically and physically. For example, a multi-column page consists of as many parts as the number of columns; there is an order on the parts (the sequence of the columns); and they are put next to each other. But this is still a restricted form of a general page which looks as follows:

A page is built up of logical areas. I will use the phrase 'logical area' for the logical units of the page (articles, pictures, etc.). Each area is a list of boxes (you may want to divide an article in two or more parts). For example let's consider the page of Figure 1.

There are six areas on the page:

- the TITLE
- a PICTURE
- article 1, which consists of three boxes: BOX 1.1, BOX 1.2 and BOX 1.3
- article 2, which consists of just one box: BOX 2.1
- article 3, which consists of two boxes: BOX 3.1 and BOX 3.2
- an ADVERTISEMENT

As you can see it is rather difficult to describe this page in a multi-column format which is just one area with restricted placement of its boxes on the page. The main problem is that there is no proper ordering on the boxes (if you represent each box as a point then this is the same problem as ordering on complex numbers), so you can't make one list from the boxes. I will now give an environment to describe such a page.

How to describe (plan) the page

The strategy of planning is that first we describe the areas and the boxes on the page, then we 'fill' the boxes with their contents (text, picture, etc.). This means that the dimensions (width, height and depth) of a box are independent from its contents, contrary to TEX's boxes where you can prescribe only one of the three dimensions and the other two will be known just after putting the contents into them. (The problem of how much space a text needs is rather general, I guess.)

The description part of the macro package should allow you

1. to describe as many areas as you want to;
2. to describe the list of boxes of each area;
3. to specify vertical and horizontal sizes of each box;
4. to specify the position of each box on the page.

First let's examine these problems from the point of syntax. You should specify boxes and areas as a list of boxes. An element in the list has a data part and a pointer to the next element; this pointer points to 'null' (i.e. nowhere) at the end of the list. It means each box (I'll use the name `planbox` for these boxes to differentiate them from TEX's boxes) should have the following parameters:

a. a name (we want to handle it as an ordinary allocated `\box`),

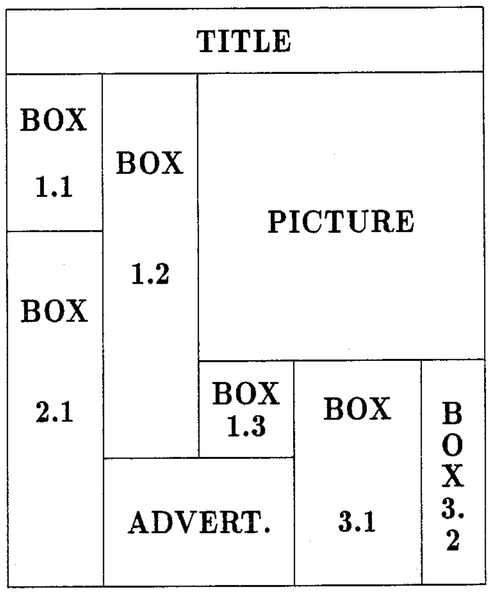


Figure 1.

b. horizontal and vertical sizes (width and depth),
c. position on the page specified as horizontal and vertical distance of the left upper corner of the box from the left upper corner of the page,
d. a pointer to the next planbox (the best way is to give the name of the next planbox);

in other words, something like this:

```
<name>,<width>,<depth>,<hdis>,<vdis>,<next>;
```

where `<next>` is either a `<name>` or the constant `\null`. The semicolon is redundant but you can read the source code easier with it. In this format we can describe the example page above with the `\plan` macro as follows (the parameter of `\plan` is the total number of planboxes on the page):

```
\plan 9; % total number of planboxes
\TITLE     ,\hsize, 2cm, 0pt, 0pt,\null;
\ArtIone   ,   3cm,10cm, 0pt, 2cm,\ArtItwo;
\ArtItwo   ,   3cm,13cm, 3cm, 2cm,\ArtIthree;
\ArtIthree,    4cm, 2cm, 6cm,13cm,\null;
\ArtIIone  ,   3cm, 6cm, 0pt,12cm,\null;
\ArtIIIone,    4cm, 5cm,10cm,13cm,\ArtIIItwo;
\ArtIIItwo,    2cm, 5cm,14cm,13cm,\null;
\AD        ,   7cm, 3cm, 3cm,15cm,\null;
\PICTURE   ,  10cm,11cm, 6cm, 2cm,\null;
```

This format also specifies the areas. Each area starts after the end of the previous area, i.e., after a `\null` pointer. You'll see below that this information is enough to handle the areas.

