

Prototype reimplementation of L^AT_EX 2_ε's block environments using templates

L^AT_EX Project*

v0.8h 2023-09-01

Abstract

Contents

1	Introduction	3
2	Object types and templates for blocks and lists	3
2.1	Object types	3
2.1.1	The object type ‘block’	3
2.1.2	The object type ‘para’	3
2.1.3	The object type ‘list’	4
2.1.4	The object type ‘item’	4
2.1.5	The object type ‘blockenv’	4
2.2	Templates	4
2.2.1	The <code>blockenv</code> template ‘display’	4
2.2.2	The <code>block</code> template ‘display’	6
2.2.3	The <code>para</code> template ‘std’	6
2.2.4	The <code>list</code> template ‘std’	7
2.2.5	The <code>item</code> template ‘std’	7
3	Tagging support	8
3.1	Paragraph tags	8
3.2	Tagging recipes	10

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

4	The Implementation	11
4.1	Handling <code>\par</code> after the end of the list	11
4.2	Object and template interfaces	12
4.3	Useful helper commands	14
4.3.1	Debugging	14
4.4	Implementation of the document-level block environments	15
4.4.1	Displayblock environments	15
4.4.2	Display quote environments	16
4.4.3	Verbatim environments	16
4.4.4	Standard list environments	17
4.4.5	verse environment	17
4.4.6	Theorem-like environments	19
4.5	Implementation of templates	21
4.5.1	Implementation of blockenv templates ...	21
4.5.2	Implementation of para templates ...	26
4.5.3	Implementation of block templates ...	26
4.5.4	Implementation of list templates ...	29
4.5.5	Implementation of <code>\item</code> template(s)	31
4.6	Tagging recipes	36
4.7	Blockenv instances	38
4.7.1	Basic instances	38
4.7.2	Blockquote instances	39
4.7.3	Verbatim instances	41
4.7.4	Standard list instances	41
4.8	Block instances	42
4.8.1	Displayblock instances	42
4.8.2	Verbatim instances	43
4.8.3	Quote/quotationblock instances	43
4.8.4	Block instances for the standard lists	44
4.9	List instances for the standard lists	44
4.10	Item instances	45
4.11	Para instances	45
4.12	Tagging support	46
4.12.1	List tags	51
5	Documentation from first prototype implementations	54
5.1	Open questions	54
5.2	Code cleanup	54
5.3	Tasks	54
6	Plan of attack of first prototype	55
	Index	57

1 Introduction

The list implementation in $\text{\LaTeX} 2_{\epsilon}$ 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, prohibiting or encouraging 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*) Object 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 L^AT_εX 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 *blockenvs* that can be nested into each other is restricted by the L^AT_εX 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 heading 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 (<i>tokenlist</i>)	Counter name to be used in a numbered list or empty, if the list is unnumbered	
item-label (<i>tokenlist</i>)	Label “string” for a fixed label or as generated from the current counter value	
start (<i>integer</i>)	Start value for the counter if the list is numbered, otherwise irrelevant	Default: 1
resume (<i>boolean</i>)	Should a numbered list be resumed from the last instance?	Default: false
item-instance (<i>instance</i>)	Instance of type item to be used to format the label string	Default: basic
item-skip (<i>skip</i>)	The space in front of an item in the list.	Default: \itemsep
item-indent (<i>length</i>)	Horizontal displacement of the item.	Default: Opt
item-penalty (<i>integer</i>)	Penalty for breaking before an item (except the first)	Default: \@itempenalty
label-width (<i>length</i>)	Width reserved for the formatted item label	Default: \labelwidth
label-sep (<i>length</i>)	Horizontal separation between label and following text	Default: \labelsep
legacy-support (<i>boolean</i>)	Is formatting the label via \makelabel supported?	Default: false

May need to be on a different template level

2.2.5 The item template ‘std’

Attributes:

counter-label (<i>function1</i>)	<i>unused</i>	Default: \arabic{#1}
counter-ref (<i>function1</i>)	<i>unused</i>	Default: value from counter-label
label-ref (<i>function1</i>)	<i>unused</i>	Default: #1
label-autoref (<i>function1</i>)	<i>unused</i>	Default: item #1
label-format (<i>function1</i>)	Formatting of the label, questionable the way it is used	Default: #1

<code>label-strut</code> (<i>boolean</i>)	Add a <code>\strut</code> to the label?	Default: <code>false</code>
<code>label-align</code> (<i>choice</i>)	Supported values <code>left</code> , <code>center</code> , <code>right</code> , and <code>parleft</code> . <i>Only partly implemented</i>	Default: <code>right</code>
<code>label-boxed</code> (<i>boolean</i>)	Should the label be boxed?	Default: <code>true</code>
<code>next-line</code> (<i>boolean</i>)		Default: <code>false</code>
<code>text-font</code> (<i>tokenlist</i>)	<i>unused</i>	
<code>compatibility</code> (<i>boolean</i>)		Default: <code>true</code>

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 L^AT_EX 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 possibly 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>
      <Lb1> label </Lb1>
      <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 /0 /Layout /TextAlign/Center>
    centered lines
  </text>
  <text /0 /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:

standalone This recipe does the following:

- Ensure that the `blockenv` is not inside a `<text-unit>` structure. If necessary, close the open one (and any open `<text>` structure).
- Text inside the body of the environment start with `<text-unit><text>` unless the key `para-flattened` is set to `true` (which is most likely the wrong thing to do because we then get just `<text>` as the structure).
- At the end of the environment close `</text>` and possibly an inner `</text-unit>` if open.
- Finally, ensure that after the environment a new `<text-unit>` is started, if appropriate, e.g., if text is following.

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 start 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 (``) with suitable substructures (`<Lb1>` 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>`, ``, and `</L>` (or the tag name used) are closed.
- Then the lookahead for an empty line is done as described previously.

4 The Implementation

```
1 <*package>
2 <@@=block>
3 \ProvidesPackage {latex-lab-testphase-block}
4                   [\tlabblockdate\space v\tlabblockversion\space
5                   blockenv implementation]
```

We make use of templates:

```
6 \RequirePackage{xtemplate}
```

Generell kernel changes, also loaded by the `sec` and `toc` code.

```
7 \RequirePackage{latex-lab-kernel-changes}
```

```
8 \ExplSyntaxOn
```

```
9 \tl_new:N \l_block_item_align_tl
```

```
10 \tl_new:N \l_block_legacy_env_params_tl
```

UFI: this variable(s) must be declared:

4.1 Handling `\par` after the end of the list

An empty line (or a `\par`) after a list has semantic meaning as it defines whether then following text is logically within the same paragraph as the list (no empty line) or whether it starts a new paragraph and the paragraph containing the list ends at the end of the list (empty line after the list). This is handled by \LaTeX using a legacy flag called `@endpe` and set of commands inside the generic `\end` (calling `\@doendpe`) and as part of the list environments identifying themselves as “paragraph ending environments” (by setting this flag).

For the reimplementaion of the list environments including support of tagging we need to augment that mechanism slightly and add some kernel hook(s) to add the tagging code if needed.

`\@doendpe` The original \LaTeX 2_{ϵ} command is augmented to allow for tagging.

```
11 \def\@doendpe{\@endpetrue
12   \def\par
13     {
14       \@restorepar
15       \@clubpenalty\@clubpenalty
```

At this point we add the tagging code that closes an open `<text-unit>`, `<text>` tag combination, if necessary:

```
16     \__kernel_displayblock_doendpe:
```

The standard `\par` command (`\par_end:`) acts on `@endpe` and attempts to close a still open `text-unit` and this would be wrong if it was already closed above. So we have to reset the switch to false first.

```
17     \@endpefalse
18     \everypar{}
19     \par
20   }
21   \everypar{{\setbox\z@\lastbox}
22             \everypar{}
23             \@endpefalse
24   }
25 }
```

By default we don't do any tagging:

```
26 \cs_new_eq:NN \__kernel_displayblock_doendpe: \prg_do_nothing:
```

The flag itself should be set globally not locally.

```
27 \def\@endpetrue {\global\let\if@endpe\iftrue}
28 \def\@endpefalse{\global\let\if@endpe\iffalse}
```

(End of definition for `\doendpe`. This function is documented on page ??.)

4.2 Object and template interfaces

`blockenv` (*objecttype*) All object types expect a single key–value argument used to tweak template parameters specific to a given use in the document. This section is devoted to template interfaces, `block` (*objecttype*) and the template code is covered later.

```
para (objecttype)
list (objecttype) 29 \DeclareObjectType{blockenv}{1}
item (objecttype) 30 \DeclareObjectType{block}{1}
                    31 \DeclareObjectType{para}{1}
                    32 \DeclareObjectType{list}{1}
                    33 \DeclareObjectType{item}{1}
```

`blockenv display` (*templ.*)

```
34 \DeclareTemplateInterface{blockenv}{display}{1}
35 {
36   env-name      : tokenlist ,
37   tag-name      : tokenlist ,
38   tag-class     : tokenlist ,
39   tagging-recipe : tokenlist = standard,
40   level-increase : boolean = true ,
41   setup-code    : tokenlist ,
42   block-instance : tokenlist = displayblock ,
43   para-instance  : tokenlist ,
44   inner-level-counter : tokenlist,
45   max-inner-levels : tokenlist = 4,
46   inner-instance-type : tokenlist = list ,
47   inner-instance  : tokenlist ,
48   para-flattened : boolean = false ,
49   final-code     : tokenlist = \ignorespaces ,
50 }
```

verify that this claim is actually correct!

block display (*templ.*)

```
51 \DeclareTemplateInterface{block}{display}{1}
52 {
53   heading      : tokenlist = ,                %??
54   beginsep     : skip = \topsep ,
55   begin-par-skip : skip = \partopsep ,
56   par-skip     : skip = \parsep ,
57   end-skip     : skip = \KeyValue{beginsep} ,   % conflict with name below
58   end-par-skip : skip = \KeyValue{begin-par-skip} ,
59   beginpenalty : integer = \UseName{@beginparpenalty} ,
60   endpenalty   : integer = \UseName{@endparpenalty} ,
61   leftmargin   : length = \leftmargin ,
62   rightmargin  : length = \rightmargin ,
63   parindent    : length = \listparindent ,
64   % font      : tokenlist      % maybe add? (or more general for fonts and color)
65 }
```

para std (*templ.*)

```
66 \DeclareTemplateInterface{para}{std}{1}
67 {
68   indent-width      : length = \parindent ,
69   start-skip        : skip = Opt ,
70   left-skip         : skip = Opt ,
71   right-skip        : skip = Opt ,
72   end-skip          : skip = \@flushglue ,
73   fixed-word-spaces : boolean = false ,
74   final-hyphen-demerits : integer = 5000 ,
75   cr-cmd            : tokenlist = \@normalcr ,
76   para-class        : tokenlist = justify ,
77 }
```

list std (*templ.*)

```
78 \DeclareTemplateInterface{list}{std}{1} % optional
79 {
80   counter      : tokenlist = ,
81   item-label   : tokenlist = ,
82   start        : integer = 1 ,
83   resume       : boolean = false ,
84   item-instance : instance{item} = basic ,
85   item-skip    : skip = \itemsep ,
86   item-penalty : integer = \UseName{@itempenalty} ,
87   item-indent  : length = Opt , % was \itemindent
88   label-width  : length = \labelwidth ,
89   label-sep    : length = \labelsep ,
90   legacy-support : boolean = false ,
91 }
```

item std (*templ.*)

```
92 \DeclareTemplateInterface{item}{std}{1}
93 {
94   counter-label : function{1} = \arabic{#1} ,
95   counter-ref   : function{1} = \KeyValue{counter-label} ,
96   label-ref     : function{1} = #1 ,

```

```

97     label-autoref : function{1} = item-#1 ,
98     label-format  : function{1} = #1 ,
99     label-strut   : boolean = false ,
100    label-align   : choice {left,center,right,parleft} = right ,
101    label-boxed   : boolean = true ,
102    next-line     : boolean = false ,
103    text-font     : tokenlist ,
104    compatibility : boolean = true ,
105 }

```

4.3 Useful helper commands

This section collects expl3 commands that will be useful.

`_block_skip_set_to_last:N` Set a skip register to the value of an immediately preceding skip or zero if there was none
`_block_skip_remove_last:`

```

106 \cs_new_protected:Npn \_block_skip_set_to_last:N #1 {
107   \skip_set:Nn #1 { \tex_lastskip:D }
108 }

```

Remove a skip previous skip if it is directly in front (not allowed in unrestricted vertical mode).

```

109 \cs_new_eq:NN \_block_skip_remove_last: \tex_unskip:D

```

(End of definition for _block_skip_set_to_last:N and _block_skip_remove_last:.)

`\tl_if_novalue:nTF`

```

110 \cs_generate_variant:Nn \tl_if_novalue:nTF { o }

```

(End of definition for \tl_if_novalue:nTF. This function is documented on page ??.)

4.3.1 Debugging

`\g_block_debug_bool`

```

111 \bool_new:N \g_block_debug_bool

```

(End of definition for \g_block_debug_bool.)

`_block_debug:n`

`_block_debug_typeof:n`

```

112 \cs_new_eq:NN \_block_debug:n \use_none:n
113 \cs_new_eq:NN \_block_debug_typeof:n \use_none:n

```

(End of definition for _block_debug:n and _block_debug_typeof:n.)

`\block_debug_on:`

`\block_debug_off:`

`_block_debug_gset:`

```

114 \cs_new_protected:Npn \block_debug_on:
115 {
116   \bool_gset_true:N \g_block_debug_bool
117   \_block_debug_gset:
118 }
119 \cs_new_protected:Npn \block_debug_off:
120 {
121   \bool_gset_false:N \g_block_debug_bool
122   \_block_debug_gset:
123 }

```

```

124 \cs_new_protected:Npn \__block_debug_gset:
125 {
126   \cs_gset_protected:Npx \__block_debug:n ##1
127   { \bool_if:NT \g__block_debug_bool {##1} }
128   \cs_gset_protected:Npx \__block_debug_typeof:n ##1
129   { \bool_if:NT \g__block_debug_bool { \typeout{==>~ ##1} } }
130 }

```

(End of definition for `\block_debug_on:`, `\block_debug_off:`, and `__block_debug_gset:`. These functions are documented on page ??.)

```

\DebugBlocksOn
\DebugBlocksOff
131 \cs_new_protected:Npn \DebugBlocksOn { \block_debug_on: }
132 \cs_new_protected:Npn \DebugBlocksOff { \block_debug_off: }
133 \DebugBlocksOff

```

(End of definition for `\DebugBlocksOn` and `\DebugBlocksOff`. These functions are documented on page ??.)

4.4 Implementation of the document-level block environments

Most such environments are pretty simple: they take an option argument and call a `blockenv` instance to do the work. At the end of environment we call `\endblockenv` to finish.

