Creating Flow Frames for Posters, Brochures or Magazines using flowfram.sty v 1.13

Nicola L. C. Talbot

23rd January 2010

Dr Nicola Talbot
School of Computing Sciences
University of East Anglia
Norwich. Norfolk. NR4 7TJ.
United Kingdom
http://theoval.cmp.uea.ac.uk/~nlct/

Contents

1. Introduction

This chapter provides a brief overview of the package, the package options and the various frame types.

The flowfram package is a LaTeX 2ε package designed to enable you to create text frames in a document such that the contents of the document environment flow from one frame to the next in the order that they were defined. This is useful for creating posters or magazines or any other form of document that does not conform to the standard one or two column layout.

The flowfram package provides three types of frame: flow frames, static frames and dynamic frames with dimensions and positions specified by the user[1]. The main contents of the document environment flow from one flow frame to the next in the order of definition, whereas the contents of the static and dynamic frames are set explicitly using commands described in chapter 3. Note that unless otherwise stated, all co-ordinates are relative to the bottom left hand corner of the typeblock. If you have a two-sided document, the absolute position of the typeblock may vary depending on the values of `\oddsidemargin` and `\evensidemargin`, and all the frames will shift accordingly unless otherwise indicated.

Since floats (such as figures and tables) can only go in flow frames, this package provides the additional environments: staticfigure and statictable which can be used in static frames and dynamic frames. Unlike their figure and table counterparts, they are fixed in place, and so do not take an optional placement specifier. The `\caption` and `\label` commands can be used within staticfigure and statictable as usual, but remember that if the frame is displayed on multiple pages, you may end up with multiply defined labels.

The standard figure and table commands will behave as usual in the flow frames, but their starred versions, figure* and table* behave no differently from figure and table[2].

This package has only been tested with a limited number of class files and packages. Since it modifies the output routine, it is likely to conflict with any other package which also does this.

You should load flowfram *after* hyperref and any colour package (e.g. color).

1.1 Draft Option

The flowfram package has the package option draft which will draw the bounding boxes for each frame that has been defined. At the bottom right of each bounding box (except for the bounding box denoting the typeblock), a marker will be shown in the form: [⟨*T*⟩:⟨*idn*⟩;⟨*idl*⟩], where ⟨*T*⟩ is a single letter denoting the frame type, ⟨*idn*⟩ is the identification number (IDN) for the frame and ⟨*idl*⟩ is the identification label (IDL) for that frame. Values of ⟨*T*⟩ are: `F` (flow frame), `S` (static frame) or `D` (dynamic frame). Markers of the form: [M:⟨*idn*⟩] indicate that the bounding box is the area taken up by the margin for flow frame with IDN ⟨*idn*⟩. Note that even if a frame has been rotated, the bounding box will not be rotated.

If you want to show or hide specific types of bounding boxes, you can use one of the following commands:

[1] Can I have arbitrary shaped frames? See section 3.1

[2] This is because of the arbitrary layout of the flow frames.

- `\showtypeblocktrue` Display the bounding box for the typeblock.
- `\showtypeblockfalse` Do not display the bounding box for the typeblock.
- `\showmarginstrue` Display the bounding box for the margins.
- `\showmarginsfalse` Do not display the bounding box for the margins.
- `\showframebboxtrue` Display the bounding box for the frames.
- `\showframebboxfalse` Do not display the bounding box for the frames.

You can see the layout for the current page (irrespective of whether or not the draft option has been set) using the command:

```
\flowframeshowlayout
```

The flowfram package also has the options nocolor and norotate for previewers that can not process colour or rotating specials. (Otherwise you may end up with large black rectangles obscuring your text, instead of the pale background colour you were hoping for.)

1.2 Frame Stacking Order

The material on each page is placed in the following order:

1. Each static frame defined for that page in ascending order of IDN.
2. Each flow frame defined for that page in ascending order of IDN.
3. Each dynamic frame defined for that page in ascending order of IDN.
4. Bounding boxes if the draft package option has been used.

This ordering can be used to determine if you want something to overlay or underlay everything else on the page. Note that the frames do not interact with each other. If you have two or more overlapping frames, the text in each frame will not attempt to wrap around the other frames, but will simply overwrite them.[3]

[3] Can I have arbitrary shaped frames? See section 3.1.

1.3 HTML

The flowfram package now comes with a LaTeX2HTML style file `flowfram.perl`. However this style file is not meant to emulate the flowfram package, but is provided to facilitate creating a plain HTML document from the LaTeX source. All frame-related information is ignored. By default, the contents of any static or dynamic frames are ignored, but this can be changed using

```
\HTMLset{showstaticcontents}{1}
```

to show the contents of the static frames or

```
\HTMLset{showdynamiccontents}{1}
```

to show the contents of the dynamic frames (where `\HTMLset` is defined in the html package). Note that this places the text at the point in the document where the contents are set. This style file does not create HTML frames. It can therefore be used to create an accessible version of the PDF document such as the HTML version of this document, `ffuserguide.html`.

2.1 Flow Frames

The flow frame is the principle type of frame. The text of the document environment will flow from one frame to the next in order of definition. Each flow frame has an associated width, height, position on the page and optionally a border. To define a new flow frame use:

`\newflowframe[`⟨*page list*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}[`⟨*label*⟩`]`

where ⟨*width*⟩ is the width of the frame, ⟨*height*⟩ is the height of the frame, (⟨*x*⟩,⟨*y*⟩) is the position of the bottom left hand corner of the frame relative to the bottom left hand corner of the typeblock[1]. The first optional argument, ⟨*page list*⟩, indicates the list of pages for which this frame is defined.

A page list can either be specified by the keywords: `all`, `odd`, `even` or `none`, or by a comma-separated list of either individual page numbers or page ranges. If ⟨*page list*⟩ is omitted, `all` is assumed. A page range can be a closed range (e.g. `2-8`) or an open range (e.g. `<10` or `>5`). For example: `<3,5,7-11,>15` indicates pages 1, 2, 5, 7, 8, 9, 10, 11 and all pages greater than page 15. These page numbers refer to the value of the page counter[2], so if you have a page i and a page 1, they will both have the same layout (unless you change the page list setting somewhere between the two pages).

Each frame has its own unique IDN, corresponding to the order in which it was defined. So the first flow frame to be defined has IDN 1, the second has IDN 2, and so on. This number can then be used to identify the frame when you want to modify its settings. Alternatively, you can assign a unique IDL to the frame using the final optional argument ⟨*label*⟩.

You can retrieve the IDL for a given flow frame from its IDN using:

`\getflowlabel{`⟨*idn*⟩`}`

Conversely, you can retrieve the IDN for a given flow frame from its IDL using:

`\getflowid{`⟨*cmd*⟩`}{`⟨*idl*⟩`}`

where ⟨*cmd*⟩ is a control sequence which will be used to store the frame's IDN. For example:

```
The label for the first flow frame is ``\getflowlabel{1}''.
The flow frame labelled ``main'' has IDN \getflowid{\myid}{main}\myid.
```

produces: The label for the first flow frame is "main". The flow frame labelled "main" has IDN 1. (See also section 7.3.)

[1] See query 10 on page 51 if you want to convert from absolute page co-ordinates to co-ordinates relative to the typeblock

[2] why can't I use the page number format? See query 3 on page 49

2. Defining New Frames

This chapter describes how to define new frames, and how to identify and set frame contents. See also chapter 5.

Note that \getflowlabel doesn't perform any check to determine whether the supplied IDN is valid, but \getflowid will generate an error if the supplied IDL is undefined.

By default, the flow frame will not have a border, but the starred form

\newflowframe*[⟨page list⟩]{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}[⟨label⟩]

will place a plain border around the flow frame. (See chapter 3 if you want a different border.)

Note that if the document continues beyond the last defined flow frame (for example, the flow frames have only been defined on pages 1 to 10, but the document contains 11 pages) then a single flow frame will be defined, emulating one column mode for all subsequent pages.

In this document, I have used the command

```
\newflowframe{0.6\textwidth}{\textheight}{0pt}{0pt}[main]
```

to define the main flow frame[3] (i.e. this one).

2.1.1 Prematurely Ending a Flow Frame

You can force text to move immediately to the next defined flow frame using one of the commands: \newpage, \pagebreak or \framebreak. The first two work in an analogous way to the way they work in standard two column mode. The last one, \framebreak, is required when a paragraph spans two flow frames of different widths, as TEX's output routine does not adjust to the new value of \hsize until the last paragraph of the previous frame has ended. As a result, the end of the paragraph at the beginning of the new flow frame retains the width of the previous flow frame.

If you want to start a new page, rather than simply move to the next frame, use the command \clearpage, or for two-sided documents, to start on the next odd page do \cleardoublepage.

2.2 Static Frames

A static frame is a rectangular area in which text neither flows into nor flows out of.[4] The contents must be set explicitly, and once set, the contents of the static frame will remain the same on each page until it is explicitly changed. Thus, a static frame can be used, for example, to make a company logo appear in the same place on every page.

As from version 1.03 it is now possible to have static frames with non-rectangular contents, see section 3.1 for further details.

[3] the position for the even pages is set using \setflowframe defined in chapter 3

[4] By "neither flows into nor flows out of" I mean you have to explicitly set the contents of this frame. Note that it may appear to contain text if another frame overlaps it, but this text belongs to the other frame.

A new static frame is defined using the command:

```
\newstaticframe[⟨page list⟩]{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}[⟨label⟩]
```

where, as with \newflowframe, ⟨*width*⟩ is the width of the frame, ⟨*height*⟩ is the height of the frame, (⟨*x*⟩,⟨*y*⟩) is the position of the bottom left hand corner of the frame relative to the bottom left hand corner of the typeblock. The first optional argument, ⟨*page list*⟩, indicates the page list for which this static frame should appear, and the final optional argument, ⟨*label*⟩ is a unique textual IDL which you can use to identify this frame. If no label is specified, you can refer to this frame by its unique IDN. The first static frame to be defined has IDN 1, the second has IDN 2, and so on.