Until now I've written about whole pages, but you aren't restricted to plan the whole page every time. If you plan just a part of the page, the origin for the positions of planboxes is the left upper corner of the planned part (i.e., relative positions, so you can shift the plan on the page without changing them) and the planned part will be put at the current position when `\bpage` (see below) is performed.

Filling the page

The plan is ready, TeX knows the structure of the page (or the planned part of it), and we can start putting the text into planboxes. The procedure my macro offers you is the following:

- You should choose an area you want to deal with.
- You can type in your text.
- At any point you are allowed to choose another area.
- At any point you are allowed to switch to the next box within the area (like an `\eject`).
- If you don't want to switch by hand, the macro automatically switches to the next box when the current one is full. After the last box in the area it gives you a warning.

Let's consider the actions step by step.

To start the page (the planned part) and to select an area you simply type:

```
\bpage<area>;
```

where `<area>` is the `<name>` of the first planbox in an area. (You can choose not just the first but any box in the area; however, you can't switch back to the previous boxes.) The chosen planbox becomes an individual page (with its own width as `\hsize` and height as `\vsize`). This *is* a page from the view of the algorithm but it hasn't got `\headline`, `\footline`, `\footnote` or floating insertions.

Now you can type your text for this area. Having finished, you can choose another area by saying:

```
\nextarea<area>;
```

It is quite simple, isn't it? You can fill the areas one by one in any order you want (there is no restriction). Indeed you can go back to a previous area but if you do so the previous 'value' of the area will be overridden.

Inside an area you are allowed to switch to the next box with the command:

```
\nextbox;
```

There is no need to specify the next box by giving its name. With the pointer technique the macro figures it out.

At `<nextbox>` and `<nextarea>` you have a choice: you can put a `\vfill` at the end of the current planbox with `\fillON` or you can omit it with `\fillOFF`. The macro package sets `\fillOFF` at the beginning of every box.

At the end simply say:

```
\epage
```

to finish the page. All four commands can be used immediately after a paragraph, but not inside one!

Hyphenation. The macro package tries to avoid any hyphenation (for the reasons see below). But sometimes (especially in narrow planboxes) it gives very poor output (underfull hboxes). You can enable hyphenations with `\hyphensON`. However, this way the package may produce hyphenations in the midst of some lines. You can correct mistake by saying `\hbox{word}` to enclose the hyphenated word.

Automatic switch

The main advantage of the macro package is the automatic switch. If a planbox is full of text the macro package automatically switches to the next box in the area. With this feature you can e.g. have text to flow around a picture (see An example below), or plan a page like the one above.

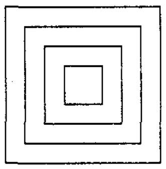
The default way to fill the page is the automatic switch. You can disable it with `\automaticOFF` and enable it again with `\automaticON`.

If there are no more boxes in an area the macro package produces an error message (a warning), and calculates and writes to the logfile the space which the rest of the text needs.

An example

This is a very simple example but it can be used often in everyday TEXing. The problem is a picture which is in the middle of the page and which is narrower or wider than a column or the page:

Here is the text above a picture of width (5pc) less than one third of the column's (18.75pc):



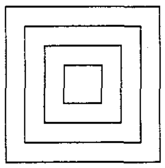
and the text continues here below the picture. The empty space around the picture is about 6.25pc×13pc. With this environment you can have the text flow around the picture. The parameters are as above (`picture height=6.1pc`):

```
\let\bs=\baselineskip
\let\hs=\hsize \let\vs=\vsize
\newdimen\pwd   \pwd=6.25pc  % picturewd
\newdimen\pht   \pht=6.1pc   % pictureht
\newdimen\ptop  \ptop=2\bs
\newdimen\pbot  \pbot=\ptop
         \advance\pbot \pht
\def\boxit#1{\vbox{\hrule\hbox{\vrule
    \kern7pt\vbox{\kern7pt#1\kern7pt}%
    \kern7pt\vrule}\hrule}}

\plan 4;
  \above,  \hs, \ptop,  0pt,   0pt,  \near;
  \near , 11pc,  \pht,  0pt, \ptop, \below;
  \below,  \hs,  2\bs,  0pt, \pbot,  \null;
  \pict , \pwd,  \pht, 12pc, \ptop,  \null;
Just a little text outside of the plan to
show the possibility of planning a part
of the page.
\bpage\above;
\automaticON \hyphensON
Here are two lines of text above the
picture. When these two lines are full
the text continues at the side of the
picture. The picture is placed on the
right side of the column as you can see.
When this box is full of text (somewhere
here at this point) the text flows
automatically to the next box which
is actually a little space below the
picture \fillON for the rest of the
paragraph.

\nextarea\pict;
\fillON
\leftline{}\nointerlineskip
\vfill
\centerline{%
\boxit{\boxit{\boxit{\boxit{}}}}%
}

\epage
And now we're outside of the planned part
again.
```