4.4.1 Displayblock environments

There are two basic block environment which are similar to L^AT_EX 2_ε's `trivlist` except that there aren't degenerated lists and thus have no hidden `\item` inside.

```

displayblock (env.)
134 \NewDocumentEnvironment{displayblock}{ !0{} }
135 { \UseInstance{blockenv}{displayblock} {#1} }
136 { \endblockenv }

```

```

displayblockflattened (env.)
137 \NewDocumentEnvironment{displayblockflattened}{ !0{} }
138 { \UseInstance{blockenv}{displayblockflattened} {#1} }
139 { \endblockenv }

```

```

center (env.)
flushleft (env.)
flushright (env.)
140 \AddToHook{begindocument/before}{
141   \RenewDocumentEnvironment{center} { !0{} }
142   { \UseInstance{blockenv}{center}{#1} }
143   { \endblockenv }
144   \RenewDocumentEnvironment{flushright} { !0{} }
145   { \UseInstance{blockenv}{flushright}{#1} }
146   { \endblockenv }
147   \RenewDocumentEnvironment{flushleft} { !0{} }
148   { \UseInstance{blockenv}{flushleft}{#1} }
149   { \endblockenv }
150 }

```

4.4.2 Display quote environments

```
quote (env.)
quotation (env.) 151 \AddToHook{begindocument/before}{
152   \RenewDocumentEnvironment{quote}{ !0{ }
153     { \UseInstance{blockenv}{quote} {#1} }
154     { \endblockenv }
155   \RenewDocumentEnvironment{quotation}{ !0{ }
156     { \UseInstance{blockenv}{quotation} {#1} }
157     { \endblockenv }
158 }
```

4.4.3 Verbatim environments

```
verbatim (env.)
verbatim* (env.) 159 \AddToHook{begindocument/before}{
160   \RenewDocumentEnvironment{verbatim}{ !0{ }
161     { \UseInstance{blockenv}{verbatim} {#1} }
```

This is the part of the code where `verbatim` and `verbatim*` differ.

```
162   \@setupverbinvisiblespace\frenchspacing\@vobeyspaces
163   \@xverbatim
164 }
165 { \endblockenv }
166 \RenewDocumentEnvironment{verbatim*}{ !0{ }
167 { \UseInstance{blockenv}{verbatim} {#1}
168   \@setupverbvisiblespace\frenchspacing\@vobeyspaces
169   \@sxverbatim
170 }
171 { \endblockenv }
172 }
```

Helper commands for verbatim

`\legacyverbatimsetup` This code resembles the L^AT_EX 2_ε `verbatim` implementation with a slight twist: in L^AT_EX 2_ε each code line was a paragraph using `\leftskip=\@totalleftmargin`. This was possible because the whole environment was implemented as a trivlist. As this is no longer the case setting `\leftskip` would alter the layout of a surrounding list. So instead we need to make sure that the paragraph end is executed in a group so that any parshape setup is preserved.

```
173 <@@=
174 \def\legacyverbatimsetup{%
175   \language\l@nohyphenation
176   \@tempwafalse
177   \def\par{%
178     \if@tempswa
179       \leavevmode \null {\@@par}\penalty\interlinepenalty
180     \else
181       \@tempwatrue
182       \ifhmode{\@@par}\penalty\interlinepenalty\fi
183     \fi}%
184   \let\do\@makeother \dospecials
185   \obeylines \verbatim@font \@noligs
```



```

186 \everypar \expandafter{\the\everypar \unpenalty}%
187 \tl_set:Nn \l__tag_para_main_tag_tl {codeline}
188 \tagtool{paratag=Code}% oder faster: \tl_set:Nn\l__tag_para_tag_tl{Code}
189 }
190 <@@=block>

```

(End of definition for `\legacyverbatimsetup`. This function is documented on page ??.)

`\@setupverbinvisiblespace` In the pdf_TE_X engine we need to use `\pdffakespace` chars for the invisible spaces.

```

191 \newcommand\@setupverbinvisiblespace{}
192 \tag_if_active:T {
193   \bool_if:NF\g__tag_mode_lua_bool
194   {
195     \renewcommand\@setupverbinvisiblespace{\def\xobeysp{\nobreakspace\pdffakespace}}
196   }
197 }

```

(End of definition for `\@setupverbinvisiblespace`. This function is documented on page ??.)

4.4.4 Standard list environments

`itemize` (*env.*) For the standard lists everything is managed by the `blockenv` instance.

```

enumerate (env.) 198 \AddToHook{begindocument/before}{
description (env.) 199   \RenewDocumentEnvironment{itemize}{!0{}}
200   { \UseInstance{blockenv}{itemize} {#1} }
201   { \endblockenv }

202   \RenewDocumentEnvironment{enumerate}{!0{}}
203   { \UseInstance{blockenv}{enumerate} {#1} }
204   { \endblockenv }

205   \RenewDocumentEnvironment{description}{!0{}}
206   { \UseInstance{blockenv}{description} {#1} }
207   { \endblockenv }
208 }

```

4.4.5 verse environment

`verse` (*env.*) The verse environment has not special tagging currently. It is defined as a simple standard list and takes the tagging from there. But it must be redefined so that `\itemindent` is correctly set.

```

209 \AddToHook{begindocument/before}{
210   \RenewDocumentEnvironment{verse}{!0{}}
211   {
212     \let\\@centercr
213     \UseInstance{blockenv}{list}
214     {
215       item-indent=-1.5em,
216       parindent=-1.5em,
217       item-skip=0pt,
218       rightmargin=\leftmargin,
219       leftmargin=\leftmargin+1.5em,
220       #1
221     }
222     \item\relax

```

```

223     }
224     { \endblockenv }
225 }

```

`list (env.)` The legacy 2e list environment is more complicated as we have to get the extra arguments accounted for.

```

226 \AddToHook{begindocument/before}{
227   \RenewDocumentEnvironment{list}{0{ } m m }
228   {

```

We do this by storing them away and then call the list instance. Inside this instance the `setup-code` key contains `\legacylistsetupcode`, which makes use of the stored values.

```

229     \tl_set:Nn \@itemlabel {#2}
230     \tl_set:Nn \l__block_legacy_env_params_tl {#3}
231     \UseInstance{blockenv}{list} {#1}
232   }
233   { \endblockenv }
234 }

```

`\l__block_env_params_tl` Declare the variable for the parameter argument; `\@itemlabel` is already declared in L^AT_EX 2_ε.

```

235 \tl_new:N \l__block_env_params_tl

```

(End of definition for `\l__block_env_params_tl`.)

`\legacylistsetupcode` And here is the extra code for use in the list instance setup inside the key `setup-code`.

```

236 \cs_new:Npn \legacylistsetupcode {

```

Reset values to defaults:

```

237   \dim_zero:N \listparindent
238   \dim_zero:N \rightmargin
239   \dim_zero:N \itemindent

```

By default a `list` environment is not numbered:

```

240   \tl_set:Nn \@listctr {}
241   \legacy_if_set_false:n { @nbrlist } % needed if lists are nested

```

By default there is a simple definition for `\makelabel`. It can be overwritten in the second mandatory argument to the list environment (stored in `\l__block_legacy_env_params_tl`) and is used if the instance sets the compatibility key to true.

```

242   \let\makelabel\@mklab % TODO: customize

```

Now we use the argument with parameter settings to update some or all of the above defaults:

```

243   \l__block_legacy_env_params_tl

```

As we don't know much about this list we can only make a guess about the nature of the list and the setting of the tag name (default `list` rolemapped to `L`) and any tag attributes may have to be overwritten in the optional key/value argument. But we do have some hints to play with.

```

244   \legacy_if:nTF { @nbrlist }
245     { \tl_set:Nn \l__tag_L_attr_class_tl {enumerate} } % numbered list
246     { \tl_if_empty:NTF \@itemlabel
247       { \tl_set:Nn \l__tag_L_attr_class_tl {list} } % no label
248       { \tl_set:Nn \l__tag_L_attr_class_tl {itemize} } % unnumbered, unordered
249     }
250 }

```

(End of definition for `\legacylistsetupcode`. This function is documented on page ??.)

`trivlist` (*env.*)

```

251 \AddToHook{begindocument/before}{
252   \RenewDocumentEnvironment{trivlist}{!O{}}{
253     { \list[#1]{
254       {
255         \dim_zero:N \leftmargin
256         \dim_zero:N \labelwidth
257         \cs_set_eq:NN \makelabel \use:n
258       }
259     }
260   { \endblockenv }
261 }

```

4.4.6 Theorem-like environments

Theorem-like environments are defined in L^AT_EX with the help of `\newtheorem` declarations. Internally they used a list with a single item. Using lists was convenient back then, but in a tagged document you end up with a strange structure. We therefore alter the mechanism.

`\newtheorem` This is a slightly streamlined version of `\newtheorem`, but it still uses a lot of the 2e code for now. Eventually this will change.

```

262 \RenewDocumentCommand \newtheorem { m O{#1} m o }
263 {
264   \expandafter\@ifdefinable\csname #1\endcsname
265   {
266     \str_if_eq:nnTF{#1}{#2}
267     {
268       \@definecounter {#2}
269       \IfNoValueTF {#4}
270       { % @ynthm
271         \tl_gset:cx { the #2 }
272         {
273           \@thmcounter{#2}
274         }
275       }
276       { % @xnthm
277         \@newctr{#1}[#4]
278         \tl_gset:cx { the #2 }
279         {
280           \expandafter\noexpand\csname the#4\endcsname
281           \@thmcountersep
282           \@thmcounter{#2}
283         }
284       }
285     }
286     { % @othm
287       \@ifundefined{c@#2}
288       { \nocounterr{#2} }
289       {
290         \tl_gset:cn { the #1 }

```

```

291         { \UseName { the #2 } }
292     }
293 }
294 \global\@namedef{#1} { \@thm{#2}{#3} }
295 \global\@namedef{end#1}{ \@endtheorem }
296 }
297 }

```

(End of definition for `\newtheorem`. This function is documented on page ??.)

`\@thm` `\@thm` executes `\refstepcounter` too early for hyperref and structure destinations: the generated target is outside the structure and can be separated from the theorem by a page break. We therefore move the anchor setting into `\@begintheorem`. `\@begintheorem` doesn't currently get the name of the counter as argument, so we store it in variable for now, to be able to pass it along.

```

298 \tl_new:N \l__block_thm_current_counter_tl
299 \def\@thm#1#2{%
300   \@kernel@refstepcounter{#1}
301   \tl_set:Nn \l__block_thm_current_counter_tl{#1}
302   \@ifnextchar[{\@ythm{#1}{#2}}{\@xthm{#1}{#2}}}]

```

To avoid that hyperref overwrites the definition again we must its patch:

```

303 \def\hyper@nopatch@thm{}

```

(End of definition for `\@thm`. This function is documented on page ??.)

`\@begintheorem` `\@opargbegintheorem` The `\@thm` command expands to either `\@begintheorem` or `\@opargbegintheorem`. For the moment we stick with this as it will help with the transition. But instead of using a `trivlist` we use a `blockenv` and some tagging for the title (as a `Caption`). We do not want potential tagging from `\textbf` here, so we use `\bfseries` to set the font. The commands set also the link targets which should be inside the main structure.

```

304 \def\@begintheorem#1#2{
305   \UseInstance{blockenv}{theorem}{}
306   \tagpdfparaOff
307   \mode_leave_vertical:
308   \MakeLinkTarget{\l__block_thm_current_counter_tl}
309   \tag_struct_begin:n{tag=Caption}
310   \group_begin:
311   \bfseries
312   \tag_mc_begin:n {}
313   #1\
314   \tag_mc_end:
315   \tag_struct_begin:n{tag=Lbl}
316   \tag_mc_begin:n {}
317   #2
318   \tag_mc_end:
319   \tag_struct_end:
320   \group_end:
321   \tag_struct_end:
322   \tagpdfparaOn
323   \__block_start_para_structure_unconditionally:n { \PARALABEL }

```

```

324 \itshape
325 \hskip\labelsep
326 \ignorespaces
327 }
328 \def\@opargbegintheorem#1#2#3{
329 \UseInstance{blockenv}{theorem}{}
330 \tagpdfparaOff
331 \mode_leave_vertical:
332 \MakeLinkTarget{\l_block_thm_current_counter_tl}
333 \tag_struct_begin:n{tag=Caption}
334 \group_begin:
335 \bfseries
336 \tag_mc_begin:n {}
337 #1\
338 \tag_mc_end:
339 \tag_struct_begin:n{tag=Lbl}
340 \tag_mc_begin:n {}
341 #2
342 \tag_mc_end:
343 \tag_struct_end:
344 \tag_mc_begin:n {}
345 \ (#3)
346 \tag_mc_end:
347 \group_end:
348 \tag_struct_end:
349 \tagpdfparaOn
350 \_block_start_para_structure_unconditionally:n { \PARALABEL }
351 \itshape
352 \hskip\labelsep
353 \ignorespaces
354 }
355 \def\@endtheorem{\endblockenv}

```

(End of definition for \@begintheorem and \@opargbegintheorem. These functions are documented on page ??.)

4.5 Implementation of templates

4.5.1 Implementation of blockenv templates ...

`\g_block_nesting_depth_int` L^AT_EX_{2 ϵ} already has a counter to record the nesting depth of blocks, but we want our own name because it isn't really tied to "lists" any more. However, `\@listdepth` is really part of the legacy interface (for example `minipage` alters it to point to a different counter) so that we are stuck with using at least indirectly for now and the following line makes this look like an L³ integer variable but internally expands to `\@listdepth`:

```

356 \cs_new:Npn \g_block_nesting_depth_int { \@listdepth } % a fake int
357 % for now

```

(End of definition for \g_block_nesting_depth_int. This function is documented on page ??.)

`blockenv display (templ.)`

```

358 \DeclareTemplateCode{blockenv}{display}{1}
359 {

```

```

360 env-name      = \l__block_env_name_tl ,
361 tag-name      = \l__block_tag_name_tl ,
362 tag-class     = \l__block_tag_class_tl ,
363 tagging-recipe = \l__block_tagging_recipe_tl ,
364 level-increase = \l__block_level_incr_bool ,
365 setup-code    = \l__block_setup_code_tl ,
366 block-instance = \l__block_block_instance_tl ,
367 para-instance = \l__block_para_instance_tl ,
368 inner-level-counter = \l__block_inner_level_counter_tl ,
369 max-inner-levels = \l__block_max_inner_levels_tl ,
370 inner-instance-type = \l__block_inner_instance_type_tl ,
371 inner-instance = \l__block_inner_instance_tl ,
372 para-flattened = \l__tag_para_flattened_bool ,
373 final-code    = \l__block_final_code_tl ,
374 }
375 {
376   \__block_debug_typeout:n{\l__block_env_name_tl -env-start}
377   %
378   \tl_if_empty:nF {#1} { \SetTemplateKeys{blockenv}{display}{#1} }
379   %

```

We need to know later if we have nested blockenvs inside a flattened environment. Whenever we start a new blockenv we increment `\l__block_flattened_level_int` if it is already different from zero. If it is zero we increment it if flattening is requested. Thus a value of 0 means no flattening requested so far and 1 means this is the first blockenv requesting flattening. In either case we have to make sure that the blockenv is surrounded by a `text-unit` tag, while for any value above 1 we have to omit the `text-unit`.

```

380   \int_compare:nNnTF \l__block_flattened_level_int > 0
381     {
382       \int_incr:N \l__block_flattened_level_int
383     }
384     {
385       \bool_if:NT \l__tag_para_flattened_bool
386         {
387           \int_incr:N \l__block_flattened_level_int
388         }
389     }
390   %
391   \tl_if_empty:NF \l__block_inner_level_counter_tl
392     {
393       \int_compare:nNnTF \l__block_inner_level_counter_tl >
394         { \l__block_max_inner_levels_tl - 1 }
395         { \@toodeep }
396         { \int_incr:N \l__block_inner_level_counter_tl } % not clean "o"?
397     }

```

Legacy defaults are only roped in if the list level changes. For display blocks that remain on the same level the current values are kept.

```

398   \bool_if:NT \l__block_level_incr_bool
399     {
400       \int_compare:nNnTF \g_block_nesting_depth_int >
401         { \c@maxblocklevels - 1 }
402         { \@toodeep }
403     }

```

```
404         \int_gincr:N \g_block_nesting_depth_int
```

If there are no legacy defaults for that level then the next line does nothing, i.e., the current values (from the last level become the defaults for the next.

```
405         \use:c { @list \int_to_roman:n { \g_block_nesting_depth_int } }
406     }
407 }
```

If we are doing tagging we load one of the available recipes for tagging, which alters various kernel hooks to add appropriate tagging structures.

```
408 \tag_if_active:T { \use:c { __block_recipe_ \l__block_tagging_recipe_tl : } }
```

Then run the setup code if any is given in the instance.

```
409 \l__block_setup_code_tl
```

Next call a block instance at the appropriate level passing it any key/value list provided in the optional argument (keys that are not recognized are ignored—currently with an error).

```
410 \__block_debug_typeout:n{use~ instance:~
411     \l__block_block_instance_tl - \int_use:N \g_block_nesting_depth_int }
412 \UseInstance{block}
413     { \l__block_block_instance_tl - \int_use:N
414       \g_block_nesting_depth_int }
415     {#1}
```

After the block instance call the para and then inner (list) instance if either or both are specified (which may not be the case).

```
416 \tl_if_empty:NF \l__block_para_instance_tl
417 {
418     \__block_debug_typeout:n{use~ para~ instance:~ \l__block_para_instance_tl }
```

For now we don't offer to alter instance parameters here so we pass an empty argument.

```
419     \UseInstance{para}{ \l__block_para_instance_tl } {}
420 }
```

The inner instance may have its own levels or none depending on which the instance name differs. Again we pass it the optional key/value list.

```
421 \tl_if_empty:NF \l__block_inner_instance_tl
422 {
423     \__block_debug_typeout:n{use~ instance:~ \l__block_inner_instance_tl
424         \tl_if_empty:NF \l__block_inner_level_counter_tl
425             { - \int_use:N \l__block_inner_level_counter_tl }}
426     \UseInstance{ \l__block_inner_instance_type_tl }
427         { \l__block_inner_instance_tl
428             \tl_if_empty:NF \l__block_inner_level_counter_tl
429                 { - \int_use:N \l__block_inner_level_counter_tl } % not clean
430                                                         % use "o"?
431         }
432     {#1}
433 }
```

We finish off with `\l__block_final_code_tl` which defaults to `\ignorespaces` so that spaces between `\begin{...}` and the start of the text are ignored.

```
434 \l__block_final_code_tl
435 }
```

`\l_block_flattened_level_int` Count the levels of nested blockenvs starting with the first that is “flattened”.

```
436 \int_new:N \l_block_flattened_level_int
```

(End of definition for `\l_block_flattened_level_int`.)

