Prototype reimplementation of LaTeX 2$_\varepsilon$'s block environments using templates

LaTeX Project[*]

v0.8p 2024-08-11

Abstract

Contents

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

1   Introduction

The list implementation in LaTeX 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, prohibitying or encuraging line breaks, and so forth.

2.1.2   The object type 'para'

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

Semantics:

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

2.1.3 The object type 'list'

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

Semantics:

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

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

2.1.4 The object type 'item'

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

Semantics:

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

2.1.5 The object type 'blockenv'

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

Semantics:

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

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

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

2.2 Templates

2.2.1 The `blockenv` template 'display'

Attributes:

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

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

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

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

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

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

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

para-instance (*tokenlist*)

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

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

inner-instance-type (*tokenlist*) 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 LaTeX 2ε name).

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

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

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

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

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

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

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

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

2.2.2 The block template 'display'

Attributes:

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

beginsep (*skip*)   Default: \topsep

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

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

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

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

beginpenalty (*integer*)   Default: \@beginparpenalty

endpenalty (*integer*)   Default: \@endparpenalty

leftmargin (*length*)   Default: \leftmargin

rightmargin (*length*)   Default: \rightmargin

parindent (*length*)   Default: \listparindent

Semantics & Comments: The idea of a heading key needs some further thoughts. Maybe instead the object type should accept a second argument and receive input for such a 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 (*tokenlist*) Counter name to be used in a numbered list or empty, if the list is unnumbered

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

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

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

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

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

> May need to be on a different template level

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

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

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

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

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

2.2.5 The `item` template 'std'

Attributes:

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

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

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

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

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

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

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

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

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

text-font (*tokenlist*) *unused*

compatibility (*boolean*)   Default: `true`

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

3   Tagging support

3.1   Paragraph tags

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

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

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

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

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

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

```
<text-unit>
  <text>
    The paragraph text before the display element ...
  </text>
  <display element structure>
    Content of the display structure 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>
        <Lbl> label </Lbl>
        <LBody>
          The text of the first item involving <text-unit> as necessary ...
        </LBody>
      </LI>
      <LI>
        The second item ...
      </LI>
      ... further items ...
    </itemize>
  </text-unit>
```

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

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

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

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

will be tagged as follows:

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

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

3.2 Tagging recipes

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

standalone This recipe does the following:

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

basic This recipe does the following:

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

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

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

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

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

4 Debugging

`\DebugBlocksOn`
`\DebugBlocksOff`
`\block_debug_on:`
`\block_debug_off:`

These commands enable/disable debugging messages.

5 New and redefined kernel command

`\@doendpe`   The original LaTeX 2$_\varepsilon$ command is augmented to allow for tagging.

`\legacyverbatimsetup`   *to be documented*
`\legacylistsetupcode`

`\@setupverbinvisiblespace`   A counterpart definition to the kernel command `\@setupverbinvisiblespace`, needed as we need to handle real space chars in verbatim.

endblockenv   *to be documented*
`\g_block_nesting_depth_int`

`\newtheorem`   Redefined to make theorems tagging aware.
`\@thm`
`\@begintheorem`

`\item`   The `\item` is redefined.
`\@itemlabel`

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

\begin The \begin is slightly redefine to handle \@doendpe better. TODO: move to kernel

\para_end: TODO: consider name, document

para/begin The para/begin hook is enhanced to support list ends

6 The Implementation

```
1 ⟨*package⟩
2 ⟨@@=block⟩

3 \ProvidesPackage {latex-lab-testphase-block}
4                  [\ltlabblockdate\space v\ltlabblockversion\space
5                        blockenv implementation]
```

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

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

7 \ExplSyntaxOn
```

UFi:this variable(s) must be declared:

```
8 \tl_new:N \l__block_item_align_tl
9 \tl_new:N\l__block_legacy_env_params_tl
```

6.1 Handling \par after the end of the list

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

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

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

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

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

```
15        \__kernel_displayblock_doendpe:
```

12

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.

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

By default we don't do any tagging:

```
25  \cs_new_eq:NN \__kernel_displayblock_doendpe: \prg_do_nothing:
```

The flag itself should be set globally not locally.

```
26  \def\@endpetrue {%
27    \global\let\if@endpe\iftrue
28  % if called inside a group propagate to the outside
29    \ifnum\currentgrouplevel>\z@
30        \aftergroup\propagate@doendpe
31    \fi
32  }
```

If `\if@endpe` is still true run `\@doendpe` that in turn runs another `\@endpetrue` (besides other things), thus propagating further, if necessary However, if the endpe situation got resolved nothing further happens

```
33  \def\propagate@doendpe{%
34    \if@endpe
35        \__block_debug_typeout:n{run~ (another)~ @doendpe~ at~
36          group~ level~ \the\currentgrouplevel}%
37      \@doendpe
38    \else
39        \__block_debug_typeout:n{do~ not~ run~ (another)~ @doendpe}%
40    \fi
41  }
```

With the above approach we do not need the propagation in `\end...` any longer (since this is now handled by `\aftergroup\propagate@doendpe`) so here a simple `\endgroup` is enough. This replaces the definition in ...:

```
42  \@namedef{end }#1{%
43    \romannumeral
44      \IfHookEmptyTF{env/#1/end}%
45          {\expandafter\z@}%
46          {\z@\UseHook{env/#1/end}}%
47      \csname end#1\endcsname\@checkend{#1}%
48  %   \expandafter\endgroup\if@endpe\@doendpe\fi
49      \endgroup
50      \UseHook{env/#1/after}%
51      \if@ignore\@ignorefalse\ignorespaces\fi
52  }
```

```
53  \def\@endpefalse{\global\let\if@endpe\iffalse}
```

(*End of definition for* `\@doendpe`. *This function is documented on page* *11*.)

verify that this claim is actually correct!

6.2 Object and template interfaces

blockenv (*objecttype*)
block (*objecttype*)
para (*objecttype*)
list (*objecttype*)
item (*objecttype*)

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

```
54 \NewTemplateType{blockenv}{1}
55 \NewTemplateType{block}{1}
56 \NewTemplateType{para}{1}
57 \NewTemplateType{list}{1}
58 \NewTemplateType{item}{1}
```

blockenv display (*templ.*)

```
59 \DeclareTemplateInterface{blockenv}{display}{1}
60 {
61   env-name        : tokenlist ,
62   tag-name        : tokenlist ,
63   tag-class       : tokenlist ,
64   tagging-recipe : tokenlist = standard,
65   level-increase : boolean = true ,
66   setup-code      : tokenlist ,
67   block-instance : tokenlist = displayblock ,
68   para-instance  : tokenlist ,
69   inner-level-counter : tokenlist,
70   max-inner-levels    : tokenlist = 4,
71   inner-instance-type : tokenlist = list ,
72   inner-instance      : tokenlist ,
73   para-flattened : boolean = false ,
74   final-code      : tokenlist = \ignorespaces ,
75 }
```

block display (*templ.*)

```
76 \DeclareTemplateInterface{block}{display}{1}
77 {
78   heading       : tokenlist = ,                          %??
79   beginsep      : skip = \topsep ,
80   begin-par-skip : skip = \partopsep ,
81   par-skip      : skip = \parsep ,
82   end-skip      : skip = \KeyValue{beginsep} ,           % conflict with name below
83   end-par-skip  : skip = \KeyValue{begin-par-skip} ,
84   beginpenalty  : integer = \UseName{@beginparpenalty} ,
85   endpenalty    : integer = \UseName{@endparpenalty} ,
86   leftmargin    : length = \leftmargin ,
87   rightmargin   : length = \rightmargin ,
88   parindent     : length = \listparindent ,
89 %  font          : tokenlist         % maybe add? (or more general for fonts and color)
90 }
```

para std (*templ.*)

```
91 \DeclareTemplateInterface{para}{std}{1}
92 {
93   indent-width         : length = \parindent ,
94   start-skip           : skip = 0pt  ,
95   left-skip            : skip = 0pt ,
96   right-skip           : skip = 0pt ,
```

14

```
97   end-skip              : skip = \@flushglue ,
98   fixed-word-spaces     : boolean = false ,
99   final-hyphen-demerits : integer = 5000 ,
100  cr-cmd                : tokenlist = \@normalcr ,
101  para-class            : tokenlist = justify ,
102 }
```

list std (*templ.*)

```
103 \DeclareTemplateInterface{list}{std}{1}      % optional
104 {
105   counter        : tokenlist = ,
106   item-label     : tokenlist = ,
107   start          : integer = 1 ,
108   resume         : boolean = false ,
109   item-instance  : instance{item} = basic ,
110   item-skip      : skip = \itemsep ,
111   item-penalty   : integer = \UseName{@itempenalty} ,
112   item-indent    : length = \itemindent ,
113   label-width    : length = \labelwidth ,
114   label-sep      : length = \labelsep ,
115   legacy-support : boolean = false ,
116 }
```

item std (*templ.*)

```
117 \DeclareTemplateInterface{item}{std}{1}
118   {
119     counter-label : function{1} = \arabic{#1} ,
120     counter-ref   : function{1} = \KeyValue{counter-label} ,
121     label-ref     : function{1} = #1 ,
122     label-autoref : function{1} = item~#1 ,
123     label-format  : function{1} = #1 ,
124     label-strut   : boolean = false ,
125     label-align   : choice {left,center,right,parleft} = right ,
126     label-boxed   : boolean = true ,
127     next-line     : boolean = false ,
128     text-font     : tokenlist ,
129     compatibility : boolean = true ,
130   }
```

6.3   Useful helper commands

This section collects expl3 commands that will be useful.

__block_skip_set_to_last:N
__block_skip_remove_last:

Set a skip register to the value of an immediately preceding skip or zero if there was none.