This code yields (notice that the whole plan is in the middle of a multicolumn environment):

Just a little text outside of the plan to show the possibility of planning a part of the page.

Here are two lines of text above the picture. When these two lines are full the text continues at the side of the picture. The picture is placed on the right side of the column as you can see. When this box is full of text (somewhere here at this point) the text flows automatically to the next box which is actually a little space below the picture for the rest of the paragraph.



And now we're outside of the planned part again.

TEXnicalities

The rest of the paper looks behind the screens and explains how this package works.

Planning the page

When the user invokes `\plan` the package should store all the information about the structure of the page. This includes the attributes of the planboxes and some supplementary information used by the package. The attributes require registers for their values while the system information supplies some notes on these registers. First of all, `\plan` stores the number of planboxes on the page in `\planb@xno` and the current insertion number in `\maxplanb@xno` (the reason for this will be explained later).

```
\newcount\planb@xno
\newcount\maxplanb@xno
\def\plan#1; {\bgroup \global\planb@xno #1
    \global\maxplanb@xno\insc@unt
    \makeplanb@x}
```

Information about each planbox is stored by the recursive macro `\makeplanb@x`. This is done in two steps: first allocating the necessary registers (`\planboxall@c`), then setting their values (`\setsiz@s`):

```
% creating a planbox
% #1 name, #2 width, #3 depth
% #4 hdis, #5 vdis,  #6 next
\def\makeplanb@x#1,#2,#3,#4,#5,#6; {%
    \planboxall@c#1#6
    \setsiz@s#2,#3,#4,#5;
    \advance\planb@xno \m@ne
    \ifnum\planb@xno>\z@
        \let\next\makeplanb@x
    \else \let\next\pl@nrest \fi \next}
```

Allocation. We need six registers for a planbox: a box register named `<name>` for the contents; four dimension registers for `<width>`, `<depth>`, `<hdis>` and `<vdis>`; and a token list register for `<next>` which contains the `<name>` of the next planbox. The package should figure out from the `<name>` the other registers for this planbox. Using an insertion, we get just a box register, but no token list register and instead of four just one dimension register. The token list register can be 'appended' to the insertion but we have to allocate the dimension registers separately from the other registers and use a 'pointer' to these dimensions. This pointer comes from the count register of the insertion. It always points to the highest of the four consecutive dimension registers. The first version of this would look like this:

```
\def\planboxall@c#1#2{%
    \newdimen#1    % the names
    \newdimen#1    % of these
    \newdimen#1    % registers are
    \newdimen#1    % unimportant
    \newinsert#1
    \global\toksdef#2=\allocationnumber
    \global\toks\allocationnumber{#2}
            % the token list register
    \global\count#1=\count11
            % the pointer to the
            % highest allocated dimen
}
```