`\c@maxblocklevels` A counter to increase or decrease the number of supported level. If increased, one needs to supply additional level instances.

```
437 \newcounter{maxblocklevels}
```

```
438 \setcounter{maxblocklevels}{6}
```

(End of definition for `\c@maxblocklevels`. This function is documented on page ??.)

`\endblockenv` The code executed when a blockenv ends is 99% the same for all blockenvs (at least up to now). Small differences exist, though. They are accounted for first in the conditionals.

We make this a public command so that new block environments can be set up without the need to resort to L3 layer programming.

```
439 \cs_new:Npn \endblockenv {
```

```
440   \__block_debug_typeout:n{blockenv~ common~ ending \on@line}
```

If this block was incrementing the level we have to decrement it now again:

```
441   \bool_if:NT \l_block_level_incr_bool
```

```
442     { \int_gdecr:N \g_block_nesting_depth_int }
```

If this block was a list and there are still `\item` labels to be placed we move to horizontal mode to get them typeset.

```
443   \legacy_if:nT { @inlabel }
```

```
444     {
```

```
445       \mode_leave_vertical:
```

```
446       \legacy_if_gset_false:n { @inlabel }
```

```
447     }
```

In a pure “displayblock” scenario `@newlist` will be always false and the code bypassed, but we may have an outer list followed immediately by a displayblock (with the `\item` missing)

```
448   \legacy_if:nT { @newlist }
```

```
449     {
```

```
450       \@noitemerr
```

```
451       \legacy_if_gset_false:n { @newlist }
```

```
452     }
```

```
453   \mode_if_horizontal:TF
```

```
454     { \__block_skip_remove_last: \__block_skip_remove_last: \par }
```

```
455     { \@inmatherr{\end{\@currenenv}} }
```

Once we are back in vertical mode we can add the appropriate closing tagging structure(s), if we are doing tagging.

```
456   \__kernel_displayblock_end:
```

What to do in terms of vertical spacing in different situations is still somewhat open to debate, right now this is more or less implementing what L^AT_EX 2_ε list environment have been doing.

```
457 %   \__block_debug_typeout:n{@nparlist =
```

```
458 %                                     \legacy_if:nTF { @nparlist }{true}{false}}
```

```
459 \legacy_if:nF { @nparlist }
```

```
460   {
```

```
461     \__block_skip_set_to_last:N \l_tmpa_skip
```

name is bad

some redesign/extensions here?


```

462 \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
463 {
464   \skip_vertical:n { - \l_tmpa_skip }
465   \skip_vertical:n { \l_tmpa_skip + \parskip - \@outerparskip }
466 }
467 \addpenalty \@endparpenalty
468 \addvspace \l_block_topsepadd_skip

```

L^AT_EX 2_ε triggered the paragraph handling after a list at this point here, i.e., only if the list didn't start a paragraph. One can make a case for that, but it can be somewhat surprising to the user and there is a good argument that even such a list could be followed explanatory text that is part of the same paragraph and doesn't start a new one.

decide which logic we want to use! If the old logic is used we need to close the text-unit ourselves in the true branch

decide

```

469 % \legacy_if_gset_true:n { @endpe }
470 }

```

So this is for now always done. Probably `\l_block_topsepadd_skip` above should be added only if the paragraph ends here and not if it continues, so this need some further

cleanup.

Finally, we have a socket that handles the `\par` handling after the block. Normally, we use it with the `on` plug (check for a following `\par`) but in the case of standalone environments we assign it the `off` plug.

```

471 \socket_use:n {tagsupport/block-endpe}
472 }

```

(End of definition for `\endblockenv`. This function is documented on page ??.)

`_kernel_displayblock_end:` The kernel hook for tagging at the end of the block.

```

473 \cs_new:Npn \_kernel_displayblock_end: {
474   \_block_debug_typeout:n{\detokenize{\_kernel_displayblock_end:}}
475 }

```

(End of definition for `_kernel_displayblock_end:`)

`tagsupport/block-endpe` (*socket*) This socket is responsible for the end environment `\par` handling. We define two plugs for it (`on` and `off`).

```

476 \socket_new:nn {tagsupport/block-endpe}{0}

```

`on` (*plug*) The plugs set the legacy `@endpe` switch. This must always happen because block environments with different settings can be nested and should not inherit the setting from the outer environment.

`off` (*plug*)

```

477 \socket_new_plug:nnn{tagsupport/block-endpe}{on}
478   { \legacy_if_gset_true:n { @endpe } }
479 \socket_new_plug:nnn{tagsupport/block-endpe}{off}
480   { \legacy_if_gset_false:n { @endpe } }
481 \socket_assign_plug:nm{tagsupport/block-endpe}{on}

```

4.5.2 Implementation of para templates ...

para std (*templ.*)

```

482 \DeclareTemplateCode{para}{std}{1}
483 {
484   indent-width      = \parindent ,
485   start-skip        = \l__par_start_skip ,           % name??
486   left-skip         = \leftskip ,
487   right-skip        = \rightskip ,
488   end-skip          = \parfillskip ,
489   fixed-word-spaces = \l__par_fixed_word_spaces_bool , % name??
490   final-hyphen-demerits = \finalhyphendemerits ,
491   cr-cmd            = \\ ,
492   para-class        = \l_tag_para_attr_class_tl ,
493 }
494 {
495   \tl_if_empty:nF {#1} { \SetTemplateKeys{para}{std}{#1} }
496   \skip_set:Nn \@rightskip \rightskip
497 }

```

4.5.3 Implementation of block templates ...

block display (*templ.*)

```

498 \DeclareTemplateCode{block}{display}{1}
499 {
500   heading           = \l__block_heading_tl ,
501   beginsep          = \topsep ,
502   begin-par-skip    = \partopsep ,
503   par-skip          = \parsep ,
504   end-skip          = \l__block_botsep_skip ,
505   end-par-skip      = \l__block_parbotsep_skip ,
506   beginpenalty      = \@beginparpenalty ,
507   endpenalty        = \@endparpenalty ,
508   rightmargin       = \rightmargin ,
509   leftmargin        = \leftmargin ,
510   parindent         = \listparindent ,
511 }
512 {
513   \tl_if_empty:nF {#1} { \SetTemplateKeys{block}{display}{#1} }
514   \tl_if_blank:oF \l__block_heading_tl
515   { \mode_leave_vertical: \textbf{\l__block_heading_tl} } % TODO customize

```

The code largely follows the logic of L^AT_EX 2_ε's `trivlist` implementation as far as it applicable for the “display block” but coded using the L³ programming layer. However, we keep all the legacy variables (e.g., `@noskipsec`) if there is some chance that they are set in classes or packages.

```

516   \legacy_if:nT { @noskipsec } { \mode_leave_vertical: }
517   \skip_set:Nn \l__block_topsepadd_skip { \topsep }
518   \mode_if_vertical:TF
519   {
520     \skip_add:Nn \l__block_topsepadd_skip { \partopsep }

```

generalize heading usage
(or drop?)

At this point it is safe to add tagging structure(s) so we have a kernel-owned hook here for tagging. This is used to possibly start a paragraph structure (to surround the block, for example, in case of lists) and possibly do some other preparation for tagging the block.

```

521     \__kernel_displayblock_beginpar_vmode:
522     }
523     {

```

If we are in horizontal mode then the displayblock has to return to vertical mode now (after removing any immediately preceding skip or kern. But before we actually issue the `\par` we execute a kernel hook in which we can add tagging code. This hook is “weird” because by default it does nothing, but if tagging is wanted it takes an argument and grabs the following `\par` in order to put tagging code before and after the `\par`.

```

524     \__block_skip_remove_last: \__block_skip_remove_last:
525     \__kernel_displayblock_beginpar_hmode:w \par
526     }

```

Now we are back to legacy list implementation ...

```

527     \legacy_if:nTF { @inlabel }
528     {
529         \legacy_if_set_true:n { @noparitem }
530         \legacy_if_set_true:n { @noparlist }
531     }
532     {
533         \legacy_if:nT { @newlist } { \@noitemerr }
534         \legacy_if_set_false:n { @noparlist }
535         \skip_set_eq:NN \l__block_effective_top_skip \l__block_topseppadd_skip
536     }
537     \skip_add:Nn \l__block_effective_top_skip { \parskip }

```

Next lines set some paragraph defaults, this may get overwritten if there is a `para`-instance specified on the `blockenv`.

```

538     \skip_zero:N \leftskip
539     \skip_set_eq:NN \rightskip \@rightskip
540     \skip_set_eq:NN \parfillskip \@flushglue

```

The next lines establish a parshape which is retained across paragraphs by executing `\para_end:` within a group and thus reestablishing the parshape for the next paragraph again. In case a list got started `\par` is ignored until we have seen an `\item` (or we have executed `\par` one thousand times.

```

541     \int_zero:N \par@deathcycles
542     \@setpar
543     {
544         \legacy_if:nTF { @newlist }
545         {
546             \int_incr:N \par@deathcycles
547             \int_compare:nNnTF \par@deathcycles > { 1000 }
548             { \@noitemerr
549               { \para_end: }
550             }
551         }
552         {
553             { \para_end: }
554         }
555     }

```

```

556 \skip_set_eq:NN \@outerparskip \parskip
557 \skip_set_eq:NN \parskip \parsep
558 \dim_set_eq:NN \parindent \listparindent
559 \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
560 \dim_add:Nn \@totalleftmargin { \leftmargin }
561 \tex_parshape:D 1 ~ \@totalleftmargin \linewidth

```

This is the point where we are ready to add the tagging structure for the block, e.g., an <L>, a <Figure> or some other structure.

```

562 \__kernel_displayblock_begin:

```

Finally, we have to output the vertical separation and penalty at the start of the block and make corrections for a change in \parskip and some other housekeeping, unless this block is inside a list and the list \item has not yet placed. In that case the vertical space and penalty us suppressed. This is controled through the legacy switches @noperitem, minipage, and @nobreak.

```

563 \legacy_if:nTF { @noperitem }
564 {
565   \legacy_if_set_false:n { @noperitem }
566   \hbox_gset:Nn \g__block_labels_box
567   {
568     \skip_horizontal:n { - \leftmargin }
569     \hbox_unpack_drop:N \g__block_labels_box
570     \skip_horizontal:n { \leftmargin }
571   }
572   \legacy_if:nF { @minipage } % Why this chunk of code?
573   {
574     \__block_skip_set_to_last:N \l__block_tmpa_skip
575     \skip_vertical:n { - \l__block_tmpa_skip }
576     \skip_vertical:n { \l__block_tmpa_skip + \@outerparskip - \parskip }
577   }
578 }
579 {
580   \legacy_if:nTF { @nobreak }
581   { \addvspace{\skip_eval:n{\@outerparskip-\parskip}} }
582   {
583     \addpenalty \@beginparpenalty
584     \addvspace \l__block_effective_top_skip
585     \addvspace{-\parskip}
586   }
587 }
588 }

```

Extra keys to support enumitem conventions:

```

589 \keys_define:nn { template/block/display }
590 {
591   ,topsep      .skip_set:N = \topsep
592   ,partopsep   .skip_set:N = \partopsep
593   ,listparindent .skip_set:N = \listparindent
594 }

```

__kernel_displayblock_begin: The internal kernel hooks for tagging.

```

\__kernel_displayblock_beginpar_hmode:w 595 \cs_new:Npn \__kernel_displayblock_begin: {
\__kernel_displayblock_beginpar_vmode: 596   \__block_debug_typeout:n{\detokenize{\__kernel_displayblock_begin:}}
597 }

```

document 2e logic used here

```

598 \cs_new:Npn \__kernel_displayblock_beginpar_hmode:w {
599   \__block_debug_typeout:n{\detokenize{\__kernel_displayblock_beginpar_hmode:w}}
600 }
601 \cs_new:Npn \__kernel_displayblock_beginpar_vmode: {
602   \__block_debug_typeout:n{\detokenize{\__kernel_displayblock_beginpar_vmode:}}
603 }

```

(End of definition for `__kernel_displayblock_begin:`, `__kernel_displayblock_beginpar_hmode:w`, and `__kernel_displayblock_beginpar_vmode:.`)

4.5.4 Implementation of list templates ...

`\@itemlabel` Both `\@itemlabel` and `\@listctr` from the L^AT_EX 2_ε list implementation are used (or `\@listctr` set) by various packages. We therefore use them too, so that these packages have a fighting chance to work with the new tagging-aware implementation for `list`.

```

604 \tl_new:N \@itemlabel      % should have a top-level definition
605 \tl_new:N \@listctr       % should have a top-level definition

```

(End of definition for `\@itemlabel` and `\@listctr`. These functions are documented on page ??.)

`list std (templ.)` This template implements numbered and unnumbered lists and can be combined with display blocks or with inline blocks.

```

606 \DeclareTemplateCode{list}{std}{1}
607 {
608   counter          = \l__block_counter_tl,
609   item-label       = \l__block_item_label_tl,
610   start            = \l__block_counter_start_int ,
611   resume           = \l__block_resume_bool ,
612   item-instance    = \__block_item_instance:n ,
613   item-skip        = \itemsep ,
614   % item-par-skip  = \parsep ,
615   item-penalty     = \@itempenalty ,
616   item-indent      = \itemindent ,
617   label-width      = \labelwidth ,
618   label-sep        = \labelsep ,
619   legacy-support   = \l__block_legacy_support_bool , % FMI questionable
620 }
621 {
622   \__block_debug_typeout:n{template:list:std}
623   %
624   \tl_if_empty:nF {#1} { \SetTemplateKeys{list}{std}{#1} }

```

Has this list a counter name defined in the instance?

```

625   \tl_if_empty:NTF \l__block_counter_tl
626   {

```

If not we check if `\@listctr` has a non-empty value to be used for the list counter.

