Prototype reimplementation of LaTeX 2ε's block environments using templates

LaTeX Project*

v0.8c 2023/05/16

Abstract

Contents

*Initial reimplementation of lists done by Bruno Le Floch, generalized second version with tagging support by Frank Mittelbach.

1 Introduction

The list implementation in LaTeX 2ε serves a dual purpose: it implements real lists such as `itemize` or `enumerate`, but it is also used as the basis for vertical blocks, i.e., to specify the vertical spacing and paragraph handling after such block, e.g., in environments like `center`, `quote`, `verbatim`, or in the theorem environments. They are all implemented as "trivial" lists with a single (hidden) item.

While this was convenient to get a consistent layout using a single implementation it is not adequate if it comes to interpreting the structure of a document, because environments based on `trivlist` should not advertise themselves as being a "list" — after all, from a semantic point of view they aren't lists.

The approach taking here is therefore to offer separate object types: *block* (horizontally or vertically oriented data that needs some handling at the start and the end), *para* (that deals with different paragraph layouts), *list* (that handles list related parameters, and *item* (for item layouts and handling), to address the independent aspects and also offer the object type *blockenv* that ties them together as necessary.

For example, a `quote` environment would make use of a (display) *block* and some *para* handling while an standard `enumerate` would make use of a display *block*, a *list*, and an *item* and *para* instance. An inline list (like `enumerate*` from the enumitem package) would be using the same *list* instance but a different (horizontally oriented) *block*.

2 Object types and templates for blocks and lists

2.1 Object types

2.1.1 The object type 'block'

Arg: 1 key/value list to alter the default block parameters

Semantics:

Handle the layout aspects of a block of data. In case of a "display" block (i.e., vertically oriented) the spacing and page breaking as well as the handling if the block starts a paragraph or ends one, that is, if text is immediately following the block without being separated by an empty line, then this text is considered to be in the same paragraph as the block.

In case of a horizontally oriented block it covers any special handling at the start and end of the block, e.g, extra spacing, prohibitying or encuraging line breaks, and so forth.

2.1.2 The object type 'para'

Arg: 1 key/value list to alter the default item parameters

Semantics:

Sets up paragraph-specific parameters for H&J, e.g., to implement justification variations, the behavior of \\ etc. The instances are used in higher-level templates, e.g., in a *block*.

2.1.3 The object type 'list'

Arg: 1 key/value list to alter the default item parameters

Semantics:

Handle the aspects related to list design, e.g., the use and formatting of counters, etc.

Note that this does not cover block-related aspects, i.e., a list instance could be used both for a display list or for an inline line.

2.1.4 The object type 'item'

Arg: 1 key/value list to alter the default item parameters

Semantics:

A sub-type used as part of *list* to easily cover alternative layout for list items.

2.1.5 The object type 'blockenv'

Arg: 1 key/value list to alter the default item parameters

Semantics:

This object type is used to implement document-level environments. It defines a *block* instance to handle the layout at the "edge" of the environment data, possibly some paragraph setup through a *para* instance, potentially an "inner" instance for more complicated environments (such as lists), and possibly some additional setup code for certain environments.

It also defines how the *blockenv* behaves with respect to nesting, e.g., does it change when nested and if so how many levels of nesting are supported, etc.

Finally, the object type defines how it appears in a tagged PDF document, what tag names are used, how they are rolemapped and whether it adds additional attributes, etc.

2.2 Templates

2.2.1 The `blockenv` template 'display'

Attributes:

`env-name` (*tokenlist*) Name of the environment used only in tracing

`tag-name` (*tokenlist*) Name of the tag in the PDF. If not explicitly given the name is defined by the `tagging-recipe`

`tag-class` (*tokenlist*) An explicit tag class attribute

`tagging-recipe` (*tokenlist*) Defines the way tagging is done. Currently the values `basic`, `standard`, and `list` are supported Default: `standard`

`level-increase` (*boolean*) Does this *blockenv* increase the block level if it is nested in an outer block? Default: `true`

`setup-code` (*tokenlist*) Initial setup code. This is executed after legacy defaults (from `\@listi`, `\@listii`, etc.) are used but before the block instance is called

`block-instance` (*tokenlist*) Part of the name of the *block* instance that is called. The full name has a `-`⟨*level*⟩ appended Default: `displayblock`

`para-instance` (*tokenlist*)

`inner-level-counter` (*tokenlist*) Name of an existing (!) counter that is incremented and used to determine final name of the `inner-instance` or empty if always the same inner instance should be used

`max-inner-levels` (*tokenlist*) Maximum number of nested environments of this kind. Only relevant if there is a `inner-level-counter` specified Default: `4`

`inner-instance-type` (*tokenlist*) Opject type of the inner instance Default: `list`

`inner-instance` (*tokenlist*) Name of the inner instance (if any).

