

Prototype reimplementation of $\text{\LaTeX} 2_{\epsilon}$'s block environments using templates

\LaTeX Project*

v0.8a 2023/03/08

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	13
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	Theorem-like environments	18
4.5	Implementation of templates	19
4.5.1	Implementation of blockenv templates ...	19
4.5.2	Implementation of para templates ...	23
4.5.3	Implementation of block templates ...	23
4.5.4	Implementation of list templates ...	26
4.5.5	Implementation of <code>\item</code> template(s)	28
4.6	Tagging recipes	33
4.7	Blockenv instances	35
4.7.1	Basic instances	35
4.7.2	Blockquote instances	36
4.7.3	Verbatim instances	37
4.7.4	Standard list instances	38
4.8	Block instances	39
4.8.1	Displayblock instances	39
4.8.2	Quote/quotationblock instances	39
4.8.3	Block instances for the standard lists	40
4.9	List instances for the standard lists	40
4.10	Item instances	41
4.11	Para instances	41
4.12	Tagging support	43
4.12.1	List tags	48
5	Documentation from first prototype implementations	50
5.1	Open questions	50
5.2	Code cleanup	50
5.3	Tasks	50
6	Plan of attack of first prototype	51
	Index	53

1 Introduction

The list implementation in L^AT_EX 2_ε 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_EX 2_ε name).

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

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

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

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

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

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

The maximum number of *blockenvs* that can be nested into each other is restricted by the L^AT_EX 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 <code>item</code> 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: 0pt
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

label-strut (<i>boolean</i>)	Add a <code>\strut</code> to the label?	Default: <code>false</code>
label-align (<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>
label-boxed (<i>boolean</i>)	Should the label be boxed?	Default: <code>true</code>
next-line (<i>boolean</i>)		Default: <code>false</code>
text-font (<i>tokenlist</i>)	<i>unused</i>	
compatibility (<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:

basic This recipe does the following:

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

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

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

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

- the inner structure is a list (`<L>`).
- Furthermore everything is set up so that we have list items (``) 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-tagging}
4                 [\ltblocksdate\space \ltblocksversion\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 L^AT_EX 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 reimplementing 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 L^AT_EX 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 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
`block` (*objecttype*) specific to a given use in the document. This section is devoted to template interfaces,
`para` (*objecttype*) and the template code is covered later.

```
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 }
```

`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)
```

verify that this claim is actually correct!

```

65 }

para std (templ.)

66 \DeclareTemplateInterface{para}{std}{1}
67 {
68   indent-width      : length = \parindent ,
69   start-skip        : skip = 0pt ,
70   left-skip         : skip = 0pt ,
71   right-skip        : skip = 0pt ,
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 = 0pt ,          % 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
`_block_skip_remove_last:` none

```

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 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 definition for `\tl_if_novalue:nTF`. This function is documented on page ??.)

`\tag_if_active:T` If tagging support is not loaded then we shouldn't try to execute any tagging related
 can vanish one day commands. Eventually this can go once the basic support is available in the kernel.

```

111 \cs_if_exist:NF \tag_if_active:T
112   { \cs_new_eq:NN \tag_if_active:T \use_none:n }

```

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

4.3.1 Debugging

`\g_block_debug_bool`

```

113 \bool_new:N \g_block_debug_bool

```

(End definition for `\g_block_debug_bool`.)

`_block_debug:n`

`_block_debug_typeout:n`

```

114 \cs_new_eq:NN \_block_debug:n \use_none:n
115 \cs_new_eq:NN \_block_debug_typeout:n \use_none:n

```

(End definition for `_block_debug:n` and `_block_debug_typeout:n`.)

`\block_debug_on:`

`\block_debug_off:`

`_block_debug_gset:`

```

116 \cs_new_protected:Npn \block_debug_on:
117   {
118     \bool_gset_true:N \g_block_debug_bool
119     \_block_debug_gset:
120   }
121 \cs_new_protected:Npn \block_debug_off:
122   {
123     \bool_gset_false:N \g_block_debug_bool
124     \_block_debug_gset:
125   }
126 \cs_new_protected:Npn \_block_debug_gset:
127   {
128     \cs_gset_protected:Npx \_block_debug:n ##1
129     { \bool_if:NT \g_block_debug_bool {##1} }
130     \cs_gset_protected:Npx \_block_debug_typeout:n ##1
131     { \bool_if:NT \g_block_debug_bool { \typeout{==>~ ##1} } }
132   }

```

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

```
\DebugBlocksOn
\DebugBlocksOff 133 \cs_new_protected:Npn \DebugBlocksOn { \block_debug_on: }
134 \cs_new_protected:Npn \DebugBlocksOff { \block_debug_off: }
135 \DebugBlocksOff
```

(End 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 $\text{\LaTeX 2}_{\epsilon}$'s `trivlist` except that there aren't degenerated lists and thus have no hidden `\item` inside.

```
displayblock (env.)
136 \NewDocumentEnvironment{displayblock}{!0{}} {
137   { \UseInstance{blockenv}{displayblock} {#1} }
138   { \endblockenv }

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

center (env.)
flushleft (env.)
flushright (env.) 142 \AddToHook{begindocument/before}{
143   \RenewDocumentEnvironment{center}{!0{}} {
144     { \UseInstance{blockenv}{center}{#1} }
145     { \endblockenv }

146   \RenewDocumentEnvironment{flushright}{!0{}} {
147     { \UseInstance{blockenv}{flushright}{#1} }
148     { \endblockenv }

149   \RenewDocumentEnvironment{flushleft}{!0{}} {
150     { \UseInstance{blockenv}{flushleft}{#1} }
151     { \endblockenv }
152 }
```

4.4.2 Display quote environments

```

quote (env.)
quotation (env.) 153 \AddToHook{begindocument/before}{
154 \RenewDocumentEnvironment{quote}{!O{}}
155 { \UseInstance{blockenv}{quote} {#1} }
156 { \endblockenv }
157 \RenewDocumentEnvironment{quotation}{!O{}}
158 { \UseInstance{blockenv}{quotation} {#1} }
159 { \endblockenv }
160 }

```

4.4.3 Verbatim environments

```

verbatim (env.)
verbatim* (env.) 161 \AddToHook{begindocument/before}{
162 \RenewDocumentEnvironment{verbatim}{!O{}}
163 { \UseInstance{blockenv}{verbatim} {#1}
164 \xverbatim
165 }
166 { \endblockenv }
167 \RenewDocumentEnvironment{verbatim*}{!O{}}
168 { \UseInstance{blockenv}{verbatim} {#1}
169 \@setupverbvisiblespace\frenchspacing\@vobeyspaces
170 \@sxverbatim
171 }
172 { \endblockenv }
173 }

```

Helper commands for verbatim

```

\legacyverbatimsetup
174 <@@=
175 \def\legacyverbatimsetup{%
176 \language\l@nohyphenation
177 \@tempwafalse
178 \def\par{%
179 \if@tempswa
180 \leavevmode \null \@@par\penalty\interlinepenalty
181 \else
182 \@tempwattrue
183 \ifhmode\@@par\penalty\interlinepenalty\fi
184 \fi}%
185 \let\do\@makeoother \dospecials
186 \obeylines \verbatim@font \@noligs
187 \everypar \expandafter{\the\everypar \unpenalty}%
188 \tl_set:Nn \l__tag_para_main_tag_tl {codeline}
189 \tagtool{paratag=Code}% oder faster: \tl_set:Nn \l__tag_para_tag_tl {Code}
190 \@setupverbinvisiblespace
191 \frenchspacing \@vobeyspaces
192 }
193 <@@=block>

```


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

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

```

194 \newcommand\@setupverbinvisiblespace{
195 \tag_if_active:T {
196   \bool_if:NF\g__tag_mode_lua_bool
197   {
198     \renewcommand\@setupverbinvisiblespace{\def\xobeysp{\nobreakspace\pdffakespace}}
199   }
200 }
```

(End 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.) 201 \AddToHook{begindocument/before}{
description (env.) 202 \RenewDocumentEnvironment{itemize}{!0{}}
203   { \UseInstance{blockenv}{itemize} {#1} }
204   { \endblockenv }
205 \RenewDocumentEnvironment{enumerate}{!0{}}
206   { \UseInstance{blockenv}{enumerate} {#1} }
207   { \endblockenv }
208 \RenewDocumentEnvironment{description}{!0{}}
209   { \UseInstance{blockenv}{description} {#1} }
210   { \endblockenv }
211 }
```

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

```

212 \AddToHook{begindocument/before}{
213 \RenewDocumentEnvironment{list}{0{ } m m }
214   {
```

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.

```

215   \tl_set:Nn \@itemlabel {#2}
216   \tl_set:Nn \l__block_legacy_env_params_tl {#3}
217   \UseInstance{blockenv}{list} {#1}
218   }
219   { \endblockenv }
220 }
```

Again something that should probably elsewhere: the rolemapping.

```

221 \tag_if_active:T {
222 \tagpdfsetup{add-new-tag={tag=list,role=L}}
223 }
```

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

```

224 \tl_new:N \l__block_env_params_tl
```

(End 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`.

```
225 \cs_new:Npn \legacylistsetupcode {
```

Reset values to defaults:

```
226 \dim_zero:N \listparindent
227 \dim_zero:N \rightmargin
228 \dim_zero:N \itemindent
```

By default a list environment is not numbered:

```
229 \tl_set:Nn \@listctr {}
230 \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.