Unfortunately there are some problems with this construction. First of all the `\new...` commands *cannot* be used inside a macro because they are defined to be `\outer`. Redefinition would be a good solution for this but not the other problems. Namely in this case you would find in the log file many messages about insertion and dimension allocations but no information about planbox allocation. Furthermore this is not an efficient construction and its form is quite different from the other kinds of allocation in `plain`. Yet planbox is something like an insertion: not just a set of certain registers but also logical connections among them. For this reasons my solution for the allocation is the following, which is a simple merge of a `\newinsert`, a `\newtoks` and four `\newdimen` commands with only one `\ch@ck` for each kind of register and with the appropriate message to the log file. I also set the value of the token list register here because I want to keep the assignments connected with `\allocationnumber` together:

```
% allocation for width,depth,
%                hdis,vdis,name,next
\def\planboxall@c#1#2{%
    \global\advance\insc@unt by\m@ne
    \global\advance\count11 4
    \ch@ck0\insc@unt\count
    \ch@ck1\insc@unt\dimen
    \ch@ck4\insc@unt\box
    \ch@ck5\insc@unt\toks
    \allocationnumber=\insc@unt
    \global\chardef#1=\allocationnumber
    \global\toksdef#2=\allocationnumber
    \global\count#1=\count11
    {\advance\count#1 -3
    \wlog{\string#1=\string\insert
                    \the\allocationnumber;
                  \dimen\the\count#1...
                  \dimen\the\count11 .}}
    \global\toks\allocationnumber{#2}}
```

For storing the values, it would be better to use the pointer of the planbox, but instead I use the 'dirty information' that right after the allocation `\count11` points to the same register as the pointer of the planbox would do (this way I save an indirection):

```
% storing width, depth, hdis and vdis
% in the appropriate registers
\def\setsiz@s#1,#2,#3,#4; {%
    {\global\dimen\count11 #4
        \advance\count11 \m@ne
     \global\dimen\count11 #3
        \advance\count11 \m@ne
     \global\dimen\count11 #2
```

```
        \advance\count11 \m@ne
      \global\dimen\count11 #1}}
```

After getting the attributes of the planboxes, \pl@nrest is invoked. It stores the current insertion number in \minplanb@xno. This register and \maxplanb@xno point out the place of the planboxes; this information will be used at the end of each use of the package to release the occupied registers. I also allocate a planbox for \null. It's a trick and seems to be useless here but you'll see its importance below. Nevertheless, notice that \null is not a real null pointer but a planbox!

```
\newcount\minplanb@xno
\def\pl@nrest{%
    \global\minplanb@xno\insc@unt
    \planboxall@c\null\zero
    \setsiz@s\hsize,\maxdimen,\z@,\z@;
    \egroup}
```

Filling the page

The structure is ready, all the registers have been allocated and all the logical connections are set; we can start to fill the planboxes with their contents.

Let's consider the actions step by step:

The first macro invoked is \bpage:

```
\def\bpage#1; {\bgroup \s@vepagesofar
    \tolerance=10000
    \showboxbreadth1 \showboxdepth1
       % there are many Underfull hboxes
       % while processing
    \advance\baselineskip 0pt
                     plus .3pt minus .1pt
    \wlog{Beginning of Page.}
    \def\par{\endgraf\egroup\pl@npar}
    \output{\fullb@xoutput}\topskip\z@
    \bb@x#1; }
```

Its main task is some preparation and initialization. Before any action, it saves the part of the page which is ready at this time in \s@vepagesofar:

```
\newbox\p@gesofarbox
\def\s@vepagesofar{\output{%
    \global\setbox\p@gesofarbox\vbox{%
           \unvbox255}}\eject}
```

Afterwards, it sets \tolerance=10000 to avoid overfull hboxes in the planned page. The package produces many underfull boxes without any visible reason. Thus \showboxbreadth and \showboxdepth are set to their minimal values. The little stretchability and shrinkability of \baselineskip is needed because of the relatively small height of the planboxes. After the message to the log file comes the essential part of the macro.

The algorithm of the process. The idea is that the package proceeds through the entire text paragraph by paragraph. Each paragraph is put in a vbox. If this paragraph has room in the current planbox, then it is simply added to the material so far; otherwise the paragraph has to be split up into two parts. The first part goes to the current planbox and the second one to the next planbox. At the same time we should finish the current planbox and switch to the next one. Then the next paragraph is processed.