You can retrieve the IDL for a given static frame from its IDN using:

```
\getstaticlabel{⟨idn⟩}
```

Conversely, you can retrieve the IDN for a given static frame from its IDL using:

```
\getstaticid{⟨cmd⟩}{⟨idl⟩}
```

where ⟨*cmd*⟩ is a control sequence which will be used to store the frame's IDN. For example:

```
The label for the first static frame is ``\getstaticlabel{1}''.
The static frame labelled ``backleft'' has IDN
\getstaticid{\myid}{backleft}\myid.
```

produces: The label for the first static frame is "backleft". The static frame labelled "backleft" has IDN 1.

Note that \getstaticlabel doesn't perform any check to determine whether the supplied IDN is valid, but \getstaticid will generate an error if the supplied IDL is undefined.

As with \newflowframe, there is a starred version

```
\newstaticframe*[⟨page list⟩]{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}[⟨label⟩]
```

which will place a border around that static frame.

To set the contents of a particular static frame, you can either use the staticcontents environment:

```
\begin{staticcontents}{⟨IDN⟩}
⟨contents⟩
\end{staticcontents}
```

where ⟨*IDN*⟩ is the unique IDN associated with that static frame and ⟨*contents*⟩ is the contents of the static frame, or you can use the command:

```
\setstaticcontents{⟨IDN⟩}{⟨contents⟩}
```

which will do the same thing.

There are starred versions available for both the environment and the command to enable you to identify the static frame by its associated IDL rather than its IDN:

```
\begin{staticcontents*}{⟨IDL⟩}
⟨contents⟩
\end{staticcontents*}
```

or the equivalent:

```
\setstaticcontents*{⟨IDL⟩}{⟨contents⟩}
```

In the body of staticcontents or staticcontents*, or in the second argument of `\setstaticcontents` or `\setstaticcontents*`, you can move onto another static frame using:

```
\continueonframe[⟨continuation text⟩]{⟨id⟩}
```

If staticcontents* or `\setstaticcontents*` are being used, ⟨*id*⟩ refers to the IDL of the next static frame, otherwise ⟨*id*⟩ refers to the IDN of the next static frame. The optional argument specifies some continuation text to place at the end of the first static frame. For example, suppose I have defined two static frames labelled “frame1” and “frame2”, then

```
\begin{staticcontents*}{frame1}
Some text in the first frame. (Let's
assume this frame is somewhere on the
left half of the page.)
\continueonframe[Continued on the right]{frame2}
This is some text in the second frame. (Let's
assume this frame is somewhere on the
right half of the same page.)
\end{staticcontents*}
```

is equivalent to:

```
\begin{staticcontents*}{frame1}
Some text in the first frame. (Let's
assume this frame is somewhere on the
left half of the page.)
\ffcontinuedtextlayout{Continued on the right}
\end{staticcontents*}
\begin{staticcontents*}{frame2}\par\noindent
This is some text in the second frame. (Let's
assume this frame is somewhere on the
right half of the same page.)
\end{staticcontents*}
```

where

`\ffcontinuedtextlayout{`⟨*text*⟩`}`

governs how the continuation text should be displayed. The font used to display the continuation text is given by

`\ffcontinuedtextfont{`⟨*text*⟩`}`

Note that this assumes that it should appear that no paragraph break occurs in the transition between the two frames. If you want a paragraph break you need to explicitly put one before and after `\continueonframe`. For example:

```
\begin{staticcontents*}{frame1}
Some text in the first frame. (Let's
assume this frame is somewhere on the
left half of the page.)

\continueonframe[Continued on the right]{frame2}

This is some text in the second frame. (Let's
assume this frame is somewhere on the
right half of the same page.)
\end{staticcontents*}
```

2.2.1 Important Notes

- When you set the contents of a static frame, the contents are immediately typeset and stored in a box until it is time to put the contents on the page. This means that if you use any information that varies throughout the document (such as the page number) the value that is current when you set the static frame's contents will be the value used.
- However, if `\label` is used inside a static frame, the label information will be written to the auxiliary file each time the static frame is displayed until the contents of that frame have been changed. This means that you may end up with multiply defined labels.

2.3 Dynamic Frames

A dynamic frame is similar to a static frame except that its contents are re-typeset on each page. (A static frame stores its contents in a savebox, whereas a dynamic frame stores its contents in a macro.[5])

As from version 1.03 it is now possible to have dynamic frames with non-rectangular contents, see section 3.1 for further details.

To create a new dynamic frame, use the command:

`\newdynamicframe[`⟨*page list*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}[`⟨*label*⟩`]`

The parameters are exactly the same as for `\newflowframe` and `\newstaticframe`. Again, each dynamic frame has an associated unique IDN, starting from 1 for the first dynamic frame to be defined, and a unique IDL can also be set using the final optional argument ⟨*label*⟩.

You can retrieve the IDL for a given dynamic frame from its IDN using:

`\getdynamiclabel{`⟨*idn*⟩`}`

Conversely, you can retrieve the IDN for a given dynamic frame from its IDL using:

`\getdynamicid{`⟨*cmd*⟩`}{`⟨*idl*⟩`}`

where ⟨*cmd*⟩ is a control sequence which will be used to store the frame's IDN.

For example:

```
The label for the first dynamic frame is ``\getdynamiclabel{1}''.
```

[5] which means that you can have verbatim text in the body of the staticcontents environment but not in the body of the dynamiccontents environment (see page 12)

```
The dynamic frame labelled ``chaphead'' has IDN
\getdynamicid{\myid}{chaphead}\myid.
```

produces: The label for the first dynamic frame is “chaphead”. The dynamic frame labelled “chaphead” has IDN 1.

Note that `\getdynamiclabel` doesn't perform any check to determine whether the supplied IDN is valid, but `\getdynamicid` will generate an error if the supplied IDL is undefined.

As with the other frame types, there is also a starred version

`\newdynamicframe*[`⟨*page list*⟩`]{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}[`⟨*label*⟩`]`

which will place a plain border around the dynamic frame. For example, in this document I have used the command

```
\newdynamicframe{0.38\textwidth}{\textheight}{0.62\textwidth}{0pt}[chaphead]
```

which has created the frame on the right on odd pages and on the left on even pages. (The position for the even pages is set using `\setdynamicframe` defined in chapter 3.)

The contents of a dynamic frame are set using the command:

`\setdynamiccontents{`⟨*id*⟩`}{`⟨*contents*⟩`}`

where ⟨*id*⟩ is the unique IDN associated with that dynamic frame, and ⟨*contents*⟩ is the contents of the dynamic frame. Alternatively, if you have assigned an IDL, ⟨*label*⟩, to the dynamic frame, you can use the starred version:

`\setdynamiccontents*{`⟨*label*⟩`}{`⟨*contents*⟩`}`

As with most LaTeX commands, you can't use verbatim text in ⟨*contents*⟩.

As from version 1.09, the contents can also be set using the dynamiccontents environment:

```
\begin{dynamiccontents}{⟨id⟩}
⟨contents⟩
\end{dynamiccontents}
```

or the dynamiccontents* environment:

```
\begin{dynamiccontents*}{label}
⟨contents⟩
\end{dynamiccontents*}
```

Note that you can't use verbatim text within the dynamiccontents or dynamiccontents* environments.

You can additionally append text to a dynamic frame using either:

```
\appenddynamiccontents{⟨id⟩}{⟨contents⟩}
```

or:

```
\appenddynamiccontents*{⟨label⟩}{⟨contents⟩}
```

If `\chapter` is defined, you can make the chapter titles appear in a dynamic frame using the command

```
\dfchaphead{⟨IDN⟩}
```

where ⟨*IDN*⟩ is the IDN of the dynamic frame. There is also a starred version of this command if you want to use the IDL instead of the IDN. For example, in this document, I used the command:

```
\dfchaphead*{chaphead}
```

The headers and footers can be turned into dynamic frames using the command

```
\makedfheaderfooter
```

This will create two dynamic frames with IDLs `header` and `footer`. The page style will be used as usual, but you can then move or resize the header and footer using `\setdynamicframe` (described in chapter 3).

In the body of dynamiccontents or dynamiccontents*, you can move onto another dynamic frame using:

```
\continueonframe[⟨continuation text⟩]{id}
```

If this command occurs within dynamiccontents*, ⟨*id*⟩ refers to the IDL of the new frame, otherwise it refers to the IDN of the new frame. The optional argument specifies some continuation text to place at the end of the first dynamic frame. For example, suppose I have defined two dynamic frames labelled "frame1" and "frame2", then

```
\begin{dynamiccontents*}{frame1}
Some text in the first frame. (Let's
assume this frame is somewhere on the
left half of the page.)
```

```
\continueonframe[Continued on the right]{frame2}
This is some text in the second frame. (Let's
assume this frame is somewhere on the
right half of the same page.)
\end{dynamiccontents*}
```

is equivalent to:

```
\begin{dynamiccontents*}{frame1}
Some text in the first frame. (Let's
assume this frame is somewhere on the
left half of the page.)
\ffcontinuedtextlayout{Continued on the right}
\end{dynamiccontents*}
\begin{dynamiccontents*}{frame2}\par\noindent
This is some text in the second frame. (Let's
assume this frame is somewhere on the
right half of the same page.)
\end{dynamiccontents*}
```

where

`\ffcontinuedtextlayout{`⟨*text*⟩`}`

governs how the continuation text should be displayed. The font used to display the continuation text is given by

`\ffcontinuedtextfont{`⟨*text*⟩`}`

Note that this assumes that it should appear that no paragraph break occurs in the transition between the two frames. If you want a paragraph break you need to explicitly put one before and after `\continueonframe`. For example:

```
\begin{dynamiccontents*}{frame1}
Some text in the first frame. (Let's
assume this frame is somewhere on the
left half of the page.)

\continueonframe[Continued on the right]{frame2}
```

```
This is some text in the second frame. (Let's
assume this frame is somewhere on the
right half of the same page.)
\end{dynamiccontents*}
```

2.3.1 Important Notes

- Verbatim text can't be used in a dynamic frame. This includes the body of the dynamiccontents and dynamiccontents* environments.
- `\continueonframe` can't be used in the argument of any of the commands that set the contents of a dynamic frame, such as `\setdynamiccontents`.
- Dynamic frames are painted on the page after all the static and flow frames. If the location of a dynamic frame overlaps the location of any static or flow frames, the contents of the dynamic frame will obscure the contents of the overlapping frames.