```
231 \let\makelabel\@mklab % TODO: customize
```

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

```
232 \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.

```
233 \legacy_if:nTF { @nbrlist }
234 { \tl_set:Nn \l_tag_L_attr_class_tl {enumerate} } % numbered list
235 { \tl_if_empty:NTF \@itemlabel
236   { \tl_set:Nn \l_tag_L_attr_class_tl {list} } % no label
237   { \tl_set:Nn \l_tag_L_attr_class_tl {itemize} } % unnumbered, unordered
238 }
239 }
```

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

`trivlist (env.)`

```
240 \AddToHook{begindocument/before}{
241   \RenewDocumentEnvironment{trivlist}{!O{}} {
242     { \list[#1]{}
243       {
244         \dim_zero:N \leftmargin
245         \dim_zero:N \labelwidth
246         \cs_set_eq:NN \makelabel \use:n
247       }
248     }
249   { \endblockenv }
250 }
```

4.4.5 Theorem-like environments

Theorem-like environments are defined in L^AT_EX with the help of `\newtheorem` declarations. Internally they use a list with a single item. For now we keep this approach and only add appropriate tagging support within the internal commands.

```

\@begintheorem We use <theorem-like> as the structure name and rollmap it to a list (<L>).
\@opargbegintheorem
251 \tag_if_active:T {
252 \tagpdfsetup{add-new-tag={tag=theorem-like,role=L}}
253 }

```

With that done, we only have to tell the trivlist what tag-name it should use.

```

254 \def\@begintheorem#1#2{\trivlist[tag-name=theorem-like]%
255 \item[\hskip \labelsep{\bfseries #1\ #2}]\itshape}

256 \def\@opargbegintheorem#1#2#3{\trivlist[tag-name=theorem-like]
257 \item[\hskip \labelsep{\bfseries #1\ #2\ (#3)}]\itshape}

```

(End 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:

```

258 \cs_new:Npn \g_block_nesting_depth_int { \@listdepth } % a fake int
259 % for now

```

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

blockenv display (templ.)

```

260 \DeclareTemplateCode{blockenv}{display}{1}
261 {
262   env-name      = \l__block_env_name_tl ,
263   tag-name      = \l__block_tag_name_tl ,
264   tag-class     = \l__block_tag_class_tl ,
265   tagging-recipe = \l__block_tagging_recipe_tl ,
266   level-increase = \l__block_level_incr_bool ,
267   setup-code    = \l__block_setup_code_tl ,
268   block-instance = \l__block_block_instance_tl ,
269   para-instance  = \l__block_para_instance_tl ,
270   inner-level-counter = \l__block_inner_level_counter_tl ,
271   max-inner-levels = \l__block_max_inner_levels_tl ,
272   inner-instance-type = \l__block_inner_instance_type_tl ,
273   inner-instance = \l__block_inner_instance_tl ,
274   para-flattened = \l__tag_para_flattened_bool ,
275   final-code    = \l__block_final_code_tl ,
276 }
277 {
278   \__block_debug_typeout:n{\l__block_env_name_tl -env-start}
279 %
280 \tl_if_empty:nF {#1} { \SetTemplateKeys{blockenv}{display}{#1} }
281 %

```

We need to know later if we haved 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`.

```

282 \int_compare:nNnTF \l__block_flattened_level_int > 0
283 {
284   \int_incr:N \l__block_flattened_level_int
285 }
286 {
287   \bool_if:NT \l__tag_para_flattened_bool
288   {
289     \int_incr:N \l__block_flattened_level_int
290   }
291 }
292 %
293 \tl_if_empty:NF \l__block_inner_level_counter_tl
294 {
295   \int_compare:nNnTF \l__block_inner_level_counter_tl >
296   { \l__block_max_inner_levels_tl - 1 }
297   { \@toodeep }
298   { \int_incr:N \l__block_inner_level_counter_tl } % not clean "o"?
299 }

```

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.

```

300 \bool_if:NT \l__block_level_incr_bool
301 {
302   \int_compare:nNnTF \g_block_nesting_depth_int >
303   { \c@maxblocklevels - 1 }
304   { \@toodeep }
305   {
306     \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.

```

307   \use:c { @list \int_to_roman:n { \g_block_nesting_depth_int } }
308 }
309 }

```

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

```

310 \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.

```

311 \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).

```

312 \__block_debug_typeout:n{use~ instance:~
313   \l__block_block_instance_tl - \int_use:N \g_block_nesting_depth_int }
314 \UseInstance{block}
315 { \l__block_block_instance_tl - \int_use:N

```

```

316         \g_block_nesting_depth_int }
317     {#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).

```

318     \tl_if_empty:NF \l__block_para_instance_tl
319     {
320         \__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.

```

321         \UseInstance{para}{ \l__block_para_instance_tl } {}
322     }

```

In 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.

```

323     \tl_if_empty:NF \l__block_inner_instance_tl
324     {
325         \__block_debug_typeout:n{use~ instance:~ \l__block_inner_instance_tl
326             \tl_if_empty:NF \l__block_inner_level_counter_tl
327             { - \int_use:N \l__block_inner_level_counter_tl }}
328         \UseInstance{ \l__block_inner_instance_type_tl }
329         { \l__block_inner_instance_tl
330             \tl_if_empty:NF \l__block_inner_level_counter_tl
331             { - \int_use:N \l__block_inner_level_counter_tl } } % not clean
332                                     % use "o"?
333         }
334         {#1}
335     }

```

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.

```

336     \l__block_final_code_tl
337 }

```

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

```

338 \int_new:N \l__block_flattened_level_int

```

(End 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.

```

339 \newcounter{maxblocklevels}
340 \setcounter{maxblocklevels}{6}

```

(End 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.

```

341 \cs_new:Npn \endblockenv {
342     \__block_debug_typeout:n{blockenv~ common~ ending \on@line}

```

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

```

343     \bool_if:NT \l__block_level_incr_bool
344     { \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.

```

345 \legacy_if:nT { @inlabel }
346 {
347   \mode_leave_vertical:
348   \legacy_if_gset_false:n { @inlabel }
349 }

```

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)

```

350 \legacy_if:nT { @newlist }
351 {
352   \@noitemerr
353   \legacy_if_gset_false:n { @newlist }
354 }
355 \mode_if_horizontal:TF
356 { \__block_skip_remove_last: \__block_skip_remove_last: \par }
357 { \@inmatherr{\end{\@currenvir}} }

```

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

```

358 \__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.

```

359 % \__block_debug_typeout:n{@nparlist =
360 % \legacy_if:nTF { @nparlist }{true}{false}}
361 \legacy_if:nF { @nparlist }
362 {
363   \__block_skip_set_to_last:N \l_tmpa_skip
364   \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
365   {
366     \skip_vertical:n { - \l_tmpa_skip }
367     \skip_vertical:n { \l_tmpa_skip + \parskip - \@outerparskip }
368   }
369   \addpenalty \@endparpenalty
370   \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 expository text that is part of the same paragraph and doesn’t start a new one.

```

371 % \legacy_if_gset_true:n { @endpe }
372 }

```

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.

```

373 \legacy_if_gset_true:n { @endpe }
374 }

```

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

some redesign/extensions here?

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

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

```

375 \cs_new:Npn \_kernel\_displayblock\_end: {
376   \_block\_debug\_typeout:n{\detokenize{\_kernel\_displayblock\_end:}}
377 }

```

(End definition for `_kernel_displayblock_end:`.)

4.5.2 Implementation of para templates ...

`para std (templ.)`

```

378 \DeclareTemplateCode{para}{std}{1}
379 {
380   indent-width      = \parindent ,
381   start-skip        = \l__par_start_skip ,           % name??
382   left-skip         = \leftskip ,
383   right-skip        = \rightskip ,
384   end-skip          = \parfillskip ,
385   fixed-word-spaces = \l__par_fixed_word_spaces_bool , % name??
386   final-hyphen-demerits = \finalhyphendemerits ,
387   cr-cmd            = \\ ,
388   para-class        = \l_tag_para_attr_class_tl ,
389 }
390 {
391   \tl_if_empty:nF {#1} { \SetTemplateKeys{para}{std}{#1} }
392   \skip_set:Nn \@rightskip \rightskip
393 }

```

4.5.3 Implementation of block templates ...

`block display (templ.)`

```

394 \DeclareTemplateCode{block}{display}{1}
395 {
396   heading          = \l__block_heading_tl ,
397   beginsep         = \topsep ,
398   begin-par-skip   = \partopsep ,
399   par-skip         = \parsep ,
400   end-skip         = \l__block_botsep_skip ,
401   end-par-skip     = \l__block_parbotsep_skip ,
402   beginpenalty     = \@beginparpenalty ,
403   endpenalty       = \@endparpenalty ,
404   rightmargin      = \rightmargin ,
405   leftmargin       = \leftmargin ,
406   parindent        = \listparindent ,
407 }
408 {
409   \tl_if_empty:nF {#1} { \SetTemplateKeys{block}{display}{#1} }
410   \tl_if_blank:oF \l__block_heading_tl
411   { \mode_leave_vertical: \textbf{\l__block_heading_tl} } % TODO customize

```

generalize heading usage
(or drop?)

The code largely follows the logic of L^AT_EX 2_ε's `trivlist` implementation as far as it is 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.

```

412 \legacy_if:nT { @noskipsec } { \mode_leave_vertical: }
413 \skip_set:Nn \l__block_topsepadd_skip { \topsep }
414 \mode_if_vertical:TF
415 {
416   \skip_add:Nn \l__block_topsepadd_skip { \partopsep }

```

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.

```

417   \__kernel_displayblock_beginpar_vmode:
418 }
419 {

```

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`.

```

420   \__block_skip_remove_last: \__block_skip_remove_last:
421   \__kernel_displayblock_beginpar_hmode:w \par
422 }

```

Now we are back to legacy list implementation ...

```

423 \legacy_if:nTF { @inlabel }
424 {
425   \legacy_if_set_true:n { @noparitem }
426   \legacy_if_set_true:n { @noparlist }
427 }
428 {
429   \legacy_if:nT { @newlist } { \@noitemerr }
430   \legacy_if_set_false:n { @noparlist }
431   \skip_set_eq:NN \l__block_effective_top_skip \l__block_topsepadd_skip
432 }
433 \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`.

```

434 \skip_zero:N \leftskip
435 \skip_set_eq:NN \rightskip \@rightskip
436 \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.

```

437 \int_zero:N \par@deathcycles
438 \@setpar
439 {
440   \legacy_if:nTF { @newlist }
441   {
442     \int_incr:N \par@deathcycles
443     \int_compare:nNnTF \par@deathcycles > { 1000 }
444     { \@noitemerr
445       { \para_end: }
446     }

```



```

447     }
448     {
449         { \para_end: }
450     }
451 }
452 \skip_set_eq:NN \@outerparskip \parskip
453 \skip_set_eq:NN \parskip \parsep
454 \dim_set_eq:NN \parindent \listparindent
455 \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
456 \dim_add:Nn \@totalleftmargin { \leftmargin }
457 \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.

