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

LaTeX Project*

v0.8n 2024-03-23

Abstract

Contents

---

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 2_ε name).

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

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

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

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

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

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

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

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

2.2.2 The block template 'display'

Attributes:

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

beginsep (*skip*)   Default: \topsep

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

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

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

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

beginpenalty (*integer*)   Default: \@beginparpenalty

endpenalty (*integer*)   Default: \@endparpenalty

leftmargin (*length*)   Default: \leftmargin

rightmargin (*length*)   Default: \rightmargin

parindent (*length*)   Default: \listparindent

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

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

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

2.2.3 The para template 'std'

Attributes:

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

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

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

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

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

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

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

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

para-class (*tokenlist*)   Default: justify

2.2.4 The `list` template 'std'

Attributes:

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

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

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

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

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

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 possiblly involving inner <text-unit> tags
  </display element structure>
  <text>
```

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

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

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

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

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

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

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

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

will be tagged as follows:

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

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

3.2 Tagging recipes

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

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_ε 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]
```

Generell 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:
```

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.

> verify that this claim is actually correct!

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

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

6.2   Object and template interfaces

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

```
28 \NewTemplateType{blockenv}{1}
29 \NewTemplateType{block}{1}
30 \NewTemplateType{para}{1}
31 \NewTemplateType{list}{1}
32 \NewTemplateType{item}{1}
```

blockenv display (*templ.*)

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

13

block display (*templ.*)

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

para std (*templ.*)

```
65 \DeclareTemplateInterface{para}{std}{1}
66 {
67    indent-width        : length = \parindent ,
68    start-skip          : skip = 0pt  ,
69    left-skip           : skip = 0pt ,
70    right-skip          : skip = 0pt ,
71    end-skip            : skip = \@flushglue ,
72    fixed-word-spaces   : boolean = false ,
73    final-hyphen-demerits : integer = 5000 ,
74    cr-cmd              : tokenlist = \@normalcr ,
75    para-class          : tokenlist = justify ,
76 }
```

list std (*templ.*)

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

item std (*templ.*)

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

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

6.3 Useful helper commands

This section collects expl3 commands that will be useful.

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

```
105 \cs_new_protected:Npn \__block_skip_set_to_last:N #1 {
106   \skip_set:Nn #1 { \tex_lastskip:D }
107 }
```

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

```
108 \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:.)

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

6.3.1 Debugging

\g__block_debug_bool

```
110 \bool_new:N \g__block_debug_bool
```

(*End of definition for* \g__block_debug_bool.)

__block_debug:n
__block_debug_typeout:n

```
111 \cs_new_eq:NN \__block_debug:n \use_none:n
112 \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:

```
113 \cs_new_protected:Npn \block_debug_on:
114   {
115     \bool_gset_true:N \g__block_debug_bool
116     \__block_debug_gset:
117   }
118 \cs_new_protected:Npn \block_debug_off:
119   {
120     \bool_gset_false:N \g__block_debug_bool
121     \__block_debug_gset:
122   }
```

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

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

\DebugBlocksOn
\DebugBlocksOff

```
130  \cs_new_protected:Npn \DebugBlocksOn  { \block_debug_on:  }
131  \cs_new_protected:Npn \DebugBlocksOff { \block_debug_off: }

132  \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 2$_\varepsilon$'s `trivlist` except that there aren't degenerated lists and thus have no hidden `\item` inside.

displayblock (*env.*)

```
133  \NewDocumentEnvironment{displayblock}{ !O{} }
134    { \UseInstance{blockenv}{displayblock} {#1} }
135    { \endblockenv }
```

displayblockflattened (*env.*)

```
136  \NewDocumentEnvironment{displayblockflattened}{ !O{} }
137    { \UseInstance{blockenv}{displayblockflattened} {#1} }
138    { \endblockenv }
```

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

```
139  \AddToHook{begindocument/before}{
140    \RenewDocumentEnvironment{center} { !O{} }
141    { \UseInstance{blockenv}{center}{#1} }
142    { \endblockenv }

143    \RenewDocumentEnvironment{flushright} { !O{} }
144    { \UseInstance{blockenv}{flushright}{#1} }
145    { \endblockenv }

146    \RenewDocumentEnvironment{flushleft} { !O{} }
147    { \UseInstance{blockenv}{flushleft}{#1} }
148    { \endblockenv }
149  }
```

16

6.4.2 Display quote environments

```
150 \AddToHook{begindocument/before}{
151   \RenewDocumentEnvironment{quote}{ !O{} }
152     { \UseInstance{blockenv}{quote} {#1} }
153     { \endblockenv }
154   \RenewDocumentEnvironment{quotation}{ !O{} }
155     { \UseInstance{blockenv}{quotation} {#1} }
156     { \endblockenv }
157 }
```

6.4.3 Verbatim environments

```
158 \AddToHook{begindocument/before}{
159   \RenewDocumentEnvironment{verbatim}{ !O{} }
160     { \UseInstance{blockenv}{verbatim} {#1}
```