```
131 \cs_new_protected:Npn \__block_skip_set_to_last:N #1 {
132   \skip_set:Nn #1 { \tex_lastskip:D }
133 }
```

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

```
134 \cs_new_eq:NN \__block_skip_remove_last: \tex_unskip:D
```

(*End of definition for* __block_skip_set_to_last:N *and* __block_skip_remove_last:*.*)

```
135 \cs_generate_variant:Nn \tl_if_novalue:nTF { o }
```

15

6.3.1 Debugging

\g__block_debug_bool

```
136 \bool_new:N \g__block_debug_bool
```

(*End of definition for* \g__block_debug_bool.)

__block_debug:n
__block_debug_typeout:n

```
137 \cs_new_eq:NN \__block_debug:n \use_none:n
138 \cs_new_eq:NN \__block_debug_typeout:n \use_none:n
```

(*End of definition for* __block_debug:n *and* __block_debug_typeout:n.)

\block_debug_on:
\block_debug_off:
__block_debug_gset:

```
139 \cs_new_protected:Npn \block_debug_on:
140   {
141     \bool_gset_true:N \g__block_debug_bool
142     \__block_debug_gset:
143   }
144 \cs_new_protected:Npn \block_debug_off:
145   {
146     \bool_gset_false:N \g__block_debug_bool
147     \__block_debug_gset:
148   }
149 \cs_new_protected:Npn \__block_debug_gset:
150   {
151     \cs_gset_protected:Npx \__block_debug:n ##1
152       { \bool_if:NT \g__block_debug_bool {##1} }
153     \cs_gset_protected:Npx \__block_debug_typeout:n ##1
154       { \bool_if:NT \g__block_debug_bool { \typeout{==>~ ##1} } }
155   }
```

(*End of definition for* \block_debug_on: , \block_debug_off: , *and* __block_debug_gset: . *These functions are documented on page* 11.)

\DebugBlocksOn
\DebugBlocksOff

```
156 \cs_new_protected:Npn \DebugBlocksOn  { \block_debug_on:  }
157 \cs_new_protected:Npn \DebugBlocksOff { \block_debug_off: }
```

```
158 \DebugBlocksOff
```

(*End of definition for* \DebugBlocksOn *and* \DebugBlocksOff. *These functions are documented on page* 11.)

6.4 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.

6.4.1 Displayblock environments

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

displayblock (*env.*)

```
159 \NewDocumentEnvironment{displayblock}{ !O{} }
160   { \UseInstance{blockenv}{displayblock} {#1} }
161   { \endblockenv }
```

displayblockflattened (*env.*)

```
162 \NewDocumentEnvironment{displayblockflattened}{ !O{} }
163   { \UseInstance{blockenv}{displayblockflattened} {#1} }
164   { \endblockenv }
```

center (*env.*)
flushleft (*env.*)
flushright (*env.*)

```
165 \AddToHook{begindocument/before}{
166   \RenewDocumentEnvironment{center} { !O{} }
167   { \UseInstance{blockenv}{center}{#1} }
168   { \endblockenv }
169   \RenewDocumentEnvironment{flushright} { !O{} }
170   { \UseInstance{blockenv}{flushright}{#1} }
171   { \endblockenv }
172   \RenewDocumentEnvironment{flushleft} { !O{} }
173   { \UseInstance{blockenv}{flushleft}{#1} }
174   { \endblockenv }
175 }
```

6.4.2 Display quote environments

quote (*env.*)
quotation (*env.*)

```
176 \AddToHook{begindocument/before}{
177   \RenewDocumentEnvironment{quote}{ !O{} }
178     { \UseInstance{blockenv}{quote} {#1} }
179     { \endblockenv }
180   \RenewDocumentEnvironment{quotation}{ !O{} }
181     { \UseInstance{blockenv}{quotation} {#1} }
182     { \endblockenv }
183 }
```

6.4.3 Verbatim environments

verbatim (*env.*)
verbatim* (*env.*)

```
184 \AddToHook{begindocument/before}{
185   \RenewDocumentEnvironment{verbatim}{ !O{} }
186     { \UseInstance{blockenv}{verbatim} {#1}
```

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

```
187         \@setupverbinvisiblespace\frenchspacing\@vobeyspaces
188         \@xverbatim
189       }
190     { \endblockenv }
```

```
191   \RenewDocumentEnvironment{verbatim*}{ !O{} }
192     { \UseInstance{blockenv}{verbatim} {#1}
193       \@setupverbvisiblespace\frenchspacing\@vobeyspaces
194       \@sxverbatim
195     }
196     { \endblockenv }
197 }
```

Helper commands for verbatim

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

```
198 ⟨@@=⟩
199 \def\legacyverbatimsetup{%
200   \language\l@nohyphenation
201   \@tempswafalse
202   \def\par{%
203     \if@tempswa
204       \leavevmode \null {\@@par}\penalty\interlinepenalty
205     \else
206       \@tempswatrue
207       \ifhmode{\@@par}\penalty\interlinepenalty\fi
208     \fi}%
209   \let\do\@makeother \dospecials
210   \obeylines \verbatim@font \@noligs
211   \everypar \expandafter{\the\everypar \unpenalty}%
212   \tl_set:Nn \l__tag_para_main_tag_tl {codeline}
213   \tagtool{paratag=Code}%   oder faster: \tl_set:Nn\l__tag_para_tag_tl{Code}
214 }
215 ⟨@@=block⟩
```

(*End of definition for* \legacyverbatimsetup. *This function is documented on page 11.*)

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

```
216 \newcommand\@setupverbinvisiblespace{}
217 \tag_if_active:T {
218   \bool_if:NF\g__tag_mode_lua_bool
219     {
220       \renewcommand\@setupverbinvisiblespace{\def\@xobeysp{\nobreakspace\pdffakespace}}
221     }
222 }
```

(*End of definition for* \@setupverbinvisiblespace. *This function is documented on page 11.*)

6.4.4 Standard list environments

itemize (*env.*) For the standard lists everything is managed by the blockenv instance.
enumerate (*env.*)
description (*env.*)
```
223 \AddToHook{begindocument/before}{
224   \RenewDocumentEnvironment{itemize}{!O{}}
```

18

```
225      { \UseInstance{blockenv}{itemize} {#1} }
226      { \endblockenv }

227    \RenewDocumentEnvironment{enumerate}{!O{}}
228      { \UseInstance{blockenv}{enumerate} {#1} }
229      { \endblockenv }

230    \RenewDocumentEnvironment{description}{!O{}}
231      { \UseInstance{blockenv}{description} {#1} }
232      { \endblockenv }
233  }
```

6.4.5 verse environment

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

```
234  \AddToHook{begindocument/before}{
235    \RenewDocumentEnvironment{verse}{ !O{} }
236      {
237        \let\\\@centercr
238        \UseInstance{blockenv}{list}
239          {
240            item-indent=-1.5em,
241            parindent=-1.5em,
242            item-skip=0pt,
243            rightmargin=\leftmargin,
244            leftmargin=\leftmargin+1.5em,
245            #1
246          }
247        \item\relax
248      }
249      { \endblockenv }
250    }
```

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

```
251  \AddToHook{begindocument/before}{
252    \RenewDocumentEnvironment{list}{O{} m m }
253      {
```

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.

```
254        \tl_set:Nn \@itemlabel {#2}
255        \tl_set:Nn \l__block_legacy_env_params_tl {#3}

256        \UseInstance{blockenv}{list} {#1}
257      }
258      { \endblockenv }
259  }
```

`\l__block_env_params_tl` Declare the variable for the parameter argument; `\@itemlabel` is already declared in LaTeX 2ε.

```
260  \tl_new:N \l__block_env_params_tl
```

(*End of definition for* `\l__block_env_params_tl`.)

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

```
261 \cs_new:Npn \legacylistsetupcode {
```

Reset values to defaults:

```
262     \dim_zero:N \listparindent
263     \dim_zero:N \rightmargin
264     \dim_zero:N \itemindent
```

By default a `list` environment is not numbered:

```
265     \tl_set:Nn \@listctr {}
266     \legacy_if_set_false:n { @nmbrlist } % 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.

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

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

```
268     \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.

```
269     \legacy_if:nTF { @nmbrlist }
270       { \tl_set:Nn \l__tag_L_attr_class_tl {enumerate} }   % numbered list
271       { \tl_if_empty:NTF \@itemlabel
272         { \tl_set:Nn \l__tag_L_attr_class_tl {list}     } % no label
273         { \tl_set:Nn \l__tag_L_attr_class_tl {itemize} } % unnumbered, unordered
274       }
275 }
```

(*End of definition for* `\legacylistsetupcode`. *This function is documented on page 11.*)

trivlist (*env.*)

```
276 \AddToHook{begindocument/before}{
277   \RenewDocumentEnvironment{trivlist}{ !O{} }
278                           { \list[#1]{}
279                             {
280                               \dim_zero:N \leftmargin
281                               \dim_zero:N \labelwidth
282                               \cs_set_eq:NN \makelabel \use:n
283                             }
284                           }
285     { \endblockenv }
286 }
```

6.4.6 Theorem-like environments

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

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

```
287 \RenewDocumentCommand \newtheorem { m O{#1} m o }
288 {
289   \expandafter\@ifdefinable\csname #1\endcsname
290     {
291       \str_if_eq:nnTF{#1}{#2}
292         {
293           \@definecounter {#2}
294           \IfNoValueTF {#4}
295             {  % @ynthm
296               \tl_gset:ce { the #2 }
297                 {
298                   \@thmcounter{#2}
299                 }
300             }
301             {  % @xnthm
302               \@newctr{#1}[#4]
303               \tl_gset:ce { the #2 }
304                 {
305                   \expandafter\noexpand\csname the#4\endcsname
306                   \@thmcountersep
307                   \@thmcounter{#2}
308                 }
309             }
310         }
311         {  % @othm
312           \@ifundefined{c@#2}
313             { \@nocounterr{#2} }
314             {
315               \tl_gset:cn { the #1 }
316                 { \UseName { the #2 } }
317             }
318         }
319       \global\@namedef{#1}    { \@thm{#2}{#3} }
320       \global\@namedef{end#1}{ \@endtheorem  }
321     }
322 }
```

(*End of definition for* \newtheorem. *This function is documented on page 11.*)

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