We better test for blank not empty in case somebody had defined `\@listctr` using `\renewcommand` or `\cs_set:Npn`.

```

627     \tl_if_blank:oF \@listctr
628     {

```

In that case `@nmbriest` should have been set too, for example, through `\usecounter`, so we do not set it explicitly. However, we check if we should resume a previous list.

```

629     \bool_if:NF \l__block_resume_bool
630     {
631         \int_gset:cn{ c@ \@listctr }
632         { \l__block_counter_start_int - 1 }
633     }
634 }

```

If `\@listctr` is not set then we have definitely an unnumbered list.

```

635     { \@nmbriestfalse }
636 }

```

If a counter is set in the list instance we use that one. This should be the name of a \LaTeX counter that is already allocated externally—no runtime check is made for this: if it is not declared one will get “no such counter” error when the list is used.

```

637     {
638         \@nmbriesttrue
639         \tl_set_eq:NN \@listctr \l__block_counter_tl
640         \bool_if:NF \l__block_resume_bool
641         {
642             \int_gset:cn{ c@ \@listctr }
643             { \l__block_counter_start_int - 1 }
644         }
645     }

```

Does the current instance has an item label representation? This would be possible whether or not we have a numbered list. If yes, then we use this for `\@itemlabel`, otherwise we expect that `\@itemlabel` is provided from the outside, e.g., as part of the `list` environment argument.

```

646     \tl_if_empty:NF \l__block_item_label_tl
647     {
648         \tl_set_eq:NN \@itemlabel \l__block_item_label_tl
649     }

```

finally, we signal that we are at the start of a new list (which effects how the first `\item` is handled and how `\par` commands are interpreted).

```

650     \legacy_if_gset_true:n { @newlist }

651     \__block_debug_typeout:n{template:list:std~end}
652 }

```

Extra keys to support enumitem conventions:

```

653 \keys_define:nn { template/list/std }
654 {
655     ,nosep .code:n =
656         \dim_zero:N \itemsep
657         \dim_zero:N \parsep
658         \dim_zero:N \topsep
659         \dim_zero:N \l__block_botsep_skip
660         \dim_zero:N \l__block_parbotsep_skip
661     ,midsep .skip_set:N = \topsep
662 }

```

4.5.5 Implementation of \item template(s)

`item std (templ.)` The item template has one hidden key `label` which is not available on the template for setting because it is only used to receive any optional data passed to the `\item` command. We therefore declare it with `\keys_define:nn` and ensure that the optional argument data to `\item` (if it is not a key/value list already) is passed to this `label` key.

```

663 \keys_define:nn { template/item/std }
664           { label .tl_set:N = \l__block_label_given_tl }
665 \DeclareTemplateCode{item}{std}{1}
666   {
667     counter-label = \__block_counter_label:n ,
668     counter-ref   = \__block_counter_ref:n ,
669
670     label-ref     = \__block_label_ref:n ,
671     label-autoref = \__block_label_autoref:n ,
672     label-format  = \__block_label_format:n ,
673     label-strut   = \l__block_label_strut_bool ,
674     label-boxed   = \l__block_label_boxed_bool ,
675     next-line     = \l__block_next_line_bool ,
676     text-font     = \l__block_text_font_tl ,
677     compatibility = \l__block_item_compatibility_bool ,

```

alignment is mostly wrong
(test short medium and
multiline labels)

next set of key not yet
used

complete

This probably needs a different implementation (and needs completing)

```

677     label-align   = {
678       left    = \tl_set:Nn \l__block_item_align_tl { \relax \hss } ,
679       center  = \tl_set:Nn \l__block_item_align_tl { \hss \hss } ,
680       right   = \tl_set:Nn \l__block_item_align_tl { \hss \relax } ,
681       parleft = \NOT_IMPLEMENTED ,
682     } ,
683   }

```

Then typeset the label at its natural width by applying `__block_make_label_box:n` to the label given or to a label constructed from the counter. If it is boxed and reasonably short, add padding to make it at least of size `\labelwidth`, then add another layer of box. This way, when we unpack it in `\g__block_labels_box` it correctly remains boxed in those cases. Afterwards, in the `nextline` case add `\newline` if the label did not fit in the allotted space.

```

684   {
685     \__block_debug_typeout:n{template:item:std}

```

First deal with the key-value input, which in particular may provide a value for the label (the usual optional argument of `\item`). For this we set `\l__block_label_given_tl` to `\c_novalue_tl` so that we can identify if an optional argument was given.

```

686     \tl_set_eq:NN \l__block_label_given_tl \c_novalue_tl
687     \tl_if_empty:nF{#1}{ \SetTemplateKeys{item}{std}{#1} }

```

If no optional argument was given then `\l__block_label_given_tl` is still equal to `\c_novalue_tl` and so we can distinguish that from `\item[]`.

```

688     \tl_if_novalue:oTF \l__block_label_given_tl
689     {

```

fix

The rest of the code for this template needs work and is both incomplete and partly wrong.

```

690     \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
691     \bool_if:NTF \l__block_item_compatibility_bool % not sure that conditional
692               % makes sense
693     { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{\@itemlabel } } % TODO ?
694     { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{\__block_counter_label:n
695     }
696     {
697     \__block_debug_typeout:n{item~ with~ optional}
698     \__block_make_label_box:n { \l__block_label_given_tl } }
699 \bool_if:nT
700   {
701     \l__block_label_boxed_bool
702     && \dim_compare_p:n { \box_wd:N \l__block_one_label_box <= \linewidth } % TODO: is \
703     }
704     {
705     \dim_compare:nNnT
706     { \box_wd:N \l__block_one_label_box } < \linewidth
707     {
708     \hbox_set_to_wd:Nnn \l__block_one_label_box { \linewidth }
709     {
710     \exp_after:wN \use_i:nn \l__block_item_align_tl
711 %
712     \hbox_unpack_drop:N \l__block_one_label_box %TODO: customize?
713     \box_use_drop:N \l__block_one_label_box
714     \exp_after:wN \use_ii:nn \l__block_item_align_tl
715     }
716     \hbox_set:Nn \l__block_one_label_box
717     { \box_use_drop:N \l__block_one_label_box }
718     }
719 \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \linewidth
720 { \bool_set_true:N \l__block_long_label_bool }
721 { \bool_set_false:N \l__block_long_label_bool }
722 \hbox_gset:Nn \g__block_labels_box
723 {
724 \hbox_unpack_drop:N \g__block_labels_box
725 \skip_horizontal:n { \itemindent - \labelsep - \linewidth }
726 \hbox_unpack_drop:N \l__block_one_label_box
727 \skip_horizontal:n { \labelsep }
728 \bool_if:NT \l__block_next_line_bool
729 { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
730 % version of \newline inside an hbox that will be unpacked
731 }
732 % \skip_set_eq:NN \parsep \l__block_item_parsep_skip TODO??? FMI
733 % what's that?
734 \dim_set_eq:NN \parindent \listparindent

```

FMI: L^AT_EX 2_ε keeps the label boxed inside (not unboxed). This means that the content stays rigid and does not vary based on glue setting in the line with the label. There are cases where we do want the unboxed version (I think enumitem offers that in some cases too) but it should probably not be the default.

Placing the list label(s) is done when the paragraph for the `\item` is started, which executes `__block_item_everypar:` inside `para/begin`. By default this command does nothing, now we change it to attach the pending label or labels.

```
735 \cs_set_eq:NN \__block_item_everypar: \__block_item_everypar_std:
736 }
```

`\l_block_one_label_box` Each label is typeset in `\l_block_one_label_box` to be measured. Once this is ready, it is put (boxed or unboxed) in `\g_block_labels_box`, together with any pending labels (for the case where a list begins just after `\item`). This is an analogue of L^AT_EX 2_ε's `\@labels`, but it is always unboxed before use, to support both boxed and unboxed labels.

```
737 \box_new:N \l_block_one_label_box
738 \box_new:N \g_block_labels_box
```

(End of definition for `\l_block_one_label_box` and `\g_block_labels_box`.)

`\l_block_long_label_bool` Track whether the `\l_block_one_label_box` is larger than `\labelwidth`.

```
739 \bool_new:N \l_block_long_label_bool
```

(End of definition for `\l_block_long_label_bool`.)

`__block_make_label_box:n` Make one label, wrapped in `__block_label_format:n`, with an appropriate `\strut` and possibly `\makeLabel` in compatibility mode (used for the list environment).

```
740 \cs_new_protected:Npn \__block_make_label_box:n #1
741 {
742   \hbox_set:Nn \l_block_one_label_box
743   {
```

If we do tagging then the contents of this box may need to be wrapped into a structure, e.g., `<Lbl>`.

```
744   \__kernel_list_label_begin:
745   \__block_label_format:n
746   {
747     \bool_if:NT \l_block_label_strut_bool { \strut }
748     \bool_if:NTF \l_block_legacy_support_bool
749       \makeLabel
750       \use:n
751       {#1}
752   }
```

And what gets opened also needs closing:

```
753   \__kernel_list_label_end:
754 }
755 }
```

(End of definition for `__block_make_label_box:n` and `__block_label_format:e`.)

`__kernel_list_label_begin:` If we aren't doing tagging the kernel hooks do nothing.

```
\__kernel_list_label_end:
756 \cs_new_eq:NN \__kernel_list_label_begin: \prg_do_nothing:
757 \cs_new_eq:NN \__kernel_list_label_end: \prg_do_nothing:
```

(End of definition for `__kernel_list_label_begin:` and `__kernel_list_label_end:.`)

`_block_item_everypar:` The `_block_item_everypar:` command is executed as part of `para/begin` but most of the time does nothing, i.e., it has the following default definition.

```
758 \cs_new_eq:NN \_block\_item\_everypar: \prg\_do\_nothing:
759 \AddToHook{para/begin}[lists]{\_block\_item\_everypar:}
```

Note that we have to make sure that the above code is executed after the hook chunk from `tagpdf` because the latter uses `@inlabel` to make a decision.

By the end of the day both should probably move into the kernel hook instead!

```
760 \DeclareHookRule{para/begin}[lists]{after}{tagpdf}
```

What follows is the version that resets various legacy booleans and puts the label box in the right place and finally resets itself to do nothing next time. `_block_item_everypar:` is set to this by the item template so that the next paragraph start runs the code below.

```
761 \cs_new_protected:Npn \_block\_item\_everypar\_std: {
762   \_block\_debug\_typeout:n{item~ everypar \on@line }
763   \legacy\_if\_set\_false:n { @minipage }
764   \legacy\_if\_gset\_false:n { @newlist }
765   \legacy\_if:nT { @inlabel }
766   {
767     \legacy\_if\_gset\_false:n { @inlabel }
768     \box\_if\_empty:NT \g\_para\_indent\_box { \kern - \itemindent }
769     \para\_omit\_indent:
770     \box\_use\_drop:N \g\_block\_labels\_box
```

After the labels are placed we start a paragraph structure (if appropriate). This is handled in the following kernel hook:

```
771   \_kernel\_list\_label\_after:
772   \penalty \c\_zero\_int
773   }
774   \legacy\_if:nTF { @nobreak }
775   {
776     \legacy\_if\_gset\_false:n { @nobreak }
777     \int\_set:Nn \clubpenalty { 10000 }
778   }
779   {
780     \int\_set\_eq:NN \clubpenalty \@clubpenalty
```

Once the label(s) are typeset and we are past any special `@nobreak` handling we reset `_block_item_everypar:` to do nothing.

```
781   \cs\_set\_eq:NN \_block\_item\_everypar: \prg\_do\_nothing:
782   }
783 }
```

(End of definition for `_block_item_everypar:` and `_block_item_everypar_std:.`)

`_kernel_list_label_after:`

```
784 \cs\_new\_eq:NN \_kernel\_list\_label\_after: \prg\_do\_nothing:
```

(End of definition for `_kernel_list_label_after:.`)

`\l_block_tmpa_skip`

```
785 \skip\_new:N \l\_block\_tmpa\_skip
```

(End of definition for `\l_block_tpa_skip`.)

`\l_block_topsepad_skip` `\l_block_effective_top_skip` Variables equivalent to L^AT_EX 2_ε's `\@topsepad` and `\@topsep`. Roughly equal to a mixture of `topsep`, `partopsep`, and various `parskip` at different nesting levels in lists. The code is really elaborate when `@inlabel` is true.

```
786 \skip_new:N \l_block_topsepad_skip
787 \skip_new:N \l_block_effective_top_skip
```

(End of definition for `\l_block_topsepad_skip` and `\l_block_effective_top_skip`.)

`\item` Here we already have all the building blocks. Complain in math mode. Distinguish between first item (do necessary tagging) and later items `_block_inter_item:` to cleanly close what's before, then call `_block_item_instance:n` (which calls `\UseInstance{item}{(instance)}`) to prepare the upcoming item: it will be actually inserted only once some later material triggers `\everypar`.

```
788 \AddToHook{begindocument/before}{
789   \RenewDocumentCommand{\item}{={label}o }
790   {
791     \@inmatherr \item
```

TODO: Test for being outside of a list needs updating!

```
792     \tl_if_empty:oTF \_block_item_instance:n %%FMi?
793     { \msg_error:nnn { _block } { item-in-nonlist } { \item[#{1}] } }
794     {
795       \legacy_if:nTF { @newlist }
796       { \_kernel_list_item_begin: }
797       { \_block_inter_item: }

```

To avoid unnecessary key/val processing we make a quick check if there was an optional argument.

```
798     \tl_if_novalue:nTF {#{1}} % avoids reparsing label={ }
799     { \_block_item_instance:n { } }
800     { \_block_item_instance:n {#{1}} }
```

Set the legacy switch that signals that we have a pending item label:

```
801     \legacy_if_gset_true:n { @inlabel }
802     \ignorespaces
803   }
804 }
805 }
```

(End of definition for `\item`. This function is documented on page ??.)

`_block_inter_item:` Between items. If the previous item had no content then we need to trigger `\everypar`. Otherwise we simply close the previous item with `\par` after removing some horizontal space. Between items, there is a penalty and some space.

```
806 \cs_new_protected:Npn \_block_inter_item: {
807   \legacy_if:nT { @inlabel }
808   { \indent \par } % case of \item\item
```

`\par` may have a strange definition and may not get us back to vertical mode in one go, so we better do not treat the next line as an else case to the above conditional (for now).

```
809   \mode_if_horizontal:T { \_block_skip_remove_last:
810     \_block_skip_remove_last: \par }
```

End any LI-tag, then start the next LI-tag (if doing tagging):

```

811 \__kernel_list_item_end:
812 \__kernel_list_item_begin:
813 \addpenalty \@itempenalty
814 \addvspace \itemsep
815 }

```

(End of definition for __block_inter_item:.)

```

\__kernel_list_item_begin:
\__kernel_list_item_end:
816 \cs_new_eq:NN \__kernel_list_item_begin: \prg_do_nothing:
817 \cs_new_eq:NN \__kernel_list_item_end: \prg_do_nothing:

```

(End of definition for __kernel_list_item_begin: and __kernel_list_item_end:.)

4.6 Tagging recipes

__block_recipe_basic: The basic recipe simply ensures that the block is inside a `text-unit` structure and if necessary starts one. When the block ends and is followed by a blank line the `text-unit` structure is closed too, otherwise it remains open and further text starts with just a `<text>` structure.

There is otherwise no inner structure so `__kernel_displayblock_begin:` and `__kernel_displayblock_end:` do nothing—`blockenvs` with inner structure use the `standard` or `list` recipe instead.

```

818 \cs_new:Npn \__block_recipe_basic: {
819 \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
820 \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:N
821 \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
822 \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
823 \let \__kernel_displayblock_begin: \prg_do_nothing:
824 \let \__kernel_displayblock_end: \prg_do_nothing:

```

End environment `\par` handling:

```

825 \socket_assign_plug:nn{tagsupport/block-endpe}{on}
826 }

```

(End of definition for __block_recipe_basic:.)

__block_recipe_standalone: The `standalone` recipe produces a block that ensures that a previous `text-unit` ends and that after the block a new `text-unit` starts.

```

827 \cs_new:Npn \__block_recipe_standalone: {
828 \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
829 \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:N
830 \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
831 \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
832 \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
833 \cs_set_eq:NN \__kernel_displayblock_end: \__block_inner_end:

```

End environment `\par` handling:

```

834 \socket_assign_plug:nn{tagsupport/block-endpe}{off}
835 \tl_if_empty:NTF \l__block_tag_name_tl
836 { \tl_set:Nn \l__block_tag_inner_tag_tl {Sect} }
837 { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
838 }

```

(End of definition for `_block_recipe_standalone:`)

`_block_recipe_standard:` The **standard** recipe does the following:

- surround the block with a `text-unit`-structure if not already in a `text-unit`. In the latter case end the MC and the `<text>` but leave the `text-unit` open.
If we are producing flattened paragraphs, just close any `<text>` but do not open a `text-unit`.
- Then open an new (inner) structure (by default `Figure` but typically the one specified on the instance).
- At the end of the block close the the inner structure (`Figure` or explicit one) but leave the `text-unit` open to be either continued or closed due to a following `\par`.

```
839 \cs_new:Npn \_block_recipe_standard:
840 {
841   \cs_set_eq:NN \_kernel_displayblock_beginpar_hmode:w
842                                     \_block_beginpar_hmode:N
843   \cs_set_eq:NN \_kernel_displayblock_beginpar_vmode:
844                                     \_block_beginpar_vmode:
845   \cs_set_eq:NN \_kernel_displayblock_begin: \_block_inner_begin:
846   \cs_set_eq:NN \_kernel_displayblock_end:  \_block_inner_end:
```

End environment `\par` handling:

```
847   \socket_assign_plug:nn{tagssupport/block-endpe}{on}
848   \tl_if_empty:NTF \l__block_tag_name_tl
849     { \tl_set:Nn \l__block_tag_inner_tag_tl {Figure}          }
850     { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
851 }
```

(End of definition for `_block_recipe_standard:`)

`\l__block_tag_inner_tag_tl`

```
852 \tl_new:N \l__block_tag_inner_tag_tl
```

(End of definition for `\l__block_tag_inner_tag_tl`.)