Redefinition of \par is the essence of this idea. Namely, it finishes the vbox by \egroup and does the necessary actions through \pl@npar (see Accumulating the paragraphs below). The last of these actions is starting a new paragraph and also a new vbox. The output routine is also redefined (this will be explained later) and \topskip is set to zero because we're making not a whole page, just a part of it. Believe it or not, no more preparation is needed; we can start the current planbox (in \bb@x):

```
\newcount\curplanb@xno
\newdimen\curplanb@xsofar
\newif\iffill@
\def\bb@x#1; {\initb@x#1; \fillOFF \st@rtpar}
\def\newsiz@s#1{%
    \advance\count#1 -3
    \hsize\dimen\count#1
    \advance\count#1 \@ne
    \vsize\dimen\count#1
    \advance\count#1 \tw@}
\def\initb@x#1; {%
    \wlog{The next planbox is #1.}
    \global\curplanb@xno#1
    \curplanb@xsofar\z@
    \newsiz@s\curplanb@xno}
```

Again, first some initialization for the box (\initb@x). This means a message to the log file, a note on the current planbox to \curplanb@xno, resetting the height of the material in the planbox so far to zero (there is no material at all) and setting \hsize and \vsize to the <width> and <depth> of the planbox. The last step of the initialization (\fillOFF) is the decision that at the end of the planbox we don't want to fill the rest space with \vfil (detailed explanation will come below). Finishing the initialization we can start to process the first paragraph:

```
\def\st@rtpar{%
    \advance\curplanb@xsofar \parskip
    \vskip\parskip
    \setbox\c@rrpar\vbox\bgroup}
```

The reason for putting `\parskip` into the vertical list by hand is that with our macros TeX sees only vboxes and not paragraphs, since we put every paragraph into `\c@rrpar`, which is the box finished in `\par`.

The normal way of processing is simply to accumulate the paragraphs. Two things may happen which can break this accumulation. The first occurs when the planbox is full, and the second when a user command is encountered. The former causes the automatic switch to be invoked (see the next section). The user commands can be divided in two classes: the first class contains the commands related to the parameters of the package. The second is formed by `\nextbox`, `\nextarea` and `\epage`. As I mentioned before, the commands of the second class can be performed only between two paragraphs, but not inside one.

Switching by hand

The commands `\nextbox` and `\nextarea` have much the same code:

```
\def\nextarea#1; {\endplanb@x
    \wlog{New area.}\bb@x#1; }
\def\nextbox{\endplanb@x
    \bb@x\the\toks\curplanb@xno; }
\def\endplanb@x{\iffill@ \vfill \fi \break}
```

Both of them should finish the current planbox (`\endplanb@x`) and start the next one (`\bb@x`). At the time `\endplanb@x` is invoked we are in vertical mode (between two paragraphs), hence `\break` causes a page break, i.e., it causes the output routine to be invoked (still see below). But before this we should decide if a `\vfill` is needed at the bottom of the current planbox. In `\bb@x` the decision is no (`\fillOFF`) but you can change it (the ultimate explanation will come soon).

To start the new planbox in `\nextbox`, `\bb@x` uses the `<next>` attribute of the planbox while `\nextarea` works with its parameter, the `<name>` of another planbox. Let me remind you that this `<name>` can be the `<name>` of *any* planbox with its successors (see above).

The third command of the second class is `\epage`:

```
\def\epage{\endplanb@x\egroup
    \box\p@gesofarbox
    \nointerlineskip
    \vskip\aboveplanskip
    \hbox{%
      \loop
        \advance\maxplanb@xno \m@ne
        \ifvoid\maxplanb@xno
        \else \dimen@\wd\maxplanb@xno
          {\advance\count
                \maxplanb@xno \m@ne
          \kern\dimen\count \maxplanb@xno}%
                % kern <hdis>
          \lower\dimen\count \maxplanb@xno
                \hbox{\box\maxplanb@xno}%
                % lower <vdis>
          {\advance\count
                \maxplanb@xno \m@ne
          \kern-\dimen\count \maxplanb@xno}%
                % kern -<hdis>
          \kern-\dimen@
                % kern -<width>
        \fi
      \ifnum\maxplanb@xno>\minplanb@xno
      \repeat}\wlog{End of Page.}%
      \rele@seplan}
\def\rele@seplan{%
    \global\insc@unt\maxplanb@xno
    \advance\maxplanb@xno \m@ne
    \advance\count\maxplanb@xno -4
    \global\count11\count\maxplanb@xno}
```

It has a more difficult job to do. After finishing the last planbox, `\epage` should construct the whole page, i.e., put each planbox in its place on the page. But first it leaves the group started in `\bpage` and puts back the part which was ready before the planned part. The variable `\aboveplanskip` has the same function as `\topskip` for whole pages. After putting it on the vertical list, an `\hbox` is started in order to keep the planned part together, and separately from the other material.