`para-flattened` (*boolean*) *describe* Default: `false`

`final-code` (*tokenlist*) Final setup code Default: `\ignorespaces`

Semantics & Comments: This *blockenv* template supports the legacy list setting that are found in many document classes in the macros `\@listi`, `\@listii`, up to `\@listvi`. It also uses the counter `\@listdepth` to track nesting of block, again mainly to support legacy setups (internally it gives it a more appropriate name but it remains accessible through the LaTeX 2_ε name).

It first checks that nothing is too deeply nested. If the level should increase then the increments the `\@listdepth` counter and calls the corresponding `\@list...` macro to update the legacy defaults. If `level-increase` is set to false this is bypassed.

It then sets up the tagging via the `tagging-recipe` setting and executes any code in `setup-code`.

Afterwards it calls the appropriate *block* instance based on `block-instance` and current level, e.g., `displayblock-1`. Then it sets up paragraph parameters if a `para-instance` was specified (otherwise they stay as they are).

If a `inner-instance` was specified this is called next, or more precisely: if no `inner-level-counter` was specified the instance `inner-instance` is called.

Otherwise, the `inner-level-counter` is incremented and the instance with the name `inner-instance-inner-level-counter` is called.

Finally, the `final-code` is executed (by default `\ignorespaces`).

The maximum number of *blockenv*s that can be nested into each other is is restricted by the LaTeX counter `maxblocklevels` with a default value of `6`. If this value is increased then it is necessary to provide additional instances, e.g., `displayblock-7`, etc. Decreasing is, of course, always possible, then some of the instances defined are not used and instead the user gets an error that there is too much nesting going on.

If the key `level-increase` is set to `false` then such an environment doesn't alter the nesting level and therefore you can nest those environments as often as you like (a typical example would be `flushleft` anywhere in the nesting hierarchy, that would have no effect on hitting the boundary).

2.2.2 The `block` template 'display'

Attributes:

`heading` (*tokenlist*) *not really used yet*

`beginsep` (*skip*) Default: `\topsep`

`begin-par-skip` (*skip*) Default: `\partopsep`

`par-skip` (*skip*) Default: `\parsep`

`end-skip` (*skip*) Default: value from `beginsep`

`end-par-skip` (*skip*) Default: value from `begin-par-skip`

`beginpenalty` (*integer*) Default: `\@beginparpenalty`

`endpenalty` (*integer*) Default: `\@endparpenalty`

`leftmargin` (*length*) Default: `\leftmargin`

`rightmargin` (*length*) Default: `\rightmargin`

`parindent` (*length*) Default: `\listparindent`

Semantics & Comments: The idea of a `heading` key needs some further thoughts. Maybe instead the object type should accept a second argument and receive input for such a headding from the document level instead.

The names of the keys need further thoughts and some decision. Right now it is a mixture of those with hyphens and those that match legacy register names (the way enumitem did its keys).

Also `parindent` conflicts with `indent-width`!

2.2.3 The `para` template 'std'

Attributes:

`indent-width` (*length*) Default: `\parindent`

`start-skip` (*skip*) Default: `0pt`

`left-skip` (*skip*) Default: `0pt`

`right-skip` (*skip*) Default: `0pt`

`end-skip` (*skip*) Default: `\@flushglue`

`fixed-word-spaces` (*boolean*) Default: `false`

`final-hyphen-demerits` (*integer*) Default: `5000`

`cr-cmd` (*tokenlist*) Default: `\@normalcr`

`para-class` (*tokenlist*) Default: `justify`

2.2.4 The `list` template 'std'

Attributes:

`counter` (*tokenlist*) Counter name to be used in a numbered list or empty, if the list is unnumbered

`item-label` (*tokenlist*) Label "string" for a fixed label or as generated from the current `counter` value

`start` (*integer*) Start value for the counter if the list is numbered, otherwise irrelevant
Default: `1`

`resume` (*boolean*) Should a numbered list be resumed from the last instance?
Default: `false`

`item-instance` (*instance*) Instance of type `item` to be used to format the label string
Default: `basic`

May need to be on a different template level

`item-skip` (*skip*) The space in front of an item in the list. Default: `\itemsep`

`item-indent` (*length*) Horizontal displacement of the item. Default: `0pt`

`item-penalty` (*integer*) Penalty for breaking before an item (except the first)
Default: `\@itempenalty`

`label-width` (*length*) Width reserved for the formatted item labelDefault: `\labelwidth`

`label-sep` (*length*) Horizontal separation between label and following text
Default: `\labelsep`

`legacy-support` (*boolean*) Is formatting the label via `\makelabel` supported?
Default: `false`

2.2.5 The `item` template 'std'

Attributes:

`counter-label` (*function1*) *unused* Default: `\arabic{#1}`

`counter-ref` (*function1*) *unused* Default: value from `counter-label`

`label-ref` (*function1*) *unused* Default: `#1`

`label-autoref` (*function1*) *unused* Default: `item #1`

`label-format` (*function1*) Formatting of the label, questionable the way it is used
Default: `#1`

`label-strut` (*boolean*) Add a `\strut` to the label? Default: `false`

`label-align` (*choice*) Supported values `left`,`center`, `right`, and `parleft`. *Only partly implemented* Default: `right`

`label-boxed` (*boolean*) Should the label be boxed? Default: `true`

`next-line` (*boolean*) Default: `false`

`text-font` (*tokenlist*) *unused*

`compatibility` (*boolean*) Default: `true`

Semantics & Comments: This template is only rudimentary implemented at the moment. It probably needs other keys and the existing ones need a proper implementation.