`_block_recipe_list:` The **list** recipe does the following.

- It opens a `<text-unit>`-structure or keeps the current one open (only closing the MC).
- It then starts a new structure rolemapped to L-structure and arranges for handling list items, e.g., `Li`, `Lbl` and `LBody` structures.
- At the end it closes open list structures as needed but keeps the `<text-unit>`-structure open to continue the paragraph after the list, if necessary.

```
853 \cs_new:Npn \_block_recipe_list:
854 {
855   \cs_set_eq:NN \_kernel_displayblock_beginpar_hmode:w
856                                     \_block_beginpar_hmode:N
857   \cs_set_eq:NN \_kernel_displayblock_beginpar_vmode:
858                                     \_block_beginpar_vmode:
859   \cs_set_eq:NN \_kernel_displayblock_begin: \_block_list_begin:
860   \cs_set_eq:NN \_kernel_displayblock_end:  \_block_list_end:
```

The next two lines could be done globally, because they are only called if we do have `\items`, i.e., if we are in a list. It is therefore also not necessary to reset them in other recipes (right now—this may change if we get more templates (like inline lists)).

```
861 \cs_set_eq:NN \__kernel_list_item_begin: \__block_list_item_begin:
862 \cs_set_eq:NN \__kernel_list_item_end: \__block_list_item_end:
```

End environment `\par` handling:

```
863 \socket_assign_plug:nn{tag-support/block-endpe}{on}
```

Handle the tag name and attribute classes using the key values from the current list instance.

```
864 \tl_if_empty:NTF \l__block_tag_name_tl
865   { \tl_set:Nn \l__tag_L_tag_tl {L} }
866   { \tl_set_eq:NN \l__tag_L_tag_tl \l__block_tag_name_tl }
867 \tl_if_empty:NTF \l__block_tag_class_tl
868   { \tl_set:Nn \l__tag_L_attr_class_tl {} }
869   { \tl_set_eq:NN \l__tag_L_attr_class_tl \l__block_tag_class_tl }
870 }
```

(End of definition for `__block_recipe_list:`.)

4.7 Blockenv instances

4.7.1 Basic instances

`blockenv displayblock` (*inst.*)

```
871 \DeclareInstance{blockenv}{displayblock}{display}
872 {
873   env-name      = displayblock,
874   tag-name      = ,
875   tag-class     = ,
876   tagging-recipe = standard,
877   inner-level-counter = ,
878   level-increase = false,
879   setup-code    = ,
880   block-instance = displayblock ,
881   inner-instance = ,
882 }
```

`blockenv displayblockflattened` (*inst.*)

```
883 \DeclareInstance{blockenv}{displayblockflattened}{display}
884 {
885   env-name      = displayblockflattened,
886   tag-name      = ,
887   tag-class     = ,
888   tagging-recipe = basic,
889   inner-level-counter = ,
890   level-increase = false,
891   setup-code    = ,
892   block-instance = displayblock ,
893   para-flattened = true ,
894   inner-instance = ,
895 }
```

blockenv center (*inst.*)

```
896 \DeclareInstance{blockenv}{center}{display}
897 {
898   env-name      = center,
899   tag-name      = ,
900   tag-class     = ,
901   tagging-recipe = basic,
902   inner-level-counter = ,
903   level-increase = false,
904   setup-code    = ,
905   block-instance = displayblock ,
906   para-flattened = true ,
907   para-instance = center ,
908   inner-instance = ,
909 }
```

blockenv flushleft (*inst.*)

```
910 \DeclareInstance{blockenv}{flushleft}{display}
911 {
912   env-name      = flushleft,
913   tag-name      = ,
914   tag-class     = ,
915   tagging-recipe = basic,
916   inner-level-counter = ,
917   level-increase = false,
918   setup-code    = ,
919   block-instance = displayblock ,
920   para-flattened = true ,
921   para-instance = raggedright ,
922   inner-instance = ,
923 }
```

blockenv flushright (*inst.*)

```
924 \DeclareInstance{blockenv}{flushright}{display}
925 {
926   env-name      = flushleft,
927   tag-name      = ,
928   tag-class     = ,
929   tagging-recipe = basic,
930   inner-level-counter = ,
931   level-increase = false,
932   setup-code    = ,
933   block-instance = displayblock ,
934   para-flattened = true ,
935   para-instance = raggedleft ,
936   inner-instance = ,
937 }
```

4.7.2 Blockquote instances

blockenv quotation (*inst.*)

```
938 \DeclareInstance{blockenv}{quotation}{display}
939 {
```

```

940 env-name      = quotation,
941 tag-name      = quotation,
942 tag-class     = ,
943 tagging-recipe = standard,
944 inner-level-counter = ,
945 level-increase = true,
946 setup-code    = ,
947 block-instance = quotationblock ,
948 inner-instance = ,
949 }

```

blockenv quote (*inst.*)

```

950 \DeclareInstance{blockenv}{quote}{display}
951 {
952   env-name      = quote,
953   tag-name      = quote,
954   tag-class     = ,
955   tagging-recipe = standard,
956   inner-level-counter = ,
957   level-increase = true,
958   setup-code    = ,
959   block-instance = quoteblock ,
960   inner-instance = ,
961 }

```

I guess the setup code is still executed too early, have to check.

An alternative setup for quotations, using the displayblock instance and just overwrite a bit in the setup code. This would be less flexible but would ensure visual consistency, because the displayblock settings are used throughout.

```

962 % \DeclareInstance{blockenv}{quotation}{display}
963 % {
964 %   env-name      = quotation,
965 %   tag-name      = ,
966 %   tag-class     = ,
967 %   tagging-recipe = blockquote,
968 %   inner-level-counter = ,
969 %   level-increase = true,
970 %   setup-code    = \setlength\rightmargin{\leftmargin}
971 %                  \setlength\parsep{1.5em} ,
972 %   block-instance = displayblock ,
973 %   inner-instance = ,
974 % }
975 % \DeclareInstance{blockenv}{quote}{display}
976 % {
977 %   env-name      = quote,
978 %   tag-name      = ,
979 %   tag-class     = ,
980 %   tagging-recipe = blockquote,
981 %   inner-level-counter = ,
982 %   level-increase = true,
983 %   setup-code    = \setlength\rightmargin{\leftmargin} ,
984 %   block-instance = displayblock ,
985 %   inner-instance = ,
986 % }

```



```

987 \DeclareInstance{blockenv}{theorem}{display}
988 {
989   env-name      = theorem-like,
990   tag-name      = theorem-like,
991   tag-class     = ,
992   tagging-recipe = standalone,
993   inner-level-counter = ,
994   level-increase = false,
995   setup-code    = ,
996   block-instance = displayblock ,
997   % inner-instance-type = innerblock ,
998   % inner-instance = theorem,
999 }

```

We use <theorem-like> as the structure name and rolemap it to a <Sect> because that can hold a <Caption>.

4.7.3 Verbatim instances

`blockenv verbatim` (*inst.*) The rolemapping is current verbatim to P and codeline to Sub (which is role mapped to Span in pdf 1.7. Alternatives for PDF 1.7: Div and P.

```

1000 \DeclareInstance{blockenv}{verbatim}{display}
1001 {
1002   env-name      = verbatim,
1003   tag-name      = verbatim,
1004   tag-class     = ,
1005   tagging-recipe = standard,
1006   inner-level-counter = ,
1007   level-increase = false,
1008   setup-code    = ,
1009   block-instance = verbatimblock ,
1010   inner-instance = ,
1011   final-code    = \legacyverbatimsetup ,
1012 }

```

4.7.4 Standard list instances

`blockenv itemize` (*inst.*)

```

1013 \DeclareInstance{blockenv}{itemize}{display}
1014 {
1015   env-name      = itemize,
1016   tag-name      = itemize,
1017   tag-class     = itemize,
1018   tagging-recipe = list,
1019   inner-level-counter = \@itemdepth,
1020   level-increase = true,
1021   max-inner-levels = 4,
1022   setup-code    = ,
1023   block-instance = list ,
1024   inner-instance = itemize ,
1025 }

```

`blockenv enumerate` (*inst.*)

```

1026 \DeclareInstance{blockenv}{enumerate}{display}
1027 {
1028   env-name           = enumerate,
1029   tag-name           = enumerate,
1030   tag-class          = enumerate,
1031   tagging-recipe     = list,
1032   level-increase     = true,
1033   setup-code         = ,
1034   block-instance     = list ,
1035   inner-level-counter = \@enumdepth,
1036   max-inner-levels   = 4,
1037   inner-instance     = enum ,
1038 }

```

`blockenv description` (*inst.*)

```

1039
1040 \DeclareInstance{blockenv}{description}{display}
1041 {
1042   env-name           = description,
1043   tag-name           = description,
1044   tag-class          = description,
1045   tagging-recipe     = list,
1046   inner-level-counter = ,
1047   level-increase     = true,
1048   setup-code         = ,
1049   block-instance     = list ,
1050   inner-instance     = description ,
1051 }

```

`blockenv list` (*inst.*) The general (legacy) list environment does some of its setup in the `setup-code` key.

```

1052 \DeclareInstance{blockenv}{list}{display}
1053 {
1054   env-name           = list,
1055   tag-name           = list,
1056   tag-class          = ,
1057   tagging-recipe     = list,
1058   level-increase     = true,
1059   setup-code         = \legacylistsetupcode ,
1060   block-instance     = list ,
1061   inner-level-counter = ,
1062   inner-instance     = legacy ,
1063 }

```

4.8 Block instances

4.8.1 Displayblock instances

We provide 6 nesting levels (as in L^AT_EX 2_ε). If you want to provide more you need to change the `maxblocklevels` counter, offer further `displayblock-xx` instances but also define further (legacy) `\list<romannumeral>` commands for the defaults. If not, then the settings from the previous level are reused automatically—which may or may not be good enough).

```

1064 \setcounter{maxblocklevels}{6}

```

block displayblock-0 (*inst.*) Here we need level zero as well in case a flattened displayblock (like the center env) it is used on top-level.

```

block displayblock-2 (inst.) 1065 \DeclareInstance{block}{displayblock-0}{display}
block displayblock-3 (inst.) 1066 {
block displayblock-4 (inst.) 1067     leftmargin      = Opt ,
block displayblock-5 (inst.) 1068     parindent       = Opt ,
block displayblock-6 (inst.) 1069 }

1070 \DeclareInstanceCopy{block}{displayblock-1}{displayblock-0}
1071 \DeclareInstanceCopy{block}{displayblock-2}{displayblock-0}
1072 \DeclareInstanceCopy{block}{displayblock-3}{displayblock-0}
1073 \DeclareInstanceCopy{block}{displayblock-4}{displayblock-0}
1074 \DeclareInstanceCopy{block}{displayblock-5}{displayblock-0}
1075 \DeclareInstanceCopy{block}{displayblock-6}{displayblock-0}

```

4.8.2 Verbatim instances

Verbatim instances have their own levels so that one can specify specific indentations or vertical separations between lines.

```

block verbatimblock-0 (inst.)
block verbatimblock-1 (inst.) 1076 \DeclareInstance{block}{verbatimblock-0}{display}
block verbatimblock-2 (inst.) 1077 {
block verbatimblock-3 (inst.) 1078     leftmargin      = Opt ,
block verbatimblock-4 (inst.) 1079     parindent       = Opt ,
block verbatimblock-5 (inst.) 1080     par-skip        = Opt ,
block verbatimblock-6 (inst.) 1081 }

1082 \DeclareInstanceCopy{block}{verbatimblock-1}{verbatimblock-0}
1083 \DeclareInstanceCopy{block}{verbatimblock-2}{verbatimblock-0}
1084 \DeclareInstanceCopy{block}{verbatimblock-3}{verbatimblock-0}
1085 \DeclareInstanceCopy{block}{verbatimblock-4}{verbatimblock-0}
1086 \DeclareInstanceCopy{block}{verbatimblock-5}{verbatimblock-0}
1087 \DeclareInstanceCopy{block}{verbatimblock-6}{verbatimblock-0}

```

4.8.3 Quote/quotationblock instances

Quote and quotation are not flattened, i.e., they change levels, thus they start with level 1 not 0.

```

block quoteblock-1 (inst.) Default layout is to indent equally from both sides.
block quoteblock-2 (inst.) 1088 \DeclareInstance{block}{quoteblock-1}{display}
block quoteblock-3 (inst.) 1089 { rightmargin = \KeyValue{leftmargin} }
block quoteblock-4 (inst.) 1090 \DeclareInstanceCopy{block}{quoteblock-2}{quoteblock-1}
block quoteblock-5 (inst.) 1091 \DeclareInstanceCopy{block}{quoteblock-3}{quoteblock-1}
block quoteblock-6 (inst.) 1092 \DeclareInstanceCopy{block}{quoteblock-4}{quoteblock-1}
1093 \DeclareInstanceCopy{block}{quoteblock-5}{quoteblock-1}
1094 \DeclareInstanceCopy{block}{quoteblock-6}{quoteblock-1}

```

```

block quotationblock-1 (inst.) Quotation additionally changes the parindent.
block quotationblock-2 (inst.) 1095 \DeclareInstance{block}{quotationblock-1}{display}
block quotationblock-3 (inst.) 1096 { parindent = 1.5em , rightmargin = \KeyValue{leftmargin} }
block quotationblock-4 (inst.)
block quotationblock-5 (inst.)
block quotationblock-6 (inst.)

```

```

1097 \DeclareInstanceCopy{block}{quotationblock-2}{quotationblock-1}
1098 \DeclareInstanceCopy{block}{quotationblock-3}{quotationblock-1}
1099 \DeclareInstanceCopy{block}{quotationblock-4}{quotationblock-1}
1100 \DeclareInstanceCopy{block}{quotationblock-5}{quotationblock-1}
1101 \DeclareInstanceCopy{block}{quotationblock-6}{quotationblock-1}

```

4.8.4 Block instances for the standard lists

block list-1 (*inst.*) The block instances for the various list environments use the same underlying instance **block list-2** (*inst.*) (well by default) and nothing needs to be set up specifically (because that is already done **block list-3** (*inst.*) in the legacy `\list{romannumeral}` unless a different layout is wanted.

```

block list-4 (inst.) 1102 \DeclareInstance{block}{list-1}{display}{
block list-5 (inst.) 1103 % heading = ,
block list-6 (inst.) 1104 % beginsep = \topsep ,
1105 % begin-par-skip = \partopsep ,
1106 % par-skip = \parsep ,
1107 % end-skip = \KeyValue{beginsep} ,
1108 % end-par-skip = \KeyValue{begin-par-skip} ,
1109 % beginpenalty = \UseName{@beginparpenalty} ,
1110 % endpenalty = \UseName{@endparpenalty} ,
1111 % leftmargin = \leftmargin ,
1112 % rightmargin = \rightmargin ,
1113 % parindent = \listparindent ,
1114 }
1115 \DeclareInstance{block}{list-2}{display}{}
1116 \DeclareInstance{block}{list-3}{display}{}
1117 \DeclareInstance{block}{list-4}{display}{}
1118 \DeclareInstance{block}{list-5}{display}{}
1119 \DeclareInstance{block}{list-6}{display}{}

```

4.9 List instances for the standard lists

For all list instances we have to say what kind of label we want (`label-instance`) and how it should be formatted.

list itemize-1 (*inst.*) For `itemize` environments this is all we need to do and we refer back to the external **list itemize-2** (*inst.*) definitions rather than defining the `item-label` code in the instance to ensure that old **list itemize-3** (*inst.*) documents still work.

```

list itemize-4 (inst.) 1120 \DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi }
1121 \DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii }
1122 \DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
1123 \DeclareInstance{list}{itemize-4}{std}{ item-label = \labelitemiv }