```

458 \__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 is suppressed. This is controlled through the legacy switches @noparitem, minipage, and @nobreak.

```

459 \legacy_if:nTF { @noparitem }
460 {
461     \legacy_if_set_false:n { @noparitem }
462     \hbox_gset:Nn \g__block_labels_box
463     {
464         \skip_horizontal:n { - \leftmargin }
465         \hbox_unpack_drop:N \g__block_labels_box
466         \skip_horizontal:n { \leftmargin }
467     }
468
469     \legacy_if:nF { @minipage } % Why this chunk of code?
470     {
471         \__block_skip_set_to_last:N \l__block_tmpa_skip
472         \skip_vertical:n { - \l__block_tmpa_skip }
473         \skip_vertical:n { \l__block_tmpa_skip + \@outerparskip - \parskip }
474     }
475 }
476
477 \legacy_if:nTF { @nobreak }
478 {
479     { \addvspace{\skip_eval:n{\@outerparskip-\parskip}} }
480     {
481         \addpenalty \@beginparpenalty
482         \addvspace \l__block_effective_top_skip
483         \addvspace{-\parskip}
484     }
485 }
486 }

```

Extra keys to support enumitem conventions:

```

485 \keys_define:nn { template/block/display }
486 {
487     ,topsep .skip_set:N = \topsep
488     ,partopsep .skip_set:N = \partopsep
489     ,listparindent .skip_set:N = \listparindent

```

document 2e logic used
here

490 }

_kernel_displayblock_begin:
_kernel_displayblock_beginpar_hmode:w
_kernel_displayblock_beginpar_vmode:

The internal kernel hooks for tagging.

```
491 \cs_new:Npn \_kernel\_displayblock\_begin: {
492   \_block\_debug\_typeout:n{\detokenize{\_kernel\_displayblock\_begin:}}
493 }
494 \cs_new:Npn \_kernel\_displayblock\_beginpar\_hmode:w {
495   \_block\_debug\_typeout:n{\detokenize{\_kernel\_displayblock\_beginpar\_hmode:w}}
496 }
497 \cs_new:Npn \_kernel\_displayblock\_beginpar\_vmode: {
498   \_block\_debug\_typeout:n{\detokenize{\_kernel\_displayblock\_beginpar\_vmode:}}
499 }
```

(End 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 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.

```
500 \tl_new:N \@itemlabel      % should have a top-level definition
501 \tl_new:N \@listctr       % should have a top-level definition
```

(End 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.

```
502 \DeclareTemplateCode{list}{std}{1}
503 {
504   counter      = \l_block_counter_tl,
505   item-label   = \l_block_item_label_tl,
506   start        = \l_block_counter_start_int ,
507   resume       = \l_block_resume_bool ,
508   item-instance = \_block_item_instance:n ,
509   item-skip     = \itemsep ,
510   % item-par-skip = \parsep ,
511   item-penalty = \@itempenalty ,
512   item-indent  = \itemindent ,
513   label-width  = \labelwidth ,
514   label-sep    = \labelsep ,
515   legacy-support = \l_block_legacy_support_bool , % FMI questionable
516 }
517 {
518   \_block\_debug\_typeout:n{template:list:std}
519   %
520   \tl_if_empty:nF {#1} { \SetTemplateKeys{list}{std}{#1} }
```

Has this list a counter name defined in the instance?

```
521 \tl_if_empty:NTF \l_block_counter_tl
522 {
```

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`.

```
523     \tl_if_blank:oF \@listctr
524     {
```

In that case `@nmbrlist` 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.

```
525         \bool_if:NF \l__block_resume_bool
526         {
527             \int_gset:cn{ c@ \@listctr }
528             { \l__block_counter_start_int - 1 }
529         }
530     }
```

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

```
531         { \@nmbrlistfalse }
532     }
```

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.

```
533     {
534         \@nmbrlisttrue
535         \tl_set_eq:NN \@listctr \l__block_counter_tl
536         \bool_if:NF \l__block_resume_bool
537         {
538             \int_gset:cn{ c@ \@listctr }
539             { \l__block_counter_start_int - 1 }
540         }
541     }
```

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.

```
542     \tl_if_empty:NF \l__block_item_label_tl
543     {
544         \tl_set_eq:NN \@itemlabel \l__block_item_label_tl
545     }
```

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).

```
546     \legacy_if_gset_true:n { @newlist }
547     \__block_debug_typeout:n{template:list:std~end}
548 }
```

Extra keys to support `enumitem` conventions:

```
549 \keys_define:nn { template/list/std }
550 {
551     ,nosep .code:n =
552     \dim_zero:N \itemsep
553     \dim_zero:N \parsep
554     \dim_zero:N \topsep
555     \dim_zero:N \l__block_botsep_skip
```

```

556 \dim_zero:N \l__block_parbotsep_skip
557 ,midsep .skip_set:N = \topsep
558 }

```

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.

```

559 \keys_define:nn { template/item/std }
560 { label .tl_set:N = \l__block_label_given_tl }
561 \DeclareTemplateCode{item}{std}{1}
562 {
563   counter-label = \__block_counter_label:n ,
564   counter-ref   = \__block_counter_ref:n ,
565   label-ref     = \__block_label_ref:n ,
566   label-autoref = \__block_label_autoref:n ,
567   label-format  = \__block_label_format:n ,
568   label-strut   = \l__block_label_strut_bool ,
569   label-boxed   = \l__block_label_boxed_bool ,
570   next-line     = \l__block_next_line_bool ,
571   text-font     = \l__block_text_font_tl ,
572   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)

```

573   label-align = {
574     left  = \tl_set:Nn \l__block_item_align_tl { \relax \hss } ,
575     center = \tl_set:Nn \l__block_item_align_tl { \hss \hss } ,
576     right  = \tl_set:Nn \l__block_item_align_tl { \hss \relax } ,
577     parleft = \NOT_IMPLEMENTED ,
578   } ,
579 }

```

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.

```

580 {
581   \__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.

```

582 \tl_set_eq:NN \l__block_label_given_tl \c_novalue_tl
583 \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[]`.

```

584 \tl_if_novalue:oTF \l__block_label_given_tl
585 {

```

fix

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

```

586 \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
587 \bool_if:NTF \l__block_item_compatibility_bool % not sure that conditional
588 % makes sense
589 { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{\@itemlabel } } % TODO ?
590 { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{\__block_counter_label:n
591 }
592 {
593 \__block_debug_typeout:n{item~ with~ optional}
594 \__block_make_label_box:n { \l__block_label_given_tl } }
595 \bool_if:nT
596 {
597 \l__block_label_boxed_bool
598 && \dim_compare_p:n { \box_wd:N \l__block_one_label_box <= \linewidth } % TODO: is \
599 }
600 {
601 \dim_compare:nNnT
602 { \box_wd:N \l__block_one_label_box } < \labelwidth
603 {
604 \hbox_set_to_wd:Nnn \l__block_one_label_box { \labelwidth }
605 {
606 \exp_after:wN \use_i:nn \l__block_item_align_tl
607 % \hbox_unpack_drop:N \l__block_one_label_box %TODO: customize?
608 \box_use_drop:N \l__block_one_label_box
609 \exp_after:wN \use_ii:nn \l__block_item_align_tl
610 }
611 }
612 \hbox_set:Nn \l__block_one_label_box
613 { \box_use_drop:N \l__block_one_label_box }
614 }
615 \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \labelwidth
616 { \bool_set_true:N \l__block_long_label_bool }
617 { \bool_set_false:N \l__block_long_label_bool }
618 \hbox_gset:Nn \g__block_labels_box
619 {
620 \hbox_unpack_drop:N \g__block_labels_box
621 \skip_horizontal:n { \itemindent - \labelsep - \labelwidth }
622 \hbox_unpack_drop:N \l__block_one_label_box
623 \skip_horizontal:n { \labelsep }
624 \bool_if:NT \l__block_next_line_bool
625 { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
626 % version of \newline inside an hbox that will be unpacked
627 }
628 % \skip_set_eq:NN \parsep \l__block_item_parsep_skip TODO??? FMi
629 % what's that?
630 \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.

```
631 \cs_set_eq:NN \__block_item_everypar: \__block_item_everypar_std:
632 }
```

`\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.

```
633 \box_new:N \l_block_one_label_box
634 \box_new:N \g_block_labels_box
```

(End 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`.

```
635 \bool_new:N \l_block_long_label_bool
```

(End 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).

```
636 \cs_new_protected:Npn \__block_make_label_box:n #1
637 {
638   \hbox_set:Nn \l_block_one_label_box
639   {
```

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

```
640   \__kernel_list_label_begin:
641   \__block_label_format:n
642   {
643     \bool_if:NT \l_block_label_strut_bool { \strut }
644     \bool_if:NTF \l_block_legacy_support_bool
645       \makelabel
646       \use:n
647     {#1}
648   }
```

And what gets opened also needs closing:

```
649   \__kernel_list_label_end:
650 }
651 }
```

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

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

```
\__kernel_list_label_end:
652 \cs_new_eq:NN \__kernel_list_label_begin: \prg_do_nothing:
653 \cs_new_eq:NN \__kernel_list_label_end: \prg_do_nothing:
```

(End 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.

```
654 \cs_new_eq:NN \_block\_item\_everypar: \prg\_do\_nothing:
655 \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!

```
656 \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.

```
657 \cs_new_protected:Npn \_block\_item\_everypar\_std: {
658   \_block\_debug\_typeout:n{item~ everypar \on@line }
659   \legacy\_if\_set\_false:n { @minipage }
660   \legacy\_if\_gset\_false:n { @newlist }
661   \legacy\_if:nT { @inlabel }
662   {
663     \legacy\_if\_gset\_false:n { @inlabel }
664     \box\_if\_empty:NT \g\_para\_indent\_box { \kern - \itemindent }
665     \para\_omit\_indent:
666     \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:

```
667   \_kernel\_list\_label\_after:
668   \penalty \c\_zero\_int
669   }
670   \legacy\_if:nTF { @nobreak }
671   {
672     \legacy\_if\_gset\_false:n { @nobreak }
673     \int\_set:Nn \clubpenalty { 10000 }
674   }
675   {
676     \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.

```
677   \cs\_set\_eq:NN \_block\_item\_everypar: \prg\_do\_nothing:
678   }
679 }
```

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

`_kernel_list_label_after:`

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

(End definition for `_kernel_list_label_after:.`)

`\l_block_tmpa_skip`

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

(End definition for \l_block_tmpa_skip.)

\l_block_topsepadd_skip Variables equivalent to L^AT_EX 2_ε's \@topsepadd 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.

```
682 \skip_new:N \l_block_topsepadd_skip
683 \skip_new:N \l_block_effective_top_skip
```

(End definition for \l_block_topsepadd_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 \l_block_inter_item: to cleanly close what's before, then call \l_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.

```
684 \AddToHook{begindocument/before}{
685   \RenewDocumentCommand{\item}{={label}o }
686   {
687     \@inmatherr \item
```

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

```
688     \tl_if_empty:OTF \l_block_item_instance:n %%FMi?
689     { \msg_error:nnn { __block } { item-in-nonlist } { \item[{#1}] } }
690     {
691       \legacy_if:nTF { @newlist }
692       { \kernel_list_item_begin: }
693       { \l_block_inter_item: }
694     }
```

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

```
694     \tl_if_novalue:nTF {#1} % avoids reparsing label={}
695     { \l_block_item_instance:n { } }
696     { \l_block_item_instance:n {#1} }
```

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

```
697     \legacy_if_gset_true:n { @inlabel }
698     \ignorespaces
699   }
700 }
701 }
```

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

\l_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.

```
702 \cs_new_protected:Npn \l_block_inter_item: {
703   \legacy_if:nT { @inlabel }
704   { \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).

