Paragraph designer with galley approach

Oleg Parashchenko

Abstract

The LaTeX package `paravesp` controls the space above and below paragraphs.

The python script `parades.py` generates paragraph styles with support of space above, space below and tabulators.

The system imposes the galley approach on the document.

1 Introduction

One layout specification defined the space above and below paragraphs. This is not how does TeX work. To satisfy the requirement, the package `paravesp` (PARAgraph VErtical SPace) was developed.

The solution imposes the galley approach on the document. The paragraphs should be wrapped by a tracking code, which controls how the matherial is added into the TeX vertical list.

The paragraph designer appeared as a generalization of the tracking code to other paragraph properties. The user describes the formatting options in a python file. The program `parades.py` converts the definitions into TeX code.

The system successfully works in the production, but so far is limited to my needs. The complete set of the paragraph properties is not a momentary goal. A switch to the package `xgalley` from the LaTeX3 project might be a step in the future development.

The article starts with the definition what the space between paragraphs is and how it is implemented. The example demonstrates the use of the commands, which are then described using pseudocode.

The paragraph designer is first illustrated by a sample LaTeX fragment, which uses the paragraph styles. For each three types of styles, there are given a sample definition in Python and the result of translation to TeX code, with explanation. Finally, the reference lists all the supported paragraph properties and the commands of the Python `parades.py` tool.

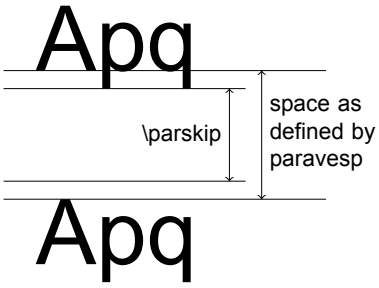
The article concludes with information on how to get the code and run it.

2 Space between paragraphs

The notion "space between paragraphs" can be defined differently.

In one definition, the space between paragraphs is the amount of additional space comparing to what happens inside a paragraph. This is what the most typesetting engines implement, and what is named `parskip` in TeX.

The definition of the package `paravesp` is: the space between paragraphs is the distance between the baseline of the preceeding paragraph and the top of the next paragraph. The code cares that this distance is larger than `prevdepth`.



2.1 Usage

The package `paravesp` imposes restrictions on how to construct a document. Otherwise it can't guarantee the desired space above or below paragraphs.

- Switches between the vertical and horizontal modes be controlled. TeX automation is partially forbidden.
- The register `\parskip` belongs to the controlling code.
- The commands rely on automatic insertion of `parskip` glue by TeX.

The guidelines for the controlling code are:

- At the end of a paragraph (after `\par`) use the command `\ParaSpaceBelow`.
- At the begin of a paragraph, still in the vertical mode, use `\ParaSpaceAbove`.
- At the begin of a block content, for which TeX will not insert `\parskip` automatically, use the both `\ParaSpaceAbove` and `\IssueParaSpace`.

An example:

```
...
\ParaSpaceAbove{20pt}%
{\HeadingStyle Heading}\par
\ParaSpaceBelow{20pt}%
\ParaSpaceAbove{10pt}%
An usual paragraph of text...\par
\ParaSpaceBelow{10pt}%
\ParaSpaceAbove{10pt}%
Yet another usual paragraph of text...\par
\ParaSpaceBelow{10pt}%
\ParaSpaceAbove{20pt}\IssueParaSpace
\vbox{\fbox{Some info-box}}%
\ParaSpaceBelow{20pt}%
...
```

2.2 Technical details

Below is the simplified approximation what happens. The special cases are not shown.

After `\ParaSpaceBelow{length}`:

- vertical list is not changed
- `parskip` := *length* - `prevdepth`
- `prevdepth` is not changed

The command `\ParaSpaceBelow` splits its argument on two lengths, `prevdepth` and `parskip`. This is a precaution for the case if the the next element in the vertical list is not controlled by galley. Thanks to the retained `prevdepth`, a possible layout corruption is avoded.

After `\ParaSpaceAbove{length}`:

- vertical list: `vskip -prevdepth`, penalty as before `vskip`
- `parskip` := max(*length*, *old_ length*)
- `prevdepth` := -1000pt

The command `\ParaSpaceAbove`, which preceedes a paragraph, can't know how much interline glue induced by `baselineskip` will be added. As a solution, the command disables this glue completely by setting `prevdepth` to minus infinity.