```

list enumerate-1 (*inst.*) `enumerate` environments are similar, except that we also have to say which counter to **list enumerate-2** (*inst.*) use on every level.

```

list enumerate-3 (inst.) 1124 \DeclareInstance{list}{enum-1}{std}
list enumerate-4 (inst.) 1125 { item-label = \labelenumi , counter = enumi }
1126 \DeclareInstance{list}{enum-2}{std}
1127 { item-label = \labelenumii , counter = enumii }
1128 \DeclareInstance{list}{enum-3}{std}
1129 { item-label = \labelenumiii , counter = enumiii }
1130 \DeclareInstance{list}{enum-4}{std}
1131 { item-label = \labelenumiv , counter = enumiv }

```

`list legacy` (*inst.*) For the legacy `list` environment there is only one instance which is reused on all levels. This is done this way one because the legacy `list` environment sets all its parameters through its arguments. So this instances shouldn't really be touched. It sets the `legacy-support` key to true, which means that the list code uses `\makeLabel` for formatting the label

```
1132 \DeclareInstance{list}{legacy}{std} {
1133   item-instance = basic ,
1134   legacy-support = true ,
1135 }
```

`list description` (*inst.*) The `description` lists also use only a single list instance with only one key not using the default:

```
1136 \DeclareInstance{list}{description}{std} { item-instance = description }
```

4.10 Item instances

`item basic` (*inst.*) There two item instances set up: `description` for use with the `description` environment and `basic` for use with all other lists (up to now).

```
1137 \DeclareInstance{item}{basic}{std}
1138 {
1139   label-align = right ,
1140 }
1141 \DeclareInstance{item}{description}{std}
1142 {
1143   label-format = \normalfont\bfseries #1 ,
1144 }
```

4.11 Para instances

```
1145 \tag_if_active:T
1146 {
1147   \tagpdfsetup
1148   {
1149     newattribute = {justify}      {/O /Layout /TextAlign/Justify},
1150     newattribute = {center}      {/O /Layout /TextAlign/Center},
1151     newattribute = {raggedright}{/O /Layout /TextAlign/Start},
1152     newattribute = {raggedleft}  {/O /Layout /TextAlign/End},
1153   }
1154 }
```

`para center` (*inst.*)

```
1155 \DeclareInstance{para}{center}{std}
1156 {
1157   indent-width           = Opt ,
1158   start-skip              = Opt ,
1159   left-skip               = \@flushglue ,
1160   right-skip              = \@flushglue ,
1161   end-skip                = \@skip ,
1162   final-hyphen-demerits = 0 ,
1163   cr-cmd                  = \@centercr ,
1164   para-class              = center ,
1165 }
```

```

1166 \DeclareInstance{para}{raggedright}{std}
1167 {
1168   indent-width      = Opt ,
1169   start-skip        = Opt ,
1170   left-skip         = \z@skip ,
1171   right-skip        = \@flushglue ,
1172   end-skip          = \z@skip ,
1173   final-hyphen-demerits = 0 ,
1174   cr-cmd            = \@centercr ,
1175   para-class        = raggedright ,
1176 }
1177 \DeclareInstance{para}{raggedleft}{std}
1178 {
1179   indent-width      = Opt ,
1180   start-skip        = Opt ,
1181   left-skip         = \@flushglue ,
1182   right-skip        = \z@skip ,
1183   end-skip          = \z@skip ,
1184   final-hyphen-demerits = 0 ,
1185   cr-cmd            = \@centercr ,
1186   para-class        = raggedleft ,
1187 }
1188 \DeclareInstance{para}{justify}{std}
1189 {
1190 % indent-width      = Opt ,
1191   start-skip        = Opt ,
1192   left-skip         = \z@skip ,
1193   right-skip        = \z@skip ,
1194   end-skip          = \@flushglue ,
1195   final-hyphen-demerits = 5000 ,
1196   cr-cmd            = \@normalcr ,
1197   para-class        = justify ,
1198 }
1199 \DeclareRobustCommand\centering {\UseInstance{para}{center}{}}
1200 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}{}}
1201 \DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}{}}
1202 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}{}}
1203
1204 \justifying

```

4.12 Tagging support

In this section we provide code to the various kernel hooks to support the tagging of the different displayblock environments.

All of the following definitions should only be made if tagging is active!

```

1205 \tag_if_active:TF {

```

`_block_beginpar_vmode:` When a block starts out in vertical mode, i.e., is not yet part of a paragraph, we have to start a paragraph structure. However, this is not the case if we are already flattening paragraphs, thus in this case we do nothing. We also do nothing if `@endpe` is currently true, because that means we are right now just after the end of a `blockenv` and in the process of looking if we have to end the current `text-unit`, i.e., it is already open.

```

1206 \cs_set:Npn \__block_beginpar_vmode: {
1207     \__block_debug_typeout:n
1208     { @endpe = \legacy_if:nTF { @endpe }{true}{false}
1209     \on@line }
1210 \legacy_if:nTF { @endpe }
1211 {
1212     \legacy_if_gset_false:n { @endpe }
1213 }

```

We test for <2 because the first flattened environment has to surround itself with a `text-unit`. Only any inner ones then have to avoid adding another `text-unit`.

```

1214 {
1215     \int_compare:nNnT \l__block_flattened_level_int < 2
1216     {
1217         \int_gincr:N \g__tag_para_main_begin_int
1218         \tagstructbegin{tag=\l__tag_para_main_tag_tl}
1219     }
1220 }
1221 }

```

(End of definition for __block_beginpar_vmode:.)

`__block_beginpar_hmode:N` If the block is already part of a part of a paragraph, i.e., when it has some text directly in front, then the first thing to do is to return to vertical mode. However, that should be done without inserting a paragraph end tag, so before calling `\par` to do its normal work, we disable paragraph tagging and restarting afterwards again. The argument to this config point simply gobbles the `\par` following it in the code above (which is used when there is no tagging going on).

```

1222 \cs_set:Npn \__block_beginpar_hmode:N #1
1223 {
1224     \tag_mc_end:
1225     \int_gincr:N \g__tag_para_end_int
1226     \__block_debug_typeout:n{increment~ /P \on@line }
1227     \bool_if:NT \l__tag_para_show_bool
1228     { \tag_mc_begin:n{artifact}
1229       \rlap{\color_select:n{red}\tiny\ \int_use:N\g__tag_para_end_int}
1230       \tag_mc_end:
1231     }
1232     \tag_struct_end:
1233     \tagpdfparaOff \par \tagpdfparaOn
1234 }

```

(End of definition for __block_beginpar_hmode:N.)

`__kernel_displayblock_doendpe:` If a display block ends and is followed by a blank line we have to end the enclosing paragraph tagging structure.

```

1235 \cs_set:Npn \__kernel_displayblock_doendpe: {
1236     \bool_if:NT \l__tag_para_bool
1237     {

```

Given that restoring `\par` through the legacy L^AT_EX 2_ε method can take a few iterations (for example, in case of nested lists, e.g., `... \end{itemize} \item ... \par` it can happen that `__kernel_displayblock_doendpe:` is called while `@endpe` is already handled and then we should not attempt to close a `text-unit` structure). So we need to check for this.

```

1238     \legacy_if:nT { @endpe }
1239     {

```

If the display block currently ending was “flattened” (i.e., uses simplified paragraphs that are not tagged by a combination of `text-unit` followed by `<text>`, but simply with a `<text>`), then we don’t have to do anything, because the `<text>` is already closed.

```

1240         \__block_debug_typeof:n
1241         { flattened= \bool_if:NTF
1242           \l__tag_para_flattened_bool {true}{false}
1243           \on@line }
1244     \bool_if:NF \l__tag_para_flattened_bool
1245     {
1246       \__block_debug_typeof:n{Structure-end~
1247         \l__tag_para_main_tag_tl\space after~ displayblock \on@line }
1248       \int_gincr:N \g__tag_para_main_end_int
1249       \tag_struct_end: %text-unit
1250     }
1251   }
1252 }
1253 }

```

(End of definition for `__kernel_displayblock_doendpe:.`)

para/begin Paragraph tagging is mainly done using the paragraph hooks (will get moved eventually). The default hook setting is not good enough when lists get supported: we need to delay starting the paragraph tagging if we still have to place the list label. We therefore remove the existing hook data and replace it with an augmented version (this will get combined eventually).

```

1254 \RemoveFromHook{para/begin}[tagpdf]
1255 \AddToHook{para/begin}[tagpdf]{
1256   \bool_if:NT \l__tag_para_bool {

```

if we are still waiting to typeset the list label we do nothing (the paragraph tagging then happens when the list is finally typeset).

```

1257     \legacy_if:nF { @inlabel }
1258     {

```

Otherwise, we start a `<text>` tag structure but only if we are not starting a paragraph immediately *after* a list, in which case we only start a new MC (because the `<text>` tag is still open from before the list — one of the reasons why lists are always put “inside” paragraphs).

We do this in a separate command, because it is needed elsewhere too.

```

1259     \__block_start_para_structure:n { \PARALABEL }
1260     }
1261   }
1262 }

```

```

\__block_start_para_structure:n 1263 \cs_new_protected:Npn \__block_start_para_structure:n #1 {
1264   \__block_debug_typeof:n
1265   { @endpe = \legacy_if:nTF { @endpe }{true}{false}
1266   \on@line }
1267   \legacy_if:nF { @endpe }
1268   {

```



```

1269     \bool_if:NF \l__tag_para_flattened_bool
1270     {
1271         \int_gincr:N \g__tag_para_main_begin_int
1272         \tag_struct_begin:n{tag=\l__tag_para_main_tag_tl}
1273     }
1274 }
1275 \int_gincr:N \g__tag_para_begin_int
1276 \__block_debug_typeout:n{increment~ P \on@line }
1277 \tag_struct_begin:n
1278 {
1279     tag=\l__tag_para_tag_tl
1280     ,attribute-class=\l_tag_para_attr_class_tl
1281 }
1282 \__tag_check_para_begin_show:nn {green}{#1}
1283 \tag_mc_begin:n {}
1284 }

```

The same code, but without testing @endpe. This is not needed in the standalone case and wrong inside lists.

```

1285 \cs_new_protected:Npn \__block_start_para_structure_unconditionally:n #1 {
1286     \bool_if:NF \l__tag_para_flattened_bool
1287     {
1288         \int_gincr:N \g__tag_para_main_begin_int
1289         \tag_struct_begin:n{tag=\l__tag_para_main_tag_tl}
1290     }
1291     \int_gincr:N \g__tag_para_begin_int
1292     \__block_debug_typeout:n{increment~ P \on@line }
1293     \tag_struct_begin:n
1294     {
1295         tag=\l__tag_para_tag_tl
1296         ,attribute-class=\l_tag_para_attr_class_tl
1297     }
1298     \__tag_check_para_begin_show:nn {green}{#1}
1299     \tag_mc_begin:n {}
1300 }

1301 \RemoveFromHook{para/end}[tagpdf]
1302 \AddToHook{para/end}
1303 {
1304     \bool_if:NT \l__tag_para_bool
1305     {
1306         \int_gincr:N \g__tag_para_end_int
1307         \__block_debug_typeout:n{increment~ /P \on@line }
1308         \tag_mc_end:
1309         \__tag_check_para_end_show:nn {red}{}
1310         \tag_struct_end:
1311         \bool_if:NF \l__tag_para_flattened_bool
1312         {
1313             \int_gincr:N \g__tag_para_main_end_int
1314             \tag_struct_end:
1315         }
1316     }
1317 }

1318 \def\PARALABEL{NP-}

```

(End of definition for `para/begin` and `__block_start_para_structure:n`. This function is documented on page ??.)

`\para_end:` If we see a `\par` in `vmode` and a `text-unit` is still open we need to close that. For this we check if a request for `@endpe` was made (but the `\par` redefinition got lost due to (bad?) coding).

```

1319 \cs_set_protected:Npn \para_end: {
1320   \scan_stop:
1321   \mode_if_horizontal:TF {
1322     \mode_if_inner:F {
1323       \tex_unskip:D
1324       \hook_use:n{para/end}
1325       \@kernel@after@para@end
1326       \mode_if_horizontal:TF {
1327         \if_int_compare:w 11 = \tex_lastnodetype:D
1328           \tex_hskip:D \c_zero_dim
1329         \fi:
1330         \tex_par:D
1331         \hook_use:n{para/after}
1332         \@kernel@after@para@after
1333       }
1334       { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
1335     }
1336   }
1337   {
1338     \__kernel_endpe_vmode:      % should do nothing if no tagging
1339     \tex_par:D
1340   }
1341 }
1342 \cs_set_eq:NN \par      \para_end:
1343 \cs_set_eq:NN \__blockpar \para_end:
1344 \cs_set_eq:NN \endgraf \para_end:

```

(End of definition for `\para_end:`. This function is documented on page ??.)

`\begin` We need to do a little more than canceling `@endpe` now.

```

1345 \DeclareRobustCommand*\begin[1]{%
1346   \UseHook{env/#1/before}%
1347   \@ifundefined{#1}%
1348     {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
1349     {\def\reserved@a{\def\@currenvir{#1}%
1350       \edef\@currencline{\on@line}%
1351       \@execute@begin@hook{#1}%
1352       \csname #1\endcsname}}%
1353   \@ignorefalse
1354   \begingroup
1355     \__kernel_endpe_vmode:
1356     \reserved@a}

```

(End of definition for `\begin`. This function is documented on page ??.)

`__kernel_endpe_vmode:` Close an open `text-unit` if `@endpe` is true and we are in `vmode`. Used in `\para_end:` and `\begin`.

```

1357 \cs_new:Npn \__kernel_endpe_vmode: {

```

```

1358     \if@endpe \ifvmode
1359         \bool_if:NT \l__tag_para_bool
1360     {
1361     \bool_if:NF \l__tag_para_flattened_bool
1362         {
1363             \int_gincr:N \g__tag_para_main_end_int
1364             \tag_struct_end:
1365         }
1366     \@endpefalse
1367     }
1368     \fi \fi
1369     }

```

(End of definition for `__kernel_endpe_vmode:`.)

`__kernel_list_label_after:` If starting the text-unit/text tags got delayed because of a pending label we have to do it after the label got typeset

```

1370 \cs_set:Npn \__kernel_list_label_after: {
1371     \bool_if:NT \l__tag_para_bool
1372     {
1373         \__block_start_para_structure_unconditionally:n { LI- }
1374     }
1375 }

```

(End of definition for `__kernel_list_label_after:`.)

`__block_inner_begin:` Start a block that has an inner structure if it isn't also a list.

```

1376 \cs_new:Npn \__block_inner_begin: {
1377     \tag_structbegin{tag=\l__block_tag_inner_tag_tl}
1378 }

```

(End of definition for `__block_inner_begin:`.)

`__block_inner_end:` End a block (which isn't also a list).

```

1379 \cs_new:Npn \__block_inner_end: {
1380     \__block_debug_typeout:n{block-end \on@line}
1381     \legacy_if:nT { @endpe }
1382     {
1383         \int_gincr:N \g__tag_para_main_end_int
1384         \__block_debug_typeout:n{close~ /text-unit \on@line}
1385         \tag_structend
1386     }
1387     \tag_structend           % end inner structure
1388 }

```

(End of definition for `__block_inner_end:`.)

4.12.1 List tags

```

1389 \tl_new:N \l__tag_L_tag_tl
1390 \tl_set:Nn \l__tag_L_tag_tl {L}
1391
1392 \tl_new:N \l__tag_L_attr_class_tl
1393 \tl_set:Nn \l__tag_L_attr_class_tl {list}
1394 \tag_if_active:T

```

```

1395 {
1396   \tagpdfsetup
1397   {
1398     % default if unknown
1399     newattribute = {list}{/0 /List /ListNumbering/None},
1400     newattribute = {itemize}{/0 /List /ListNumbering/Unordered},
1401     newattribute = {enumerate}{/0 /List /ListNumbering/Ordered},
1402     newattribute = {description}{/0 /List /ListNumbering/Description},
1403   }
1404 }

```

```

1405 \def\LItag{LI}

```

`_block_list_begin:` Start a list ...

```

1406 \cs_set:Npn \_block\_list\_begin: {
1407   \tagstructbegin
1408   {
1409     tag=\l\_tag\_L\_tag\_tl
1410     ,attribute-class=\l\_tag\_L\_attr\_class\_tl
1411   }
1412 }

```

(End of definition for _block_list_begin:.)