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

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

Helper commands for verbatim

This code resembles the LaTeX 2$_\varepsilon$ verbatim implementation with a slight twist: in LaTeX 2$_\varepsilon$ 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.

```
172 ⟨@@=⟩
173 \def\legacyverbatimsetup{%
174   \language\l@nohyphenation
175   \@tempswafalse
176   \def\par{%
177     \if@tempswa
178       \leavevmode \null {\@@par}\penalty\interlinepenalty
179     \else
180       \@tempswatrue
181       \ifhmode{\@@par}\penalty\interlinepenalty\fi
182     \fi}%
183   \let\do\@makeother \dospecials
184   \obeylines \verbatim@font \@noligs
```

17

```
185    \everypar \expandafter{\the\everypar \unpenalty}%
186    \tl_set:Nn \l__tag_para_main_tag_tl {codeline}
187    \tagtool{paratag=Code}%   oder faster: \tl_set:Nn\l__tag_para_tag_tl{Code}
188 }
189 ⟨@@=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.

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

(*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.*)
```
197 \AddToHook{begindocument/before}{
198   \RenewDocumentEnvironment{itemize}{!O{}}
199     { \UseInstance{blockenv}{itemize} {#1} }
200     { \endblockenv }
201   \RenewDocumentEnvironment{enumerate}{!O{}}
202     { \UseInstance{blockenv}{enumerate} {#1} }
203     { \endblockenv }
204   \RenewDocumentEnvironment{description}{!O{}}
205     { \UseInstance{blockenv}{description} {#1} }
206     { \endblockenv }
207 }
```

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.

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

18

```
222        }
223      { \endblockenv }
224    }
```

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

```
225  \AddToHook{begindocument/before}{
226    \RenewDocumentEnvironment{list}{O{} m m }
227      {
```

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.

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

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

```
234  \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`.

```
235  \cs_new:Npn \legacylistsetupcode {
```

Reset values to defaults:

```
236        \dim_zero:N \listparindent
237        \dim_zero:N \rightmargin
238        \dim_zero:N \itemindent
```

By default a `list` environment is not numbered:

```
239        \tl_set:Nn \@listctr {}
240        \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.

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

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

```
242        \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.

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

trivlist (*env.*)

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

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.

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

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

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

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

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

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

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

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

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

`\@begintheorem`   The `\@thm` command expands to either `\@beginthorem` or `\@opargbegintheorem`. For
`\@opargbegintheorem`   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.

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

322    \__block_start_para_structure_unconditionally:n { \PARALABEL }
```

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

349    \__block_start_para_structure_unconditionally:n { \PARALABEL }

350      \itshape
351      \hskip\labelsep
352      \ignorespaces
353 }

354 \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` LaTeX2_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`:

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

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

blockenv display (*templ.*)

```
357 \DeclareTemplateCode{blockenv}{display}{1}
358 {
```

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

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

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

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.

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

```
403          \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.

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

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

```
407   \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.

```
408   \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).

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

```
415   \tl_if_empty:NF \l__block_para_instance_tl
416     {
417        \__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.

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

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.

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

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.

```
433   \l__block_final_code_tl
434 }
```

`\l__block_flattened_level_int` Count the levels of nested blockenvs starting with the first that is "flattened".

```
435 \int_new:N \l__block_flattened_level_int
```

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

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

```
436 \newcounter{maxblocklevels}
437 \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

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

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

```
440   \bool_if:NT \l__block_level_incr_bool
441     { \int_gdecr:N \g_block_nesting_depth_int }
```

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

```
442   \legacy_if:nT { @inlabel }
443     {
444       \mode_leave_vertical:
445       \legacy_if_gset_false:n { @inlabel }
446     }
```

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)

```
447   \legacy_if:nT { @newlist }
448     {
449       \@noitemerr
450       \legacy_if_gset_false:n { @newlist }
451     }
452   \mode_if_horizontal:TF
453     { \__block_skip_remove_last: \__block_skip_remove_last: \par }
454     { \@inmatherr{\end{\@currenvir}} }
```

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

```
455   \__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 2_ε list environment have been doing.

> some redesign/extensions here?

```
456 %   \__block_debug_typeout:n{@noparlist =
457 %                      \legacy_if:nTF { @noparlist }{true}{false}}
458   \legacy_if:nF { @noparlist }
459     {
460       \__block_skip_set_to_last:N \l_tmpa_skip
```

25

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

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

```
468  %           \legacy_if_gset_true:n { @endpe }
469          }
```

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.

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

(*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.

```
472  \cs_new:Npn \__kernel_displayblock_end: {
473    \__block_debug_typeout:n{\detokenize{__kernel_displayblock_end:}}
474  }
```

(*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`).

```
475  \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.