```
323 \tl_new:N \l__block_thm_current_counter_tl
324 \def\@thm#1#2{%
325   \@kernel@refstepcounter{#1}
326   \tl_set:Nn \l__block_thm_current_counter_tl{#1}
327   \@ifnextchar[{\@ythm{#1}{#2}}{\@xthm{#1}{#2}}}
```

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

```
328 \def\hyper@nopatch@thm{}
```

`\@begintheorem`
`\@opargbegintheorem`

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

```
329 \def\@begintheorem#1#2{
330   \UseInstance{blockenv}{theorem}{}
331   \tagpdfparaOff
332   \mode_leave_vertical:
333   \MakeLinkTarget{\l__block_thm_current_counter_tl}
334   \tag_struct_begin:n{tag=Caption}
335   \group_begin:
336   \bfseries
337   \tag_mc_begin:n {}
338     #1\
339   \tag_mc_end:
340     \tag_struct_begin:n{tag=Lbl}
341       \tag_mc_begin:n {}
342         #2
343       \tag_mc_end:
344     \tag_struct_end:
345     \group_end:
346   \tag_struct_end:
347   \tagpdfparaOn

348   \__block_start_para_structure_unconditionally:n { \PARALABEL }

349   \itshape
350   \hskip\labelsep
351   \ignorespaces
352 }
353 \def\@opargbegintheorem#1#2#3{
354   \UseInstance{blockenv}{theorem}{}
355   \tagpdfparaOff
356   \mode_leave_vertical:
357   \MakeLinkTarget{\l__block_thm_current_counter_tl}
358   \tag_struct_begin:n{tag=Caption}
359   \group_begin:
360   \bfseries
361   \tag_mc_begin:n {}
362     #1\
363   \tag_mc_end:
364   \tag_struct_begin:n{tag=Lbl}
365     \tag_mc_begin:n {}
366       #2
367     \tag_mc_end:
368   \tag_struct_end:
369     \tag_mc_begin:n {}
370     \ (#3)
371     \tag_mc_end:
372   \group_end:
373   \tag_struct_end:
374   \tagpdfparaOn
```

22

```
375    \__block_start_para_structure_unconditionally:n { \PARALABEL }
376    \itshape
377    \hskip\labelsep
378    \ignorespaces
379  }
380  \def\@endtheorem{\endblockenv}
```

(*End of definition for* \@begintheorem *and* \@opargbegintheorem*. These functions are documented on page 11.*)

6.5   Implementation of templates

6.5.1   Implementation of blockenv templates . . .

\g_block_nesting_depth_int   LaTeX 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:

```
381  \cs_new:Npn \g_block_nesting_depth_int { \@listdepth }  % a fake int
382                                                          % for now
```

(*End of definition for* \g_block_nesting_depth_int*. This function is documented on page 11.*)

blockenv display (*templ.*)

```
383  \DeclareTemplateCode{blockenv}{display}{1}
384  {
385    env-name        = \l__block_env_name_tl ,
386    tag-name        = \l__block_tag_name_tl ,
387    tag-class       = \l__block_tag_class_tl ,
388    tagging-recipe = \l__block_tagging_recipe_tl ,
389    level-increase = \l__block_level_incr_bool ,
390    setup-code      = \l__block_setup_code_tl ,
391    block-instance = \l__block_block_instance_tl ,
392    para-instance  = \l__block_para_instance_tl ,
393    inner-level-counter = \l__block_inner_level_counter_tl ,
394    max-inner-levels    = \l__block_max_inner_levels_tl ,
395    inner-instance-type = \l__block_inner_instance_type_tl ,
396    inner-instance      = \l__block_inner_instance_tl ,
397    para-flattened = \l__tag_para_flattened_bool ,
398    final-code      = \l__block_final_code_tl ,
399  }
400  {
401    \__block_debug_typeout:n{\l__block_env_name_tl -env-start}
402  %
403    \tl_if_empty:nF {#1} { \SetTemplateKeys{blockenv}{display}{#1} }
404  %
```

We need to know later if we have nested blockenvs inside a flattened environment. Whenever we start a new blockenv we increment \l__tag_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.

23

```
405    \int_compare:nNnTF \l__tag_block_flattened_level_int > 0
406        {
407          \int_incr:N \l__tag_block_flattened_level_int
408        }
409        {
410          \bool_if:NT \l__tag_para_flattened_bool
411              {
412                \int_incr:N \l__tag_block_flattened_level_int
413              }
414        }
415 %
416    \tl_if_empty:NF \l__block_inner_level_counter_tl
417        {
418          \int_compare:nNnTF  \l__block_inner_level_counter_tl >
419                             { \l__block_max_inner_levels_tl - 1 }
420            { \@toodeep }
421            { \int_incr:N \l__block_inner_level_counter_tl }  % not clean "o"?
422        }
```

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.

```
423    \bool_if:NT \l__block_level_incr_bool
424        {
425          \int_compare:nNnTF  \g_block_nesting_depth_int >
426                             { \c@maxblocklevels - 1 }
427            { \@toodeep }
428            {
429              \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.

```
430              \use:c { @list \int_to_roman:n { \g_block_nesting_depth_int } }
431            }
432        }
```

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

```
433    \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.

```
434    \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).

```
435    \__block_debug_typeout:n{use~ instance:~
436            \l__block_block_instance_tl - \int_use:N \g_block_nesting_depth_int }
437    \UseInstance{block}
438            { \l__block_block_instance_tl - \int_use:N
439              \g_block_nesting_depth_int }
440            {#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).

```
441    \tl_if_empty:NF \l__block_para_instance_tl
442        {
```

24

```
443        \__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.

```
444        \UseInstance{para}{ \l__block_para_instance_tl } {}
445      }
```

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.

```
446    \tl_if_empty:NF \l__block_inner_instance_tl
447      {
448        \__block_debug_typeout:n{use~ instance:~ \l__block_inner_instance_tl
449              \tl_if_empty:NF \l__block_inner_level_counter_tl
450                    { - \int_use:N \l__block_inner_level_counter_tl }}
451        \UseInstance{ \l__block_inner_instance_type_tl }
452              { \l__block_inner_instance_tl
453                \tl_if_empty:NF \l__block_inner_level_counter_tl
454                    { - \int_use:N \l__block_inner_level_counter_tl }  % not clean
455                                                % use "o"?
456              }
457              {#1}
458      }
```

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.

```
459    \l__block_final_code_tl
460 }
```

\l__tag_block_flattened_level_int Count the levels of nested blockenvs starting with the first that is "flattened". The counter is defined in lttagging.dtx, but until the next release 11/24 we set it up here too

```
461 \int_if_exist:NF \l__tag_block_flattened_level_int
462   {
463     \int_new:N \l__tag_block_flattened_level_int
464   }
```

(*End of definition for* `\l__tag_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.

```
465 \newcounter{maxblocklevels}
466 \setcounter{maxblocklevels}{6}
```

(*End of definition for* `\c@maxblocklevels`. *This function is documented on page 12.*)

\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.

> name is bad

```
467 \cs_new:Npn \endblockenv {
468    \__block_debug_typeout:n{blockenv~ common~ ending \on@line}
```

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

```
469    \bool_if:NT \l__block_level_incr_bool
470      { \int_gdecr:N \g_block_nesting_depth_int }
```

25

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

```
471    \legacy_if:nT { @inlabel }
472      {
473        \mode_leave_vertical:
474        \legacy_if_gset_false:n { @inlabel }
475      }
```

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)

```
476    \legacy_if:nT { @newlist }
477      {
478        \@noitemerr
479        \legacy_if_gset_false:n { @newlist }
480      }
481    \mode_if_horizontal:TF
482      { \__block_skip_remove_last: \__block_skip_remove_last: \par }
483      { \@inmatherr{\end{\@currenvir}} }
```

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

```
484    \__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 LATEX 2ε list environment have been doing.

```
485  %    \__block_debug_typeout:n{@noparlist =
486  %                    \legacy_if:nTF { @noparlist }{true}{false}}
487    \legacy_if:nF { @noparlist }
488      {
489        \__block_skip_set_to_last:N \l_tmpa_skip
490        \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
491          {
492            \skip_vertical:n { - \l_tmpa_skip }
493            \skip_vertical:n { \l_tmpa_skip + \parskip - \@outerparskip }
494          }
495        \addpenalty \@endparpenalty
496        \addvspace \l__block_topsepadd_skip
```

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

```
497  %        \legacy_if_gset_true:n { @endpe }
498      }
```

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

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

```
499    \socket_use:n {tagsupport/block-endpe}
500  }
```

26

(End of definition for \endblockenv. This function is documented on page 11.)

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

```
501 \cs_new:Npn \__kernel_displayblock_end: {
502   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_end:}}
503 }
```

(End of definition for __kernel_displayblock_end:.)

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

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

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

```
505 \socket_new_plug:nnn{tagsupport/block-endpe}{on}
```

We can't use \legacy_if_gset_true:n because this is now doing more than setting the legacy switch

```
506                   { \@endpetrue }
507 \socket_new_plug:nnn{tagsupport/block-endpe}{off}
508                   { \@endpefalse }
509 \socket_assign_plug:nn{tagsupport/block-endpe}{on}
```

6.5.2 Implementation of para templates . . .

para std *(templ.)*

```
510 \DeclareTemplateCode{para}{std}{1}
511 {
512   indent-width       = \parindent ,
513   start-skip         = \l__par_start_skip ,              % name??
514   left-skip          = \leftskip   ,
515   right-skip         = \rightskip  ,
516   end-skip           = \parfillskip ,
517   fixed-word-spaces = \l__par_fixed_word_spaces_bool ,  % name??
518   final-hyphen-demerits = \finalhyphendemerits ,
519   cr-cmd             = \\ ,
520   para-class         = \l__tag_para_attr_class_tl ,
521 }
522 {
523   \tl_if_empty:nF {#1} { \SetTemplateKeys{para}{std}{#1} }
524   \skip_set:Nn \@rightskip \rightskip
525 }
```

6.5.3 Implementation of block templates . . .

block display *(templ.)*