`_block_list_item_begin:` Start tagging a list item.

```

1413 \cs_set:Npn \_block\_list\_item\_begin: { \tagstructbegin{tag=\LItag} }

```

(End of definition for _block_list_item_begin:.)

`_kernel_list_label_begin:` A list label needs a Lbl structure tag and an MC.

```

1414 \cs_set:Npn \_kernel\_list\_label\_begin: {
1415   %
1416   % FMi: this needs a different logic to decide when to make the label
1417   %   an artifact (after cleaning up the the \item code ), therefore
1418   %   disabled for now
1419   % \tl_if_empty:oTF \@itemlabel
1420   % {
1421   %   \tag_mc_begin:n {artifact}
1422   % }
1423   % {
1424   %   \tagstructbegin{tag=Lbl}
1425   %   \tagmcbegin{tag=Lbl}
1426   % }
1427 }

```

(End of definition for _kernel_list_label_begin:.)

`_kernel_list_label_end:` And when we are done with the label we have to close the MC and the Lbl structure. We then start the LBody. The material inside will be “paragraph” text and the tagging for that is handled by the normal para tagging.

```

1428 \cs_set:Npn \_kernel\_list\_label\_end: {
1429   \tagmcbend % end mc-Lbl or artifact
1430   % FMi: unconditionally for now
1431   % \tl_if_empty:oF \@itemlabel
1432   \tagstructend % end Lbl

```

```

1433 \tagstructbegin{tag=\LBody}
1434 }
1435 \def\LBody{\LBody}

```

(End of definition for _kernel_list_label_end:.)

`_block_list_item_end:` When a list item ends we have to close LBody and LI but also a <text> in the special case that the item material ends in a list (identifiable via @endpe).

```

1436 \cs_set:Npn \_block\_list\_item\_end: {
1437   \legacy_if:nT { @endpe }
1438   {
1439     \int_gincr:N \g__tag\_para\_main\_end\_int
1440     \tagstructend % text-unit
1441     % \_block\_debug\_typeout:n{Structure-end~ P~ at~ item-end \on@line }
1442   }
1443   \tagstructend \tagstructend % end LBody, LI
1444 }

```

(End of definition for _block_list_item_end:.)

`_block_list_end:` Finally, at the list end we have to close the open LBody, LI, L, and possibly a <text> if the last item ends with a list.

```

1445 \cs_set:Npn \_block\_list\_end: {
1446   \legacy_if:nT { @endpe }
1447   {
1448     \int_gincr:N \g__tag\_para\_main\_end\_int
1449     \tagstructend % text-unit
1450     \_block\_debug\_typeout:n{Structure-end~ P~ at~ list-end \on@line }
1451   }
1452   \tagstructend\tagstructend % end LBody, LI
1453   \tagstructend % end L
1454 }

```

(End of definition for _block_list_end:.)

End of tagging related declarations.

```

1455 }

```

These command should have a dummy declaration if tagging is not active

```

1456 {
1457   \cs_new:Npn \_block\_start\_para\_structure\_unconditionally:n #1 {}
1458 }

```

```

1459 </package>

```

```

1460 <*latex-lab>

```

```

1461 \ProvidesFile{block-latex-lab-testphase.ltx}

```

```

1462     [\ltlabblockdate\space v\ltlabblockversion\space

```

```

1463         blockenv implementation]

```

```

1464 \RequirePackage{latex-lab-testphase-block}

```

```

1465 </latex-lab>

```

5 Documentation from first prototype implementations

5.1 Open questions

- Existing questions — moved to issues —

5.2 Code cleanup

- Actually implement what's announced.
- Encapsulate most uses of `\legacy_if...` into commands with `expl3` syntax: we cannot rename these booleans for compatibility reasons but we can make the code cleaner nevertheless. — made issue —
- The `\topsep` and `\partopsep` business is tricky to reproduce exactly (see `\@topsepadd` and `\@topsep`) because of how it accumulates when lists are nested immediately.

5.3 Tasks

- Change author to LaTeX Team once it's nice enough to deserve that label.
- Reproducing exactly the standard layouts and examples in the `enumitem` documentation.
- Hooks, but do not duplicate those that already exist as environment hooks. Hence, mostly around items.
- Customization and interaction with LDB:
 - Allow arbitrary nesting depth with automatically defined styles for labels, counters etc.
 - Adapt everything to font size! (e.g. footnotes).
 - How to model the inheritance from `trivlist` to `list` to `enumerate`?
- Add key-value settings mimicking `enumitem`'s ability to set any four of five horizontal parameters and deduce the fifth by `\leftmargin + \itemindent = \labelindent + \labelwidth + \labelsep`.
- Provide good ways to customize how overlong labels are dealt with.
- Use the `.aux` file.
 - Implement the `\ref` styles that `enumitem` provides.
 - Reverse enumerations, important in publication lists and the like. Somehow avoid needing 3 compilations for references to reverse enumerations to settle?
 - Ability to calculate `\labelwidth` from the label contents. Share calculated parameters between multiple environments (cf. `resume` option).
- Related to grabbing the whole list environment, and input syntax variations:

- Other layouts: `tabular` (see `listliketab` vs `typed-checklist`), `multicolumn` and horizontally numbered (see `tasks`), inline lists, `runin` lists in the easy case where there is no intervening `\par`.
 - Formatting the item text in a box or similar (requires grabbing the whole list).
 - Filtering which items to show: hide certain items according to criteria (useful together with list reuse), see `typed-checklist`.
 - Shorthands `\iitem` for automatic nested lists, or `\1`, `\2` etc from `outlines`.
 - Support markdown input like `asciilist`.
- Check interaction with `babel` options such as `french` or `accadian` (see `FrenchItemizeSpacing`)
 - RTL and vertical typesetting.

6 Plan of attack of first prototype

Typesetting list environments involves a rather large number of parameters. They can be affected by the context such as the total list nesting level, the nesting level of the given type of list, and the font size. An environment like `enumerate` has two main aspects.

- It has a certain layout in the page, with vertical and horizontal spacing around it. This type of layout is shared with environments such as `quote`, `flushright`, or `tabbing`. This common layout is implemented in $\text{\LaTeX} 2_{\epsilon}$ through `\trivlist` (or `\list`).
- It defines how each `\item` should be typeset: how to construct the label, in particular the `counter` name, and how to format the content of the item.

This suggests defining two object types, `block` and `item` covering these two aspects.¹ While the `item` type will perhaps have a single template, one could typeset a `block` object in several ways, for instance the standard $\text{\LaTeX} 2_{\epsilon}$ way or a fancy colored box.

The *general block* template should receive the following parameters. The *plain block* template is a restricted template that freezes all item-related parameters to dummy values (`counter`, `start`, `resume`, `label-width`, `label-sep` and all `item-*`). The *list block* template is a restricted template² that omits the `heading` parameter and whose default for `item-instance` is non-empty.

- Structural parameters: the `heading` to place before, `counter` name, `start` value, whether to `resume` a previous list, and the `item-instance` (an `item` instance) to use when typesetting items.
- Vertical spacing and penalties: `beginpenalty`, `beginsep`, `begin-par-skip`, `item-penalty`, `item-skip`, `item-par-skip`, `endpenalty`, `end-skip`, `end-par-skip`.
- Horizontal spacing: `rightmargin`, `leftmargin`, `parindent`, `item-indent`, `label-width`, `label-sep`.

A document class should edit these templates (or define restricted templates) to set

¹Possibly also `endblock` to deal with decorations at the end?

²A better approach could be to have a notion of inheritance for object types, so that we end up with two different *object types*. Then we can implement other template for the list object type: `table` for lists typeset as rows/columns of a table, `inline` for lists typeset in horizontal mode within a paragraph, and `runin` for run-in lists.

up default values that depend on `\g_block_nesting_depth_int`, namely how many lists are nested overall.³ The document class should then set up an instance of these templates for each environment, with appropriate settings such as a `heading`, a suitable `item-instance`, or making `margin-right` equal to `margin-left` in a `quote` environment.

The *inline-list block* template receives many fewer parameters. Note that `beginsep`, `item-skip`, `end-skip` are now *horizontal* skips.

- Structural parameters: `counter`, `start`, `resume`, `item-instance`.
- Spacing and penalties: `beginpenalty`, `beginsep`, `item-penalty`, `item-skip`, `endpenalty`, `end-skip`.
- Horizontal spacing: `label-width`, `label-sep`.

The *std item* template should receive the following parameters. They depend on the type of list and its nesting level among lists of such type, but typically not on the total nesting level.

- Counter name (`counter`), shared with the parent *list block* template, but needed for incrementing.
- Label construction: a function `counter-label` that produces the label from the counter name, used if `\item` is given without argument.
- References: a function `counter-ref` for how the label should be referred to when it is constructed from the counter, `label-ref` and `label-autoref` used when `\item` has an optional argument.
- Label formatting: `label-format` function, `label-strut` boolean.
- Label alignment (`label-align`, `label-boxed`, `next-line`).
- Content parameters: `text-font`.
- A `compatibility` boolean that controls for instance whether `\makeLabel` is used.

The document class should set up an instance such as *enumiii* for each environment and nesting level.⁴

A given environment will adjust some nesting levels, then call the *block* instance appropriate to the environment type, passing it the *item* instance appropriate to the environment and depth. Additional context-dependence could be provided by `l3ldb`, but the main context-dependence should not rely on it for simplicity reasons and incidentally because `l3ldb` is not yet available.

³Does `xtemplate` provide a way to specify default values that are only evaluated once an instance is used?

⁴This should be made easily extendible to deeper levels.

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.

Symbols

`\` 212, 491
† internal commands:
`\i_block_flattened_level_int` ... 22
`\l` 313, 337, 345, 1229

Numbers

`\1` 55
`\2` 55

A

`\addpenalty` 467, 583, 813
`\AddToHook` 140, 151, 159,
198, 209, 226, 251, 759, 788, 1255, 1302
`\addvspace` 468, 581, 584, 585, 814
`\arabic` 7, 94

B

`\begin` 50, 1345
`\begingroup` 1354
`\bfseries` 20, 311, 335, 1143
`block` (objecttype) 29
`block` commands:
`\block_debug_off:` 114, 119, 132
`\block_debug_on:` 114, 114, 131
`\g_block_nesting_depth_int`
..... 356, 400, 404, 405, 411, 414, 442
`block display` (template) 51, 498
`block displayblock-0` (instance) ... 1065
`block displayblock-1` (instance) ... 1065
`block displayblock-2` (instance) ... 1065
`block displayblock-3` (instance) ... 1065
`block displayblock-4` (instance) ... 1065
`block displayblock-5` (instance) ... 1065
`block displayblock-6` (instance) ... 1065
`block` internal commands:
`__block_beginpar_hmode:N`
..... 820, 842, 856, 1222, 1222
`__block_beginpar_vmode:`
..... 822, 844, 858, 1206, 1206
`\l_block_block_instance_tl`
..... 366, 411, 413
`\l_block_botsep_skip` 504, 659
`_block_counter_label:n` ... 667, 694
`_block_counter_ref:n` 668
`\l_block_counter_start_int`
..... 610, 632, 643
`\l_block_counter_tl` .. 608, 625, 639
`_block_debug:n` 112, 112, 126

`\g__block_debug_bool`
..... 111, 116, 121, 127, 129
`_block_debug_gset:` 114, 117, 122, 124
`_block_debug_typeout:n`
..... 112, 113, 128, 376,
410, 418, 423, 440, 457, 474, 596,
599, 602, 622, 651, 685, 697, 762,
1207, 1226, 1240, 1246, 1264, 1276,
1292, 1307, 1380, 1384, 1441, 1450
`\l_block_effective_top_skip` ...
..... 535, 537, 584, 786
`\l_block_env_name_tl` 360, 376
`\l_block_env_params_tl` 235
`\l_block_final_code_tl` . 23, 373, 434
`\l_block_flattened_level_int` ...
..... 380, 382, 387, 436, 1215
`\l_block_heading_tl` .. 500, 514, 515
`_block_inner_begin:`
..... 832, 845, 1376, 1376
`_block_inner_end:`
..... 833, 846, 1379, 1379
`\l_block_inner_instance_tl`
..... 371, 421, 423, 427
`\l_block_inner_instance_type_tl`
..... 370, 426
`\l_block_inner_level_counter_tl`
368, 391, 393, 396, 424, 425, 428, 429
`_block_inter_item:` 35, 797, 806, 806
`\l_block_item_align_tl`
..... 9, 678, 679, 680, 710, 713
`\l_block_item_compatibility_`
`bool` 676, 691
`_block_item_everypar:`
..... 33, 34, 735, 758, 758, 759, 781
`_block_item_everypar_std:`
..... 735, 758, 761
`_block_item_instance:n`
..... 35, 612, 792, 799, 800
`\l_block_item_label_tl` 609, 646, 648
`\l_block_item_parsep_skip` 732
`_block_label_autoref:n` 670
`\l_block_label_boxed_bool` . 673, 701
`_block_label_format:n`
..... 33, 671, 740, 745
`\l_block_label_given_tl`
..... 31, 664, 686, 688, 698
`_block_label_ref:n` 669
`\l_block_label_strut_bool` . 672, 747

<code>\g__block_labels_box</code>	<code>block list-1 (instance)</code>	1102
... 31 , 33 , 566 , 569 , 722 , 724 , 737 , 770	<code>block list-2 (instance)</code>	1102
<code>\l_block_legacy_env_params_tl</code> ..	<code>block list-3 (instance)</code>	1102
..... 18 , 10 , 230 , 243	<code>block list-4 (instance)</code>	1102
<code>\l_block_legacy_support_bool</code> ...	<code>block list-5 (instance)</code>	1102
..... 619 , 748	<code>block list-6 (instance)</code>	1102
<code>\l_block_level_incr_bool</code>	<code>block quotationblock-1 (instance)</code> .	1095
..... 364 , 398 , 441	<code>block quotationblock-2 (instance)</code> .	1095
<code>__block_list_begin:</code> . 859 , 1406 , 1406	<code>block quotationblock-3 (instance)</code> .	1095
<code>__block_list_end:</code> ... 860 , 1445 , 1445	<code>block quotationblock-4 (instance)</code> .	1095
<code>__block_list_item_begin:</code>	<code>block quotationblock-5 (instance)</code> .	1095
..... 861 , 1413 , 1413	<code>block quotationblock-6 (instance)</code> .	1095
<code>__block_list_item_end:</code>	<code>block quoteblock-1 (instance)</code>	1088
..... 862 , 1436 , 1436	<code>block quoteblock-2 (instance)</code>	1088
<code>\l_block_long_label_bool</code>	<code>block quoteblock-3 (instance)</code>	1088
..... 720 , 721 , 729 , 739	<code>block quoteblock-4 (instance)</code>	1088
<code>__block_make_label_box:n</code>	<code>block quoteblock-5 (instance)</code>	1088
..... 31 , 693 , 694 , 698 , 740 , 740	<code>block quoteblock-6 (instance)</code>	1088
<code>\l_block_max_inner_levels_tl</code> ...	<code>block verbatimblock-0 (instance)</code> ..	1076
..... 369 , 394	<code>block verbatimblock-1 (instance)</code> ..	1076
<code>\l_block_next_line_bool</code> ... 674 , 728	<code>block verbatimblock-2 (instance)</code> ..	1076
<code>\l_block_one_label_box</code>	<code>block verbatimblock-3 (instance)</code> ..	1076
..... 33 , 702 , 706 , 708 ,	<code>block verbatimblock-4 (instance)</code> ..	1076
711 , 712 , 716 , 717 , 719 , 726 , 737 , 742	<code>block verbatimblock-5 (instance)</code> ..	1076
<code>\l_block_para_instance_tl</code>	<code>block verbatimblock-6 (instance)</code> ..	1076
..... 367 , 416 , 418 , 419	<code>blockenv (objecttype)</code>	29
<code>\l_block_parbotsep_skip</code> ... 505 , 660	<code>blockenv center (instance)</code>	896
<code>__block_recipe_basic:</code>	<code>blockenv description (instance)</code> ...	1039
..... 818 , 818	<code>blockenv display (template)</code>	34 , 358
<code>__block_recipe_list:</code>	<code>blockenv displayblock (instance)</code> ...	871
..... 853 , 853	<code>blockenv displayblockflattened (in-</code>	
<code>__block_recipe_standalone:</code> 827 , 827	instance)	883
<code>__block_recipe_standard:</code> .. 839 , 839	<code>blockenv enumerate (instance)</code>	1026
<code>\l_block_resume_bool</code> . 611 , 629 , 640	<code>blockenv flushleft (instance)</code>	910
<code>\l_block_setup_code_tl</code> ... 365 , 409	<code>blockenv flushright (instance)</code>	924
<code>__block_skip_remove_last:</code>	<code>blockenv itemize (instance)</code>	1013
..... 106 , 109 , 454 , 524 , 809 , 810	<code>blockenv list (instance)</code>	1052
<code>__block_skip_set_to_last:N</code>	<code>blockenv quotation (instance)</code>	938
..... 106 , 106 , 461 , 574	<code>blockenv quote (instance)</code>	950
<code>__block_start_para_structure:n</code> .	<code>blockenv verbatim (instance)</code>	1000
..... 1259 , 1263 , 1263	<code>blockpar internal commands:</code>	
<code>__block_start_para_structure_-</code>	<code>__blockpar</code>	1343
<code>unconditionally:n</code>	<code>bool commands:</code>	
..... 323 , 350 , 1285 , 1373 , 1457	<code>\bool_gset_false:N</code>	121
<code>\l_block_tag_class_tl</code> . 362 , 867 , 869	<code>\bool_gset_true:N</code>	116
<code>\l_block_tag_inner_tag_tl</code>	<code>\bool_if:NTF</code>	
..... 836 , 837 , 849 , 850 , 852 , 1377 127 , 129 , 193 , 385 , 398 , 441 ,	
<code>\l_block_tag_name_tl</code>	629 , 640 , 691 , 728 , 729 , 747 , 748 ,	
..... 361 , 835 , 837 , 848 , 850 , 864 , 866	1227 , 1236 , 1241 , 1244 , 1256 , 1269 ,	
<code>\l_block_tagging_recipe_tl</code> 363 , 408	1286 , 1304 , 1311 , 1359 , 1361 , 1371	
<code>\l_block_text_font_tl</code>	<code>\bool_if:nTF</code>	699
..... 675	<code>\bool_new:N</code>	111 , 739
<code>\l_block_thm_current_counter_tl</code>	<code>\bool_set_false:N</code>	721
..... 298 , 301 , 308 , 332	<code>\bool_set_true:N</code>	720
<code>\l_block_tmpa_skip</code> 574 , 575 , 576 , 785		
<code>\l_block_topsepadd_skip</code>		
..... 25 , 468 , 517 , 520 , 535 , 786		