After `\IssueParaSpace`:

- vertical list: `vskip parskip`, penalty as before `vskip`
- `parskip` := 0pt
- `prevdepth` := -1000pt

You need the command `\IssueParaSpace` if TEX isn't going to insert `\parskip` automatically, for example, before a box.

The command expects that it is called after `\ParaSpaceAbove`.

After `\IgnoreSpaceAboveNextPara`:

- vertical list is not changed
- parskip := -0.01pt
- prevdepth is not changed

The special case is `parskip` less than 0pt, which cancels the vertical spacing. It is useful when display content (image, list etc) is the first element inside a table cell.

3 Paragraph designer

The paragraph designer transforms Python objects with desired paragraph properties into TEX code which implements these properties.

The main benefit is that the paragraphs definitions can be constructed in such way that the repetitions (for example, font names) can be extracted into common settings.

The system proposes that every block-level element of a document should be wrapped into a command or an environment, which support the galley approach. The suggested sorts of the paragraphs:

- long body text paragraphs, wrapped by an environment,
- short paragraphs, wrapped by a command, and
- short paragraphs with tabstops, also wrapped by a command.

A document, made using this approach, looks structured. Here is an example.

```
\HeadI{Universal Declaration of Human Rights}
\HeadII{Preamble}
\begin{para}Whereas recognition...\end{para}
\begin{para}Whereas disregard
    and contempt...\end{para}
...
\HeadII{Article 14}
\begin{udhrlist}
\listitem{1}{Everyone has the right ...}
\listitem{2}{This right may not be invoked ...}
\end{udhrlist}
```

The sample is generated automatically from the XML source. The generation script, the paragraph styles as Python definition, the `.sty` code, and the PDF-result are included in the package in the directory `udhr`.

3.1 Example: the command "HeadI"

Commands are recommended for small paragraphs, such as headings and captions.

```
\HeadI{Universal Declaration of Human Rights}
```

A sample definition in Python:

```
add_style(ParagraphOptions(cmd='HeadI',
    space_above='20pt',
    space_below='20pt',
    fontsize='12pt', baseline='14pt',
    fontcmd=r'\fontseries{b}\selectfont',
    afterpar=r'\nobreak',
    ))
```

The properties of the paragraph are stored inside the object `ParagraphOptions`. Like in many other programming languages, the backslash-symbol (\) is a control symbol and should appear in strings escaped (\\). An alternative as used in the example is to prefix the string with `"r"`, which disables the control character.

The function `add_style` remembers the object in the global styles-list. At the end of the python

script, the objects in the list are converted to TEX code.

The result of the conversion:

```
\newcommand{\HeadI}[1]{{%
\fontsize{12pt}{14pt}\fontseries{b}\selectfont%
\ParaSpaceAbove{20pt}%
\noindent #1\par}%
\nobreak\ParaSpaceBelow{20pt}}
```

The pecularities are:

- The paragraph is created explicit with `\noindent #1\par`.
- The text and the pre-paragraph settings are inside a group. This way the settings such as font change work only for the given paragraph and do not affect the rest of the document.

3.2 Example: the environment "para"

Environments are recommended to wrap paragraphs in the text body.

```
\begin{para}Whereas recognition...\end{para}
\begin{para}Whereas disregard
    and contempt...\end{para}
```

A sample definition in python:

```
add_style(ParagraphOptions(cmd='paracmd',
    env='para',
    space_above='10pt plus1pt minus1pt',
    ))
```

The result of the conversion in a sty-file:

```
\newenvironment{para}{%
\ParaSpaceAbove{10pt plus1pt minus1pt}%
\noindent \ignorespaces}
{\par\global\def\pd@after@para{%
\ParaSpaceBelow{0pt}}%
\aftergroup\pd@after@para}
```

The paragraph is started explicitely with the command `\noindent` followed by `\ignorespaces` and finished also explicitely with `\par`.

The changes inside an environment, including post-paragraph settings, are local and automatically discarded when the environment group is finished. Therefore, using `\aftergroup`, the post-paragraph settings are applied after the end of the environment.