Inside the `\hbox` a `\loop` goes through all the planboxes. Apart from empty planboxes, placing a planbox means a horizontal kerning for `<hdis>`, a vertical kerning for `<vdis>`, putting the box at the point reached, and afterwards coming back to the origin. The horizontal kerning is done by a real `\kern` where `\hdis` comes from the indirection through the `\count` register of the planbox. Vertical kerning and placing are done by a `\lower` command. Again, `\vdis` is found with the same indirection. After performing `\lower` the 'cursor' of the page goes back automatically to its original place so we have to 'undo' only the horizontal kerning. This also includes the width of the planbox.

And finally we should release all the planboxes (`\rele@seplan`). This feature is missing from `plain` TeX so `\rele@seplan` has to do it explicitly. Resetting `\insc@unt` is simple because `\bpage` has stored its value in `\maxplanb@xno`. The dimen allocation register (`\count11`) was originally four less than the value of the `\count` register of the first planbox, since this count register points to the highest of the four dimension registers related to the planbox.

The output routine. This is an interesting output routine because the major part of it does nothing else but give information:

```
\def\fullb@xoutput{%
    \global\setbox\curplanb@xno
        \vtop to \vsize{%
            \line{\hfil}\nointerlineskip
            \unvbox255}%
    \ifnum\null=\the\curplanb@xno
        \errhelp{I'll forget
                  the superfluous text.}
        \errmessage{Current area is full.
                 You'll lose a part of your
                 text on the output}
        \wlog{There're no more boxes for
              this area, so I forget}
        \wlog{the superfluous text.
              The text needs about:}
        \setbox\null\vtop{\unvbox\null}
        \wlog{vertical : \the\dp\null,
            horizontal : \the\wd\null,}
        \wlog{or any equivalent space.}
    \else
        \wlog{Current planbox is full.}
    \fi}
```

Its task is to save the main vertical list in the box related to the planbox. The modification (`\line{\hfill}\nointerlineskip`) prevents the commands `\vfil`, `\vskip`, ... from getting lost.

And now comes the trick of `\null`! If there is not enough room in the area for the text, then the package switches from the last box to `\null`. Since `\null` has `<depth>=\maxdimen`, the rest of the text goes to `\null`. And at the end of `\null` the output routine is able to give you the information about the amount of the lost text.

Setting the parameters. After the interruption of the output routine let's go back to the first class of the user commands. The algorithm depends on three parameters, which are chosen to be exact. The simplest decision is mentioned twice above: the user should decide if he/she wants to fill out the space at the bottom of a planbox (handled with the `\newif` construction):

```
\newif\iffill@
\def\fillON{\global\fill@true}
\def\fillOFF{\global\fill@false}
```

The second parameter gives a choice about hyphenation (for exact explanation of how `\pretolerance` works see *The TeXbook*, p. 96):

```
\def\hyphensON{\pretolerance 300 }
\def\hyphensOFF{\pretolerance 10000 }
```

The third choice is the most important one. You can turn on and off the automatic switch:

```
\newtoks\aut@switch
\def\automaticON{\aut@switch={%
     \ifdim\curplanb@xsofar>\vsize
     \splitit@p \fi}}
\def\automaticOFF{\aut@switch={\relax}}
\def\pl@npar{%
    \advance\curplanb@xsofar \ht\c@rrpar
    \advance\curplanb@xsofar \dp\c@rrpar
    \the\aut@switch
    \unvbox\c@rrpar
    \afterassignment\wh@tnext\let\nextt=}
```

Both definitions is connected to `\pl@npar`. This leads us to the last part of this section:

Accumulating the paragraphs. I hope you remember that I haven't mentioned how to append the current paragraph to the current planbox. All I have written about is how 'to cut out' a paragraph and to put it into a vbox. But after this, `\pl@npar` is invoked in `\par`:

```
% from \bpage
...
    \def\par{\endgraf\egroup\pl@npar}
...
```

Behaviour of `\pl@npar` depends on whether `\automaticON` or `\automaticOFF` is active. In both cases it measures the vertical size of the material in the current planbox and appends the current paragraph (`\c@rrpar`) to the vertical list. When `\automaticON` is active, `\pl@npar` checks whether the current planbox is full or not. If it is, then the code for the automatic switch (`\splitit@p`) takes place (see Automatic switch below).