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

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

decide

6.5.2 Implementation of para templates . . .

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

6.5.3 Implementation of block templates . . .

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

> generalize heading usage
> (or drop?)

```
513   \tl_if_blank:oF \l__block_heading_tl
514     { \mode_leave_vertical: \textbf{\l__block_heading_tl} } % TODO customize
```

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.

```
515   \legacy_if:nT { @noskipsec } { \mode_leave_vertical: }
516   \skip_set:Nn \l__block_topsepadd_skip { \topsep }
517   \mode_if_vertical:TF
518     {
519       \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.

```
520         \__kernel_displayblock_beginpar_vmode:
521       }
522       {
```

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.

```
523         \__block_skip_remove_last: \__block_skip_remove_last:
524         \__kernel_displayblock_beginpar_hmode:w \par
525       }
```

Now we are back to legacy list implementation . . .

```
526     \legacy_if:nTF { @inlabel }
527       {
528       \legacy_if_set_true:n { @noparitem }
529       \legacy_if_set_true:n { @noparlist }
530       }
531       {
532       \legacy_if:nT { @newlist } { \@noitemerr }
533       \legacy_if_set_false:n { @noparlist }
534       \skip_set_eq:NN \l__block_effective_top_skip \l__block_topsepadd_skip
535       }
536     \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*.

```
537     \skip_zero:N \leftskip
538     \skip_set_eq:NN \rightskip \@rightskip
539     \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.

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

28

```
555    \skip_set_eq:NN \@outerparskip \parskip
556    \skip_set_eq:NN \parskip \parsep
557    \dim_set_eq:NN \parindent \listparindent
558    \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
559    \dim_add:Nn \@totalleftmargin { \leftmargin }
560    \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.

```
561    \__kernel_displayblock_begin:
```

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

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

<!-- callout: document 2e logic used here (pointing to line 570) -->

Extra keys to support enumitem conventions:
```
588  \keys_define:nn { template/block/display }
589  {
590    ,topsep        .skip_set:N = \topsep
591    ,partopsep     .skip_set:N = \partopsep
592    ,listparindent .skip_set:N = \listparindent
593  }
```

__kernel_displayblock_begin:
__kernel_displayblock_beginpar_hmode:w
__kernel_displayblock_beginpar_vmode:

The internal kernel hooks for tagging.

```
594  \cs_new:Npn \__kernel_displayblock_begin: {
595    \__block_debug_typeout:n{\detokenize{__kernel_displayblock_begin:}}
596  }
```

29

```
597 \cs_new:Npn \__kernel_displayblock_beginpar_hmode:w {
598   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_beginpar_hmode:w}}}
599 }

600 \cs_new:Npn \__kernel_displayblock_beginpar_vmode: {
601   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_beginpar_vmode:}}}
602 }
```

*(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`.

```
603 \tl_new:N \@itemlabel          % should have a top-level definition
604 \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.

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

Has this list a counter name defined in the instance?

```
624   \tl_if_empty:NTF \l__block_counter_tl
625     {
```

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.

```
626     \tl_if_blank:oF \@listctr
627       {
```

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.

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

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

```
634        { \@nmbrlistfalse }
635      }
```

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.

```
636      {
637        \@nmbrlisttrue
638        \tl_set_eq:NN \@listctr \l__block_counter_tl
639        \bool_if:NF \l__block_resume_bool
640          {
641            \int_gset:cn{ c@ \@listctr }
642                { \l__block_counter_start_int - 1 }
643          }
644      }
```

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.

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

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.

```
649      \legacy_if_gset_true:n { @newlist }

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

Extra keys to support enumitem conventions:

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

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.

```
662 \keys_define:nn { template/item/std }
663                 { label .tl_set:N = \l__block_label_given_tl }
664 \DeclareTemplateCode{item}{std}{1}
665   {
666       counter-label    = \__block_counter_label:n ,
667       counter-ref      = \__block_counter_ref:n ,

668       label-ref        = \__block_label_ref:n ,
669       label-autoref    = \__block_label_autoref:n ,
670       label-format     = \__block_label_format:n ,
671       label-strut      = \l__block_label_strut_bool ,
672       label-boxed      = \l__block_label_boxed_bool ,
673       next-line        = \l__block_next_line_bool ,
674       text-font        = \l__block_text_font_tl ,
675       compatibility    = \l__block_item_compatibility_bool ,
```

This probably needs a different implementation (and needs completing)

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

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.

```
683   {
684     \__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.

```
685     \tl_set_eq:NN \l__block_label_given_tl \c_novalue_tl
686     \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[]`.

```
687     \tl_if_novalue:oTF \l__block_label_given_tl
688       {
```

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

wrong.

```
689        \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
690        \bool_if:NTF \l__block_item_compatibility_bool   % not sure that conditional
691                                          % makes sense
692          { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}\@itemlabel } } % TODO ?
693          { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}\__block_counter_label:n
694        }
695        {
696          \__block_debug_typeout:n{item~ with~ optional}
697          \__block_make_label_box:n { \l__block_label_given_tl } }
698      \bool_if:nT
699        {
700          \l__block_label_boxed_bool
701          && \dim_compare_p:n { \box_wd:N \l__block_one_label_box <= \linewidth } % TODO: is \
702        }
703        {
704          \dim_compare:nNnT
705            { \box_wd:N \l__block_one_label_box } < \labelwidth
706            {
707              \hbox_set_to_wd:Nnn \l__block_one_label_box { \labelwidth }
708                {
709                  \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.