3 Tagging support

3.1 Paragraph tags

Paragraphs in LaTeX can be nested, e.g., you can have a paragraph containing a display quote, which in turn consists of more than one (sub)paragraph, followed by some more text which all belongs to the same outer paragraph.

In the PDF model and in the HTML model that is not supported — a limitation that conflicts with real live, given that such constructs are quite normal in spoken and written language.

The approach we take to resolve this is to model such "big" paragraphs with a structure named `<text-unit>` and use `<text>` (rollmapped to `<P>`) only for (portions of) the actual paragraph text in a way that the `<text>`s are not nested. As a result we have for a simple paragraph the structures

```
<text-unit>
  <text>
    The paragraph text ...
  </text>
</text-unit>
```

The `<text-unit>` structure is rollmapped to `<Part>` or possibly to `<Div>` so we get a valid PDF, but processors who care can identify the complete paragraphs by looking for `<text-unit>` tags.

In the case of an element, such as a display quote or a display list inside the paragraph, we then have

```
<text-unit>
  <text>
    The paragraph text before the display element ...
  </text>
  <display element structure>
    Content of the display structure possiblly involving inner <text-unit> tags
  </display element structure>
  <text>
```

```
    ... continuing the outer paragraph text
  </text>
</text-unit>
```

In other words such a display block is always embedded in a `<text-unit>` structure, possibly preceded by a `<text>...</text>` block and possibly followed by one, though both such blocks are optional.

Thus an `itemize` environment that has some introductory text but no text immediately following the list would be tagged as follows:

```
<text-unit>
  <text>
    The intro text for the itemize environment ...
  </text>
  <itemize>
    <LI>
      <Lbl> label </Lbl>
      <LBody>
        The text of the first item involving <text-unit> as necessary ...
      </LBody>
    </LI>
    <LI>
      The second item ...
    </LI>
    ... further items ...
  </itemize>
</text-unit>
```

The `<itemize>` is rollmapped to `<L>`.

For some display blocks, such as centered text, we use a simpler strategy. Such blocks still ensure that they are inside a `<text-unit>` structure but their body uses simple `<text>` blocks and not `<text-unit><text>` inside, e.g., the input

```
This is a paragraph with some
\begin{center}
   centered lines

   with a paragraph break between them
\end{center}
followed by some more text.
```

will be tagged as follows:

```
<text-unit>
  <text>
    This is a paragraph with some
  </text>
  <text /O /Layout /TextAlign/Center>
    centered lines
  </text>
  <text /O /Layout /TextAlign/Center>
    with a paragraph break between them
```

```
  </text>
  <text>
    followed by some more text.
</text-unit>
```

3.2 Tagging recipes

There are a number of different tagging recipes that implement different tagging approaches. They are selected through the `tagging-recipe` of the *blockenv* template. Currently the following values are implemented:

`basic` This recipe does the following:

- Ensure that the *blockenv* is inside a `<text-unit>` structure, if necessary, start one.
- If inside a `<text-unit><text>`, then close the `</text>` but leave the `<text-unit>` open.
- Text inside the body of the environment starts with `<text-unit><text>` if `para-flattened` is set to `false`, otherwise just with `<text>`.
- At the end of the environment close `</text>` and possibly an inner `</text-unit>` if open.
- Then look if the environment is followed by an empty line (`\par`). If so, close the outer `</text-unit>` and start any following text with `<text-unit><text>`. Otherwise, don't and following text restarts with a just a `<text>` (and no paragraph indentation)

`standard` This recipe is like the `basic` one as far as handling `<text-unit>` and `<text>` is concerned. In addition

- it starts an inner tagging structure (i.e., which is therefore a child of the outer `<text-unit>`).
- By default this structure is a `<Figure>` unless overwritten by the key `tag-name`. If that key is used, a suitable rollmap needs to be provided for the name given.
- At the end of the environment that inner structure is closed again so that we are back on the `<text-unit>` level from the outside.
- Then the lookahead for an empty line is done as described previously.

`list` This recipe is like the `standard` one except that

- the inner structure is a list (`<L>`).
- Furthermore everything is set up so that we have list items (`<LI>`) with suitable substructures (`<Lbl>` for the item labels and `<LBody>` for the item bodies).
- If the key `tag-name` is specified, this is used as the tag name for the whole list instead of `<L>`. Of course, it should then have a suitable rollmap.
- If the key `tag-class` is specified then this is used as the class attribute. Again, this requires a suitable setup on the outside.
- At the end of the environment the `</LBody>`, `</LI>`, and `</L>` (or the tag name used) are closed.
- Then the lookahead for an empty line is done as described previously.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.