```
526 \DeclareTemplateCode{block}{display}{1}
527 {
528   heading         = \l__block_heading_tl ,
529   beginsep        = \topsep ,
530   begin-par-skip  = \partopsep ,
```

27

```
531    par-skip        = \parsep ,
532    end-skip        = \l__block_botsep_skip ,
533    end-par-skip    = \l__block_parbotsep_skip ,
534    beginpenalty    = \@beginparpenalty ,
535    endpenalty      = \@endparpenalty ,
536    rightmargin     = \rightmargin ,
537    leftmargin      = \leftmargin ,
538    parindent       = \listparindent ,
539  }
540  {
541    \tl_if_empty:nF {#1} { \SetTemplateKeys{block}{display}{#1} }

542      \tl_if_blank:oF \l__block_heading_tl
543        { \mode_leave_vertical: \textbf{\l__block_heading_tl} } % TODO customize
```

generalize heading usage (or drop?)

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

```
544    \legacy_if:nT { @noskipsec } { \mode_leave_vertical: }
545    \skip_set:Nn \l__block_topsepadd_skip { \topsep }
546    \mode_if_vertical:TF
547      {
548        \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.

```
549        \__kernel_displayblock_beginpar_vmode:
550      }
551      {
```

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`.

```
552        \__block_skip_remove_last: \__block_skip_remove_last:
553        \__kernel_displayblock_beginpar_hmode:w \par
554      }
```

Now we are back to legacy list implementation . . .

```
555    \legacy_if:nTF { @inlabel }
556      {
557        \legacy_if_set_true:n { @noparitem }
558        \legacy_if_set_true:n { @noparlist }
559      }
560      {
561        \legacy_if:nT { @newlist } { \@noitemerr }
562        \legacy_if_set_false:n { @noparlist }
563        \skip_set_eq:NN \l__block_effective_top_skip \l__block_topsepadd_skip
564      }
565    \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*.

```
566        \skip_zero:N \leftskip
567        \skip_set_eq:NN \rightskip \@rightskip
568        \skip_set_eq:NN \parfillskip \@flushglue
```

The next lines establish a parshape which is retained across paragraphs be 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.

```
569        \int_zero:N \par@deathcycles
570        \@setpar
571          {
572            \legacy_if:nTF { @newlist }
573              {
574                \int_incr:N \par@deathcycles
575                \int_compare:nNnTF \par@deathcycles > { 1000 }
576                    { \@noitemerr
577                      { \para_end: }
578                    }
579              }
580              {
581                { \para_end: }
582              }
583          }
584        \skip_set_eq:NN \@outerparskip \parskip
585        \skip_set_eq:NN \parskip \parsep
586        \dim_set_eq:NN \parindent \listparindent
587        \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
588        \dim_add:Nn \@totalleftmargin { \leftmargin }
589        \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.

```
590        \__kernel_displayblock_begin:
```

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

```
591        \legacy_if:nTF { @noparitem }
592          {
593            \legacy_if_set_false:n { @noparitem }
594            \hbox_gset:Nn \g__block_labels_box
595              {
596                \skip_horizontal:n { - \leftmargin }
597                \hbox_unpack_drop:N \g__block_labels_box
598                \skip_horizontal:n { \leftmargin }
599              }
600            \legacy_if:nF { @minipage } % Why this chunk of code?
601              {
602                \__block_skip_set_to_last:N \l__block_tmpa_skip
```

document 2e logic used here

29

```
603          \skip_vertical:n { - \l__block_tmpa_skip }
604          \skip_vertical:n { \l__block_tmpa_skip + \@outerparskip - \parskip }
605        }
606      }
607      {
608        \legacy_if:nTF { @nobreak }
609          { \addvspace{\skip_eval:n{\@outerparskip-\parskip}} }
610          {
611            \addpenalty \@beginparpenalty
612            \addvspace \l__block_effective_top_skip
613            \addvspace{-\parskip}
614          }
615      }
616 }
```

Extra keys to support enumitem conventions:

```
617 \keys_define:nn { template/block/display }
618 {
619   ,topsep       .skip_set:N = \topsep
620   ,partopsep    .skip_set:N = \partopsep
621   ,listparindent .skip_set:N = \listparindent
622 }
```

__kernel_displayblock_begin: The internal kernel hooks for tagging.
__kernel_displayblock_beginpar_hmode:w
__kernel_displayblock_beginpar_vmode:

```
623 \cs_new:Npn \__kernel_displayblock_begin: {
624   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_begin:}}
625 }

626 \cs_new:Npn \__kernel_displayblock_beginpar_hmode:w {
627   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_beginpar_hmode:w}}
628 }

629 \cs_new:Npn \__kernel_displayblock_beginpar_vmode: {
630   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_beginpar_vmode:}}
631 }
```

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

6.5.4 Implementation of list templates . . .

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

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

(*End of definition for* \@itemlabel *and* \@listctr. *These functions are documented on page 11.*)

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

```
634 \DeclareTemplateCode{list}{std}{1}
635 {
636   counter        = \l__block_counter_tl,
637   item-label     = \l__block_item_label_tl,
```

```
638    start          = \l__block_counter_start_int ,
639    resume         = \l__block_resume_bool ,
640    item-instance  = \__block_item_instance:n ,
641    item-skip      = \itemsep ,
642 %  item-par-skip  = \parsep ,
643    item-penalty   = \@itempenalty ,
644    item-indent    = \itemindent ,
645    label-width    = \labelwidth ,
646    label-sep      = \labelsep ,
647    legacy-support = \l__block_legacy_support_bool , % FMi questionable
648 }
649 {
650   \__block_debug_typeout:n{template:list:std}
651 %
652   \tl_if_empty:nF {#1} { \SetTemplateKeys{list}{std}{#1} }
```

Has this list a counter name defined in the instance?

```
653   \tl_if_empty:NTF \l__block_counter_tl
654     {
```

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`.

```
655       \tl_if_blank:oF \@listctr
656         {
```

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.

```
657         \bool_if:NF \l__block_resume_bool
658           {
659             \int_gset:cn{ c@ \@listctr }
660               { \l__block_counter_start_int - 1 }
661           }
662         }
```

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

```
663       { \@nmbrlistfalse }
664     }
```

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.

```
665     {
666     \@nmbrlisttrue
667     \tl_set_eq:NN \@listctr \l__block_counter_tl
668     \bool_if:NF \l__block_resume_bool
669       {
670         \int_gset:cn{ c@ \@listctr }
671           { \l__block_counter_start_int - 1 }
672       }
673     }
```

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.

```
674    \tl_if_empty:NF \l__block_item_label_tl
675      {
676        \tl_set_eq:NN \@itemlabel \l__block_item_label_tl
677      }
```

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.

```
678      \legacy_if_gset_true:n { @newlist }
679    \__block_debug_typeout:n{template:list:std~end}
680  }
```

Extra keys to support enumitem conventions:

```
681  \keys_define:nn { template/list/std }
682  {
683    ,nosep .code:n =
684      \dim_zero:N \itemsep
685      \dim_zero:N \parsep
686      \dim_zero:N \topsep
687      \dim_zero:N \l__block_botsep_skip
688      \dim_zero:N \l__block_parbotsep_skip
689    ,midsep    .skip_set:N = \topsep
690  }
```

6.5.5   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.

```
691  \keys_define:nn { template/item/std }
692                  { label .tl_set:N = \l__block_label_given_tl }
```

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

```
693  \DeclareTemplateCode{item}{std}{1}
694    {
695      counter-label   = \__block_counter_label:n ,
696      counter-ref     = \__block_counter_ref:n ,
```

next set of key not yet used

```
697      label-ref       = \__block_label_ref:n ,
698      label-autoref   = \__block_label_autoref:n ,
699      label-format    = \__block_label_format:n ,
700      label-strut     = \l__block_label_strut_bool ,
701      label-boxed     = \l__block_label_boxed_bool ,
702      next-line       = \l__block_next_line_bool ,
703      text-font       = \l__block_text_font_tl ,
704      compatibility   = \l__block_item_compatibility_bool ,
```

complete This probably needs a different implementation (and needs completing)

```
705      label-align     = {
706        left    = \tl_set:Nn \l__block_item_align_tl { \relax \hss } ,
707        center  = \tl_set:Nn \l__block_item_align_tl { \hss \hss } ,
708        right   = \tl_set:Nn \l__block_item_align_tl { \hss \relax } ,
709        parleft = \NOT_IMPLEMENTED ,
710      } ,
711    }
```

32

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.

```
712    {
713        \__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.