```
710 %                    \hbox_unpack_drop:N \l__block_one_label_box    %TODO: customize?
711                  \box_use_drop:N \l__block_one_label_box

712                  \exp_after:wN \use_ii:nn \l__block_item_align_tl
713                }
714            }
```

Add another box level to the label box:

```
715          \hbox_set:Nn \l__block_one_label_box
716                    { \box_use_drop:N \l__block_one_label_box }
717        }
718      \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \labelwidth
719        { \bool_set_true:N \l__block_long_label_bool }
720        { \bool_set_false:N \l__block_long_label_bool }
721      \hbox_gset:Nn \g__block_labels_box
722        {
723          \hbox_unpack_drop:N \g__block_labels_box
724          \skip_horizontal:n { \itemindent - \labelsep - \labelwidth }
725          \hbox_unpack_drop:N \l__block_one_label_box
726          \skip_horizontal:n { \labelsep }
727          \bool_if:NT \l__block_next_line_bool
728            { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
729          % version of \newline inside an hbox that will be unpacked
730        }
731      % \skip_set_eq:NN \parsep \l__block_item_parsep_skip TODO??? FMi
732                                          % what's that?
733      \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.

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

\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.

```
736 \box_new:N \l__block_one_label_box
737 \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.

```
738 \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).

```
739 \cs_new_protected:Npn \__block_make_label_box:n #1
740    {
741        \hbox_set:Nn \l__block_one_label_box
742            {
```

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

```
743                \__kernel_list_label_begin:
744                \__block_label_format:n
745                  {
746                    \bool_if:NT \l__block_label_strut_bool { \strut }
747                    \bool_if:NTF \l__block_legacy_support_bool
748                                \makelabel
749                                \use:n
750                          {#1}
751                  }
```

And what gets opened also needs closing:

```
752                \__kernel_list_label_end:
753            }
754    }
```

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

__kernel_list_label_begin: If we aren't doing tagging the kernel hooks do nothing.
__kernel_list_label_end:
```
755 \cs_new_eq:NN \__kernel_list_label_begin: \prg_do_nothing:
756 \cs_new_eq:NN \__kernel_list_label_end:   \prg_do_nothing:
```

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

`\__block_item_everypar:`
`\__block_item_everypar_std:`

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.

```
757 \cs_new_eq:NN \__block_item_everypar: \prg_do_nothing:
```

```
758 \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!

```
759 \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.

```
760 \cs_new_protected:Npn \__block_item_everypar_std: {
761     \__block_debug_typeout:n{item~ everypar \on@line }
762     \legacy_if_set_false:n { @minipage }
763     \legacy_if_gset_false:n { @newlist }
764     \legacy_if:nT { @inlabel }
765         {
766             \legacy_if_gset_false:n { @inlabel }
767             \box_if_empty:NT \g_para_indent_box { \kern - \itemindent }
768             \para_omit_indent:

769             \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:

```
770             \__kernel_list_label_after:

771             \penalty \c_zero_int
772         }
773     \legacy_if:nTF { @nobreak }
774         {
775             \legacy_if_gset_false:n { @nobreak }
776             \int_set:Nn \clubpenalty { 10000 }
777         }
778         {
779             \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.

```
780             \cs_set_eq:NN \__block_item_everypar: \prg_do_nothing:
781         }
782 }
```

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

`\__kernel_list_label_after:`

```
783 \cs_new_eq:NN \__kernel_list_label_after: \prg_do_nothing:
```

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

`\l__block_tmpa_skip`

```
784 \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 2ε'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.

```
785 \skip_new:N \l__block_topsepadd_skip
786 \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. Distingusih 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`.

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

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

```
791       \cs_if_free:NTF \__block_item_instance:n
792         {
793           \@latex@error{Lonely~\string\item--perhaps~a~missing~
794           list~environment}\@ehc
795         }
796         {
797           \legacy_if:nTF { @newlist }
798             { \__kernel_list_item_begin: }
799             { \__block_inter_item:            }
```

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

```
800           \tl_if_novalue:nTF {#1}          % avoids reparsing label={}
801             { \__block_item_instance:n { } }
802             { \__block_item_instance:n {#1} }
```

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

```
803           \legacy_if_gset_true:n { @inlabel }
804           \ignorespaces
805         }
806     }
807 }
```

(*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.

```
808 \cs_new_protected:Npn \__block_inter_item: {
809   \legacy_if:nT { @inlabel }
810                  { \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).