3. Modifying Frame Attributes

This chapter describes how to modify frame attributes, such as the size and location.

Once you have defined the flow frames, static frames and dynamic frames, their attributes can be changed. The three types of frame mostly have the same set of attributes, but some are specific to a certain type.

Flow frame attributes are modified using either the command:

`\setflowframe{⟨idn list⟩}{⟨key-val list⟩}`

or the starred version:

`\setflowframe*{⟨label list⟩}{⟨key-val list⟩}`

or the attributes for all flow frames can be set using:

`\setallflowframes{⟨key-val list⟩}`

Static frame attributes are modified using either the command:

`\setstaticframe{⟨idn list⟩}{⟨key-val list⟩}`

or the starred version:

`\setstaticframe*{⟨label list⟩}{⟨key-val list⟩}`

or the attributes for all static frames can be set using:

`\setallstaticframes{⟨key-val list⟩}`

Dynamic frame attributes are modified using either the command:

`\setdynamicframe{⟨idn list⟩}{⟨key-val list⟩}`

or the starred version:

`\setdynamicframe*{⟨label list⟩}{⟨key-val list⟩}`

or the attributes for all dynamic frames can be set using:

`\setalldynamicframes{⟨key-val list⟩}`

In each of the above, ⟨*idn list*⟩ can either be one of the keywords: `all`, `odd` or `even` (indicating all

frames of that type, frames of that type whose IDN is odd or frames of that type whose IDN is even) or it can be a comma-separated list of ID numbers, or IDN ranges.

For the starred versions, ⟨*label list*⟩ should be a comma-separated list of IDLs. Note that you can't use the above keywords or have ranges in ⟨*label list*⟩.

The ⟨*key-val list*⟩ argument must be a comma-separated list of ⟨*key*⟩=⟨*value*⟩ pairs, indicating which attributes to modify. The available values are as follows:

width=⟨*length*⟩ The width of the frame.

height=⟨*length*⟩ The height of the frame.

x=⟨*length*⟩ The x-coordinate of the frame for all pages on which it is defined.

y=⟨*length*⟩ The y-coordinate of the frame for all pages on which it is defined.

evenx=⟨*length*⟩ The x-coordinate of the frame for all even pages on which it is defined, but only if the document is a two-sided document.

For example, in this document, I have used the commands

```
\setflowframe*{main}{evenx=0.4\textwidth}
\setdynamicframe*{chaphead}{evenx=0pt}
```

to switch the positions of the flow frame and dynamic frame containing the document text and chapter headings, respectively, on even pages.

You can swap the odd and even values using the commands:

```
\ffswapoddeven{⟨IDN⟩}
```

(for flow frames)

```
\sfswapoddeven{⟨IDN⟩}
```

(for static frames) or

```
\dfswapoddeven{⟨IDN⟩}
```

(for dynamic frames). These commands all have starred versions which take the frame's IDL instead of its IDN.

eveny=⟨*length*⟩ The y-coordinate of the frame for all even pages on which it is defined, but only if the document is a two-sided document.

oddx=⟨*length*⟩ The x-coordinate of the frame for all odd pages on which it is defined, if the document is two-sided.

oddy=⟨*length*⟩ The y-coordinate of the frame for all odd pages on which it is defined, if the document is two-sided.

valign=⟨*pos*⟩ Change the vertical alignment of material inside a static or dynamic frame. The value ⟨*pos*⟩ may be one of: `c`, `t` or `b`. The default for static frames is `c`, the default for dynamic frames is `t`. This key is not available for flow frames.

label=⟨*text*⟩ Assign an IDL to the frame. (If you do not specify a label when you first define a frame it will be given a label identical to its IDN.) This key is provided to allow the user to label frames that have been generated by certain predefined layout commands described in chapter 5.

border=⟨*style*⟩ The style of the border around the frame, this can take the values: `none` (no border), `plain` (plain border) or the name of a LaTeX frame making command without the preceding backslash. (I admit the notation is a little confusing, a frame making command is a command that places some kind of border around its argument, such as `\fbox`, or if you are using the fancybox package: `\doublebox`, `\ovalbox`, `\Ovalbox` and `\shadowbox`.) The value `fbox` is equivalent to `plain`.

For example, to make the first static frame have an oval border:

```
\setstaticframe{1}{border=ovalbox}
```

Or you can define your own border:

```
\newcommand{\greenyellowbox}[1]{\fcolorbox{green}{yellow}{#1}}
\setstaticframe{1}{border=greenyellowbox}
```

This next example uses the tikz package to define a fancy frame, so you need to use:

```
\usepackage{tikz}
\usetikzlibrary{snakes}
```

The border command is defined as follows:

```
\newlength\fancywidth
\newlength\fancyheight
\newlength\fancydepth

\newcommand{\fancyborder}[1]{%
\settowidth{\fancywidth}{#1}%
\settoheight{\fancyheight}{#1}%
\settodepth{\fancydepth}{#1}%
\addtolength{\fancyheight}{\fancydepth}%
\hspace{-\flowframesep}%
\tikz[baseline=0pt]{%
\draw[snake=bumps,raise snake=\flowframesep,
      line width=\flowframerule]
  (0pt,0pt) rectangle (\fancywidth,\fancyheight);
}}
```

This makes a bumpy border, but it uses `\flowframesep` to determine the gap between the border and the text and uses `\flowframerule` to set the line width. This ensures that the offset (see below) is correctly computed.

This new border can now be applied to a frame:

```
\setstaticframe{1}{border=fancyborder}
```

offset=⟨*offset*⟩ The border offset, if it is a user-defined border. This is the distance from the outer edge of the left hand border to the left edge of the bounding box of the text inside the border. The flowfram package is able to compute the border for the following known frame making commands: `\fbox`, `\ovalbox`, `\Ovalbox`, `\doublebox` and `\shadowbox`. For all other borders, the offset is assumed to be `−\flowframesep−\flowframerule`. If you define your own frame making command, you may need to specify the offset explicitly, or the flow/static/dynamic frames may end up shifted to the right or left.

The above examples can compute their own offsets, however, if you were to do, for example:

```
\newcommand{\thickgreenyellowbox}[1]{%
{\setlength{\fboxsep}{5pt}\setlength{\fboxrule}{6pt}%
\fcolorbox{green}{yellow}{#1}}}
```

Then you would have to specify the offset. In this example, the offset is $-5\text{pt}-6\text{pt}=-11\text{pt}$, so you would need to do:

```
\setstaticframe{1}{border=thickgreenyellowbox,offset=-11pt}
```

bordercolor=⟨*colour*⟩ The colour of the border if you are using a standard frame making command. The colour can either be specified as, e.g. `green`, or including the colour model, e.g. `[rgb]{0,1,0}`. For example:

```
\setallflowframes{border=doublebox,bordercolor=[rgb]{1,0,0.5}}
```

textcolor=⟨*colour*⟩ The text colour for that frame. Again, the colour can either be specified as, e.g. `green`, or including the colour model, e.g. `[rgb]{0,1,0}`.

backcolor=⟨*colour*⟩ The background colour for that frame. Again, the colour can either be specified as, e.g. `green`, or including the colour model, e.g. `[rgb]{0,1,0}`. Note that the background colour only extends as far as the bounding box, not the border. If you want it to extend as far as the border, you will need to define your own border type (see above).

pages=⟨*page list*⟩ The list of pages for which the frame should appear. This can either have the values: `all`, `even`, `odd` or `none` (the latter removes the frame from that point on—useful if you have multiple pages with the same number), or it can be a comma-separated list of single pages, or page ranges. For example:

```
\setdynamicframe{1}{pages=1,5,8-10}
```

margin=⟨*side*⟩ The side of the flow frame that its corresponding margin should go on. This can take the values `left`, `right`, `inner` or `outer`. This setting is only available for flow frames.

clear=⟨*boolean*⟩ If this value is set, the static or dynamic frame will be cleared at the start of the next page, otherwise it will only be cleared on the next occurrence of `\setstaticcontents` or the staticcontents environment, or the `\setdynamiccontents`, depending on the frame type. This value is not set by default. This setting is not available for flow frames.

For example, to prevent the chapter heading reappearing on every page, I have used the command:

```
\setdynamicframe*{chaphead}{clear}
```

If you want to put `\label` in a static or dynamic frame, you should use the clear key to prevent the label from being multiply defined.

style=⟨*cmd*⟩ This should be the name of a command *without* the preceding backslash, to be applied to the contents of the specified dynamic frame. The command may either be a declaration, for example:

```
\setalldynamicframes{style=large}
```

which will set the contents of all the dynamic frames in a large font, or it can be a command that takes a single argument, for example:

```
\setalldynamicframes{style=textbf}
```

which will make the text for all the dynamic frames come out in bold. To unset a style, do `style=none`. This setting is only available for dynamic frames.

angle=⟨*n*⟩ Rotate the contents of the frame by ⟨*n*⟩ degrees (new to version 1.02). Note that the bounding boxes will not appear rotated.

shape=⟨*shape command*⟩ Define a shape for the contents of a static frame or dynamic frame (new to version 1.03). If ⟨*shape command*⟩ is `\relax`, no paragraph shape will be applied. See section 3.1 for further details.

3.1 Non-Rectangular Frames

As from version 1.03, it is now possible to specify non-rectangular static or dynamic frames (but not flow frames). Note that the bounding box will still appear as a rectangle despite the frame's shape setting. You may use either TEX's `\parshape` command, or the `\shapepar` command defined in Donald Arseneau's shapepar package (if using `\shapepar`, remember to include the shapepar package.)

Note that it is better to use `\parshape` instead of `\shapepar` as the latter is more restrictive and requires more processing. However, `\shapepar` provides greater flexibility in the type of shape that can be used.

`\parshape` With `\parshape` you can not have cut-outs in the middle, top or bottom of a frame, however it is possible to have cut-outs in the left or right side of the frame. When used with the shape key for static or dynamic frames, the effects of `\par` and the sectioning commands are modified to allow the paragraph shape to extend beyond a single paragraph, and to allow sectioning commands (but not `\chapter` or `\part`).