On the other hand, when `\automaticOFF` is active, the material is not handled automatically. Thus the user him/herself should take care of page-setting, i.e., invoking `\nextbox` or `\nextarea` at the necessary points in the text.

You may say that no one will use `\automaticOFF` since it has only disadvantages. But this is not true. If two consecutive planboxes have the same width, then by using `\automaticOFF` the page builder of plain TeX may be executed to find another (and perhaps better) solution for page breaking than the automatic switch of my package.

And one more thing about `\automaticOFF`: it should do all other things except checking because the user may switch ON again in the same planbox where it was switched OFF (even if there is no reason for doing so).

Let us return to \pl@npar. At the end it looks ahead for the next token:

```
    \def\wh@tnext{\let\next\nextt
        \ifx\nextbox\next
        \else \ifx\nextarea\next
        \else \ifx\epage\next
        \else \let\next\st@rtpar
              \afterassignment\nextt
        \fi \fi \fi \next}
```

If the first token is one of \nextbox, \nextarea or \epage then that command should be performed, because they can be performed just outside the vbox containing the paragraph. As you can see, \wh@tnext performs at most one command before starting a new paragraph. Hence, if you want two commands to be performed, leave a blank line between them. It has just one effect: the empty line means an empty paragraph between the two commands. And if there is no command at all, a new paragraph is to be started and the token is to be put back. But this happens just after the new paragraph, i.e., the vbox has been started! Fortunately \afterassignment puts the saved token back right after starting the vbox.

Again, we are at the beginning of a new paragraph.

Automatic switch

Let's pick up the thread at \splitit@p:

```
\def\splitit@p{%
    \m@veextra
    \unvbox\c@rrpar \endplanb@x
    \initb@x\the\toks\curplanb@xno;
    \r@typeset
    \global\curplanb@xsofar=\ht\c@rrpar
    \global\advance\curplanb@xsofar
                              \dp\c@rrpar
    \ifdim\curplanb@xsofar>\vsize
        \let\next\splitit@p
    \else \let\next\relax \fi \next}
```

This macro is invoked when the current paragraph has no room in the current planbox. First \m@veextra reduces the vertical size of the paragraph to the appropriate size by removing the last lines of it. Then the remaining part is appended to the current planbox. This planbox is finished and a new one is initialized (not started!). The removed part of the paragraph is then retypeset with the new \hsize. If this amount of material is too much for this planbox, then the whole process is repeated.

The macro \m@veextra removes the necessary lines one by one with \rem@velastline:

```
\def\m@veextra{%
    \global\setbox\c@rrpar\vbox{%
           \unvbox\c@rrpar \rem@velastline}%
    \global\setbox\extrat@xt\vbox{%
           \unvbox\extrat@xt\box\l@stline}%
    \ifdim\curplanb@xsofar>\vsize
        \let\next\m@veextra
    \else \let\next\relax \fi \next}
```

It also accumulates these lines in \extrat@xt, and it goes on until the vertical size of the material is less than or equal to \vsize. One single line is removed by \rem@velastline:

```
\def\rem@velastline{%
    \global\setbox\l@stline\lastbox
    \ifvoid\l@stline
       \global\advance\curplanb@xsofar
                        -\lastskip \unskip
       \unpenalty
       \global\advance\curplanb@xsofar
                        -\lastkern \unkern
       \let\next\rem@velastline
    \else
       \global\advance\curplanb@xsofar
                        -\ht\l@stline
       \global\advance\curplanb@xsofar
                        -\dp\l@stline
       \let\next\relax
    \fi \next}
```

The macro works with TeX's \lastbox and \un... operations. If a box could be removed, the macro returns it in \l@stline. Otherwise \rem@velastline tries to remove the last item in the vertical list and updates \curplanb@xsofar. Unfortunately there is no proper \un... command for each type of item, but the commands for the missing types ("whatsit", mark, insertion) are mode-independent, so in general you can avoid their being appended to the vertical list. (I hope this feature of TeX won't cause too much trouble for you and for the package.) Moreover there is no opportunity to check whether the last item is glue or not, because there is no \ifskip command to distinguish \z@skip=0pt plus 0pt minus 0pt from let's say \parskip=0pt plus 1pt. Thus brute force is used instead of checking the last item with \if... operations.