```
811    \mode_if_horizontal:T { \__block_skip_remove_last:
812                             \__block_skip_remove_last: \par }
```

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

```
813    \__kernel_list_item_end:
814    \__kernel_list_item_begin:

815    \addpenalty \@itempenalty
816    \addvspace \itemsep
817  }
```

(*End of definition for* __block_inter_item:.)

__kernel_list_item_begin:
__kernel_list_item_end:

```
818 \cs_new_eq:NN \__kernel_list_item_begin: \prg_do_nothing:
819 \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.

```
820 \cs_new:Npn \__block_recipe_basic: {
821    \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
822                                         \__block_beginpar_hmode:N
823    \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
824                                         \__block_beginpar_vmode:
825    \let \__kernel_displayblock_begin:        \prg_do_nothing:
826    \let \__kernel_displayblock_end:          \prg_do_nothing:
```

End environment \par handling:

```
827    \socket_assign_plug:nn{tagsupport/block-endpe}{on}
828  }
```

(*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.

```
829 \cs_new:Npn \__block_recipe_standalone: {
830    \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
831                                         \prg_do_nothing:
832    \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
833                                         \prg_do_nothing:
834    \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
835    \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:
```

37

End environment `\par` handling:

```
836    \socket_assign_plug:nn{tagsupport/block-endpe}{off}
837    \tl_if_empty:NTF \l__block_tag_name_tl
838       { \tl_set:Nn    \l__block_tag_inner_tag_tl {Sect}              }
839       { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
840  }
```

(*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`.

```
841  \cs_new:Npn \__block_recipe_standard:
842  {
843    \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
844                                          \__block_beginpar_hmode:N
845    \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
846                                          \__block_beginpar_vmode:
847    \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
848    \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:
```

End environment `\par` handling:

```
849    \socket_assign_plug:nn{tagsupport/block-endpe}{on}
850    \tl_if_empty:NTF \l__block_tag_name_tl
851       { \tl_set:Nn    \l__block_tag_inner_tag_tl {Figure}            }
852       { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
853  }
```

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

`\l__block_tag_inner_tag_tl`

```
854  \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.

38

```
855 \cs_new:Npn \__block_recipe_list:
856 {
857   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
858                                          \__block_beginpar_hmode:N
859   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
860                                          \__block_beginpar_vmode:
861   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_list_begin:
862   \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)).

```
863   \cs_set_eq:NN \__kernel_list_item_begin:   \__block_list_item_begin:
864   \cs_set_eq:NN \__kernel_list_item_end:     \__block_list_item_end:
```

End environment \par handling:

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

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

```
866   \tl_if_empty:NTF \l__block_tag_name_tl
867       { \tl_set:Nn    \l__tag_L_tag_tl {L}              }
868       { \tl_set_eq:NN \l__tag_L_tag_tl \l__block_tag_name_tl }
869   \tl_if_empty:NTF \l__block_tag_class_tl
870       { \tl_set:Nn    \l__tag_L_attr_class_tl {}             }
871       { \tl_set_eq:NN \l__tag_L_attr_class_tl \l__block_tag_class_tl }
872 }
```

(*End of definition for* __block_recipe_list:.)

6.7 Blockenv instances

6.7.1 Basic instances

blockenv displayblock (*inst.*)

```
873 \DeclareInstance{blockenv}{displayblock}{display}
874 {
875   env-name        = displayblock,
876   tag-name        = ,
877   tag-class       = ,
878   tagging-recipe = standard,
879   inner-level-counter  = ,
880   level-increase = false,
881   setup-code      = ,
882   block-instance = displayblock ,
883   inner-instance = ,
884 }
```

nv displayblockflattened (*inst.*)

```
885 \DeclareInstance{blockenv}{displayblockflattened}{display}
886 {
887   env-name        = displayblockflattened,
888   tag-name        = ,
889   tag-class       = ,
890   tagging-recipe = basic,
```

39

```
891   inner-level-counter  = ,
892   level-increase = false,
893   setup-code       = ,
894   block-instance = displayblock ,
895   para-flattened = true ,
896   inner-instance = ,
897 }
```

blockenv center (*inst.*)

```
898 \DeclareInstance{blockenv}{center}{display}
899 {
900   env-name         = center,
901   tag-name         = ,
902   tag-class        = ,
903   tagging-recipe = basic,
904   inner-level-counter  = ,
905   level-increase = false,
906   setup-code       = ,
907   block-instance = displayblock ,
908   para-flattened = true ,
909   para-instance    = center ,
910   inner-instance = ,
911 }
```

blockenv flushleft (*inst.*)

```
912 \DeclareInstance{blockenv}{flushleft}{display}
913 {
914   env-name         = flushleft,
915   tag-name         = ,
916   tag-class        = ,
917   tagging-recipe = basic,
918   inner-level-counter  = ,
919   level-increase = false,
920   setup-code       = ,
921   block-instance = displayblock ,
922   para-flattened = true ,
923   para-instance    = raggedright ,
924   inner-instance = ,
925 }
```

blockenv flushright (*inst.*)