3.3 Example: tabstops in "listitem"

Paragraph with tabstops are used to implement list items, captions, table of content entries and similar elements. The list paragraphs in the following example have one tabstop to store the list numbering.

```
\listitem{1}{Everyone has the right ...}
\listitem{2}{This right
    may not be invoked ...}
```

A sample definition in python:

```
add_style(ParagraphOptions(cmd='listitem',
    moresetup='\\interlinepenalty=150\\relax',
    space_above='8pt',
    boxes=(('0cm', '0.5cm'),),
    leftskip='0.5cm'))
```

The argument `boxes` is a list of pairs. Each pair gives the offset of the tabstop from left and the width of the box. Due to pecularities of Python, one-element lists of pairs should have an extra comma inside.

The position of the paragraph text should be tuned manually to avoid overlapping with the tabstop boxes. In the example above, the left margin is set to `0.5cm` using `\leftskip`.

The result of the conversion in a sty-file is complicated:

```
\newcommand{\listitem}[2]{{%
\ParaSpaceAbove{8pt}%
\interlinepenalty=150\relax%
\noindent \advance\pd@leftskip by 0.5cm %
\hbox to 0pt{\hss\hbox to 0.5cm{#1\hss}%
\dimen0=0.5cm %
\advance\dimen0 by -0cm %
\advance\dimen0 by -0.5cm \hskip\dimen0}%
\the\everypar #2\par}%
\ParaSpaceBelow{0pt}}
```

The skeleton of the list paragraph are these elements:

```
\noindent tabstops \everypar text \par
```

The use of `\noindent` and `\par` is clear. The paragraph starts with the tabstop boxes, TEX does not insert `\everypar` automatically, therefore the code does it.

The token `\pd@leftskip` is a let-synonym for `\leftskip`. In the right-to-left document you would prefer to set the token to `\rightskip`.

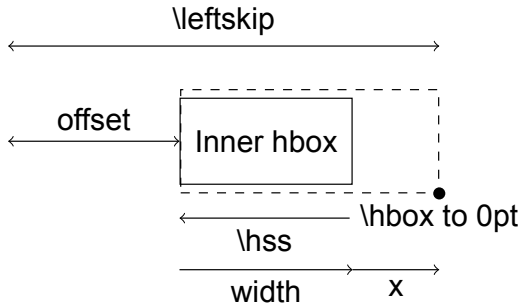
A tabstop is constructed from two nested boxes. The inner box gives the width of the tabstop and aligns the content to the left:

```
\hbox to WIDTH{CONTENT \hss}
```

The outer box puts the inner box at the specified offset.

```
\hbox to 0pt{\hss INNER_BOX%
\dimen0=LEFTSKIP
\advance\dimen0 by -OFFSET
\advance\dimen0 by -WIDTH
\hskip\dimen0}%
```

The calculation is not obvious. The following image provides the source for it.



The image reflects how the boxes, glues and lengths are related. We see that `offset`+`width`+`x` is `leftskip`, therefore `x` (`\dimen0`) is `leftskip` minus `offset` minus `width`.

4 Paragraph designer referense

Denomination. `cmd`, `env`, `stylecmd`. These are the names for the generated commands and environments.

An examples of `cmd` and `env` are already given. The command for `stylecmd` makes a character style, which affects the font and doesn't set the paragraph properties (vertical spacing, tabulars etc).

A sample paragraph definition:

```
ParagraphOptions(cmd="Caption,
  stylecmd="UseCaption", ...)
```

In a LaTeX document you can write:

```
{\UseCaption Article 1.} All human beings
are born free and equal in dignity ...
```

All the three denominators can be mixed together at once. You must specify `cmd` even if you don't need it.

Fonts. `fontsize`, `baseline`, `fontcmd`.

The only supported font properties are its size and baseline. The rest properties, such as width or serie, should be manually defined in `fontcmd`:

```
ParagraphOptions(...,
    fontcmd=r'\fontseries{b}\selectfont',
    ...)
```

Dimensions. `leftskip`, `hsize`, `space_above` and `space_below`.

The names are self-explaining.

The default value for `space_above` and also for `space_below` is `0pt`. It means that if you haven't given a value, then two consequent paragraphs will touch each other, like if `\nointerlineskip` were given between them.