`\shapepar` With `\shapepar` you may have cut-outs, but you may not have any sectioning commands, paragraph breaks, vertical spacing or mathematics. You can simulate a paragraph break using `\simpar`, but this is not recommended. The size of the shape depends on the amount of text, so the shape will expand or contract as you add or delete text. See the shapepar documentation for more details.

To restore a frame to its default rectangular setting use shape=\relax.

For those unfamiliar with TEX's \parshape command, the syntax is as follows:

\parshape=*n* i_1 l_1 i_2 l_2 ... i_n l_n

where *n* is the number of (i_j l_j) pairs and i_j specifies the left indentation for the *j*th line and l_j specifies the length of the *j*th line.

The static frame on the top right was assigned a zigzag shape using:

```
\setstaticframe*{shapedt}{shape={\parshape=20
0.6\linewidth 0.4\linewidth 0.5\linewidth 0.4\linewidth
0.4\linewidth 0.4\linewidth 0.3\linewidth 0.4\linewidth
0.2\linewidth 0.4\linewidth 0.1\linewidth 0.4\linewidth
0pt 0.4\linewidth 0.1\linewidth 0.4\linewidth
0.2\linewidth 0.4\linewidth 0.3\linewidth 0.4\linewidth
0.4\linewidth 0.4\linewidth 0.5\linewidth 0.4\linewidth
0.6\linewidth 0.4\linewidth 0.5\linewidth 0.4\linewidth
0.4\linewidth 0.4\linewidth 0.3\linewidth 0.4\linewidth
0.2\linewidth 0.4\linewidth 0.1\linewidth 0.4\linewidth
0pt 0.4\linewidth 0.1\linewidth 0.4\linewidth
}}
```

The syntax for \shapepar is more complicated, see the shapepar documentation for more details. In general:

\shapepar{⟨*shape specs*⟩}

The shapepar package has four predefined shapes: \squareshape, \diamondshape, \heartshape and \nutshape.

The static frame on the bottom right was assigned a heart shape using the command:

```
\setstaticframe*{shapedb}{shape={\shapepar\heartshape}}
```

To reset the frame back to its original rectangular shape do:

```
\setstaticframe*{shapedb}{shape=\relax}
```

The flowfram package currently does not support any other paragraph shape making commands. Any other commands would have to be used explicitly within the contents of the frame.

This is an example of a static frame with a non-rectangular shape. This zigzag shape was specified using the shape key setting in \setstaticframe. The \parshape command was used to set the shape.

Using the shape key rather than explicitly using \parshape within the staticcontents environment means that I can have paragraph breaks, sectioning commands, and even some mathematics

$$E = mc^2 \qquad (3.1)$$

whilst retaining the shape.

This example has a more complicated shape that can not be generated using TEX's \parshape command, so \shapepar was used instead. Note that this document must include the shapepar package in this instance, whereas no extra packages are required to use \parshape. No mathematics or sectioning commands are allowed here. The shape will expand as more text is added to it.

4. Locations and Dimensions

This chapter describes some of the commands provided to determine the locations and dimensions of frames.

This chapter describes some of the commands available that can be used to determine the locations and dimensions of frames. See the accompanying document `flowfram.pdf` for more details of these commands or for other commands not listed here.

4.1 Determining the Location of the Typeblock

As mentioned earlier, when you create new frames, you must specify their location relative to the typeblock, but what if you want to position a frame a set distance from the edge of the paper? The flowfram package provides the following commands that compute the distance from the typeblock to the paper boundary:

`\computeleftedgeodd{`⟨*length*⟩`}`

This computes the position of the left edge of the (odd) page, relative to the left side of the typeblock, and stores the result in ⟨*length*⟩.

`\computeleftedgeeven{`⟨*length*⟩`}`

As above, but for even pages.

`\computetopedge{`⟨*length*⟩`}`

This computes the top edge of the page, relative to the bottom of the typeblock, and stores the result in ⟨*length*⟩.

`\computebottomedge{`⟨*length*⟩`}`

This computes the bottom edge of the page, relative to the bottom of the typeblock, and stores the result in ⟨*length*⟩.

`\computerightedgeodd{`⟨*length*⟩`}`

This computes the position of the right edge of the (odd) page, relative to the left side of the typeblock, and store the result in ⟨*length*⟩.

`\computerightedgeeven{`⟨*length*⟩`}`

As above, but for even pages.

Note that in all cases ⟨*length*⟩ must be a LaTeX length command.

For example, if you want to create a frame whose bottom left corner is one inch from the left edge of the page and half an inch from the bottom edge of the page (this assumes odd and even pages have the same margins):

```
% define two new lengths to represent the x and y coords
\newlength{\myX}
\newlength{\myY}
% compute the distance from the typeblock to the paper edge
\computeleftedgeodd{\myX}
\computebottomedge{\myY}
% Add the absolute co-ordinates to get co-ordinates
% relative to the typeblock
\addtolength{\myX}{1in}
\addtolength{\myY}{0.5in}
```

4.2 Determining the Dimensions and Locations of Frames

It is possible to determine the dimensions and locations of a frame using one of the following commands:

- `\getstaticbounds{`⟨*IDN*⟩`}`
- `\getstaticbounds*{`⟨*IDL*⟩`}`
- `\getflowbounds{`⟨*IDN*⟩`}`
- `\getflowbounds*{`⟨*IDL*⟩`}`
- `\getdynamicbounds{`⟨*IDN*⟩`}`
- `\getdynamicbounds*{`⟨*IDL*⟩`}`

For each command, the starred version takes an IDL as the argument, and the unstarred version takes an IDN as the argument. Each command stores the relevant information in the lengths `\ffareawidth`, `\ffareaheight`, `\ffareax` and `\ffareay`.

For other related commands, see the section “Determining Dimensions and Locations” in the accompanying document `flowfram.pdf`.

4.3 Relative Locations

To print the relative location of one frame from another do:

```
\relativeframelocation{⟨type1⟩}{⟨idn1⟩}{⟨type2⟩}{⟨idn2⟩}
```

where ⟨*type1*⟩ and ⟨*idn1*⟩ indicate the type and IDN of the first frame, and ⟨*type2*⟩ and ⟨*idn2*⟩ indicate the type and IDN of the second frame. There is also a starred version:

```
\relativeframelocation*{⟨type1⟩}{⟨idl1⟩}{⟨type2⟩}{⟨idll⟩}
```

where ⟨*idl1*⟩ and ⟨*idl2*⟩ indicate the IDL of the first and second frames, respectively. Both the above commands will print one of the following:

- `\FFaboveleft` if the first frame is above left of the second frame.
- `\FFaboveright` if the first frame is above right of the second frame.
- `\FFabove` if the first frame is above the second frame.
- `\FFbelowleft` if the first frame is below left of the second frame.
- `\FFbelowright` if the first frame is below right of the second frame.
- `\FFbelow` if the first frame is below the second frame.
- `\FFleft` if the first frame is to the left of the second frame.
- `\FFright` if the first frame is to the right of the second frame.
- `\FFoverlap` if both frames overlap.

A frame is considered to be above another frame if the bottom edge of the first frame is higher than the top edge of the second frame.

A frame is considered to be below another frame if the top edge of the first frame is lower than the bottom edge of the second frame.

A frame is considered to be to the left of another frame if the right edge of the first frame is to the left of the left edge of the second frame.

A frame is considered to be to the right of another frame if the left edge of the first frame is to the right of the right edge of the second frame.

Note that the relative locations are taken for the current page, regardless of whether either of the two frames are displayed on that page. If the current page is odd, then the frame settings for odd pages will be compared, otherwise the frame settings for even pages will be compared. However remember that the first paragraph of each page retains the settings in place at the start of the paragraph, so if the frames have different locations for odd and even pages, then `\relativeframelocation` may use the wrong page settings if it is used at the start of the page.

For example, this document defined a flow frame labelled `main` (this one) and a dynamic frame labelled `chaphead` which is used to display the chapter headings. The following code

```
The dynamic frame is
\relativeframelocation*{dynamic}{chaphead}{flow}{main}
of the flow frame.
```

produces: The dynamic frame is on the left of the flow frame.

There are some short cut commands for frames of the same type:

`\reldynamicloc{`⟨*idn1*⟩`}{`⟨*idn2*⟩`}`

This is equivalent to:

`\relativeframelocation{dynamic}{`⟨*idn1*⟩`}{dynamic}{`⟨*idn2*⟩`}`

`\relstaticloc{`⟨*idn1*⟩`}{`⟨*idn2*⟩`}`

This is equivalent to:

`\relativeframelocation{static}{`⟨*idn1*⟩`}{static}{`⟨*idn2*⟩`}`

`\relflowloc{`⟨*idn1*⟩`}{`⟨*idn2*⟩`}`

This is equivalent to:

`\relativeframelocation{flow}{`⟨*idn1*⟩`}{flow}{`⟨*idn2*⟩`}`

Each of the above commands also has a starred version that uses the IDL instead of the IDN.

These commands may be used in the optional argument of `\continueonframe` for frames that are on the same page. For example:

```
\begin{dynamiccontents}{1}
Some text in the first dynamic frame that goes on for
```

```
quite a bit longer than this example.
\continueonframe[continued \reldynamicloc{2}{1}]{2}
This text is in the second dynamic frame which is
somewhere on the same page.
\end{dynamiccontents}
```

For additional commands that determine the relative location of one frame from another, see the section “Determining the relative location of one frame from another” in the accompanying document `flowfram.pdf`.

5. Predefined Layouts

This chapter describes commands that create frames arranged in a predefined layout.

The flowfram package has a number of commands which create frames in a predefined layout. These commands may only be used in the preamble.