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

6.7.2 Blockquote instances

blockenv quotation (*inst.*)

```
940 \DeclareInstance{blockenv}{quotation}{display}
941 {
942   env-name        = quotation,
943   tag-name        = quotation,
944   tag-class       = ,
945   tagging-recipe = standard,
946   inner-level-counter  = ,
947   level-increase = true,
948   setup-code      = ,
949   block-instance = quotationblock ,
950   inner-instance =  ,
951 }
```

blockenv quote (*inst.*)

```
952 \DeclareInstance{blockenv}{quote}{display}
953 {
954   env-name        = quote,
955   tag-name        = quote,
956   tag-class       = ,
957   tagging-recipe = standard,
958   inner-level-counter  = ,
959   level-increase = true,
960   setup-code      = ,
961   block-instance = quoteblock ,
962   inner-instance =  ,
963 }
```

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

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

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

```
984 %   level-increase = true,
985 %   setup-code     = \setlength\rightmargin{\leftmargin} ,
986 %   block-instance = displayblock ,
987 %   inner-instance =  ,
988 % }
989 \DeclareInstance{blockenv}{theorem}{display}
990 {
991   env-name        = theorem-like,
992   tag-name        = theorem-like,
993   tag-class       = ,
994   tagging-recipe = standalone,
995   inner-level-counter  = ,
996   level-increase = false,
997   setup-code      = ,
998   block-instance = displayblock ,
999 %  inner-instance-type = innerblock ,
1000 %  inner-instance = theorem,
1001 }
```

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.

```
1002 \DeclareInstance{blockenv}{verbatim}{display}
1003 {
1004   env-name        = verbatim,
1005   tag-name        = verbatim,
1006   tag-class       = ,
1007   tagging-recipe = standard,
1008   inner-level-counter  = ,
1009   level-increase = false,
1010   setup-code      = ,
1011   block-instance = verbatimblock ,
1012   inner-instance = ,
1013   final-code      = \legacyverbatimsetup ,
1014 }
```

6.7.4   Standard list instances

blockenv itemize (*inst.*)

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

```
1025    block-instance = list ,
1026    inner-instance = itemize ,
1027 }
```

blockenv enumerate (*inst.*)

```
1028 \DeclareInstance{blockenv}{enumerate}{display}
1029 {
1030    env-name           = enumerate,
1031    tag-name           = enumerate,
1032    tag-class          = enumerate,
1033    tagging-recipe     = list,
1034    level-increase     = true,
1035    setup-code         = ,
1036    block-instance     = list ,
1037    inner-level-counter = \@enumdepth,
1038    max-inner-levels   = 4,
1039    inner-instance     = enum ,
1040 }
```

blockenv description (*inst.*)

```
1041
1042 \DeclareInstance{blockenv}{description}{display}
1043 {
1044    env-name        = description,
1045    tag-name        = description,
1046    tag-class       = description,
1047    tagging-recipe  = list,
1048    inner-level-counter  = ,
1049    level-increase  = true,
1050    setup-code      = ,
1051    block-instance  = list ,
1052    inner-instance  = description ,
1053 }
```

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

```
1054 \DeclareInstance{blockenv}{list}{display}
1055 {
1056    env-name        = list,
1057    tag-name        = list,
1058    tag-class       = ,
1059    tagging-recipe  = list,
1060    level-increase  = true,
1061    setup-code      = \legacylistsetupcode ,
1062    block-instance  = list ,
1063    inner-level-counter  = ,
1064    inner-instance  = legacy ,
1065 }
```

6.8 Block instances

6.8.1 Displayblock instances

We provide 6 nesting levels (as in LaTeX 2$_\varepsilon$). If you want to provide more you need to change the maxblocklevels counter, offer further displayblock-xx instances but also

define further (legacy) \list⟨*romannumeral*⟩ commands for the defaults. If not, then the settings from the previous level are reused automatically—which may or may not be good enough).

```
1066 \setcounter{maxblocklevels}{6}
```

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

```
1067 \DeclareInstance{block}{displayblock-0}{display}
1068   {
1069     leftmargin       = 0pt ,
1070     parindent        = 0pt ,
1071   }
```

```
1072 \DeclareInstanceCopy{block}{displayblock-1}{displayblock-0}
1073 \DeclareInstanceCopy{block}{displayblock-2}{displayblock-0}
1074 \DeclareInstanceCopy{block}{displayblock-3}{displayblock-0}
1075 \DeclareInstanceCopy{block}{displayblock-4}{displayblock-0}
1076 \DeclareInstanceCopy{block}{displayblock-5}{displayblock-0}
1077 \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.

```
1078 \DeclareInstance{block}{verbatimblock-0}{display}
1079   {
1080     leftmargin       = 0pt ,
1081     parindent        = 0pt ,
1082     par-skip         = 0pt ,
1083   }
```

```
1084 \DeclareInstanceCopy{block}{verbatimblock-1}{verbatimblock-0}
1085 \DeclareInstanceCopy{block}{verbatimblock-2}{verbatimblock-0}
1086 \DeclareInstanceCopy{block}{verbatimblock-3}{verbatimblock-0}
1087 \DeclareInstanceCopy{block}{verbatimblock-4}{verbatimblock-0}
1088 \DeclareInstanceCopy{block}{verbatimblock-5}{verbatimblock-0}
1089 \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.