```
705   \mode_if_horizontal:T { \l_block_skip_remove_last:
706     \l_block_skip_remove_last: \par }
```


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

```

707 \__kernel_list_item_end:
708 \__kernel_list_item_begin:
709 \addpenalty \@itempenalty
710 \addvspace \itemsep
711 }

```

(End definition for __block_inter_item:.)

```

\__kernel_list_item_begin:
\__kernel_list_item_end:
712 \cs_new_eq:NN \__kernel_list_item_begin: \prg_do_nothing:
713 \cs_new_eq:NN \__kernel_list_item_end: \prg_do_nothing:

```

(End 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.

```

714 \cs_new:Npn \__block_recipe_basic: {
715 \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
716 \__block_beginpar_hmode:N
717 \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
718 \__block_beginpar_vmode:
719 \let \__kernel_displayblock_begin: \prg_do_nothing:
720 \let \__kernel_displayblock_end: \prg_do_nothing:
721 }

```

(End definition for __block_recipe_basic:.)

__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**.

```

722 \cs_new:Npn \__block_recipe_standard:
723 {
724 \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
725 \__block_beginpar_hmode:N
726 \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
727 \__block_beginpar_vmode:
728 \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
729 \cs_set_eq:NN \__kernel_displayblock_end: \__block_inner_end:

```

```

730 \tl_if_empty:NTF \l__block_tag_name_tl
731 { \tl_set:Nn \l__block_tag_inner_tag_tl {Figure} }
732 { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
733 }

```

(End definition for `__block_recipe_standard:`)

`\l__block_tag_inner_tag_tl`

```

734 \tl_new:N \l__block_tag_inner_tag_tl

```

(End 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 rollmapped 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.

```

735 \cs_new:Npn \__block_recipe_list:
736 {
737   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
738               \__block_beginpar_hmode:N
739   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
740               \__block_beginpar_vmode:
741   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_list_begin:
742   \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)).

```

743   \cs_set_eq:NN \__kernel_list_item_begin:   \__block_list_item_begin:
744   \cs_set_eq:NN \__kernel_list_item_end:     \__block_list_item_end:

```

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

```

745   \tl_if_empty:NTF \l__block_tag_name_tl
746   { \tl_set:Nn \l__tag_L_tag_tl {L} }
747   { \tl_set_eq:NN \l__tag_L_tag_tl \l__block_tag_name_tl }
748   \tl_if_empty:NTF \l__block_tag_class_tl
749   { \tl_set:Nn \l__tag_L_attr_class_tl {} }
750   { \tl_set_eq:NN \l__tag_L_attr_class_tl \l__block_tag_class_tl }
751 }

```

(End definition for `__block_recipe_list:`)

4.7 Blockenv instances

4.7.1 Basic instances

`blockenv displayblock (inst.)`

```
752 \DeclareInstance{blockenv}{displayblock}{display}
753 {
754   env-name      = displayblock,
755   tag-name      = ,
756   tag-class     = ,
757   tagging-recipe = standard,
758   inner-level-counter = ,
759   level-increase = false,
760   setup-code    = ,
761   block-instance = displayblock ,
762   inner-instance = ,
763 }
```

`blockenv displayblockflattened (inst.)`

```
764 \DeclareInstance{blockenv}{displayblockflattened}{display}
765 {
766   env-name      = displayblockflattened,
767   tag-name      = ,
768   tag-class     = ,
769   tagging-recipe = basic,
770   inner-level-counter = ,
771   level-increase = false,
772   setup-code    = ,
773   block-instance = displayblock ,
774   para-flattened = true ,
775   inner-instance = ,
776 }
```

`blockenv center (inst.)`

```
777 \DeclareInstance{blockenv}{center}{display}
778 {
779   env-name      = center,
780   tag-name      = ,
781   tag-class     = ,
782   tagging-recipe = basic,
783   inner-level-counter = ,
784   level-increase = false,
785   setup-code    = ,
786   block-instance = displayblock ,
787   para-flattened = true ,
788   para-instance = center ,
789   inner-instance = ,
790 }
```

`blockenv flushleft (inst.)`

```
791 \DeclareInstance{blockenv}{flushleft}{display}
792 {
793   env-name      = flushleft,
794   tag-name      = ,
```

```

795 tag-class      = ,
796 tagging-recipe = basic,
797 inner-level-counter = ,
798 level-increase = false,
799 setup-code     = ,
800 block-instance = displayblock ,
801 para-flattened = true ,
802 para-instance  = raggedright ,
803 inner-instance = ,
804 }

```

`blockenv flushright` (*inst.*)

```

805 \DeclareInstance{blockenv}{flushright}{display}
806 {
807   env-name      = flushleft,
808   tag-name      = ,
809   tag-class     = ,
810   tagging-recipe = basic,
811   inner-level-counter = ,
812   level-increase = false,
813   setup-code    = ,
814   block-instance = displayblock ,
815   para-flattened = true ,
816   para-instance  = raggedleft ,
817   inner-instance = ,
818 }

```

4.7.2 Blockquote instances

`blockenv quotation` (*inst.*)

```

819 \tag_if_active:T {
820   \tagpdfsetup{add-new-tag={tag=quote,role=BlockQuote}}
821   \tagpdfsetup{add-new-tag={tag=quotation,role=BlockQuote}}
822 }
823 \DeclareInstance{blockenv}{quotation}{display}
824 {
825   env-name      = quotation,
826   tag-name      = quotation,
827   tag-class     = ,
828   tagging-recipe = standard,
829   inner-level-counter = ,
830   level-increase = true,
831   setup-code    = ,
832   block-instance = quotationblock ,
833   inner-instance = ,
834 }

```

`blockenv quote` (*inst.*)

```

835 \DeclareInstance{blockenv}{quote}{display}
836 {
837   env-name      = quote,
838   tag-name      = quote,
839   tag-class     = ,

```

```

840 tagging-recipe = standard,
841 inner-level-counter = ,
842 level-increase = true,
843 setup-code = ,
844 block-instance = quoteblock ,
845 inner-instance = ,
846 }

```

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.

```

847 % \DeclareInstance{blockenv}{quotation}{display}
848 % {
849 %   env-name      = quotation,
850 %   tag-name      = ,
851 %   tag-class     = ,
852 %   tagging-recipe = blockquote,
853 %   inner-level-counter = ,
854 %   level-increase = true,
855 %   setup-code    = \setlength\rightmargin{\leftmargin}
856 %                  \setlength\parsep{1.5em} ,
857 %   block-instance = displayblock ,
858 %   inner-instance = ,
859 % }
860 % \DeclareInstance{blockenv}{quote}{display}
861 % {
862 %   env-name      = quote,
863 %   tag-name      = ,
864 %   tag-class     = ,
865 %   tagging-recipe = blockquote,
866 %   inner-level-counter = ,
867 %   level-increase = true,
868 %   setup-code    = \setlength\rightmargin{\leftmargin} ,
869 %   block-instance = displayblock ,
870 %   inner-instance = ,
871 % }

```

4.7.3 Verbatim instances

`blockenv verbatim` (*inst.*)

```

872 \tag_if_active:T {
873   \tagpdfsetup{add-new-tag={tag=verbatim,role=P}}
874   \tagpdfsetup{add-new-tag={tag=codeline,role=Sub}}

```

Possible alternative for PDF 1.7:

```

875 % \tagpdfsetup{add-new-tag={tag=verbatim,role=Div}}
876 % \tagpdfsetup{add-new-tag={tag=codeline,role=P}}
877 }
878 \DeclareInstance{blockenv}{verbatim}{display}
879 {
880   env-name      = verbatim,
881   tag-name      = verbatim,
882   tag-class     = ,

```

```

883 tagging-recipe = standard,
884 inner-level-counter = ,
885 level-increase = false,
886 setup-code      = ,
887 block-instance = displayblock ,
888 inner-instance = ,
889 final-code      = \legacyverbatimsetup ,
890 }

```

4.7.4 Standard list instances

`blockenv itemize (inst.)`

```

891 \DeclareInstance{blockenv}{itemize}{display}
892 {
893   env-name      = itemize,
894   tag-name      = itemize,
895   tag-class     = itemize,
896   tagging-recipe = list,
897   inner-level-counter = \@itemdepth,
898   level-increase = true,
899   max-inner-levels = 4,
900   setup-code    = ,
901   block-instance = list ,
902   inner-instance = itemize ,
903 }

```

`blockenv enumerate (inst.)`

```

904 \DeclareInstance{blockenv}{enumerate}{display}
905 {
906   env-name      = enumerate,
907   tag-name      = enumerate,
908   tag-class     = enumerate,
909   tagging-recipe = list,
910   level-increase = true,
911   setup-code    = ,
912   block-instance = list ,
913   inner-level-counter = \@enumdepth,
914   max-inner-levels = 4,
915   inner-instance = enum ,
916 }

```

`blockenv description (inst.)`

```

917
918 \DeclareInstance{blockenv}{description}{display}
919 {
920   env-name      = description,
921   tag-name      = description,
922   tag-class     = description,
923   tagging-recipe = list,
924   inner-level-counter = ,
925   level-increase = true,
926   setup-code    = ,
927   block-instance = list ,
928   inner-instance = description ,

```

929 }

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

```

930 \DeclareInstance{blockenv}{list}{display}
931 {
932   env-name      = list,
933   tag-name      = list,
934   tag-class     = ,
935   tagging-recipe = list,
936   level-increase = true,
937   setup-code    = \legacylistsetupcode ,
938   block-instance = list ,
939   inner-level-counter = ,
940   inner-instance = legacy ,
941 }
```

4.8 Block instances

4.8.1 Displayblock instances

We provide 6 nesting levels (as in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\varepsilon}$). 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).

```

942 \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.) 943 \DeclareInstance{block}{displayblock-0}{display}
block displayblock-3 (inst.) 944 {
block displayblock-4 (inst.) 945   leftmargin      = 0pt ,
block displayblock-5 (inst.) 946   parindent      = 0pt ,
block displayblock-6 (inst.) 947 }

948 \DeclareInstanceCopy{block}{displayblock-1}{displayblock-0}
949 \DeclareInstanceCopy{block}{displayblock-2}{displayblock-0}
950 \DeclareInstanceCopy{block}{displayblock-3}{displayblock-0}
951 \DeclareInstanceCopy{block}{displayblock-4}{displayblock-0}
952 \DeclareInstanceCopy{block}{displayblock-5}{displayblock-0}
953 \DeclareInstanceCopy{block}{displayblock-6}{displayblock-0}
```