box commands:		<code>\DeclareTemplateInterface</code>	
<code>\box_if_empty:N</code>	NTF	34, 51, 66, 78, 92
<code>\box_new:N</code>		11, 12,
<code>\box_use_drop:N</code>		27, 28, 174, 177, 195, 299, 303, 304,
<code>\box_wd:N</code>		328, 355, 1318, 1348, 1349, 1405, 1435
<code>\break</code>		description (env.) 198
			<code>\detokenize</code> 474, 596, 599, 602
			dim commands:
		<code>\dim_add:Nn</code> 559, 560
		<code>\dim_compare:nNnTF</code> 462, 705, 719
		<code>\dim_compare_p:n</code> 702
		<code>\dim_set_eq:NN</code> 558, 734
		<code>\dim_zero:N</code> 237, 238,
			239, 255, 256, 656, 657, 658, 659, 660
		<code>\c_zero_dim</code> 462, 1328
		<code>displayblock</code> (env.) 134
		<code>displayblockflattened</code> (env.) 137
		<code>\do</code> 184
		<code>\dospecials</code> 184
			E
		<code>\edef</code> 1350
		<code>\else</code> 180
		<code>\end</code> 11, 455
		<code>\endblockenv</code> 15, 136, 139,
			143, 146, 149, 154, 157, 165, 171,
			201, 204, 207, 224, 233, 260, 355, 439
		<code>\endcsname</code> 264, 280, 1352
		<code>\endgraf</code> 1344
		<code>enumerate</code> (env.) 198
		environments:	
		<code>center</code> 140
		<code>description</code> 198
		<code>displayblock</code> 134
		<code>displayblockflattened</code> 137
		<code>enumerate</code> 198
		<code>flushleft</code> 140
		<code>flushright</code> 140
		<code>itemize</code> 198
		<code>list</code> 226
		<code>quotation</code> 151
		<code>quote</code> 151
		<code>trivlist</code> 251
		<code>verbatim</code> 159
		<code>verbatim*</code> 159
		<code>verse</code> 209
		<code>\everypar</code> 18, 21, 22, 186
		exp commands:	
		<code>\exp_after:wN</code> 710, 713
		<code>\expandafter</code> 186, 264, 280
		<code>\ExplSyntaxOn</code> 8
			F
		<code>\fi</code> 182, 183, 1368
			C
<code>center</code> (env.)		140
<code>\centering</code>		1199
<code>\clubpenalty</code>		15, 777, 780
color commands:			
<code>\color_select:n</code>		1229
cs commands:			
<code>\cs_generate_variant:Nn</code>		110
<code>\cs_gset_protected:Npx</code>		126, 128
<code>\cs_new:Npn</code>		236,
			356, 439, 473, 595, 598, 601, 818,
			827, 839, 853, 1357, 1376, 1379, 1457
<code>\cs_new_eq:NN</code>		26, 109,
			112, 113, 756, 757, 758, 784, 816, 817
<code>\cs_new_protected:Npn</code>		106, 114, 119,
			124, 131, 132, 740, 761, 806, 1263, 1285
<code>\cs_set:Npn</code>		
		 29, 1206, 1222, 1235, 1370,
			1406, 1413, 1414, 1428, 1436, 1445
<code>\cs_set_eq:NN</code>		257,
			735, 781, 819, 821, 828, 830, 832,
			833, 841, 843, 845, 846, 855, 857,
			859, 860, 861, 862, 1342, 1343, 1344
<code>\cs_set_protected:Npn</code>		1319
<code>\csname</code>		264, 280, 1352
			D
<code>\DebugBlocksOff</code>		131
<code>\DebugBlocksOn</code>		131
<code>\DeclareHookRule</code>		760
<code>\DeclareInstance</code>		..	871, 883, 896, 910,
			924, 938, 950, 962, 975, 987, 1000,
			1013, 1026, 1040, 1052, 1065, 1076,
			1088, 1095, 1102, 1115, 1116, 1117,
			1118, 1119, 1120, 1121, 1122, 1123,
			1124, 1126, 1128, 1130, 1132, 1136,
			1137, 1141, 1155, 1166, 1177, 1188
<code>\DeclareInstanceCopy</code>		
		 1070, 1071, 1072, 1073,
			1074, 1075, 1082, 1083, 1084, 1085,
			1086, 1087, 1090, 1091, 1092, 1093,
			1094, 1097, 1098, 1099, 1100, 1101
<code>\DeclareObjectType</code>		29, 30, 31, 32, 33
<code>\DeclareRobustCommand</code>		
		 1199, 1200, 1201, 1202, 1345
<code>\DeclareTemplateCode</code>		358, 482, 498, 606, 665

<code>\int_use:N</code> . . .	411, 413, 425, 429, 1229	<code>\labelindent</code>	54
<code>\int_zero:N</code>	541	<code>\labelitemi</code>	1120
<code>\c_zero_int</code>	772	<code>\labelitemii</code>	1121
<code>\interlinepenalty</code>	179, 182	<code>\labelitemiii</code>	1122
<code>item</code> (objecttype)	29	<code>\labelitemiv</code>	1123
<code>\item</code>	15, 24, 27, 28, 30, 31, 33, 38, 55, 222, 788, 808, 1417	<code>\labelsep</code> 7, 54, 89, 325, 352, 618, 725, 727	
<code>item basic</code> (instance)	1137	<code>\labelwidth</code>	7, 54, 88, 256, 617, 706, 708, 719, 725
<code>item description</code> (instance)	1137	<code>\language</code>	175
<code>item std</code> (template)	92, 663	<code>\lastbox</code>	21
<code>\itemindent</code>	17, 54, 87, 239, 616, 725, 768	<code>\LBody</code>	1433, 1435
<code>itemize</code> (env.)	198	<code>\leavevmode</code>	179
<code>\itemsep</code>	7, 85, 613, 656, 814	<code>\leftmargin</code>	6, 54, 61, 218, 219, 255, 509, 559, 560, 568, 570, 970, 983, 1111
<code>\itshape</code>	324, 351	<code>\leftskip</code>	16, 486, 538
J			
<code>\justifying</code>	1202, 1204	legacy commands:	
K			
<code>\kern</code>	768	<code>\legacy_if:nTF</code> 244, 443, 448, 458, 459, 516, 527, 533, 544, 563, 572, 580, 765, 774, 795, 807, 1208, 1210, 1238, 1257, 1265, 1267, 1381, 1437, 1446
kernel internal commands:		<code>\legacy_if_gset_false:n</code> 446, 451, 480, 764, 767, 776, 1212
<code>__kernel_displayblock_begin:</code>	36, 562, 595, 595, 823, 832, 845, 859	<code>\legacy_if_gset_true:n</code> 469, 478, 650, 801
<code>__kernel_displayblock_beginpar_hmode:w</code>	525, 595, 598, 819, 828, 841, 855	<code>\legacy_if_set_false:n</code> 241, 534, 565, 763
<code>__kernel_displayblock_beginpar_vmode:</code>	521, 595, 601, 821, 830, 843, 857	<code>\legacy_if_set_true:n</code>	529, 530
<code>__kernel_displayblock_doendpe:</code>	47, 16, 26, 1235, 1235	<code>\legacylistsetupcode</code>	18, 236, 1059
<code>__kernel_displayblock_end:</code>	36, 456, 473, 473, 824, 833, 846, 860	<code>\legacyverbatimsetup</code>	173, 1011
<code>__kernel_endpe_vmode:</code>	1338, 1355, 1357, 1357	<code>\let</code>	27, 28, 184, 212, 242, 823, 824
<code>__kernel_list_item_begin:</code>	796, 812, 816, 816, 861	<code>\linewidth</code>	559, 561, 702
<code>__kernel_list_item_end:</code>	811, 816, 817, 862	<code>list</code> (env.)	226
<code>__kernel_list_label_after:</code>	771, 784, 784, 1370, 1370	<code>list</code> (objecttype)	29
<code>__kernel_list_label_begin:</code>	744, 756, 756, 1414, 1414	<code>\list</code>	55, 253
<code>__kernel_list_label_end:</code>	753, 756, 757, 1428, 1428	<code>list description</code> (instance)	1136
keys commands:		<code>list enumerate-1</code> (instance)	1124
<code>\keys_define:nm</code>	31, 589, 653, 663	<code>list enumerate-2</code> (instance)	1124
<code>\KeyValue</code> 57, 58, 95, 1089, 1096, 1107, 1108		<code>list enumerate-3</code> (instance)	1124
L			
<code>\labelenumi</code>	1125	<code>list enumerate-4</code> (instance)	1124
<code>\labelenumii</code>	1127	<code>list itemize-1</code> (instance)	1120
<code>\labelenumiii</code>	1129	<code>list itemize-2</code> (instance)	1120
<code>\labelenumiv</code>	1131	<code>list itemize-3</code> (instance)	1120
		<code>list itemize-4</code> (instance)	1120
		<code>list legacy</code> (instance)	1132
		<code>list std</code> (template)	78, 606
		<code>\list<romannumeral></code>	42, 44
		<code>\listparindent</code> 6, 63, 237, 510, 558, 593, 734, 1113
		<code>\LItag</code>	1405, 1413
		<code>\ltablblockdate</code>	4, 1462
		<code>\ltablblockversion</code>	4, 1462

M	
<code>\makelabel</code>	7 , 18 , 45 , 242 , 257 , 749
<code>\MakeLinkTarget</code>	308 , 332 , 693 , 694
mode commands:	
<code>\mode_if_horizontal:TF</code>	453 , 809 , 1321 , 1326
<code>\mode_if_inner:TF</code>	1322
<code>\mode_if_vertical:TF</code>	518
<code>\mode_leave_vertical:</code>	307 , 331 , 445 , 515 , 516
msg commands:	
<code>\msg_error:nnn</code>	793
<code>\msg_error:nnnn</code>	1334
N	
<code>\newcommand</code>	191
<code>\newcounter</code>	437
<code>\NewDocumentEnvironment</code>	134 , 137
<code>\newline</code>	730
<code>\newtheorem</code>	19 , 262
<code>\nobreak</code>	729
<code>\nobreakspace</code>	195
<code>\noexpand</code>	280
<code>\normalfont</code>	1143
NOT commands:	
<code>\NOT_IMPLEMENTED</code>	681
<code>\null</code>	179
O	
<code>\obeylines</code>	185
object types:	
block	29
blockenv	29
item	29
list	29
para	29
off (plug)	25 , 25 , 477
on (plug)	25 , 25 , 477
P	
<code>\par</code>	10–12 , 25 , 27 , 30 , 35–38 , 47 , 50 , 12 , 19 , 177 , 454 , 525 , 808 , 810 , 1233 , 1342
par commands:	
<code>\par_end:</code>	12
par internal commands:	
<code>\l_par_fixed_word_spaces_bool</code>	489
<code>\l_par_start_skip</code>	485
para (objecttype)	29
para center (instance)	1155
para commands:	
<code>\para_end:</code>	27 , 50 , 549 , 553 , 1319 , 1342 , 1343 , 1344
<code>\g_para_indent_box</code>	768
<code>\para_omit_indent:</code>	769
para std (template)	66 , 482
para/begin (hook)	33 , 34
para/begin	1254
<code>\PARALABEL</code>	323 , 350 , 1259 , 1318
<code>\parfillskip</code>	488 , 540
<code>\parindent</code>	6 , 68 , 484 , 558 , 734
<code>\parsep</code>	6 , 56 , 503 , 557 , 614 , 657 , 732 , 971 , 1106
<code>\parskip</code>	28 , 465 , 537 , 556 , 557 , 576 , 581 , 585
<code>\partopsep</code>	6 , 55 , 502 , 520 , 592 , 1105
<code>\pdfspaces</code>	17 , 195
<code>\penalty</code>	179 , 182 , 772
Plugs:	
off	25 , 25 , 477
on	25 , 25 , 477
prg commands:	
<code>\prg_do_nothing:</code>	26 , 756 , 757 , 758 , 781 , 784 , 816 , 817 , 823 , 824 , 829 , 831
<code>\ProvidesFile</code>	1461
<code>\ProvidesPackage</code>	3
Q	
quotation (env.)	151
quote (env.)	151
R	
<code>\raggedleft</code>	1200
<code>\raggedright</code>	1201
<code>\refstepcounter</code>	20
<code>\relax</code>	222 , 678 , 680
<code>\RemoveFromHook</code>	1254 , 1301
<code>\renewcommand</code>	29 , 195
<code>\RenewDocumentCommand</code>	262 , 789
<code>\RenewDocumentEnvironment</code>	141 , 144 , 147 , 152 , 155 , 160 , 166 , 199 , 202 , 205 , 210 , 227 , 252
<code>\RequirePackage</code>	6 , 7 , 1464
<code>\rightmargin</code>	6 , 62 , 238 , 508 , 559 , 970 , 983 , 1112
<code>\rightskip</code>	487 , 496 , 539
<code>\rlap</code>	1229
S	
scan commands:	
<code>\scan_stop:</code>	1320
<code>\setbox</code>	21
<code>\setcounter</code>	438 , 1064
<code>\setlength</code>	970 , 971 , 983
<code>\SetTemplateKeys</code>	378 , 495 , 513 , 624 , 687
skip commands:	
<code>\skip_add:Nn</code>	520 , 537
<code>\skip_eval:n</code>	581
<code>\skip_horizontal:n</code>	568 , 570 , 725 , 727
<code>\skip_new:N</code>	785 , 786 , 787

		V	
200, 203, 206, 213, 231, 305, 329,			
412, 419, 426, 1199, 1200, 1201, 1202	verbatim (env.)	<u>159</u>
	verbatim* (env.)	<u>159</u>
\UserName	verse (env.)	<u>209</u>
..... 59, 60, 86, 291, 1109, 1110			