Default layout is to indent equaly from both side.

```
1090 \DeclareInstance{block}{quoteblock-1}{display}
1091   { rightmargin = \KeyValue{leftmargin} }
```

```
1092 \DeclareInstanceCopy{block}{quoteblock-2}{quoteblock-1}
1093 \DeclareInstanceCopy{block}{quoteblock-3}{quoteblock-1}
1094 \DeclareInstanceCopy{block}{quoteblock-4}{quoteblock-1}
1095 \DeclareInstanceCopy{block}{quoteblock-5}{quoteblock-1}
1096 \DeclareInstanceCopy{block}{quoteblock-6}{quoteblock-1}
```

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

block quotationblock-2 (*inst.*)

block quotationblock-3 (*inst.*)

block quotationblock-4 (*inst.*)

block quotationblock-5 (*inst.*)

block quotationblock-6 (*inst.*)

```
1097 \DeclareInstance{block}{quotationblock-1}{display}
1098   { parindent = 1.5em , rightmargin = \KeyValue{leftmargin} }
1099 \DeclareInstanceCopy{block}{quotationblock-2}{quotationblock-1}
1100 \DeclareInstanceCopy{block}{quotationblock-3}{quotationblock-1}
1101 \DeclareInstanceCopy{block}{quotationblock-4}{quotationblock-1}
1102 \DeclareInstanceCopy{block}{quotationblock-5}{quotationblock-1}
1103 \DeclareInstanceCopy{block}{quotationblock-6}{quotationblock-1}
```

6.8.4 Block instances for the standard lists

block list-1 (*inst.*) The block instances for the various list environments use the same underlying instance

block list-2 (*inst.*) (well by default) and nothing needs to be set up specifically (because that is already done

block list-3 (*inst.*) in the legacy \list⟨*romannumeral*⟩ unless a different layout is wanted.

block list-4 (*inst.*)

block list-5 (*inst.*)

block list-6 (*inst.*)

```
1104 \DeclareInstance{block}{list-1}{display}{
1105 %   heading           = ,
1106 %   beginsep          = \topsep ,
1107 %   begin-par-skip    = \partopsep ,
1108 %   par-skip          = \parsep ,
1109 %   end-skip          = \KeyValue{beginsep} ,
1110 %   end-par-skip      = \KeyValue{begin-par-skip} ,
1111 %   beginpenalty      = \UseName{@beginparpenalty} ,
1112 %   endpenalty        = \UseName{@endparpenalty} ,
1113 %   leftmargin        = \leftmargin ,
1114 %   rightmargin       = \rightmargin ,
1115 %   parindent         = \listparindent ,
1116 }
1117 \DeclareInstance{block}{list-2}{display}{}
1118 \DeclareInstance{block}{list-3}{display}{}
1119 \DeclareInstance{block}{list-4}{display}{}
1120 \DeclareInstance{block}{list-5}{display}{}
1121 \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.

list itemize-1 (*inst.*) For `itemize` environments this is all we need to do and we refer back to the external

list itemize-2 (*inst.*) definitions rather than defining the `item-label` code in the instance to ensure that old

list itemize-3 (*inst.*) documents still work.

list itemize-4 (*inst.*)

```
1122 \DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi }
1123 \DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii }
1124 \DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
1125 \DeclareInstance{list}{itemize-4}{std}{ item-label = \labelitemiv }
```

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

list enumerate-2 (*inst.*) use on every level.

list enumerate-3 (*inst.*)

list enumerate-4 (*inst.*)

```
1126 \DeclareInstance{list}{enum-1}{std}
1127   { item-label = \labelenumi ,   counter = enumi }
1128 \DeclareInstance{list}{enum-2}{std}
1129   { item-label = \labelenumii ,   counter = enumii }
```

```
1130 \DeclareInstance{list}{enum-3}{std}
1131   { item-label = \labelenumiii , counter = enumiii }
1132 \DeclareInstance{list}{enum-4}{std}
1133   { item-label = \labelenumiv ,  counter = enumiv }
```

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

```
1134 \DeclareInstance{list}{legacy}{std} {
1135   item-instance = basic ,
1136   legacy-support = true ,
1137 }
```

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

```
1138 \DeclareInstance{list}{description}{std} { item-instance = description }
```

6.10   Item instances

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

```
1139 \DeclareInstance{item}{basic}{std}
1140   {
1141     label-align = right ,
1142   }
1143 \DeclareInstance{item}{description}{std}
1144   {
1145     label-format = \normalfont\bfseries #1 ,
1146     label-align = left
1147   }
```

6.11   Para instances