4.8.2 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 side.

```

block quoteblock-2 (inst.) 954 \DeclareInstance{block}{quoteblock-1}{display}
block quoteblock-3 (inst.) 955 { rightmargin = \KeyValue{leftmargin} }
block quoteblock-4 (inst.)
block quoteblock-5 (inst.)
block quoteblock-6 (inst.)
```

```

956 \DeclareInstanceCopy{block}{quoteblock-2}{quoteblock-1}
957 \DeclareInstanceCopy{block}{quoteblock-3}{quoteblock-1}
958 \DeclareInstanceCopy{block}{quoteblock-4}{quoteblock-1}
959 \DeclareInstanceCopy{block}{quoteblock-5}{quoteblock-1}
960 \DeclareInstanceCopy{block}{quoteblock-6}{quoteblock-1}

```

block quotationblock-1 (*inst.*) Quotation additionally changes the parindent.

```

block quotationblock-2 (inst.) 961 \DeclareInstance{block}{quotationblock-1}{display}
block quotationblock-3 (inst.) 962 { parindent = 1.5em , rightmargin = \KeyValue{leftmargin} }
block quotationblock-4 (inst.)
block quotationblock-5 (inst.) 963 \DeclareInstanceCopy{block}{quotationblock-2}{quotationblock-1}
block quotationblock-6 (inst.) 964 \DeclareInstanceCopy{block}{quotationblock-3}{quotationblock-1}
965 \DeclareInstanceCopy{block}{quotationblock-4}{quotationblock-1}
966 \DeclareInstanceCopy{block}{quotationblock-5}{quotationblock-1}
967 \DeclareInstanceCopy{block}{quotationblock-6}{quotationblock-1}

```

4.8.3 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
block list-3 (*inst.*) already done in the legacy `\list{romannumeral}` unless a different layout is wanted.

```

block list-4 (inst.) 968 \DeclareInstance{block}{list-1}{display}{
block list-5 (inst.) 969 % heading = ,
block list-6 (inst.) 970 % beginsep = \topsep ,
971 % begin-par-skip = \partopsep ,
972 % par-skip = \parsep ,
973 % end-skip = \KeyValue{beginsep} ,
974 % end-par-skip = \KeyValue{begin-par-skip} ,
975 % beginpenalty = \UseName{@beginparpenalty} ,
976 % endpenalty = \UseName{@endparpenalty} ,
977 % leftmargin = \leftmargin ,
978 % rightmargin = \rightmargin ,
979 % parindent = \listparindent ,
980 }
981 \DeclareInstance{block}{list-2}{display}{}
982 \DeclareInstance{block}{list-3}{display}{}
983 \DeclareInstance{block}{list-4}{display}{}
984 \DeclareInstance{block}{list-5}{display}{}
985 \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 beformatted.

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.) 986 \DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi }
987 \DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii }
988 \DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
989 \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 use on every level.

```
list enumerate-2 (inst.)
list enumerate-3 (inst.)
list enumerate-4 (inst.)
990 \DeclareInstance{list}{enum-1}{std}
991 { item-label = \labelenumi , counter = enumi }
992 \DeclareInstance{list}{enum-2}{std}
993 { item-label = \labelenumii , counter = enumii }
994 \DeclareInstance{list}{enum-3}{std}
995 { item-label = \labelenumiii , counter = enumiii }
996 \DeclareInstance{list}{enum-4}{std}
997 { 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

```
998 \DeclareInstance{list}{legacy}{std} {
999   item-instance = basic ,
1000   legacy-support = true ,
1001 }
```

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

```
1002 \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
`item description (inst.)` and `basic` for use with all other lists (up to now).

```
1003 \DeclareInstance{item}{basic}{std}
1004 {
1005   label-align = right ,
1006 }
1007 \DeclareInstance{item}{description}{std}
1008 {
1009   label-format = \normalfont\bfseries #1 ,
1010 }
```

4.11 Para instances

```
1011 \tag_if_active:T
1012 {
1013   \tagpdfsetup
1014   {
1015     newattribute = {justify}    {/O /Layout /TextAlign/Justify},
1016     newattribute = {center}     {/O /Layout /TextAlign/Center},
1017     newattribute = {raggedright}{/O /Layout /TextAlign/Start},
1018     newattribute = {raggedleft} {/O /Layout /TextAlign/End},
1019   }
1020 }
```

`para center (inst.)`

```

1021 \DeclareInstance{para}{center}{std}
1022 {
1023     indent-width          = Opt ,
1024     start-skip             = Opt ,
1025     left-skip              = \@flushglue ,
1026     right-skip             = \@flushglue ,
1027     end-skip               = \z@skip ,
1028     final-hyphen-demerits = 0 ,
1029     cr-cmd                 = \@centercr ,
1030     para-class             = center ,
1031 }
1032 \DeclareInstance{para}{raggedright}{std}
1033 {
1034     indent-width          = Opt ,
1035     start-skip             = Opt ,
1036     left-skip              = \z@skip ,
1037     right-skip             = \@flushglue ,
1038     end-skip               = \z@skip ,
1039     final-hyphen-demerits = 0 ,
1040     cr-cmd                 = \@centercr ,
1041     para-class             = raggedright ,
1042 }
1043 \DeclareInstance{para}{raggedleft}{std}
1044 {
1045     indent-width          = Opt ,
1046     start-skip             = Opt ,
1047     left-skip              = \@flushglue ,
1048     right-skip             = \z@skip ,
1049     end-skip               = \z@skip ,
1050     final-hyphen-demerits = 0 ,
1051     cr-cmd                 = \@centercr ,
1052     para-class             = raggedleft ,
1053 }
1054 \DeclareInstance{para}{justify}{std}
1055 {
1056     % indent-width          = Opt ,
1057     start-skip             = Opt ,
1058     left-skip              = \z@skip ,
1059     right-skip             = \z@skip ,
1060     end-skip               = \@flushglue ,
1061     final-hyphen-demerits = 5000 ,
1062     cr-cmd                 = \@normalcr ,
1063     para-class             = justify ,
1064 }
1065 \DeclareRobustCommand\centering {\UseInstance{para}{center}{}}
1066 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}{}}
1067 \DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}{}}
1068 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}{}}
1069
1070 \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!

```
1071 \tag_if_active:T {
```

`_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.

```
1072 \cs_set:Npn \_block\_beginpar\_vmode: {
1073     \_block\_debug\_typeout:n
1074     { @endpe = \legacy\_if:nTF { @endpe }{true}{false}
1075     \on@line }
1076     \legacy\_if:nTF { @endpe }
1077     {
1078     \legacy\_if\_gset\_false:n { @endpe }
1079     }
```

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`.

```
1080     {
1081     % \typeout{-->G1~ (\int\_use:N \l\_block\_flattened\_level\_int)}
1082     \int\_compare:nNnT \l\_block\_flattened\_level\_int < 2
1083     {
1084     \int\_gincr:N \g\_tag\_para\_main\_begin\_int
1085     \tagstructbegin{tag=\l\_tag\_para\_main\_tag\_tl}
1086     }
1087     }
1088 }
```

(End 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).

```
1089 \cs_set:Npn \_block\_beginpar\_hmode:N #1
1090 {
1091     \tag\_mc\_end:
1092     \int\_gincr:N \g\_tag\_para\_end\_int
1093     \_block\_debug\_typeout:n{increment~ /P \on@line }
1094     \bool\_if:NT \l\_tag\_para\_show\_bool
1095     { \tag\_mc\_begin:n{artifact}
1096     \rlap{\color\_select:n{red}\tiny\ \int\_use:N\g\_tag\_para\_end\_int}
1097     \tag\_mc\_end:
1098     }
1099     \tag\_struct\_end:
1100     \tagpdfparaOff \par \tagpdfparaOn
1101 }
```

(End 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.

```

1102 \cs_set:Npn \_kernel\_displayblock\_doendpe: {
1103     \bool_if:NT \l__tag\_para\_bool
1104     {

```

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.

```

1105     \legacy_if:nT { @endpe }
1106     {

```

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.

```

1107         \_block\_debug\_typeout:n
1108         { flattened= \bool_if:NTF
1109             \l__tag\_para\_flattened\_bool {true}{false}
1110             \on@line }
1111         \bool_if:NF \l__tag\_para\_flattened\_bool
1112         {
1113             \_block\_debug\_typeout:n{Structure-end~
1114                 \l__tag\_para\_main\_tag\_tl\space after~ list \on@line }
1115             \int_gincr:N \g__tag\_para\_main\_end\_int
1116             \tag\_struct\_end: %text-unit
1117         }
1118     }
1119 }
1120 }

```

(End 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).

```

1121 \RemoveFromHook{para/begin}[tagpdf]
1122 \AddToHook{para/begin}[tagpdf]{
1123     \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).

```

1124     \legacy_if:nF { @inlabel }
1125     {

```

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).

```

1126     \__block_debug_typeout:n
1127     { @endpe = \legacy_if:nTF { @endpe }{true}{false}
1128       \on@line }
1129   \legacy_if:nF { @endpe }
1130   {
1131     \bool_if:NF \l__tag_para_flattened_bool
1132     {
1133       \int_gincr:N \g__tag_para_main_begin_int
1134       \tag_struct_begin:n{tag=\l__tag_para_main_tag_tl}
1135     }
1136   }
1137   \int_gincr:N \g__tag_para_begin_int
1138   \__block_debug_typeout:n{increment~ P \on@line }
1139   \tag_struct_begin:n
1140   {
1141     tag=\l__tag_para_tag_tl
1142     ,attribute-class=\l_tag_para_attr_class_tl
1143   }
1144   \__tag_check_para_begin_show:nn {green}{\PARALABEL}
1145   \tag_mc_begin:n {}
1146 }
1147 }
1148 }

1149 \tag_if_active:T {
1150 % \tagpdfsetup{add-new-tag={tag=text-unit,role=Part}}
1151 }

1152 \RemoveFromHook{para/end}[tagpdf]
1153 \AddToHook{para/end}
1154 {
1155   \bool_if:NT \l__tag_para_bool
1156   {
1157     \int_gincr:N \g__tag_para_end_int
1158     \__block_debug_typeout:n{increment~ /P \on@line }
1159     \tag_mc_end:
1160     \__tag_check_para_end_show:nn {red}{}
1161     \tag_struct_end:
1162     \bool_if:NF \l__tag_para_flattened_bool
1163     {
1164       \int_gincr:N \g__tag_para_main_end_int
1165       \tag_struct_end:
1166     }
1167   }
1168 }

1169 \def\PARALABEL{NP-}

```

(End definition for para/begin. 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).