5.1 Column Styles

The standard LaTeX commands `\onecolumn` and `\twocolumn` are redefined to create one or two flow frames that fill the entire typeblock separated from each other (in the case of `\twocolumn`) by a gap of width `\columnsep`. The height of these flow frames may not be exactly as high as the typeblock, as their height is adjusted to make them an integer multiple of `\baselineskip`. You can switch off this automatic adjustment using the command:

```
\ffvadjustfalse
```

The `\onecolumn` and `\twocolumn` commands also take an optional argument which is the page list for which those flow frames are defined. In addition to `\onecolumn` and `\twocolumn`, the following commands are also defined:

```
\Ncolumn[⟨pages⟩]{⟨n⟩}
```

This creates ⟨*n*⟩ column flow frames each separated by a distance of `\columnsep`.

```
\onecolumninarea[⟨pages⟩]{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}
```

This creates a single flow frame to fill the given area, adjusting the height so that it is an integer multiple of `\baselineskip`.

```
\twocolumninarea[⟨pages⟩]{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}
```

This creates two column flow frames separated by a distance of `\columnsep` filling the entire area specified, again adjusting the height so that it is an integer multiple of `\baselineskip`. The columns are separated by a gap of `\columnsep`.

```
\Ncolumninarea[⟨pages⟩]{⟨n⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}
```

This is a more general form of `\twocolumninarea` making ⟨*n*⟩ flow frames instead of two.

5.2 Column Styles with an Additional Frame

As well as the column-style flow frames defined above, it is also possible to define ⟨*n*⟩ columns with an additional frame spanning either above or below them. There will be a vertical gap of approximately[1] `\vcolumnsep` between the columns and the extra frame. In each of the following definitions, the argument ⟨*pages*⟩ is the page list for which the frames are defined, ⟨*n*⟩ is the number of columns required, ⟨*type*⟩ is the type of frame to go above or below the columns (this may be one of: `flow`, `static` or `dynamic`). The area in which the new frames should fill is defined by ⟨*width*⟩, ⟨*height*⟩ (the width and height of the area) and ⟨*x*⟩, ⟨*y*⟩ (the position of the bottom left hand corner of the area relative to the bottom left hand corner of the typeblock.)

The height of the additional frame at the top or bottom of the columns is given by ⟨*H*⟩.

`\onecolumntopinarea[`⟨*pages*⟩`]{`⟨*type*⟩`}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This creates one flow frame with a ⟨*type*⟩ frame above it, filling the area specified.

`\twocolumntopinarea[`⟨*pages*⟩`]{`⟨*type*⟩`}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This creates two column-style flow frames with a ⟨*type*⟩ frame above them, filling the area specified.

`\Ncolumntopinarea[`⟨*pages*⟩`]{`⟨*type*⟩`}{`⟨*n*⟩`}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This creates ⟨*n*⟩ column-style flow frames with a ⟨*type*⟩ frame above them, filling the area specified.

`\onecolumnbottominarea[`⟨*pages*⟩`]{`⟨*type*⟩`}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This creates one flow frame with a ⟨*type*⟩ frame underneath it, filling the area specified.

`\twocolumnbottominarea[`⟨*pages*⟩`]{`⟨*type*⟩`}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This creates two column-style flow frames with a ⟨*type*⟩ frame below them, filling the area specified.

`\Ncolumnbottominarea[`⟨*pages*⟩`]{`⟨*type*⟩`}{`⟨*n*⟩`}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This creates ⟨*n*⟩ column-style flow frames with a ⟨*type*⟩ frame below them, filling the area specified.

[1] It may not be exact, as the flow frames are adjusted so that their height is an integer multiple of `\baselineskip`, which may increase the gap.

The following commands are special cases of the above:

`\onecolumnStopinarea[`⟨*pages*⟩`]{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This is equivalent to:
`\onecolumntopinarea[`⟨*pages*⟩`]{static}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

`\onecolumnDtopinarea[`⟨*pages*⟩`]{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This is equivalent to:
`\onecolumntopinarea[`⟨*pages*⟩`]{dynamic}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

`\onecolumntop[`⟨*pages*⟩`]{`⟨*type*⟩`}{`⟨*H*⟩`}`

As `\onecolumntopinarea` where the area is the entire typeblock.

`\onecolumnStop[`⟨*pages*⟩`]{`⟨*H*⟩`}`

This is equivalent to: `\onecolumntop[`⟨*pages*⟩`]{static}{`⟨*H*⟩`}`

`\onecolumnDtop[`⟨*pages*⟩`]{`⟨*H*⟩`}`

This is equivalent to: `\onecolumntop[`⟨*pages*⟩`]{dynamic}{`⟨*H*⟩`}`

`\twocolumnStopinarea[`⟨*pages*⟩`]{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This is equivalent to:
`\twocolumntopinarea[`⟨*pages*⟩`]{static}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

`\twocolumnDtopinarea[`⟨*pages*⟩`]{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

This is equivalent to:
`\twocolumntopinarea[`⟨*pages*⟩`]{dynamic}{`⟨*H*⟩`}{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*x*⟩`}{`⟨*y*⟩`}`

`\twocolumntop[`⟨*pages*⟩`]{`⟨*type*⟩`}{`⟨*H*⟩`}`

As `\twocolumntopinarea` where the area is the entire typeblock.

\twocolumnStop[⟨pages⟩]{⟨H⟩}

This is equivalent to: \twocolumntop[⟨pages⟩]{static}{⟨H⟩}

\twocolumnDtop[⟨pages⟩]{⟨H⟩}

This is equivalent to: \twocolumntop[⟨pages⟩]{dynamic}{⟨H⟩}

\NcolumnStopinarea[⟨pages⟩]{⟨n⟩}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

This is equivalent to:
\Ncolumntopinarea[⟨pages⟩]{static}{⟨n⟩}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

\NcolumnDtopinarea[⟨pages⟩]{⟨n⟩}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

This is equivalent to:
\Ncolumntopinarea[⟨pages⟩]{dynamic}{⟨n⟩}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

\Ncolumntop[⟨pages⟩]{⟨type⟩}{⟨n⟩}{⟨H⟩}

As \Ncolumntopinarea where the area is the entire typeblock.

\NcolumnStop[⟨pages⟩]{⟨n⟩}{⟨H⟩}

This is equivalent to: \Ncolumntop[⟨pages⟩]{static}{⟨n⟩}{⟨H⟩}

\NcolumnDtop[⟨pages⟩]{⟨n⟩}{⟨H⟩}

This is equivalent to: \Ncolumntop[⟨pages⟩]{dynamic}{⟨n⟩}{⟨H⟩}

\onecolumnSbottominarea[⟨pages⟩]{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

This is equivalent to:
\onecolumnbottominarea[⟨pages⟩]{static}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

\onecolumnDbottominarea[⟨pages⟩]{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

This is equivalent to:

\onecolumnbottominarea[⟨pages⟩]{dynamic}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

\onecolumnbottom[⟨pages⟩]{⟨type⟩}{⟨H⟩}

As \onecolumnbottominarea where the area is the entire typeblock.

\onecolumnSbottom[⟨pages⟩]{⟨H⟩}

This is equivalent to: \onecolumnbottom[⟨pages⟩]{static}{⟨H⟩}

\onecolumnDbottom[⟨pages⟩]{⟨H⟩}

This is equivalent to: \onecolumnbottom[⟨pages⟩]{dynamic}{⟨H⟩}

\twocolumnSbottominarea[⟨pages⟩]{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

This is equivalent to:
\twocolumnbottominarea[⟨pages⟩]{static}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

\twocolumnDbottominarea[⟨pages⟩]{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

This is equivalent to:
\twocolumnbottominarea[⟨pages⟩]{dynamic}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

\twocolumnbottom[⟨pages⟩]{⟨type⟩}{⟨H⟩}

As \twocolumnbottominarea where the area is the entire typeblock.

\twocolumnSbottom[⟨pages⟩]{⟨H⟩}

This is equivalent to: \twocolumnbottom[⟨pages⟩]{static}{⟨H⟩}

\twocolumnDbottom[⟨pages⟩]{⟨H⟩}

This is equivalent to: \twocolumnbottom[⟨pages⟩]{dynamic}{⟨H⟩}

\NcolumnSbottominarea[⟨pages⟩]{⟨n⟩}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}

This is equivalent to:
`\Ncolumnbottominarea[⟨pages⟩]{static}{⟨n⟩}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}`

`\NcolumnDbottominarea[⟨pages⟩]{⟨n⟩}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}`

This is equivalent to:
`\Ncolumnbottominarea[⟨pages⟩]{dynamic}{⟨n⟩}{⟨H⟩}{⟨width⟩}{⟨height⟩}{⟨x⟩}{⟨y⟩}`

`\Ncolumnbottom[⟨pages⟩]{⟨type⟩}{⟨n⟩}{⟨H⟩}`

As `\Ncolumnbottominarea` where the area is the entire typeblock.

`\NcolumnSbottom[⟨pages⟩]{⟨n⟩}{⟨H⟩}`

This is equivalent to: `\Ncolumnbottom[⟨pages⟩]{static}{⟨n⟩}{⟨H⟩}`

`\NcolumnDbottom[⟨pages⟩]{⟨n⟩}{⟨H⟩}`

This is equivalent to: `\Ncolumnbottom[⟨pages⟩]{dynamic}{⟨n⟩}{⟨H⟩}`

5.3 Right to Left Columns

The default behaviour for the commands defined above is to create the flow frames from left to right. However if you are typesetting from right to left, you will probably prefer to define the flow frames in the reverse order. This can be done via the package option RL. Alternatively you can use the command:

```
\lefttorightcolumnsfalse
```

before using commands such as `\twocolumn` or `\Ncolumn`. Two switch back to left-to-right columns use:

```
\lefttorightcolumnstrue
```

5.4 Backdrop Effects

Static frames can be used to produce a backdrop. There are a number of commands which create static frames that can be used as a backdrop. In the following definitions, ⟨*pages*⟩ is the page list for which those static frames are defined (`all` is the default). For the vertical strips: ⟨*xoffset*⟩ is the amount by which the frames should be shifted horizontally (0pt by default), ⟨*W1*⟩ is the width of the first frame, with colour specified by ⟨*C1*⟩ and IDL ⟨*L1*⟩, and so on up to ⟨*Wn*⟩ the width of the ⟨*n*⟩th frame with colour specified by ⟨*Cn*⟩ and IDL ⟨*Ln*⟩. For the vertical strips: ⟨*yoffset*⟩ is the amount by which the frames should be shifted vertically (0pt by default), ⟨*H1*⟩ is the height of the first frame, with colour specified by ⟨*C1*⟩ and IDL ⟨*L1*⟩, and so on up to ⟨*Hn*⟩ the height of the ⟨*n*⟩th frame with colour specified by ⟨*Cn*⟩ and IDL ⟨*Ln*⟩.