```
714        \tl_set_eq:NN \l__block_label_given_tl \c_novalue_tl
715        \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 distinuish that from `\item[]`.

```
716        \tl_if_novalue:oTF \l__block_label_given_tl
717            {
```

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

fix

```
718            \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
719            \bool_if:NTF \l__block_item_compatibility_bool   % not sure that conditional
720                                                        % makes sense
721            { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}\@itemlabel } } % TODO ?
722            { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}\__block_counter_label:n
723        }
724        {
725            \__block_debug_typeout:n{item~ with~ optional}
726            \__block_make_label_box:n { \l__block_label_given_tl } }
727    \bool_if:nT
728        {
729            \l__block_label_boxed_bool
730            && \dim_compare_p:n { \box_wd:N \l__block_one_label_box <= \linewidth } % TODO: is \
731        }
732        {
733            \dim_compare:nNnT
734            { \box_wd:N \l__block_one_label_box } < \labelwidth
735            {
736                \hbox_set_to_wd:Nnn \l__block_one_label_box { \labelwidth }
737                {
738                    \exp_after:wN \use_i:nn \l__block_item_align_tl
```

FMi: LaTeX 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 the default.

```
739 %                   \hbox_unpack_drop:N \l__block_one_label_box    %TODO: customize?
740                    \box_use_drop:N \l__block_one_label_box

741                    \exp_after:wN \use_ii:nn \l__block_item_align_tl
742                }
743            }
```

Add another box level to the label box:

```
744          \hbox_set:Nn \l__block_one_label_box
745                     { \box_use_drop:N \l__block_one_label_box }
746          }
747      \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \labelwidth
748        { \bool_set_true:N \l__block_long_label_bool }
749        { \bool_set_false:N \l__block_long_label_bool }
750      \hbox_gset:Nn \g__block_labels_box
751        {
752          \hbox_unpack_drop:N \g__block_labels_box
753          \skip_horizontal:n { \itemindent - \labelsep - \labelwidth }
754          \hbox_unpack_drop:N \l__block_one_label_box
755          \skip_horizontal:n { \labelsep }
756          \bool_if:NT \l__block_next_line_bool
757            { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
758          % version of \newline inside an hbox that will be unpacked
759        }
760      % \skip_set_eq:NN \parsep \l__block_item_parsep_skip TODO??? FMi
761                                                    % what's that?
762      \dim_set_eq:NN \parindent \listparindent
```

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.

```
763        \cs_set_eq:NN \__block_item_everypar: \__block_item_everypar_std:
764      }
```

\l__block_one_label_box   Each label is typeset in \l__block_one_label_box to be measured. Once this is ready, it
\g__block_labels_box   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 LaTeX 2ε's \@labels, but it is always unboxed before use, to support both boxed and unboxed labels.

```
765 \box_new:N \l__block_one_label_box
766 \box_new:N \g__block_labels_box
```

(*End of definition for* \l__block_one_label_box *and* \g__block_labels_box.)

\l__block_long_label_bool   Track whether the \l__block_one_label_box is larger than \labelwidth.

```
767 \bool_new:N \l__block_long_label_bool
```

(*End of definition for* \l__block_long_label_bool.)

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

```
768 \cs_new_protected:Npn \__block_make_label_box:n #1
769   {
770     \hbox_set:Nn \l__block_one_label_box
771       {
```

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

```
772          \__kernel_list_label_begin:
```

```
773        \__block_label_format:n
774          {
775            \bool_if:NT \l__block_label_strut_bool { \strut }
776            \bool_if:NTF \l__block_legacy_support_bool
777                         \makelabel
778                         \use:n
779                  {#1}
780          }
781        }
```

And what gets opened also needs closing:

```
781          \__kernel_list_label_end:
782        }
783    }
```

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

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

```
784 \cs_new_eq:NN \__kernel_list_label_begin: \prg_do_nothing:
785 \cs_new_eq:NN \__kernel_list_label_end:   \prg_do_nothing:
```

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

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.

```
786 \cs_new_eq:NN \__block_item_everypar: \prg_do_nothing:
```

```
787 \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!

```
788 \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.

```
789 \cs_new_protected:Npn \__block_item_everypar_std: {
790     \__block_debug_typeout:n{item~ everypar \on@line }
791     \legacy_if_set_false:n { @minipage }
792     \legacy_if_gset_false:n { @newlist }
793     \legacy_if:nT { @inlabel }
794        {
795            \legacy_if_gset_false:n { @inlabel }

796            \box_if_empty:NT \g_para_indent_box { \kern - \itemindent }
797            \para_omit_indent:

798            \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:

```
799          \__kernel_list_label_after:
```

35

```
800        \penalty \c_zero_int
801      }
802    \legacy_if:nTF { @nobreak }
803        {
804          \legacy_if_gset_false:n { @nobreak }
805          \int_set:Nn \clubpenalty { 10000 }
806        }
807        {
808          \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.

```
809          \cs_set_eq:NN \__block_item_everypar: \prg_do_nothing:
810        }
811  }
```

(*End of definition for* __block_item_everypar: *and* __block_item_everypar_std:*.*)

__kernel_list_label_after:

```
812 \cs_new_eq:NN \__kernel_list_label_after: \prg_do_nothing:
```

(*End of definition for* __kernel_list_label_after:*.*)

\l__block_tmpa_skip

```
813 \skip_new:N \l__block_tmpa_skip
```

(*End of definition for* \l__block_tmpa_skip*.*)

\l__block_topsepadd_skip
\l__block_effective_top_skip

Variables equivalent to LaTeX 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.

```
814 \skip_new:N \l__block_topsepadd_skip
815 \skip_new:N \l__block_effective_top_skip
```

(*End of 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 __block_inter_-item: to cleanly close what's before, then call __block_item_instance:n (which calls \UseInstance{item}{⟨*instance*⟩}) to prepare the upcoming item: it will be actually inserted only once some later material triggers \everypar.

```
816 \AddToHook{begindocument/before}{
817   \RenewDocumentCommand{\item}{ ={label}o }
818     {
819       \@inmatherr \item
```

TODO: Check if test for being outside of a list is sensible

```
820       \cs_if_free:NTF \__block_item_instance:n
821         {
822           \@latex@error{Lonely~\string\item--perhaps~a~missing~
823           list~environment}\@ehc
824         }
825         {
826           \legacy_if:nTF { @newlist }
827             { \__kernel_list_item_begin: }
828             { \__block_inter_item:            }
```

36

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

```
829            \tl_if_novalue:nTF {#1}          % avoids reparsing label={}
830              { \__block_item_instance:n { } }
831              { \__block_item_instance:n {#1} }
```

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

```
832            \legacy_if_gset_true:n { @inlabel }
833            \ignorespaces
834          }
835      }
836 }
```

(*End of definition for* \item. *This function is documented on page 11.*)

__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.

```
837 \cs_new_protected:Npn \__block_inter_item: {
838   \legacy_if:nT { @inlabel }
839                 { \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).

```
840    \mode_if_horizontal:T { \__block_skip_remove_last:
841                            \__block_skip_remove_last: \par }
```

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

```
842    \__kernel_list_item_end:
843    \__kernel_list_item_begin:

844    \addpenalty \@itempenalty
845    \addvspace \itemsep
846 }
```

(*End of definition for* __block_inter_item:.)

__kernel_list_item_begin:
__kernel_list_item_end:
```
847 \cs_new_eq:NN \__kernel_list_item_begin: \prg_do_nothing:
848 \cs_new_eq:NN \__kernel_list_item_end:   \prg_do_nothing:
```

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

6.6 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.

```
849 \cs_new:Npn \__block_recipe_basic: {
850   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
851                                          \__block_beginpar_hmode:N
```

```
852    \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
853                                                          \__block_beginpar_vmode:
854    \let \__kernel_displayblock_begin:              \prg_do_nothing:
855    \let \__kernel_displayblock_end:                \prg_do_nothing:
```

End environment \par handling:

```
856    \socket_assign_plug:nn{tagsupport/block-endpe}{on}
857  }
```

(*End of definition for* __block_recipe_basic:.)

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

```
858  \cs_new:Npn \__block_recipe_standalone: {
859    \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
860                                                          \prg_do_nothing:
861    \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
862                                                          \prg_do_nothing:
863    \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
864    \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:
```

End environment \par handling:

```
865    \socket_assign_plug:nn{tagsupport/block-endpe}{off}
866    \tl_if_empty:NTF \l__block_tag_name_tl
867       { \tl_set:Nn    \l__block_tag_inner_tag_tl {Sect}              }
868       { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
869  }
```

(*End of definition for* __block_recipe_standalone:.)

__block_recipe_standard: The `standard` recipe does the following:

- surround the block with a `text-unit`-structure if not already in a 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 inner structure (`Figure` or explicit one) but leave the `text-unit` open to be either continued or closed due to a following \par.

```
870  \cs_new:Npn \__block_recipe_standard:
871  {
872    \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
873                                                          \__block_beginpar_hmode:N
874    \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
875                                                          \__block_beginpar_vmode:
876    \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
877    \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:
```

End environment \par handling:

```
878    \socket_assign_plug:nn{tagsupport/block-endpe}{on}
```

```
879    \tl_if_empty:NTF \l__block_tag_name_tl
880      { \tl_set:Nn    \l__block_tag_inner_tag_tl {Figure}           }
881      { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
882  }
```

(*End of definition for* __block_recipe_standard:.)

\l__block_tag_inner_tag_tl

```
883  \tl_new:N \l__block_tag_inner_tag_tl
```

(*End of definition for* \l__block_tag_inner_tag_tl.)

__block_recipe_list: The list recipe does the following.

- It opens a `<text-unit>`-structure or keeps the current one open (only closing the MC).

- It then starts a new structure rolemapped to L-structure and arranges for handling list items, e.g., Li, Lbl and LBody structures.

- At the end it closes open list structures as needed but keeps the `<text-unit>`-structure open to continue the paragraph after the list, if necessary.

```
884  \cs_new:Npn \__block_recipe_list:
885  {
886    \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
887                                                \__block_beginpar_hmode:N
888    \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
889                                                \__block_beginpar_vmode:
890    \cs_set_eq:NN \__kernel_displayblock_begin: \__block_list_begin:
891    \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)).

```
892    \cs_set_eq:NN \__kernel_list_item_begin:    \__block_list_item_begin:
893    \cs_set_eq:NN \__kernel_list_item_end:      \__block_list_item_end:
```

End environment \par handling:

```
894    \socket_assign_plug:nn{tagsupport/block-endpe}{on}
```

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

```
895    \tl_if_empty:NTF \l__block_tag_name_tl
896      { \tl_set:Nn    \l__tag_L_tag_tl {L}                }
897      { \tl_set_eq:NN \l__tag_L_tag_tl \l__block_tag_name_tl }
898    \tl_if_empty:NTF \l__block_tag_class_tl
899      { \tl_set:Nn    \l__tag_L_attr_class_tl {}              }
900      { \tl_set_eq:NN \l__tag_L_attr_class_tl \l__block_tag_class_tl }
901  }
```

(*End of definition for* __block_recipe_list:.)