Notice that \extrat@xt contains only lines of text, i.e., neither glue items nor kerns nor penalties, and the lines are placed in reverse order. They will be reversed again in \r@typeset:

```
\def\r@typeset{%
    \global\setbox\extrat@xt\vbox{%
           \unvbox\extrat@xt
           \global\setbox\n@xtline\lastbox}%
    \setbox\n@xtline\hbox{\unhbox\n@xtline
                              \unskip}%
    \global\setbox\c@rrpar\vbox{\noindent
           \unhbox\n@xtline}%
    {\parskip\z@skip
    \loop
       \global\setbox\extrat@xt\vbox{%
           \unvbox\extrat@xt
           \global\setbox\n@xtline\lastbox}%
       \global\setbox\c@rrpar\vbox{%
           \unvbox\c@rrpar \rem@velastline
       \@penhbox\l@stline
       \setbox\n@xtline\hbox{\unhbox\n@xtline
                                 \unskip}%
       \noindent \unhbox\l@stline\ %
       \unhbox\n@xtline}%
    \ifdim\ht\extrat@xt>\z@ \repeat}}
\def\@penhbox#1{\setbox#1 \hbox{\unhbox#1%
    \unskip \unskip \unpenalty}}
```

Before examining the code, let's go through the idea. It seemes to be simple: join the lines again and let the line breaking algorithm form the new paragraph. Unfortunately the task is more difficult. The main problem is that because of the different `\hsize`, the breakpoints in the new paragraph will be at other points than they were in the original paragraph. The line breaking algorithm puts `\rightskip` at the end of every line. So `\rightskip` is to be removed from the original ends of lines.

At the end of a line a hyphenation may occur, too. The trade-off is to check every line end with a number of complicated macros or to leave the task of correcting the bad hyphenations to the user. Because of the easy correction I decided to use the latter option.

On the other hand, joining the lines means that the first couple of lines form the beginning of the new paragraph and the next line is joined to the last line of the partial paragraph. And the last line of a paragraph contains not just `\rightskip` but three more items related to `\par`, namely, `\penalty10000`, `\hskip\parfillskip` and `\penalty-10000` (*The TeXbook*, p. 100). The third item is discarded at the line break but the two other items also should be removed before the join. The macro `\@penhbox` removes all three items from its parameter box.

Last but not least, there is no space between the last word of a particular line and the first word of the next line. Hence we should put a space before each line except the first one. The first line differs from the others in another respect: We don't need to apply the joining algorithm just discussed because there is no last line of the empty paragraph.

Let's go back to the code! In the code up to the `{` before the `\parskip\z@skip` command the first line is retypeset in three steps. First the line is removed from `\extrat@xt` (there is no need to use `\rem@velastline` because `\extrat@xt` only contains the lines). Then `\rightskip` is removed. Finally, the line is put into `\c@rrpar`. Because `\c@rrpar` now contains not a real paragraph but just the second part of it, `\noindent` is inserted before the line.

The other lines will be appended according to the algorithm. First the line is removed from `\extrat@xt` the same way as the first line. Then the last line of the paragraph is removed with `\rem@velastline`. Both lines are 'peeled' and then joined with a space between them. We should insert `\noindent` before `\l@stline` because this line may be the first in the paragraph; so the paragraph may be restarted at this point. This is the reason for setting `\parskip` to `\z@`. When a paragraph starts `\parskip` is automatically inserted. In our case the paragraph may start again but we don't need another `\parskip`.

The whole action is repeated until all the lines have been joined together.

And in the end...

This package is developed to handle pure text. It was created in such a way that it avoids interfering with `\plain` TeX whenever possible. On the other hand the package has to change things deep inside TeX. I'm sure these changes mean restrictions of the usage but only experiments can discover all of them. However, I think that independently from the restrictions the package is useful and helps to create documents with a better look.

Already at the moment there is one possible improvement: the multipage version of the package. This needs only technical, and not fundamental changes, but it gives the possibility of making whole newspapers. I hope sooner or later that version will come out.

My only reference was *The TeXbook.* If you find my article does not explain a notion you will find the best possible answer in this book.

At last: I hope you will enjoy this 'pagemaker'.

◇ Péter Huszár
Budapest
Bogdánffy út 10.b.
H-1117 Hungary
`h1612hus@ella.hu`