NOTE: unlike the earlier commands, these commands are all relative to the actual page, not the typeblock. So an *x* offset of 0pt indicates the first vertical frame is flush with the left hand edge of the page, and a *y* offset of 0pt indicates the first horizontal frame is flush with the bottom edge of the page. This is because backdrops tend to span the entire page, not just the typeblock.

The colour specification must be completely enclosed in braces, for example `{[rgb]{1,0,1}}` not `[rgb]{1,0,1}`.

5.4.1 Vertical stripe effects

`\vtwotone[`⟨*pages*⟩`][`⟨*xoffset*⟩`]{`⟨*W1*⟩`}{`⟨*C1*⟩`}{`⟨*L1*⟩`}{`⟨*W2*⟩`}{`⟨*C2*⟩`}{`⟨*L2*⟩`}`

Defines two static frames to create a two tone vertical strip effect. (This command was used to create the coloured background on the title page of this document.)

`\vNtone[`⟨*pages*⟩`][`⟨*xoffset*⟩`]{`⟨*n*⟩`}{`⟨*W1*⟩`}{`⟨*C1*⟩`}{`⟨*L1*⟩`}...{`⟨*Wn*⟩`}{`⟨*Cn*⟩`}{`⟨*Ln*⟩`}`

This is similar to `\vtwotone` but with ⟨*n*⟩ static frames instead of two.

`\vtwotonebottom[`⟨*pages*⟩`][`⟨*xoffset*⟩`]{`⟨*H*⟩`}{`⟨*W1*⟩`}{`⟨*C1*⟩`}{`⟨*L1*⟩`}{`⟨*W2*⟩`}{`⟨*C2*⟩`}{`⟨*L2*⟩`}`

This is similar to `\vtwotone` but the static frames are only ⟨*H*⟩ high, instead of the entire height of the page. The frames are aligned along the bottom edge of the page.

`\vtwotonetop[`⟨*pages*⟩`][`⟨*xoffset*⟩`]{`⟨*H*⟩`}{`⟨*W1*⟩`}{`⟨*C1*⟩`}{`⟨*L1*⟩`}{`⟨*W2*⟩`}{`⟨*C2*⟩`}{`⟨*L2*⟩`}`

This is similar to `\vtwotone` but the static frames are only ⟨*H*⟩ high, instead of the entire height of the

page. The frames are aligned along the top edge of the page. (This command was used to create the border effect along the top of every page in this document. Two \vtwotonetop commands were used, one for the even pages, and the other for the odd pages.)

\vNtonebottom[⟨pages⟩][⟨xoffset⟩]{⟨H⟩}{⟨n⟩}{⟨W1⟩}{⟨C1⟩}{⟨L1⟩}...{⟨Wn⟩}{⟨Cn⟩}{⟨Ln⟩}

This is a general version of \vtwotonebottom for ⟨n⟩ frames instead of two.

\vNtonetop[⟨pages⟩][⟨xoffset⟩]{⟨H⟩}{⟨n⟩}{⟨W1⟩}{⟨C1⟩}{⟨L1⟩}...{⟨Wn⟩}{⟨Cn⟩}{⟨Ln⟩}

This is a general version of \vtwotonetop for ⟨n⟩ frames instead of two.

5.4.2 Horizontal stripe effect

\htwotone[⟨pages⟩][⟨yoffset⟩]{⟨H1⟩}{⟨C1⟩}{⟨L1⟩}{⟨H2⟩}{⟨C2⟩}{⟨L2⟩}

This defines two static frames to create a two tone horizontal strip effect.

\hNtone[⟨pages⟩][⟨yoffset⟩]{⟨n⟩}{⟨H1⟩}{⟨C1⟩}{⟨L1⟩}...{⟨Hn⟩}{⟨Cn⟩}{⟨Ln⟩}

This is similar to \htwotone but with ⟨n⟩ static frames instead of two.

\htwotoneleft[⟨pages⟩][⟨yoffset⟩]{⟨W⟩}{⟨H1⟩}{⟨C1⟩}{⟨L1⟩}{⟨H2⟩}{⟨C2⟩}{⟨L2⟩}

This is similar to \htwotone but the static frames are only ⟨W⟩ wide, instead of the entire width of the page. The frames are aligned along the left edge of the page.

\htwotoneright[⟨pages⟩][⟨yoffset⟩]{⟨W⟩}{⟨H1⟩}{⟨C1⟩}{⟨L1⟩}{⟨H2⟩}{⟨C2⟩}{⟨L2⟩}

This is similar to \htwotone but the static frames are only ⟨W⟩ wide, instead of the entire width of the page. The frames are aligned along the right edge of the page.

\hNtoneleft[⟨pages⟩][⟨yoffset⟩]{⟨W⟩}{⟨n⟩}{⟨H1⟩}{⟨C1⟩}{⟨L1⟩}...{⟨Hn⟩}{⟨Cn⟩}{⟨Ln⟩}

This is a general version of \htwotoneleft for ⟨n⟩ frames instead of two.

\hNtoneright[⟨pages⟩][⟨yoffset⟩]{⟨W⟩}{⟨n⟩}{⟨H1⟩}{⟨C1⟩}{⟨L1⟩}...{⟨Hn⟩}{⟨Cn⟩}{⟨Ln⟩}

This is a general version of \htwotoneright for ⟨*n*⟩ frames instead of two.

5.4.3 Background Frame

To make a single static frame covering the entire page, use:

```
\makebackgroundframe[⟨pages⟩][⟨IDL⟩]
```

Note that this frame should be created before any other static frame as it will obscure all other static frames created before it if it is given a background colour.

5.4.4 Vertical and Horizontal Rules

You can create vertical or horizontal rules between two frames using the commands:

```
\insertvrule[⟨y top⟩][⟨y bottom⟩]{⟨T1⟩}{⟨IDN1⟩}{⟨T2⟩}{⟨IDN2⟩}
```

This creates a new static frame which fits between ⟨*T1*⟩ frame with IDN ⟨*IDN1*⟩ and ⟨*T2*⟩ frame with IDN ⟨*IDN2*⟩, and places a vertical rule in it extending from the highest point of the highest frame to the lowest point of the lowest frame. The first optional argument ⟨*y top*⟩ (default 0pt) extends the rule by that much above the highest point, and the second optional argument ⟨*y bottom*⟩ (default 0pt) extends the rule by that much below the lowest point. If either of the optional arguments are negative, the rule will be shortened instead of extended. The width of the rule is given by

```
\ffcolumnseprule
```

Note that this has changed as from version 1.09: versions prior to 1.09 used \columnseprule.

The vertical rule drawn by \insertvrule is created using the command:

```
\ffvrule{offset}{width}{height}
```

This can be redefined if a more ornate rule is required (see below).

```
\inserthrule[⟨x left⟩][⟨x right⟩]{⟨T1⟩}{⟨IDN1⟩}{⟨T2⟩}{⟨IDN2⟩}
```

This creates a new static frame which fits between ⟨*T1*⟩ frame with IDN ⟨*IDN1*⟩ and ⟨*T2*⟩ frame with IDN ⟨*IDN2*⟩, and places a horizontal rule in it extending from the leftmost point of the left frame to the rightmost point of the right frame. The first optional argument ⟨*x left*⟩ (default 0pt) extends the rule by

that much before the leftmost point, and the second optional argument ⟨*x right*⟩ (default 0pt) extends the rule by that much beyond the rightmost point. If either of the optional arguments are negative, the rule will be shortened instead of extended. The height of the rule is given by

```
\ffcolumnseprule
```

The horizontal rule drawn by `\inserthrule` is created using the command:

```
\ffhrule{offset}{width}{height}
```

This can be redefined if a more ornate rule is required (see below).

The default value for `\ffcolumnseprule` is 2pt. Both `\insertvrule` and `\inserthrule` have starred versions which allow you to identify the frame by IDL instead of IDN. The frame types, ⟨*T1*⟩ and ⟨*T2*⟩ can be one of the following keywords: `flow`, `static` or `dynamic`.

The command

```
\ffruledeclarations
```

can be redefined to set declarations that affect how the rule is drawn. The most likely use of this command is to set the rule colour. For example:

```
\twocolumnStop{2in}

\renewcommand{\ffruledeclarations}{\color{red}}
\insertvrule{flow}{1}{flow}{2}

\renewcommand{\ffruledeclarations}{\color{blue}}
\inserthrule{static}{1}{flow}{1}
```

This will create a layout with two columns (flow frames 1 and 2) with a static frame above. A red vertical rule is placed in a static frame between flow frames 1 and 2, and a blue horizontal rule is placed between the static frame and the first flow frame. (However the horizontal rule will span both flow frames since that is the width of the static frame.)

In the following example, the rules have been redefined to use a zigzag pattern (which is obtained using the tikz package):

```
\usepackage{flowfram}
\usepackage{tikz}
\usetikzlibrary{snakes}
```

```
\twocolumnStop

\renewcommand{\ffvrule}[3]{%
\hfill
\tikz{\draw[snake=zigzag,line width=#2,yshift=-#1] (0,0) -- (0pt,#3);}%
\hfill\mbox{}}

\insertvrule{flow}{1}{flow}{2}

\renewcommand{\ffhrule}[3]{%
\tikz{\draw[snake=zigzag,line width=#3,xshift=-#1] (0,0) -- (#2,0pt);}}

\inserthrule{static}{1}{flow}{1}
```

6. Thumbtabs and Minitocs

This chapter describes how to create thumbtabs and minitocs, such as those used in this document.

6.1 Thumbtabs

On the right hand side of this page, there is a blue rectangle with the chapter number in it. This is a thumbtab, and it gives you a rough idea whereabouts in the document you are when you quickly flick through the pages. Each thumbtab is in fact a dynamic frame, and you can control whether to make the number and/or title appear in the thumbtab by using one or more of the package options: ttbtitle (show title—default), ttbnotitle (don't show the title), ttbnum (show the number) and ttbnonum (don't show the number—default).