6.7 Blockenv instances

6.7.1 Basic instances

blockenv displayblock *(inst.)*

```
902 \DeclareInstance{blockenv}{displayblock}{display}
903 {
904   env-name        = displayblock,
905   tag-name        = ,
906   tag-class       = ,
907   tagging-recipe = standard,
908   inner-level-counter  = ,
909   level-increase = false,
910   setup-code      = ,
911   block-instance = displayblock ,
912   inner-instance = ,
913 }
```

nv displayblockflattened *(inst.)*

```
914 \DeclareInstance{blockenv}{displayblockflattened}{display}
915 {
916   env-name        = displayblockflattened,
917   tag-name        = ,
918   tag-class       = ,
919   tagging-recipe = basic,
920   inner-level-counter  = ,
921   level-increase = false,
922   setup-code      = ,
923   block-instance = displayblock ,
924   para-flattened = true ,
925   inner-instance = ,
926 }
```

blockenv center *(inst.)*

```
927 \DeclareInstance{blockenv}{center}{display}
928 {
929   env-name        = center,
930   tag-name        = ,
931   tag-class       = ,
932   tagging-recipe = basic,
933   inner-level-counter  = ,
934   level-increase = false,
935   setup-code      = ,
936   block-instance = displayblock ,
937   para-flattened = true ,
938   para-instance = center ,
939   inner-instance = ,
940 }
```

blockenv flushleft *(inst.)*

```
941 \DeclareInstance{blockenv}{flushleft}{display}
942 {
943   env-name        = flushleft,
944   tag-name        = ,
```

40

```
945    tag-class      = ,
946    tagging-recipe = basic,
947    inner-level-counter  = ,
948    level-increase = false,
949    setup-code     = ,
950    block-instance = displayblock ,
951    para-flattened = true ,
952    para-instance  = raggedright ,
953    inner-instance = ,
954  }
```

blockenv flushright (*inst.*)

```
955  \DeclareInstance{blockenv}{flushright}{display}
956  {
957    env-name       = flushleft,
958    tag-name       = ,
959    tag-class      = ,
960    tagging-recipe = basic,
961    inner-level-counter  = ,
962    level-increase = false,
963    setup-code     = ,
964    block-instance = displayblock ,
965    para-flattened = true ,
966    para-instance  = raggedleft ,
967    inner-instance = ,
968  }
```

6.7.2   Blockquote instances

blockenv quotation (*inst.*)

```
969  \DeclareInstance{blockenv}{quotation}{display}
970  {
971    env-name       = quotation,
972    tag-name       = quotation,
973    tag-class      = ,
974    tagging-recipe = standard,
975    inner-level-counter  = ,
976    level-increase = true,
977    setup-code     = ,
978    block-instance = quotationblock ,
979    inner-instance =   ,
980  }
```

blockenv quote (*inst.*)

```
981  \DeclareInstance{blockenv}{quote}{display}
982  {
983    env-name       = quote,
984    tag-name       = quote,
985    tag-class      = ,
986    tagging-recipe = standard,
987    inner-level-counter  = ,
988    level-increase = true,
989    setup-code     = ,
990    block-instance = quoteblock ,
```

41

```
991    inner-instance =  ,
992 }
```

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.

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

```
993 % \DeclareInstance{blockenv}{quotation}{display}
994 % {
995 %    env-name       = quotation,
996 %    tag-name       = ,
997 %    tag-class      = ,
998 %    tagging-recipe = blockquote,
999 %    inner-level-counter  = ,
1000 %   level-increase = true,
1001 %   setup-code     = \setlength\rightmargin{\leftmargin}
1002 %                    \setlength\parsep{1.5em} ,
1003 %   block-instance = displayblock ,
1004 %   inner-instance =  ,
1005 % }
1006 % \DeclareInstance{blockenv}{quote}{display}
1007 % {
1008 %    env-name       = quote,
1009 %    tag-name       = ,
1010 %    tag-class      = ,
1011 %    tagging-recipe = blockquote,
1012 %    inner-level-counter  = ,
1013 %   level-increase = true,
1014 %   setup-code     = \setlength\rightmargin{\leftmargin} ,
1015 %   block-instance = displayblock ,
1016 %   inner-instance =  ,
1017 % }
1018 \DeclareInstance{blockenv}{theorem}{display}
1019 {
1020   env-name       = theorem-like,
1021   tag-name       = theorem-like,
1022   tag-class      = ,
1023   tagging-recipe = standalone,
1024   inner-level-counter  = ,
1025   level-increase = false,
1026   setup-code     = ,
1027   block-instance = displayblock ,
1028 % inner-instance-type = innerblock ,
1029 % inner-instance = theorem,
1030 }
```

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

6.7.3   Verbatim instances

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

```
1031 \DeclareInstance{blockenv}{verbatim}{display}
```

```
1032 {
1033   env-name       = verbatim,
1034   tag-name       = verbatim,
1035   tag-class      = ,
1036   tagging-recipe = standard,
1037   inner-level-counter  = ,
1038   level-increase = false,
1039   setup-code     = ,
1040   block-instance = verbatimblock ,
1041   inner-instance = ,
1042   final-code     = \legacyverbatimsetup ,
1043   para-instance  = justify
1044 }
```

6.7.4 Standard list instances

blockenv itemize (*inst.*)

```
1045 \DeclareInstance{blockenv}{itemize}{display}
1046 {
1047   env-name       = itemize,
1048   tag-name       = itemize,
1049   tag-class      = itemize,
1050   tagging-recipe = list,
1051   inner-level-counter  = \@itemdepth,
1052   level-increase = true,
1053   max-inner-levels = 4,
1054   setup-code     = ,
1055   block-instance = list ,
1056   inner-instance = itemize ,
1057 }
```

blockenv enumerate (*inst.*)

```
1058 \DeclareInstance{blockenv}{enumerate}{display}
1059 {
1060   env-name            = enumerate,
1061   tag-name            = enumerate,
1062   tag-class           = enumerate,
1063   tagging-recipe      = list,
1064   level-increase      = true,
1065   setup-code          = ,
1066   block-instance      = list ,
1067   inner-level-counter = \@enumdepth,
1068   max-inner-levels    = 4,
1069   inner-instance      = enum ,
1070 }
```

blockenv description (*inst.*)

```
1071
1072 \DeclareInstance{blockenv}{description}{display}
1073 {
1074   env-name       = description,
1075   tag-name       = description,
1076   tag-class      = description,
1077   tagging-recipe = list,
```

43

```
1078    inner-level-counter  = ,
1079    level-increase = true,
1080    setup-code       = ,
1081    block-instance = list ,
1082    inner-instance = description ,
1083  }
```

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

```
1084  \DeclareInstance{blockenv}{list}{display}
1085  {
1086    env-name        = list,
1087    tag-name        = list,
1088    tag-class       = ,
1089    tagging-recipe = list,
1090    level-increase = true,
1091    setup-code       = \legacylistsetupcode ,
1092    block-instance = list ,
1093    inner-level-counter  = ,
1094    inner-instance = legacy ,
1095  }
```

6.8   Block instances

6.8.1   Displayblock instances

We provide 6 nesting levels (as in LaTeX 2ε). 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).

```
1096  \setcounter{maxblocklevels}{6}
```

block displayblock-0 (*inst.*)
block displayblock-1 (*inst.*)
block displayblock-2 (*inst.*)
block displayblock-3 (*inst.*)
block displayblock-4 (*inst.*)
block displayblock-5 (*inst.*)
block displayblock-6 (*inst.*)

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

```
1097  \DeclareInstance{block}{displayblock-0}{display}
1098    {
1099      leftmargin        = 0pt ,
1100      parindent         = 0pt ,
1101    }
1102  \DeclareInstanceCopy{block}{displayblock-1}{displayblock-0}
1103  \DeclareInstanceCopy{block}{displayblock-2}{displayblock-0}
1104  \DeclareInstanceCopy{block}{displayblock-3}{displayblock-0}
1105  \DeclareInstanceCopy{block}{displayblock-4}{displayblock-0}
1106  \DeclareInstanceCopy{block}{displayblock-5}{displayblock-0}
1107  \DeclareInstanceCopy{block}{displayblock-6}{displayblock-0}
```

6.8.2   Verbatim instances

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

block verbatimblock-0 (*inst.*)
block verbatimblock-1 (*inst.*)
block verbatimblock-2 (*inst.*)
block verbatimblock-3 (*inst.*)
block verbatimblock-4 (*inst.*)
block verbatimblock-5 (*inst.*)
block verbatimblock-6 (*inst.*)

```
1108 \DeclareInstance{block}{verbatimblock-0}{display}
1109   {
1110     leftmargin      = 0pt ,
1111     parindent       = 0pt ,
1112     par-skip        = 0pt ,
1113   }
1114 \DeclareInstanceCopy{block}{verbatimblock-1}{verbatimblock-0}
1115 \DeclareInstanceCopy{block}{verbatimblock-2}{verbatimblock-0}
1116 \DeclareInstanceCopy{block}{verbatimblock-3}{verbatimblock-0}
1117 \DeclareInstanceCopy{block}{verbatimblock-4}{verbatimblock-0}
1118 \DeclareInstanceCopy{block}{verbatimblock-5}{verbatimblock-0}
1119 \DeclareInstanceCopy{block}{verbatimblock-6}{verbatimblock-0}
```

6.8.3 Quote/quotationblock instances

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

block quoteblock-1 (*inst.*)
block quoteblock-2 (*inst.*)
block quoteblock-3 (*inst.*)
block quoteblock-4 (*inst.*)
block quoteblock-5 (*inst.*)
block quoteblock-6 (*inst.*)

Default layout is to indent equally from both sides.

```
1120 \DeclareInstance{block}{quoteblock-1}{display}
1121   { rightmargin = \KeyValue{leftmargin} }
1122 \DeclareInstanceCopy{block}{quoteblock-2}{quoteblock-1}
1123 \DeclareInstanceCopy{block}{quoteblock-3}{quoteblock-1}
1124 \DeclareInstanceCopy{block}{quoteblock-4}{quoteblock-1}
1125 \DeclareInstanceCopy{block}{quoteblock-5}{quoteblock-1}
1126 \DeclareInstanceCopy{block}{quoteblock-6}{quoteblock-1}
```

block quotationblock-1 (*inst.*)
block quotationblock-2 (*inst.*)
block quotationblock-3 (*inst.*)
block quotationblock-4 (*inst.*)
block quotationblock-5 (*inst.*)
block quotationblock-6 (*inst.*)