```

1170 \cs_set_protected:Npn \para_end: {
1171   \scan_stop:
1172   \mode_if_horizontal:TF {

```

```

1173 \mode_if_inner:F {
1174     \tex_unskip:D
1175     \hook_use:n{para/end}
1176     \@kernel@after@para@end
1177     \mode_if_horizontal:TF {
1178         \if_int_compare:w 11 = \tex_lastnodetype:D
1179             \tex_hskip:D \c_zero_dim
1180         \fi:
1181         \tex_par:D
1182         \hook_use:n{para/after}
1183         \@kernel@after@para@after
1184     }
1185     { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
1186 }
1187 }
1188 {
1189     \__kernel_endpe_vmode:      % should do nothing if no tagging
1190     \tex_par:D
1191 }
1192 }
1193 \cs_set_eq:NN \par \para_end:
1194 \cs_set_eq:NN \__blockpar \para_end:
1195 \cs_set_eq:NN \endgraf \para_end:

```

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

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

```

1196 \DeclareRobustCommand*\begin[1]{%
1197     \UseHook{env/#1/before}%
1198     \@ifundefined{#1}%
1199         {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
1200         {\def\reserved@a{\def\@currentenv{#1}%
1201             \edef\@currentvline{\on@line}%
1202             \@execute@begin@hook{#1}%
1203             \csname #1\endcsname}}%
1204     \@ignorefalse
1205     \begingroup
1206     \__kernel_endpe_vmode:
1207     \reserved@a}

```

(End 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.

```

1208 \cs_new:Npn \__kernel_endpe_vmode: {
1209     \if@endpe \ifvmode
1210         \bool_if:NT \l__tag_para_bool
1211     {
1212         \bool_if:NF \l__tag_para_flattened_bool
1213         {
1214             \int_gincr:N \g__tag_para_main_end_int
1215             \tag_struct_end:
1216         }
1217         \@endpefalse

```

```

1218 }
1219 \fi \fi
1220 }

```

(End 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

```

1221 \cs_set:Npn \_kernel_list_label_after: {
1222   \bool_if:NT \l__tag_para_bool
1223   {
1224     \bool_if:NF \l__tag_para_flattened_bool
1225     {
1226       % \typeout{-->G3}
1227       \int_gincr:N \g__tag_para_main_begin_int
1228       \tag_struct_begin:n{tag=\l__tag_para_main_tag_tl}
1229     }
1230     \int_gincr:N \g__tag_para_begin_int
1231     \__block_debug_typeout:n{increment~ P \on@line }
1232     \tag_struct_begin:n {
1233       tag=\l__tag_para_tag_tl
1234       ,attribute-class=\l__tag_para_attr_class_tl
1235     }
1236     \bool_if:NT \l__tag_para_show_bool
1237     {
1238       \tag_mc_begin:n {artifact}
1239       \llap {\color_select:n {blue}\tiny Li-\int_use:N \g__tag_para_begin_int \ }
1240       \tag_mc_end:
1241     }
1242     \tag_mc_begin:n {tag=P}
1243   }
1244 }

```

(End definition for _kernel_list_label_after:.)

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

```

1245 \cs_new:Npn \_block_inner_begin: {
1246   \tag_structbegin{tag=\l__block_tag_inner_tag_tl}
1247 }

```

(End definition for _block_inner_begin:.)

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

```

1248 \cs_new:Npn \_block_inner_end: {
1249   \__block_debug_typeout:n{block-end \on@line}
1250   \legacy_if:nT { @endpe }
1251   {
1252     \int_gincr:N \g__tag_para_main_end_int
1253     \__block_debug_typeout:n{close~ /text-unit \on@line}
1254     \tag_structend
1255   }
1256   \tag_structend % end inner structure
1257 }

```

(End definition for _block_inner_end:.)

4.12.1 List tags

```

1258 \tl_new:N \l__tag_L_tag_tl
1259 \tl_set:Nn \l__tag_L_tag_tl {L}
1260
1261 \tl_new:N \l__tag_L_attr_class_tl
1262 \tl_set:Nn \l__tag_L_attr_class_tl {list}
1263
1264 \tag_if_active:T
1265 {
1266   \tagpdfsetup
1267   {
1268     % default if unknown
1269     newattribute = {list}{/0 /List /ListNumbering/None},
1270     newattribute = {itemize}{/0 /List /ListNumbering/Unordered},
1271     newattribute = {enumerate}{/0 /List /ListNumbering/Ordered},
1272     newattribute = {description}{/0 /List /ListNumbering/Description},
1273   }
1274 }
1275
1276 \def\LItag{LI}

```

`__block_list_begin:` Start a list ...

```

1275 \cs_set:Npn \__block_list_begin: {
1276   \tagstructbegin
1277   {
1278     tag=\l__tag_L_tag_tl
1279     ,attribute-class=\l__tag_L_attr_class_tl
1280   }
1281 }

```

(End definition for __block_list_begin:.)

`__block_list_item_begin:` Start tagging a list item.

```

1282 \cs_set:Npn \__block_list_item_begin: { \tagstructbegin{tag=\LItag} }

```

(End definition for __block_list_item_begin:.)

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

```

1283 \cs_set:Npn \__kernel_list_label_begin: {
1284   %
1285   % FMI: this needs a different logic to decide when to make the label
1286   %   an artifact (after cleaning up the the \item code ), therefore
1287   %   disabled for now
1288   % \tl_if_empty:oTF \@itemlabel
1289   % {
1290   %   \tag_mc_begin:n {artifact}
1291   % }
1292   % {
1293   %   \tagstructbegin{tag=Lbl}
1294   %   \tagmcbegin{tag=Lbl}
1295   % }
1296 }

```

(End 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.

```

1297 \cs_set:Npn \__kernel_list_label_end: {
1298   \tagmccend                                % end mc-Lbl or artifact
1299   % FMi: unconditionally for now
1300   % \tl_if_empty:oF \@itemlabel
1301   \tagstructend % end Lbl
1302   \tagstructbegin{tag=\LBody}
1303 }
1304 \def\LBody{\LBody}

```

(End 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).

```

1305 \cs_set:Npn \__block_list_item_end: {
1306   \legacy_if:nT { @endpe }
1307   {
1308     \int_gincr:N \g__tag_para_main_end_int
1309     \tagstructend % text-unit
1310     % \__block_debug_typeout:n{Structure-end~ P~ at~ item-end \on@line }
1311   }
1312   \tagstructend \tagstructend % end LBody, LI
1313 }

```

(End 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.

```

1314 \cs_set:Npn \__block_list_end: {
1315   \legacy_if:nT { @endpe }
1316   {
1317     \int_gincr:N \g__tag_para_main_end_int
1318     \tagstructend % text-unit
1319     \__block_debug_typeout:n{Structure-end~ P~ at~ list-end \on@line }
1320   }
1321   \tagstructend\tagstructend % end LBody, LI
1322   \tagstructend % end L
1323 }

```

(End definition for __block_list_end:.)

End of tagging related declarations.

```

1324 }
1325 \</package>
1326 \<*latex-lab>
1327 \ProvidesFile{block-tagging-latex-lab-testphase.ltx}
1328   [\ltblocksdate\space \ltblocksversion\space
1329     blockenv implementation]
1330 \RequirePackage{latex-lab-testphase-block-tagging}
1331 \</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	
<code>\</code>	387
† internal commands:	
<code>\i_block_flattened_level_int</code>	20
<code>\l</code>	255, 257, 1096, 1239
Numbers	
<code>\1</code>	51
<code>\2</code>	51
A	
<code>\addpenalty</code>	369, 479, 709
<code>\AddToHook</code>	142, 153, 161, 201, 212, 240, 655, 684, 1122, 1153
<code>\addvspace</code>	370, 477, 480, 481, 710
<code>\arabic</code>	7, 94
B	
<code>\begin</code>	46, 1196
<code>\begingroup</code>	1205
<code>\bfseries</code>	255, 257, 1009
<code>block</code> (objecttype)	29
block commands:	
<code>\block_debug_off:</code>	116, 121, 134
<code>\block_debug_on:</code>	116, 116, 133
<code>\g_block_nesting_depth_int</code>	258, 302, 306, 307, 313, 316, 344
<code>block display</code> (template)	51, 394
<code>block displayblock-0</code> (instance)	943
<code>block displayblock-1</code> (instance)	943
<code>block displayblock-2</code> (instance)	943
<code>block displayblock-3</code> (instance)	943
<code>block displayblock-4</code> (instance)	943
<code>block displayblock-5</code> (instance)	943
<code>block displayblock-6</code> (instance)	943
block internal commands:	
<code>__block_beginpar_hmode:N</code>	716, 725, 738, 1089, 1089
<code>__block_beginpar_vmode:</code>	718, 727, 740, 1072, 1072
<code>\l_block_block_instance_tl</code>	268, 313, 315
<code>\l_block_botsep_skip</code>	400, 555
<code>__block_counter_label:n</code>	563, 590
<code>__block_counter_ref:n</code>	564
<code>\l_block_counter_start_int</code>	506, 528, 539
<code>\l_block_counter_tl</code>	504, 521, 535
<code>__block_debug:n</code>	114, 114, 128
<code>\g__block_debug_bool</code>	113, 118, 123, 129, 131
<code>__block_debug_gset:</code>	116, 119, 124, 126
<code>__block_debug_typeout:n</code>	114, 115, 130, 278, 312, 320, 325, 342, 359, 376, 492, 495, 498, 518, 547, 581, 593, 658, 1073, 1093, 1107, 1113, 1126, 1138, 1158, 1231, 1249, 1253, 1310, 1319
<code>\l_block_effective_top_skip</code>	431, 433, 480, 682
<code>\l_block_env_name_tl</code>	262, 278
<code>\l_block_env_params_tl</code>	224
<code>\l_block_final_code_tl</code>	21, 275, 336
<code>\l_block_flattened_level_int</code>	282, 284, 289, 338, 1081, 1082
<code>\l_block_heading_tl</code>	396, 410, 411
<code>__block_inner_begin:</code>	728, 1245, 1245
<code>__block_inner_end:</code>	729, 1248, 1248
<code>\l_block_inner_instance_tl</code>	273, 323, 325, 329
<code>\l_block_inner_instance_type_tl</code>	272, 328
<code>\l_block_inner_level_counter_tl</code>	270, 293, 295, 298, 326, 327, 330, 331
<code>__block_inter_item:</code>	32, 693, 702, 702
<code>\l_block_item_align_tl</code>	9, 574, 575, 576, 606, 609
<code>\l_block_item_compatibility_bool</code>	572, 587
<code>__block_item_everypar:</code>	30, 31, 631, 654, 654, 655, 677
<code>__block_item_everypar_std:</code>	631, 654, 657
<code>__block_item_instance:n</code>	32, 508, 688, 695, 696
<code>\l_block_item_label_tl</code>	505, 542, 544
<code>\l_block_item_parsep_skip</code>	628
<code>__block_label_autoref:n</code>	566
<code>\l_block_label_boxed_bool</code>	569, 597
<code>__block_label_format:n</code>	30, 567, 636, 641
<code>\l_block_label_given_tl</code>	28, 560, 582, 584, 594
<code>__block_label_ref:n</code>	565
<code>\l_block_label_strut_bool</code>	568, 643
<code>\g_block_labels_box</code>	28, 30, 462, 465, 618, 620, 633, 666