If you want thumbtabs in your document, you need to use the command

```
\makethumbtabs[⟨y offset⟩]{⟨height⟩}[⟨section type⟩]
```

in the document preamble. By default, the topmost thumbtab is level with the top of the typeblock, but can be shifted vertically using the first optional argument ⟨*y offset*⟩. Each thumbtab will be ⟨*height*⟩ high, and will correspond to the sectioning type ⟨*section type*⟩. If ⟨*section type*⟩ is omitted, chapters will be used if the `\chapter` command is defined, otherwise sections will be used. The width of the thumbtabs is given by the length

```
\thumbtabwidth
```

which is 1cm by default. The command

```
\thumbtabindex
```

will display the thumbtab index (all thumbtabs) on the current page. You then need to use

```
\enablethumbtabs
```

to start the individual thumbtabs and

```
\disablethumbtabs
```

to make them go away. You can align the table of contents with the thumbtabs[1] using the command

```
\tocandthumbtabindex
```

instead of the commands `\tableofcontents` and `\thumbtabindex`. If you are using the hyperref package, the text on the thumbtab index will be a hyperlink to the corresponding part of the document.

[1] but only do this if there is enough room on the page!

Note that when using \tocandthumbtabindex you may need to shift the thumbtabs vertically up or down to make sure that they align correctly with the table of contents.

The format of the text on the thumbtabs is given by the command

```
\thumbtabindexformat
```

for the thumbtab index entries, and

```
\thumbtabformat
```

for the individual thumbtabs. By default the text on the thumbtabs will be rotated, but as rotating is not implemented by some previewers, the package option norotate is provided, which will stack the letters vertically. This does not look as good as the rotated text. Note also that some previewers do not put the hyperlink in the correct place when the link has been rotated, so this may also cause a problem.

The thumbtab attributes can be changed using

```
\setthumbtab{⟨n⟩}{⟨key value list⟩}
```

where ⟨*n*⟩ is the thumbtab number starting from 1 (for the top thumbtab) to the value given by the counter maxthumbtabs (for the bottom thumbtab). Note that these numbers are not related to the associated frame IDN. You may also use the keyword `all` instead of ⟨*n*⟩ to indicate that the new attributes should apply to all thumbtabs.

To just change the settings for the thumbtab index, use

```
\setthumbtabindex{⟨n⟩}{⟨key value list⟩}
```

The ⟨*key value list*⟩ for both these commands is the same as that for `\setdynamicframe`. Again ⟨*n*⟩ may either be the thumbtab index or the keyword `all`.

By default, the thumbtabs have a grey background. In this document, I have used:

```
\setthumbtab{1}{backcolor=[rgb]{0.15,0.15,1}}
\setthumbtab{2}{backcolor=[rgb]{0.2,0.2,1}}
\setthumbtab{3}{backcolor=[rgb]{0.25,0.25,1}}
\setthumbtab{4}{backcolor=[rgb]{0.3,0.3,1}}
\setthumbtab{5}{backcolor=[rgb]{0.35,0.35,1}}
\setthumbtab{6}{backcolor=[rgb]{0.4,0.4,1}}
\setthumbtab{7}{backcolor=[rgb]{0.45,0.45,1}}
\setthumbtab{8}{backcolor=[rgb]{0.5,0.5,1}}
```

to change the thumbtab background colour to shades of blue.

I have also changed the style of the thumbtab text using:

```
\newcommand{\thumbtabstyle}[1]{{\hypersetup{linkcolor=white}%
\textbf{\large\sffamily #1}}}
\setthumbtab{all}{style=thumbtabstyle,textcolor=white}
```

Note that the style uses `\hypersetup`[2] to change the colour of the hyperlink text, since the hyperlink overrides the text colour.

6.2 Minitocs

In this document, after each chapter heading, there is a mini table of contents for that chapter. To enable minitocs, use the command

`\enableminitoc[`⟨*section type*⟩`]`

The default ⟨*section type*⟩ is the same as that used by the thumbtabs.

If you want the minitocs to appear in a dynamic frame, you can use

`\appenddfminitoc{`⟨*IDN*⟩`}`

where ⟨*IDN*⟩ is the IDN of the appropriate dynamic frame. There is also a starred version available if you want to use the IDL instead of the IDN.

For example, in this document I have used the command:

```
\appenddfminitoc*{chaphead}
```

in the preamble, which has appended the minitocs to the dynamic frame with IDL `chaphead`.

The style of the minitoc text is given by the command

`\minitocstyle{`⟨*contents*⟩`}`

where the argument is the contents of the minitoc. This command may be redefined if you want to change the minitoc style. The gap before the minitoc is given by the length

`\beforeminitocskip`

[2] defined by the hyperref package

and the gap after the minitoc is given by the length

```
\afterminitocskip
```

These lengths may be changed using `\setlength`.

7. Global Settings

This section describes style macros, lengths and counters used by the flowfram package.

7.1 Macros

The following macros can be changed using `\renewcommand`:

- `\setffdraftcolor`

  This sets the colour of the bounding box when it is displayed in draft mode. The default value is: `\color[gray]{0.8}`. For example, if you want a darker grey, do:

  ```
  \renewcommand{\setffdraftcolor}{\color[gray]{0.3}}
  ```

- `\setffdrafttypeblockcolor`

  This sets the colour of the bounding box of the typeblock when it is displayed in draft mode. The default value is: `\color[gray]{0.9}`. For example, if you want a medium grey, do:

  ```
  \renewcommand{\setffdrafttypeblockcolor}{\color[gray]{0.5}}
  ```

- `\fflabelfont`

  This sets the font size for the bounding box markers in draft mode. The default value is: `\small\sffamily`. For example, if you want a larger font, do:

  ```
  \renewcommand{\fflabelfont}{\large\sffamily}
  ```

- `\ffruledeclarations`

  This sets the declarations that affect the rules created using `\insertvrule` and `\inserthrule`. The default definition does nothing. See subsection 5.4.4 for further details.

- `\ffcontinuedtextfont{`⟨*text*⟩`}`

  This sets ⟨*text*⟩ in the continuation font. The default definition does `\emph{\small` ⟨*text*⟩`}`. See section 2.2 and section 2.3 for further details.

7.2 Lengths

The following are lengths, which can be changed using `\setlength`:

- `\fflabelsep`

  This is the distance from the right hand side of the bounding box at which to place the bounding box marker. The default value is: `1pt`

- `\flowframesep`

  This is the gap between the text of the frame and its border, for the standard border types.

- `\flowframerule`

  This is the width of the frame's border, if using a border given by a frame making command that uses `\fboxsep` to set its border width (e.g. `\fbox`).

- `\sdfparindent`

  This is the paragraph indentation within static or dynamic frames. The default value is 0pt.

- `\vcolumnsep`

  This is the approximate vertical distance between the top frame and the column frames when using `\Ncolumntop` etc. (The height of the flow frame may be adjusted to make it an integer multiple of `\baselineskip`.)

- `\columnsep`

  This is the horizontal distance between the column frames when using `\Ncolumn` or `\Ncolumntop` etc

- `\ffcolumnseprule`

  This is the width of vertical rules created using `\insertvrule` or the height of horizontal rules created using `\inserthrule`.

- `\beforeminitocskip`

  This is the vertical distance before the minitoc.

- `\afterminitocskip`

  This is the vertical distance after the minitoc.

7.3 Counters

The following are counters that can be accessed via `\value{`⟨*counter name*⟩`}` or via `\the`⟨*counter name*⟩. However the value of these counters should not be modified.

maxflow The total number of flow frames that have been defined so far.

thisframe Stores the IDN of the current flow frame. You can label and reference the IDN using

```
\labelflowid{⟨label⟩}
```

This is analogous to the standard `\label` command, but will always refer to the IDN of the current flow frame. It can then be referenced using `\ref{⟨label⟩}`. Note that this will always refer to the current flow frame even when used in the contents of a static or dynamic frame.

Don't use more than one instance of `\labelflowid` in a given flow frame for a given page or you will get a "multiply defined references" warning.

displayedframe Stores the index of the currently displayed flow frame. This will be the same as the IDN if all flow frames are displayed on the current page, but if some are hidden, the values may be different. You can label this counter using

```
\labelflow{⟨label⟩}
```

and reference it elsewhere in the document using `\ref{⟨label⟩}`. For example, if you are using a column layout, you might want to do something like:

```
This text is about hippos\labelflow{hippos}.

% Somewhere else in the document
See column~\ref{hippos} on page~\pageref{hippos}
for information on hippos.
```

Don't use more than one instance of `\labelflow` in a given flow frame for a given page or you will get a "multiply defined references" warning. Note that `\labelflow` will always refer to the current flow frame even when used in the contents of a static or dynamic frame.

maxstatic The total number of static frames that have been defined so far.

maxdynamic The total number of dynamic frames that have been defined so far.

maxthumbtabs The total number of thumbtabs.

8. Troubleshooting

This chapter should be consulted if you experience any problems using the flowfram package.

For an up-to-date list of frequently asked questions, see http://theoval.cmp.uea.ac.uk/~nlct/latex/packages/faq/flowframfaq.html. If you have a query that is not addressed here, please try there before contacting me.

8.1 General Queries

1. If all my flow frames are only defined on, say, pages 1-10, what happens if I then add some extra text so that the document exceeds 10 pages?

   The output routine will create a new flow frame the size of the typeblock and use that.

2. Can I use the formatted page number in page lists?

   No.

3. Why not?

   When the output routine finishes with one flow frame it looks for the next flow frame defined on that page. If there are none left, it then searches through the page list of all the defined flow frames to see if the next page lies in that range. If there are none defined on that page, it ships out that page, and tries the next page. This gives rise to two problems:

   (a) LaTeX is not clairvoyant. If it is currently on page 14, and on the next page the page numbering changes to A, it has no way of knowing this until it has reached that point, which it hasn't yet. So it is looking for a flow frame defined on page 15, not on page A.

   (b) How does LaTeX tell if page C lies between pages A and D? It would require an algorithm that can convert from a formatted number back to an integer. Given that there are many different ways of formatting the value of a counter (besides the standard Roman and alphabetical formats) it would be impossible to write an algorithm to do this for some arbitrary format.

4. Can I have an arbitrarily shaped frame?

   You can assign certain irregular shapes to static or dynamic frames, using the shape key (see section 3.1). Note that the bounding box will still appear as a rectangle with the dimensions of the given frame which may not correspond to the assigned shape. This function is not available for flow frames.