Quotation additionally changes the parindent.

```
1127 \DeclareInstance{block}{quotationblock-1}{display}
1128   { parindent = 1.5em , rightmargin = \KeyValue{leftmargin} }
1129 \DeclareInstanceCopy{block}{quotationblock-2}{quotationblock-1}
1130 \DeclareInstanceCopy{block}{quotationblock-3}{quotationblock-1}
1131 \DeclareInstanceCopy{block}{quotationblock-4}{quotationblock-1}
1132 \DeclareInstanceCopy{block}{quotationblock-5}{quotationblock-1}
1133 \DeclareInstanceCopy{block}{quotationblock-6}{quotationblock-1}
```

6.8.4 Block instances for the standard lists

block list-1 (*inst.*)
block list-2 (*inst.*)
block list-3 (*inst.*)
block list-4 (*inst.*)
block list-5 (*inst.*)
block list-6 (*inst.*)

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

```
1134 \DeclareInstance{block}{list-1}{display}{
1135 %    heading          = ,
1136 %    beginsep         = \topsep ,
1137 %    begin-par-skip   = \partopsep ,
1138 %    par-skip         = \parsep ,
1139 %    end-skip         = \KeyValue{beginsep} ,
1140 %    end-par-skip     = \KeyValue{begin-par-skip} ,
1141 %    beginpenalty     = \UseName{@beginparpenalty} ,
1142 %    endpenalty       = \UseName{@endparpenalty} ,
1143 %    leftmargin       = \leftmargin ,
```

```
1144 %   rightmargin      = \rightmargin ,
1145 %   parindent        = \listparindent ,
1146 }
1147 \DeclareInstance{block}{list-2}{display}{}
1148 \DeclareInstance{block}{list-3}{display}{}
1149 \DeclareInstance{block}{list-4}{display}{}
1150 \DeclareInstance{block}{list-5}{display}{}
1151 \DeclareInstance{block}{list-6}{display}{}
```

6.9   List instances for the standard lists

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

<div style="float:left"><code>list itemize-1</code> (*inst.*)
<code>list itemize-2</code> (*inst.*)
<code>list itemize-3</code> (*inst.*)
<code>list itemize-4</code> (*inst.*)</div>

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

```
1152 \DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi }
1153 \DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii }
1154 \DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
1155 \DeclareInstance{list}{itemize-4}{std}{ item-label = \labelitemiv }
```

<div style="float:left"><code>list enumerate-1</code> (*inst.*)
<code>list enumerate-2</code> (*inst.*)
<code>list enumerate-3</code> (*inst.*)
<code>list enumerate-4</code> (*inst.*)</div>

`enumerate` environments are similar, except that we also have to say which counter to
use on every level.

```
1156 \DeclareInstance{list}{enum-1}{std}
1157   { item-label = \labelenumi ,   counter = enumi }
1158 \DeclareInstance{list}{enum-2}{std}
1159   { item-label = \labelenumii ,  counter = enumii }
1160 \DeclareInstance{list}{enum-3}{std}
1161   { item-label = \labelenumiii , counter = enumiii }
1162 \DeclareInstance{list}{enum-4}{std}
1163   { item-label = \labelenumiv ,  counter = enumiv }
```

<div style="float:left"><code>list legacy</code> (*inst.*)</div>

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 parame-
ters 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 for-
matting the label

```
1164 \DeclareInstance{list}{legacy}{std} {
1165   item-instance = basic ,
1166   legacy-support = true ,
1167 }
```

<div style="float:left"><code>list description</code> (*inst.*)</div>

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

```
1168 \DeclareInstance{list}{description}{std} { item-instance = description }
```

46

6.10 Item instances

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

```
1169 \DeclareInstance{item}{basic}{std}
1170   {
1171     label-align = right ,
1172   }
1173 \DeclareInstance{item}{description}{std}
1174   {
1175     label-format = \normalfont\bfseries #1 ,
1176     label-align = left
1177   }
```

6.11 Para instances

```
1178 \tag_if_active:T
1179 {
1180   \tagpdfsetup
1181     {
1182       role/new-attribute = {justify}    {/O /Layout /TextAlign/Justify},
1183       role/new-attribute = {center}     {/O /Layout /TextAlign/Center},
1184       role/new-attribute = {raggedright}{/O /Layout /TextAlign/Start},
1185       role/new-attribute = {raggedleft} {/O /Layout /TextAlign/End},
1186     }
1187 }
```

```
1188 \DeclareInstance{para}{center}{std}
1189 {
1190   indent-width          = 0pt ,
1191   start-skip            = 0pt ,
1192   left-skip             = \@flushglue ,
1193   right-skip            = \@flushglue ,
1194   end-skip              = \z@skip ,
1195   final-hyphen-demerits = 0 ,
1196   cr-cmd                = \@centercr ,
1197   para-class            = center ,
1198 }
1199 \DeclareInstance{para}{raggedright}{std}
1200 {
1201   indent-width          = 0pt ,
1202   start-skip            = 0pt ,
1203   left-skip             = \z@skip ,
1204   right-skip            = \@flushglue ,
1205   end-skip              = \z@skip ,
1206   final-hyphen-demerits = 0 ,
1207   cr-cmd                = \@centercr ,
1208   para-class            = raggedright ,
1209 }
1210 \DeclareInstance{para}{raggedleft}{std}
1211 {
1212   indent-width          = 0pt ,
```

```
1213     start-skip              = 0pt ,
1214     left-skip               = \@flushglue ,
1215     right-skip              = \z@skip ,
1216     end-skip                = \z@skip ,
1217     final-hyphen-demerits = 0 ,
1218     cr-cmd                  = \@centercr ,
1219     para-class              = raggedleft ,
1220 }
1221 \DeclareInstance{para}{justify}{std}
1222 {
1223 %   indent-width            = 0pt ,
1224     start-skip              = 0pt ,
1225     left-skip               = \z@skip ,
1226     right-skip              = \z@skip ,
1227     end-skip                = \@flushglue ,
1228     final-hyphen-demerits =  5000 ,
1229     cr-cmd                  = \@normalcr ,
1230     para-class              = justify ,
1231 }
1232 \DeclareRobustCommand\centering   {\UseInstance{para}{center}{}}
1233 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}{}}
1234 \DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}{}}
1235 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}{}}
1236
1237 \justifying
```

6.12   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!

```
1238 \tag_if_active:TF {
```

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

```
1239     \cs_set:Npn \__block_beginpar_vmode: {
1240             \__block_debug_typeout:n
1241                 { @endpe = \legacy_if:nTF { @endpe }{true}{false}
1242                 \on@line }
1243     \legacy_if:nTF { @endpe }
1244         {
1245         \legacy_if_gset_false:n { @endpe }
1246         }
```

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.

```
1247         {
1248         \int_compare:nNnT \l__tag_block_flattened_level_int < 2
1249             {
```

```
1250                    \__tag_gincr_para_main_begin_int:
1251                    \tag_struct_begin:n
1252                      {
1253                        tag=\l__tag_para_main_tag_tl,
1254                        attribute-class=\l__tag_para_main_attr_class_tl,
1255                      }
1256                    \__tag_para_main_store_struct:
1257                  }
1258            }
1259      }
```

(*End of definition for* __block_beginpar_vmode:.)

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

```
1260    \cs_set:Npn \__block_beginpar_hmode:N #1
1261      {
1262        \tag_mc_end:
1263        \__tag_gincr_para_end_int:
1264        \__block_debug_typeout:n{increment~ /P \on@line }
1265        \bool_if:NT \l__tag_para_show_bool
1266          { \tag_mc_begin:n{artifact}
1267            \rlap{\color_select:n{red}\tiny\ \int_use:N\g__tag_para_end_int}
1268            \tag_mc_end:
1269          }
1270        \tag_struct_end:
1271        \tagpdfparaOff \par \tagpdfparaOn
1272      }
```

(*End of definition for* __block_beginpar_hmode:N.)

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

```
1273    \cs_set:Npn \__kernel_displayblock_doendpe: {
1274        \bool_if:NT \l__tag_para_bool
1275          {
```

Given that restoring \par through the legacy LaTeX 2ε 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.

```
1276            \legacy_if:nT { @endpe }
1277              {
```

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.

```
1278                \__block_debug_typeout:n
1279                  { flattened= \bool_if:NTF
```

49

```
1280                                    \l__tag_para_flattened_bool {true}{false}
1281                      \on@line }
1282                  \bool_if:NF \l__tag_para_flattened_bool
1283                    {
1284                      \__block_debug_typeout:n{Structure-end~
1285                        \l__tag_para_main_tag_tl\space after~ displayblock \on@line }
1286                      \__tag_gincr_para_main_end_int:
1287                      \tag_struct_end: %text-unit
1288                    }
1289                }
1290          }
1291 }
```

(*End of definition for* __kernel_displayblock_doendpe:.)

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

```
1292 \RemoveFromHook{para/begin}[tagpdf]
```

```
1293 \AddToHook{para/begin}[tagpdf]{
1294   \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).