\l_block_legacy_env_params_tl	18, 10, 216, 232	block quotationblock-5 (instance)	961
\l_block_legacy_support_bool	515, 644	block quotationblock-6 (instance)	961
\l_block_level_incr_bool	266, 300, 343	block quoteblock-1 (instance)	954
_block_list_begin:	741, 1275, 1275	block quoteblock-2 (instance)	954
_block_list_end:	742, 1314, 1314	block quoteblock-3 (instance)	954
_block_list_item_begin:	743, 1282, 1282	block quoteblock-4 (instance)	954
_block_list_item_end:	744, 1305, 1305	block quoteblock-5 (instance)	954
\l_block_long_label_bool	616, 617, 625, 635	block quoteblock-6 (instance)	954
_block_make_label_box:n	28, 589, 590, 594, 636, 636	blockenv (objecttype)	29
\l_block_max_inner_levels_tl	271, 296	blockenv center (instance)	777
\l_block_next_line_bool	570, 624	blockenv description (instance)	917
\l_block_one_label_box	30, 598, 602, 604, 607, 608, 612, 613, 615, 622, 633, 638	blockenv display (template)	34, 260
\l_block_para_instance_tl	269, 318, 320, 321	blockenv displayblock (instance)	752
\l_block_parbotsep_skip	401, 556	blockenv displayblockflattened (instance)	764
_block_recipe_basic:	714, 714	blockenv enumerate (instance)	904
_block_recipe_list:	735, 735	blockenv flushleft (instance)	791
_block_recipe_standard:	722, 722	blockenv flushright (instance)	805
\l_block_resume_bool	507, 525, 536	blockenv itemize (instance)	891
\l_block_setup_code_tl	267, 311	blockenv list (instance)	930
_block_skip_remove_last:	106, 109, 356, 420, 705, 706	blockenv quotation (instance)	819
_block_skip_set_to_last:N	106, 106, 363, 470	blockenv quote (instance)	835
\l_block_tag_class_tl	264, 748, 750	blockenv verbatim (instance)	872
\l_block_tag_inner_tag_tl	731, 732, 734, 1246	blockpar internal commands:	
\l_block_tag_name_tl	263, 730, 732, 745, 747	_blockpar	1194
\l_block_tagging_recipe_tl	265, 310	bool commands:	
\l_block_text_font_tl	571	\bool_gset_false:N	123
\l_block_tpa_skip	470, 471, 472, 681	\bool_gset_true:N	118
\l_block_topsepadd_skip	22, 370, 413, 416, 431, 682	\bool_if:NTF	129, 131, 196, 287, 300, 343, 525, 536, 587, 624, 625, 643, 644, 1094, 1103, 1108, 1111, 1123, 1131, 1155, 1162, 1210, 1212, 1222, 1224, 1236
block list-1 (instance)	968	\bool_if:nTF	595
block list-2 (instance)	968	\bool_new:N	113, 635
block list-3 (instance)	968	\bool_set_false:N	617
block list-4 (instance)	968	\bool_set_true:N	616
block list-5 (instance)	968	box commands:	
block list-6 (instance)	968	\box_if_empty:NTF	664
block quotationblock-1 (instance)	961	\box_new:N	633, 634
block quotationblock-2 (instance)	961	\box_use_drop:N	608, 613, 666
block quotationblock-3 (instance)	961	\box_wd:N	598, 602, 615
block quotationblock-4 (instance)	961	\break	625
		C	
		center (env.)	142
		\centering	1065
		\clubpenalty	15, 673, 676
		color commands:	
		\color_select:n	1096, 1239
		cs commands:	
		\cs_generate_variant:Nn	110
		\cs_gset_protected:Npx	128, 130
		\cs_if_exist:NTF	111

I

if commands:

\if_int_compare:w	1178
\iffalse	28
\ifhmode	183
\iftrue	27
\ifvmode	1209
\ignorespaces	5, 21, 49, 698
\indent	704

instances:

block displayblock-0	943
block displayblock-1	943
block displayblock-2	943
block displayblock-3	943
block displayblock-4	943
block displayblock-5	943
block displayblock-6	943
block list-1	968
block list-2	968
block list-3	968
block list-4	968
block list-5	968
block list-6	968
block quotationblock-1	961
block quotationblock-2	961
block quotationblock-3	961
block quotationblock-4	961
block quotationblock-5	961
block quotationblock-6	961
block quoteblock-1	954
block quoteblock-2	954
block quoteblock-3	954
block quoteblock-4	954
block quoteblock-5	954
block quoteblock-6	954
blockenv center	777
blockenv description	917
blockenv displayblock	752
blockenv displayblockflattened	764
blockenv enumerate	904
blockenv flushleft	791
blockenv flushright	805
blockenv itemize	891
blockenv list	930
blockenv quotation	819
blockenv quote	835
blockenv verbatim	872
item basic	1003
item description	1003
list description	1002
list enumerate-1	990
list enumerate-2	990
list enumerate-3	990
list enumerate-4	990

list itemize-1	986
list itemize-2	986
list itemize-3	986
list itemize-4	986
list legacy	998
para center	1021

int commands:

\int_compare:nNnTF	282, 295, 302, 443, 1082
\int_gdecr:N	344
\int_gincr:N	306, 1084, 1092, 1115, 1133, 1137, 1157, 1164, 1214, 1227, 1230, 1252, 1308, 1317
\int_gset:Nn	527, 538
\int_incr:N	284, 289, 298, 442
\int_new:N	338
\int_set:Nn	673
\int_set_eq:NN	676
\int_to_roman:n	307
\int_use:N	313, 315, 327, 331, 1081, 1096, 1239
\int_zero:N	437
\c_zero_int	668
\interlinepenalty	180, 183
item (objecttype)	29
\item	15, 22, 24, 25, 27, 28, 30, 34, 51, 255, 257, 684, 704, 1286
item basic (instance)	1003
item description (instance)	1003
item std (template)	92, 559
\itemindent	50, 87, 228, 512, 621, 664
itemize (env.)	201
\itemsep	7, 85, 509, 552, 710
\itshape	255, 257

J

\justifying	1068, 1070
-------------	------------

K

\kern	664
kernel internal commands:	
_kernel_displayblock_begin:	33, 458, 491, 491, 719, 728, 741
_kernel_displayblock_beginpar_hmode:w	421, 491, 494, 715, 724, 737
_kernel_displayblock_beginpar_vmode:	417, 491, 497, 717, 726, 739
_kernel_displayblock_doendpe:	44, 16, 26, 1102, 1102
_kernel_displayblock_end:	33, 358, 375, 375, 720, 729, 742
_kernel_endpe_vmode:	1189, 1206, 1208, 1208

<code>__kernel_list_item_begin:</code>	<code>list description (instance)</code>	1002
. 692 , 708 , 712 , 712 , 743	<code>list enumerate-1 (instance)</code>	990
<code>__kernel_list_item_end:</code>	<code>list enumerate-2 (instance)</code>	990
. 707 , 712 , 713 , 744	<code>list enumerate-3 (instance)</code>	990
<code>__kernel_list_label_after:</code>	<code>list enumerate-4 (instance)</code>	990
. 667 , 680 , 680 , 1221 , 1221	<code>list itemize-1 (instance)</code>	986
<code>__kernel_list_label_begin:</code>	<code>list itemize-2 (instance)</code>	986
. 640 , 652 , 652 , 1283 , 1283	<code>list itemize-3 (instance)</code>	986
<code>__kernel_list_label_end:</code>	<code>list itemize-4 (instance)</code>	986
. 649 , 652 , 653 , 1297 , 1297	<code>list legacy (instance)</code>	998
keys commands:	<code>list std (template)</code>	78 , 502
<code>\keys_define:nn</code> 28 , 485 , 549 , 559	<code>\list<romannumeral></code>	39 , 40
<code>\KeyValue</code> 57 , 58 , 95 , 955 , 962 , 973 , 974	<code>\listparindent</code>	
 6 , 63 , 226 , 406 , 454 , 489 , 630 , 979	
L	<code>\Litag</code>	1274 , 1282
<code>\labelenumi</code>	<code>\llap</code>	1239
<code>\labelenumii</code>	<code>\ltblocksdate</code>	4 , 1328
<code>\labelenumiii</code>	<code>\ltblocksversion</code>	4 , 1328
<code>\labelenumiv</code>		
<code>\labelindent</code>	M	
<code>\labelitemi</code>	<code>\makelabel</code>	7 , 18 , 41 , 231 , 246 , 645
<code>\labelitemii</code>	<code>\MakeLinkTarget</code>	589 , 590
<code>\labelitemiii</code>	mode commands:	
<code>\labelitemiv</code>	<code>\mode_if_horizontal:TF</code>	
<code>\labelsep</code> 7 , 50 , 89 , 255 , 257 , 514 , 621 , 623 355 , 705 , 1172 , 1177	
<code>\labelwidth</code> 7 ,	<code>\mode_if_inner:TF</code>	1173
50 , 88 , 245 , 513 , 602 , 604 , 615 , 621	<code>\mode_if_vertical:TF</code>	414
<code>\language</code>	<code>\mode_leave_vertical:</code>	347 , 411 , 412
<code>\lastbox</code>	msg commands:	
<code>\LBody</code>	<code>\msg_error:nnn</code>	689
<code>\leavevmode</code>	<code>\msg_error:nnnn</code>	1185
<code>\leftmargin</code> 6 , 50 , 61 , 244 ,		
405 , 455 , 456 , 464 , 466 , 855 , 868 , 977	N	
<code>\leftskip</code>	<code>\newcommand</code>	194
legacy commands:	<code>\newcounter</code>	339
<code>\legacy_if:nTF</code>	<code>\NewDocumentEnvironment</code>	136 , 139
. 233 , 345 , 350 , 360 , 361 , 412 ,	<code>\newline</code>	626
423 , 429 , 440 , 459 , 468 , 476 , 661 ,	<code>\newtheorem</code>	18
670 , 691 , 703 , 1074 , 1076 , 1105 ,	<code>\nobreak</code>	625
1124 , 1127 , 1129 , 1250 , 1306 , 1315	<code>\nobreakspace</code>	198
<code>\legacy_if_gset_false:n</code>	<code>\normalfont</code>	1009
. 348 , 353 , 660 , 663 , 672 , 1078	NOT commands:	
<code>\legacy_if_gset_true:n</code>	<code>\NOT_IMPLEMENTED</code>	577
. 371 , 373 , 546 , 697	<code>\null</code>	180
<code>\legacy_if_set_false:n</code>		
. 230 , 430 , 461 , 659	O	
<code>\legacy_if_set_true:n</code>	<code>\obeylines</code>	186
. 425 , 426	object types:	
<code>\legacylistsetupcode</code>	<code>block</code>	29
<code>\legacyverbatimsetup</code>	<code>blockenv</code>	29
<code>\let</code>	<code>item</code>	29
<code>\linewidth</code>	<code>list</code>	29
<code>list (env.)</code>	<code>para</code>	29
<code>list (objecttype)</code>		
<code>\list</code>		
. 51 , 242		