5. Why has the text from my flow frame appeared in a static frame or dynamic frame?

   Assuming you haven't inadvertently set that text as the contents of the static or dynamic frame, the frames are most likely overlapping (see section 1.2). In an attempt to clarify what's going on,

suppose you have defined a static frame, a dynamic frame and two flow frames. The following is an approximate[1] analogy: TEX has a sheet of paper on the table, and has pencilled[2] in a rectangle denoting the typeblock. The paper is put to one side for now. TEX also has four rectangular sheets of transparent paper. The first (which I shall call sheet 1) represents the static frame, the next two (which I shall call sheets 2 and 3) represent the flow frames, and the last one (which I shall call sheet 4) represents the dynamic frame. TEX starts work on filling sheet 2 with the document text. Once it has put as much text on that sheet as it considers possible (according to its views on aesthetics), it puts sheet 2 into the "in tray", and then continues on sheet 3. While it's filling in sheets 2 and 3, if it encounters a command or environment that tells it what to put in the static frame, it fills in sheet 1 and then puts sheet 1 into the "in tray" and resumes where it left off on sheet 2 or 3. Similarly, if it encounters a command that tells it what to put in the dynamic frame, it stops what it's doing, fills in sheet 4, then puts sheet 4 into the "in tray", and resumes where it left off. Only when it has finished sheet 3 (the last flow frame defined on that page), will it gather together all the transparent sheets, and fix them onto the page starting with sheet 1 through to sheet 4, measuring the bottom left hand corner of each transparent sheet relative to the bottom left hand corner of the typeblock. TEX will then put that page aside, and start work on the next page. If two or more of the transparent sheets overlap, you will see through the top one into the one below (unless of course the top one has been painted over, either by setting a background colour, or by adding an image that has a non-transparent background.)

Note that it's also possible that the overlap is caused by an overfull hbox that's causing the text to poke out the side of the flow frame into a neighbouring frame. Check the log file for warnings or use the draft option to the document class which will place a filled rectangle on the end of overfull lines.

6. Why do I get lots of overfull hbox messages?

   Probably because you have narrow frames. It's better to use ragged right formatting when the frames are narrow.

7. Why do I keep getting multiply-defined warnings?

   Probably because you have used `\label` in a static or dynamic frame that is displayed on more than one page. Try using the clear key to ensure that the frame is always cleared at the end of each page.

8. What happens if I use a command or environment that switches to two-column mode (e.g. theindex)?

[1] The pedantic may point out that TEX may make several attempts to fill in the flow frames depending on penalties and so on.
[2] actually it hasn't drawn anything really, but it has in its mind's eye.

As from version 1.01, any \onecolumn or \twocolumn commands that occur outside of the preamble will print the contents of the optional argument, and issue a warning. It is recommended that you set up your own frames for use in the index. See the source code of this document, ffuserguide.tex, for an example.

9. How do I change the vertical alignment of material inside a static or dynamic frame?

   Use the valign key in \setstaticframe or \setdynamicframe (new to version 1.03).

10. How do I compute the distance from the edge of the page instead of the typeblock?

    See section 4.1.

11. Is there a GUI I can use to make it easier to create the frames?

    Yes, Jpgfdraw which can be downloaded from: http://theoval.cmp.uea.ac.uk/~nlct/jpgfdraw/

8.2 Unexpected Output

1. The lines at the beginning of my flow frames are the wrong width.

   This is a problem that will occur if you have flow frames with different widths, as the change in \hsize does not come into effect until a paragraph break. So if you have a paragraph that spans two flow frames, the end of the paragraph at the beginning of the second flow frame will retain the width it had at the start of the paragraph at the bottom of the previous flow frame. You can fix the problem by inserting \framebreak at the point where the frame break occurs (see subsection 2.1.1).

2. My frames shift to the right when I add a border.

   This may occur if you use a border that is not recognised by the flowfram package. You will need to set the offset using the offset key (see chapter 3).

3. I have a vertical white strip along the right hand side of every page.

   This can happen if you have, say, an A4 document, and ghostscript has letter as the default paper size. You can change the default paper size by editing the file gs_init.ps. Change:

   ```
   % Optionally choose a default paper size other than U.S. letter.
   % (a4)
   ```

   to:

```
% Optionally choose a default paper size other than U.S. letter.
(a4)
```

4. I don't have any output.

   All your flow frames are empty. TEX doesn't put the frames onto the page until it has finished putting text into the flow frames. So if there is no text to go in the flow frames it won't output the page. If you only want the static frames or dynamic frames filled in, and nothing outside of them, just do `\mbox{}\clearpage`. This will put an invisible something with zero area into your flow frame, but it's enough to convince TEX that the document contains some text.

5. The last page hasn't appeared.

   See the previous answer.

6. There is no paragraph indentation inside my static or dynamic frames.

   The paragraph indentation in static or dynamic frames is governed by the length `\sdfparindent` which is set to 0pt by default. To make the indentation the same as that used by flow frames place the following in the preamble:

   ```
   \setlength{\sdfparindent}{\parindent}
   ```

7. My section numbering is in the wrong order.

   Remember that the contents of the dynamic frames are not set until the page is shipped out, and the contents will be set in the order of IDN, so if you have any sectioning commands occuring within dynamic frames, they may not be set in the same order as they are in your input file.

8. The contents of my static or dynamic frame have shifted to the left when I used `\parshape`.

   This will happen if your `\parshape` specification exceeds the linewidth. For example:

   ```
   \parshape=1 0.4\linewidth 0.7\linewidth
   ```

   This specifies a line with overall length `1.1\linewidth` which is too long.

8.3 Error Messages

1. `Illegal unit of measure (pt inserted)`

   All lengths must have units. Remember to include the units when defining new frames. The following keys require lengths: `width`, `height`, `x`, `y` and `offset`[3].

[3] `offset` can also have the value `compute`

2. `Missing number, treated as zero`

   LaTeX is expecting a number. There are a number of possible causes:

   (a) You have used an IDL instead of an IDN. If you want to refer to a frame by its label, you need to remember to use the starred versions of the `\set`⟨*type*⟩`frame` commands, or when setting the contents of static frames or dynamic frames.

   (b) When specifying page lists, you have mixed keywords with page ranges. For example: `1,even` is invalid.

3. `Flow frame IDL '`⟨*label*⟩`' already defined`

   All IDLs within each frame type must be unique. There are similar error messages for duplicate IDLs for static frames and dynamic frames.

4. `Can't find flow frame id`

   You have specified a non-existent flow frame IDL. There are similar error messages for static frames and dynamic frames. Check to make sure you have spelt the label correctly, and check you are using the correct frame type command. (For example, if a static frame has the IDL `mylabel`, and you attempt to do `\setflowframe*{mylabel}{`⟨*options*⟩`}`, then you will get this error, because `mylabel` refers to a static frame not a flow frame.)

5. `Key 'clear' is boolean`

   The `clear` key can only have the values `true` or `false`.

6. `Key 'clear' not available`

   The `clear` key is only available for static or dynamic frames.

7. `Key 'style' not available`

   The `style` key is only available for dynamic frames.

8. `Key 'margin' not available`

   The `margin` key is only available for flow frames.

9. `Key 'shape' not available`

   The `shape` key is only available for static or dynamic frames.

10. `Dynamic frame style '`⟨*style*⟩`' not defined`

    The specified style ⟨*style*⟩ must be the name of a command without the preceding backslash. It is possible that you have mis-spelt the name, or you have forgotten to define the command.

11. `Argument of \fbox has an extra }`

    This error will occur if you do, say, `border=\fbox` instead of `border=fbox`. Remember not to include the initial backslash.

12. `Not in outer par mode`

    You can not have floats (such as figures, tables or marginal notes) in static or dynamic frames. If you want a figure or table within a static or dynamic frame use staticfigure or statictable.

13. `Somethings wrong---maybe missing \item`

    Assuming that all your list type of environments start with `\item`, this may be caused by something going wrong with the toc (table of contents), ttb (thumbtab) or aux (auxiliary) files in the previous run. Try deleting them, and try again.

14. `No room for a new \skip`

    You have exceeded TEX's 256 register limit. Use the etex package.

15. `\verb illegal in command argument`

    As a general rule, you can't use verbatim text in a command argument. This rule applies to all the commands defined by the flowfram package. See also below.

16. I get `\verb illegal in command argument` when using verbatim text inside the dynamiccontents environment.

    You can not use verbatim text inside either the starred or unstarred version of the dynamiccontents environment. (See page 12.)

Glossary

bounding box
The bounding box of a frame is the area allocated for the contents of that frame. However the text may not completely fill that area, and it is possible that the text may overflow that area.

dynamic frame
Frames in which text is fixed in place, but the contents are re-typeset each time the frame is displayed.

flow frame
The frames in a document such that the contents of the document environment flow from one frame to the next in the order that they were defined. There must be at least one flow frame on every page.

frame making command
A LaTeX command which places some kind of border around its argument. For example: `\fbox`.

frame
A rectangular area of the page in which text can be placed (not to be confused with a frame making command). There are three types: flow, static and dynamic.

identification label (IDL)
A unique label which can be assigned to a frame, enabling you to refer to the frame by label instead of by its IDN.

identification number (IDN)
A unique number assigned to each frame, which you can use to identify the frame when modifying its appearance. Example: if you have defined 3 flow frames, 2 static frames and 1 dynamic frame, the flow frames will have IDNs 1, 2 and 3, the static frames will have IDNs 1 and 2, and the dynamic frame will have IDN 1.

page list
A list of pages. This can either be a single keyword: `all`, `odd`, `even` or `none`, or it can be a comma-separated list of individual page numbers or page ranges. For example: `<3,5,7-11,>15` indicates pages 1,2,5,7,8,9,10,11 and all pages after page 15. Note that these numbers refer to the actual value of the page counter, not the absolute physical page number.

page range
Page ranges can be closed, e.g. `5-10`, or open, e.g. `<7` or `>9`.

static frame
: Frames in which text is fixed in place. The contents are fixed until explicitly changed or cleared via the `\clear` key in `\setstaticcontents`.

typeblock
: The area of the page where the main body of the text goes. The width and height of this area are given by `\textwidth` and `\textheight`.

Index

Index

L

M

N

O

P

Index

Index

V

Index