Use the special value `#natural` to disable the use of `\ParaSpaceAbove` or `\ParaSpaceBelow` and activate instead the default TeX behaviour.

```
ParagraphOptions(...,
  space_above='#natural',
  space_below='#natural', ...)
```

Tuning. `moresetup`, `afterpar`, `preamble_arg1`, `preamble_arg2`, `preamble_arg3`, `preamble_arg4`.

The content of `moresetup` is literally copied into the style difinition at the end of the paragraph setup, just before `\noindent`. A few ideas what can be set in `moresetup`:

- A color of the paragraph text,
- `\penalty` to suggest a page break,
- `\interlinepenalty` for list item paragraphs, to avoid a page breaks inside.

The content of `afterpar` is literally copied into the style definition directly after `{...\par}`. This place is good to put `\nobreak` or some other penalty.

The content of `preamble_argN` is copied literally into the style definition directly before `#N`. Possible applications:

- Add `\ignorespaces` if the text might contain spuorious spaces at the beginning.
- For list item paragraphs, `\hfil` centers the tabulator box content, `\hfill` aligns to the right.

Tabstops. Tabstops are hboxes of a given width at given offset. All the offsets are relative to the left border of the text flow.

```
ParagraphOptions(...,
  boxes=(
    (OFFSET1,WIDTH1),
    (OFFSET2,WIDTH2),
    ...,
    (OFFSETn,WIDTHn)),
  ...)
```

Due to Python pecularities, one-element list of lists should have an additional comma, otherwise Python unwraps one level of parentheses. The correct way is:

```
ParagraphOptions(...,
  boxes=((OFFSET,WIDTH),), # Comma inside
  ...)
```

The content of the boxes is left-aligned. To center or right-align the content, add `\hfil` or `\hfill` through the parameter `preamble_argN`.

Inheritance. The parameter `parent` uses an already existing paragraph object as the starting point for the paragraph being defined. The properties, which are not specified in the new paragraph definition, are taken from the parent.

```
head_i = ParagraphOptions(
    cmd='HeadI',
    fontsize='12pt', baseline='14pt',
    fontcmd=r'\fontseries{b}\selectfont',
    ... )
```

```
ParagraphOptions(cmd='HeadII',
    parent=head_i, # Inheritance
    fontsize='11pt', baseline='13pt',
    ... )
```

In the example, the paragraph `HeadII` inherits `fontcmd` from `HeadI`, but uses the custom font size and baseline.

The infrastructure. A python file with definitions: (1) starts by importing the support code; (2) continues with collecting the definitions; and (3) finishes by the command to dump the TEX result.

```
from parades import * # (1)

add_style(ParagraphOptions(...)) # (2)
add_style(ParagraphOptions(...))
...
add_style(ParagraphOptions(...))

main('paras') # (3)
```

The parameter of the function `main` (in this example `paras`) is the name of the generated sty-package as given by `\ProvidesPackage`.

5 Getting and running the code

All the files, including the example, are contained in the CTAN package `parades`. Alternatevely, one gets the source code from `github` in the repository `http://github.com/olpa/tex`, in the folder `paragraph_designer`.

Put the file `paravesp.sty` into a directory in which TEX will find it. Put the file `parades.py` into a directory in which Python will find it.

The paragraph generator runs from the command line.

```
$ python input-defs.py [output-defs.sty]
```

The script `input-defs.py` is the file with the python definitions of the paragraphs. The optional arguments is the name of the `.sty`-file with the generated TEX definitions. If the output file is not specified, the code is dumped to the standard output.

The directory `udhr` contains a sample project. Refer to the file `README` in this directory for details how to use it.

6 Conclusion

The paragraph designer helps both on the technical and organization levels. On the technical level, it helps generating code for paragraph styles. It would be an unpleasant error-prone task to write this code manually:

- Space above and below a paragraph.
- Paragraphs with tabstops such as list items, table of content entries, headers.

On the organization level, the python-scripts allow to have a common code base and adapt it to the needs of specific layouts.

The LATEX package `paravesp` can be used independently of the paragraph designer to implement vertical spacing.

There are problems with the package `paravesp` and the paragraph designer:

- Many features are not implemented and some need rework.
- The LATEX code written in the galley style is too verbose to be typeset manually.

The paragraph designer is used in a production system for years. The benefits compensate the problems.