P	
\par . . . 10, 11, 24, 27, 32, 33, 43–45, 12, 19, 178, 356, 421, 704, 706, 1100, 1193	
par commands:	
\par_end: 11	
par internal commands:	
\l__par_fixed_word_spaces_bool . 385	
\l__par_start_skip 381	
para (objecttype) 29	
para center (instance) 1021	
para commands:	
\para_end: 24, 46, 445, 449, 1170, 1170, 1193, 1194, 1195	
\g_para_indent_box 664	
\para_omit_indent: 665	
para std (template) 66, 378	
para/begin 1121	
\PARALABEL 1144, 1169	
\parfillskip 384, 436	
\parindent 6, 68, 380, 454, 630	
\parsep 6, 56, 399, 453, 510, 553, 628, 856, 972	
\parskip 25, 367, 433, 452, 453, 472, 477, 481	
\partopsep 6, 55, 398, 416, 488, 971	
\pdfakespace 17, 198	
\penalty 180, 183, 668	
prg commands:	
\prg_do_nothing: 26, 652, 653, 654, 677, 680, 712, 713, 719, 720	
\ProvidesFile 1327	
\ProvidesPackage 3	
Q	
quotation (env.) 153	
quote (env.) 153	
R	
\raggedleft 1066	
\raggedright 1067	
\relax 574, 576	
\RemoveFromHook 1121, 1152	
\renewcommand 27, 198	
\RenewDocumentCommand 685	
\RenewDocumentEnvironment 143, 146, 149, 154, 157, 162, 167, 202, 205, 208, 213, 241	
\RequirePackage 6, 7, 1330	
\rightmargin 6, 62, 227, 404, 455, 855, 868, 978	
\rightskip 383, 392, 435	
\rlap 1096	
S	
scan commands:	
\scan_stop: 1171	
\setbox 21	
\setcounter 340, 942	
\setlength 855, 856, 868	
\SetTemplateKeys 280, 391, 409, 520, 583	
skip commands:	
\skip_add:Nn 416, 433	
\skip_eval:n 477	
\skip_horizontal:n 464, 466, 621, 623	
\skip_new:N 681, 682, 683	
\skip_set:Nn 107, 392, 413	
\skip_set_eq:NN 431, 435, 436, 452, 453, 628	
\skip_vertical:n 366, 367, 471, 472	
\skip_zero:N 434	
\l_tmpa_skip 363, 364, 366, 367	
\space 4, 1114, 1328	
\strut 8, 643	
T	
tag commands:	
\tag_if_active:TF 111, 111, 112, 195, 221, 251, 310, 819, 872, 1011, 1071, 1149, 1263	
\tag_mc_begin:n 1095, 1145, 1238, 1242, 1290	
\tag_mc_end: 1091, 1097, 1159, 1240	
\l_tag_para_attr_class_tl 388, 1142, 1234	
\tag_struct_begin:n 1134, 1139, 1228, 1232	
\tag_struct_end: 1099, 1116, 1161, 1165, 1215	
tag internal commands:	
__tag_check_para_begin_show:nn 1144	
__tag_check_para_end_show:nn . 1160	
\l__tag_L_attr_class_tl 234, 236, 237, 749, 750, 1261, 1262, 1279	
\l__tag_L_tag_tl 746, 747, 1258, 1259, 1278	
\g__tag_mode_lua_bool 196	
\g__tag_para_begin_int 1137, 1230, 1239	
\l__tag_para_bool 1103, 1123, 1155, 1210, 1222	
\g__tag_para_end_int 1092, 1096, 1157	
\l__tag_para_flattened_bool 274, 287, 1109, 1111, 1131, 1162, 1212, 1224	
\g__tag_para_main_begin_int 1084, 1133, 1227	
\g__tag_para_main_end_int 1115, 1164, 1214, 1252, 1308, 1317	
\l__tag_para_main_tag_tl 188, 1085, 1114, 1134, 1228	
\l__tag_para_show_bool 1094, 1236	

<code>\l__tag_para_tag_tl</code> ..	189, 1141, 1233	<code>\@makeother</code>	185
<code>\tagmcbegin</code>	1294	<code>\@mklab</code>	231
<code>\tagmccend</code>	1298	<code>\@nmbrlistfalse</code>	531
<code>\tagpdfparaOff</code>	1100	<code>\@nmbrlisttrue</code>	534
<code>\tagpdfparaOn</code>	1100	<code>\@noitemerr</code>	352, 429, 444
<code>\tagpdfsetup</code>	222, 252, 820, 821, 873, 874, 875, 876, 1013, 1150, 1265	<code>\@noligs</code>	186
<code>\tagstructbegin</code> 1085, 1246, 1276, 1282, 1293, 1302	<code>\@normalcr</code>	6, 75, 1062
<code>\tagstructend</code>	1254, 1256, 1301, 1309, 1312, 1318, 1321, 1322	<code>\@opargbegintheorem</code>	251
<code>\tagtool</code>	189	<code>\@outerparskip</code>	367, 452, 472, 477
templates:		<code>\@restorepar</code>	14
<code>block display</code>	51, 394	<code>\@rightskip</code>	392, 435
<code>blockenv display</code>	34, 260	<code>\@setpar</code>	438
<code>item std</code>	92, 559	<code>\@setupverbinvisible space</code> ..	190, 194
<code>list std</code>	78, 502	<code>\@setupverbvisible space</code>	169
<code>para std</code>	66, 378	<code>\@sxverbatim</code>	170
\TeX and \LaTeX 2 ϵ commands:		<code>\@tempwafalse</code>	177
<code>\@@par</code>	180, 183	<code>\@tempwattrue</code>	182
<code>\@beginparpenalty</code>	6, 402, 479	<code>\@toodeep</code>	297, 304
<code>\@begintheorem</code>	251	<code>\@topsep</code>	32, 50
<code>\@centercr</code>	1029, 1040, 1051	<code>\@topsepadd</code>	32, 50
<code>\@clubpenalty</code>	15, 676	<code>\@totalleftmargin</code>	456, 457
<code>\@currenvir</code>	357, 1200	<code>\@vobeyspaces</code>	169, 191
<code>\@currenvline</code>	1201	<code>\@xobeysp</code>	198
<code>\@doendpe</code>	11, 11	<code>\@xverbatim</code>	164
<code>\@eha</code>	1199	<code>\g_block_nesting_depth_int</code>	52
<code>\@endparpenalty</code>	6, 369, 403	<code>\c@maxblocklevels</code>	303, 339
<code>\@endpefalse</code>	17, 23, 28, 1217	<code>\everypar</code>	32
<code>\@endpetrue</code>	11, 27	<code>\if@endpe</code>	27, 28, 1209
<code>\@enumdepth</code>	913	<code>\if@tempswa</code>	179
<code>\@execute@begin@hook</code>	1202	<code>\iitem</code>	51
<code>\@flushglue</code>	6, 72, 436, 1025, 1026, 1037, 1047, 1060	<code>\item</code>	30, 52
<code>\@ifundefined</code>	1198	<code>\l@nohyphenation</code>	176
<code>\@ignorefalse</code>	1204	<code>\labelwidth</code>	28, 30, 50
<code>\@inmatherr</code>	357, 687	<code>\makelabel</code>	30, 52
<code>\@itemdepth</code>	897	<code>\newline</code>	28
<code>\@itemlabel</code>	17, 26, 27, 215, 235, 500, 544, 589, 1288, 1300	<code>\on@line</code>	342, 658, 1075, 1093, 1110, 1114, 1128, 1138, 1158, 1201, 1231, 1249, 1253, 1310, 1319
<code>\@itempenalty</code>	7, 511, 709	<code>\par</code>	32, 51
<code>\@kernel@after@para@after</code>	1183	<code>\par@deathcycles</code>	437, 442, 443
<code>\@kernel@after@para@end</code>	1176	<code>\partopsep</code>	50
<code>\@kernel@refstepcounter</code>	586	<code>\ref</code>	50
<code>\@labels</code>	30	<code>\reserved@a</code>	1199, 1200, 1207
<code>\@latex@error</code>	1199	<code>\strut</code>	30
<code>\@list...</code>	5	<code>\topsep</code>	50
<code>\@listctr</code>	26, 27, 229, 500, 523, 527, 535, 538, 586, 589, 590	<code>\verbatim@font</code>	186
<code>\@listdepth</code>	5, 19, 258	<code>\z@</code>	21
<code>\@listi</code>	5	<code>\z@skip</code>	1027, 1036, 1038, 1048, 1049, 1058, 1059
<code>\@listii</code>	5	tex commands:	
<code>\@listvi</code>	5	<code>\tex_hskip:D</code>	1179
		<code>\tex_lastnodetype:D</code>	1178
		<code>\tex_lastskip:D</code>	107
		<code>\tex_par:D</code>	1181, 1190