```
1295        \legacy_if:nF { @inlabel }
1296          {
```

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

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

```
1297              \__block_start_para_structure:n { \PARALABEL }
1298            }
1299        }
1300 }
```

__block_start_para_structure:n
```
1301 \cs_new_protected:Npn \__block_start_para_structure:n #1 {
1302   \__block_debug_typeout:n
1303     { @endpe = \legacy_if:nTF { @endpe }{true}{false}
1304       \on@line }
1305   \legacy_if:nF { @endpe }
1306     {
1307       \bool_if:NF \l__tag_para_flattened_bool
1308         {
1309           \__tag_gincr_para_main_begin_int:
1310           \tag_struct_begin:n
1311             {
1312               tag=\l__tag_para_main_tag_tl,
1313               attribute-class=\l__tag_para_main_attr_class_tl,
```

```
1314              }
1315            \__tag_para_main_store_struct:
1316          }
1317        }
1318      \__tag_gincr_para_begin_int:
1319      \__block_debug_typeout:n{increment~ P \on@line }
1320      \tag_struct_begin:n
1321        {
1322          tag=\l__tag_para_tag_tl
1323          ,attribute-class=\l__tag_para_attr_class_tl
1324        }
1325      \__tag_check_para_begin_show:nn {green}{#1}
1326      \tag_mc_begin:n {}
1327  }
```

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

```
1328  \cs_new_protected:Npn \__block_start_para_structure_unconditionally:n #1 {
1329      \bool_if:NF \l__tag_para_flattened_bool
1330        {
1331          \__tag_gincr_para_main_begin_int:
1332          \tag_struct_begin:n
1333            {
1334              tag=\l__tag_para_main_tag_tl,
1335              attribute-class=\l__tag_para_main_attr_class_tl,
1336            }
1337          \__tag_para_main_store_struct:
1338        }
1339      \__tag_gincr_para_begin_int:
1340      \__block_debug_typeout:n{increment~ P \on@line }
1341      \tag_struct_begin:n
1342        {
1343          tag=\l__tag_para_tag_tl
1344          ,attribute-class=\l__tag_para_attr_class_tl
1345        }
1346      \__tag_check_para_begin_show:nn {green}{#1}
1347      \tag_mc_begin:n {}
1348  }

1349  \RemoveFromHook{para/end}[tagpdf]
1350  \AddToHook{para/end}
1351    {
1352      \bool_if:NT \l__tag_para_bool
1353        {
1354          \__tag_gincr_para_end_int:
1355          \__block_debug_typeout:n{increment~ /P \on@line }
1356          \tag_mc_end:
1357          \__tag_check_para_end_show:nn {red}{}
1358          \tag_struct_end:
1359          \bool_if:NF \l__tag_para_flattened_bool
1360            {
1361              \__tag_gincr_para_main_end_int:
1362              \tag_struct_end:
1363            }
1364        }
```

```
1365      }

1366 \def\PARALABEL{NP-}
```

(*End of definition for* para/begin *and* __block_start_para_structure:n. *This function is documented on page 12.*)

\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).

```
1367 \cs_set_protected:Npn \para_end: {
1368    \scan_stop:
1369    \mode_if_horizontal:TF {
1370      \mode_if_inner:F {
1371          \tex_unskip:D
1372          \hook_use:n{para/end}
1373          \@kernel@after@para@end
1374          \mode_if_horizontal:TF {
1375            \if_int_compare:w 11 = \tex_lastnodetype:D
1376              \tex_hskip:D \c_zero_dim
1377            \fi:
1378            \tex_par:D
1379            \hook_use:n{para/after}
1380            \@kernel@after@para@after
1381          }
1382          { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
1383      }
1384    }
1385    {
1386      \__kernel_endpe_vmode:        % should do nothing if no tagging
1387      \tex_par:D
1388    }
1389 }
1390 \cs_set_eq:NN \par        \para_end:
1391 \cs_set_eq:NN \__blockpar    \para_end:
1392 \cs_set_eq:NN \endgraf \para_end:
```

(*End of definition for* \para_end:. *This function is documented on page 12.*)

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

```
1393 \DeclareRobustCommand*\begin[1]{%
1394    \UseHook{env/#1/before}%
1395    \@ifundefined{#1}%
1396      {\def\reserved@a{\@latex@error{Environment~#1~undefined}\@eha}}%
1397      {\def\reserved@a{\def\@currenvir{#1}%
1398          \edef\@currenvline{\on@line}%
1399          \@execute@begin@hook{#1}%
1400          \csname #1\endcsname}}%
1401    \@ignorefalse
1402    \begingroup
1403      \__kernel_endpe_vmode:
1404      \reserved@a}
```

(*End of definition for* \begin. *This function is documented on page 12.*)

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

```
1405 \cs_new:Npn \__kernel_endpe_vmode: {
1406     \if@endpe \ifvmode
1407         \bool_if:NT \l__tag_para_bool
1408 {
1409     \bool_if:NF \l__tag_para_flattened_bool
1410     {
1411         \__tag_gincr_para_main_end_int:
1412         \tag_struct_end:
1413     }
1414     \@endpefalse
1415 }
1416     \fi \fi
1417 }
```

(*End of definition for* `\__kernel_endpe_vmode:`.)

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

```
1418 \cs_set:Npn \__kernel_list_label_after: {
1419     \bool_if:NT \l__tag_para_bool
1420         {
1421             \__block_start_para_structure_unconditionally:n { LI- }
1422         }
1423 }
```

(*End of definition for* `\__kernel_list_label_after:`.)

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

```
1424 \cs_new:Npn \__block_inner_begin: {
1425     \tagstructbegin{tag=\l__block_tag_inner_tag_tl}
1426 }
```

(*End of definition for* `\__block_inner_begin:`.)

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

```
1427 \cs_new:Npn \__block_inner_end: {
1428     \__block_debug_typeout:n{block-end \on@line}
1429     \legacy_if:nT { @endpe }
1430         {
1431             \__tag_gincr_para_main_end_int:
1432             \__block_debug_typeout:n{close~ /text-unit \on@line}
1433             \tagstructend
1434         }
1435     \tagstructend        % end inner structure
1436 }
```

(*End of definition for* `\__block_inner_end:`.)

6.12.1 List tags

```
1437 \tl_new:N  \l__tag_L_tag_tl
1438 \tl_set:Nn \l__tag_L_tag_tl {L}
1439
1440 \tl_new:N\l__tag_L_attr_class_tl
1441 \tl_set:Nn \l__tag_L_attr_class_tl {list}
1442 \tag_if_active:T
1443 {
1444   \tagpdfsetup
1445       {
1446         role/new-attribute = {itemize}{/O /List /ListNumbering/Unordered},
1447         role/new-attribute = {enumerate}{/O /List /ListNumbering/Ordered},
1448         role/new-attribute = {description}{/O /List /ListNumbering/Description},
```

Initially, we had `/None` for the basic `list` environment, but that is not allowed in PDF/UA-2 if the list contains any Lbl tags. So now we default to `Unordered`.

```
1449         % default if unknown
1450         role/new-attribute = {list}{/O /List /ListNumbering/Unordered},
1451       }
1452 }
1453 \def\LItag{LI}
```

`\__block_list_begin:`   Start a list ...

```
1454 \cs_set:Npn \__block_list_begin: {
1455   \tagstructbegin
1456       {
1457          tag=\l__tag_L_tag_tl
1458         ,attribute-class=\l__tag_L_attr_class_tl
1459       }
1460 }
```

(*End of definition for* `\__block_list_begin:`*.*)

`\__block_list_item_begin:`   Start tagging a list item.

```
1461 \cs_set:Npn \__block_list_item_begin: { \tagstructbegin{tag=\LItag} }
```

(*End of definition for* `\__block_list_item_begin:`*.*)

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

```
1462 \cs_set:Npn \__kernel_list_label_begin: {
1463 %
1464 % FMi: this needs a different logic to decide when to make the label
1465 %    an artifact (after cleaning up the \item code  ), therefore
1466 %    disabled for now
1467 %  \tl_if_empty:oTF \@itemlabel
1468 %      {
1469 %        \tag_mc_begin:n {artifact}
1470 %      }
1471 %      {
1472         \tagstructbegin{tag=Lbl}
1473         \tagmcbegin{tag=Lbl}
1474 %      }
1475 }
```

(*End of definition for* `\__kernel_list_label_begin:`.)

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

```
1476 \cs_set:Npn \__kernel_list_label_end: {
1477   \tagmcend                                    % end mc-Lbl or artifact
1478 % FMi: unconditionally for now
1479 %  \tl_if_empty:oF \@itemlabel
1480        \tagstructend    % end   Lbl
1481   \tagstructbegin{tag=\LBody}
1482 }
1483 \def\LBody{LBody}
```

(*End of definition for* `\__kernel_list_label_end:`.)

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

```
1484 \cs_set:Npn \__block_list_item_end: {
1485   \legacy_if:nT { @endpe }
1486     {
1487        \__tag_gincr_para_main_end_int:
1488        \tagstructend                             % text-unit
1489 %       \__block_debug_typeout:n{Structure-end~ P~ at~ item-end \on@line }
1490     }
1491   \tagstructend \tagstructend   % end LBody, LI
1492 }
```

(*End of definition for* `\__block_list_item_end:`.)

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

```
1493 \cs_set:Npn \__block_list_end: {
1494   \legacy_if:nT { @endpe }
1495     {
1496        \__tag_gincr_para_main_end_int:
1497        \tagstructend                          % text-unit
1498        \__block_debug_typeout:n{Structure-end~ P~ at~ list-end \on@line }
1499     }
1500   \tagstructend\tagstructend  % end LBody, LI
1501   \tagstructend              % end L
1502 }
```

(*End of definition for* `\__block_list_end:`.)

End of tagging related declarations.

```
1503 }
```

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

```
1504 {
1505   \cs_new:Npn \__block_start_para_structure_unconditionally:n #1 {}
1506 }
```

```
1507 ⟨/package⟩
```

```
1508 ⟨∗latex-lab⟩
1509 \ProvidesFile{block-latex-lab-testphase.ltx}
1510        [\ltlabblockdate\space v\ltlabblockversion\space
1511                          blockenv implementation]
1512 \RequirePackage{latex-lab-testphase-block}
1513 ⟨/latex-lab⟩
```

7 Documentation from first prototype implementations

7.1 Open questions

- Existing questions — moved to issues —

7.2 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.

7.3 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.

8   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 LaTeX 2_ε 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 LaTeX 2_ε 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 template[2] 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.

---

[1] Possibly also *endblock* to deal with decorations at the end?

[2] A 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.

- 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 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.

---

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

[4]This 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.

61

63

64

65