```
1148 \tag_if_active:T
1149 {
1150   \tagpdfsetup
1151     {
1152       role/new-attribute = {justify}    {/O /Layout /TextAlign/Justify},
1153       role/new-attribute = {center}     {/O /Layout /TextAlign/Center},
1154       role/new-attribute = {raggedright}{/O /Layout /TextAlign/Start},
1155       role/new-attribute = {raggedleft} {/O /Layout /TextAlign/End},
1156     }
1157 }
```

para center (*inst.*)

```
1158 \DeclareInstance{para}{center}{std}
1159 {
1160   indent-width        = 0pt ,
1161   start-skip          = 0pt ,
1162   left-skip           = \@flushglue ,
```

46

```
1163    right-skip            = \@flushglue ,
1164    end-skip              = \z@skip ,
1165    final-hyphen-demerits = 0 ,
1166    cr-cmd                = \@centercr ,
1167    para-class            = center ,
1168 }
1169 \DeclareInstance{para}{raggedright}{std}
1170 {
1171    indent-width          = 0pt ,
1172    start-skip            = 0pt ,
1173    left-skip             = \z@skip ,
1174    right-skip            = \@flushglue ,
1175    end-skip              = \z@skip ,
1176    final-hyphen-demerits = 0 ,
1177    cr-cmd                = \@centercr ,
1178    para-class            = raggedright ,
1179 }
1180 \DeclareInstance{para}{raggedleft}{std}
1181 {
1182    indent-width          = 0pt ,
1183    start-skip            = 0pt ,
1184    left-skip             = \@flushglue ,
1185    right-skip            = \z@skip ,
1186    end-skip              = \z@skip ,
1187    final-hyphen-demerits = 0 ,
1188    cr-cmd                = \@centercr ,
1189    para-class            = raggedleft ,
1190 }
1191 \DeclareInstance{para}{justify}{std}
1192 {
1193 %  indent-width          = 0pt ,
1194    start-skip            = 0pt ,
1195    left-skip             = \z@skip ,
1196    right-skip            = \z@skip ,
1197    end-skip              = \@flushglue ,
1198    final-hyphen-demerits =  5000 ,
1199    cr-cmd                = \@normalcr ,
1200    para-class            = justify ,
1201 }

1202 \DeclareRobustCommand\centering  {\UseInstance{para}{center}{}}
1203 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}{}}
1204 \DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}{}}
1205 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}{}}
1206
1207 \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!

```
1208 \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.

```
1209    \cs_set:Npn \__block_beginpar_vmode: {
1210        \__block_debug_typeout:n
1211          { @endpe = \legacy_if:nTF { @endpe }{true}{false}
1212            \on@line }
1213    \legacy_if:nTF { @endpe }
1214      {
1215        \legacy_if_gset_false:n { @endpe }
1216      }
```

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.

```
1217      {
1218        \int_compare:nNnT \l__block_flattened_level_int < 2
1219          {
1220            \__tag_gincr_para_main_begin_int:
1221            \tag_struct_begin:n
1222              {
1223                tag=\l__tag_para_main_tag_tl,
1224                attribute-class=\l__tag_para_main_attr_class_tl,
1225              }
1226            \__tag_para_main_store_struct:
1227          }
1228      }
1229    }
```

(*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.

```
1230    \cs_set:Npn \__block_beginpar_hmode:N #1
1231      {
1232        \tag_mc_end:
1233        \__tag_gincr_para_end_int:
1234        \__block_debug_typeout:n{increment~ /P \on@line }
1235        \bool_if:NT \l__tag_para_show_bool
1236          { \tag_mc_begin:n{artifact}
1237            \rlap{\color_select:n{red}\tiny\ \int_use:N\g__tag_para_end_int}
1238            \tag_mc_end:
1239          }
1240        \tag_struct_end:
1241        \tagpdfparaOff \par \tagpdfparaOn
1242      }
```

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

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

```
1243 \cs_set:Npn \__kernel_displayblock_doendpe: {
1244     \bool_if:NT \l__tag_para_bool
1245         {
```

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.

```
1246         \legacy_if:nT { @endpe }
1247             {
```

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.

```
1248             \__block_debug_typeout:n
1249                 { flattened= \bool_if:NTF
1250                             \l__tag_para_flattened_bool {true}{false}
1251                 \on@line }
1252             \bool_if:NF \l__tag_para_flattened_bool
1253                 {
1254                 \__block_debug_typeout:n{Structure-end~
1255                     \l__tag_para_main_tag_tl\space after~ displayblock \on@line }
1256                 \__tag_gincr_para_main_end_int:
1257                 \tag_struct_end: %text-unit
1258                 }
1259             }
1260         }
1261 }
```

(*End of definition for* __kernel_displayblock_doendpe:.)

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).

```
1262 \RemoveFromHook{para/begin}[tagpdf]
```

```
1263 \AddToHook{para/begin}[tagpdf]{
1264     \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).

```
1265         \legacy_if:nF { @inlabel }
1266             {
```

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.

```
1267              \__block_start_para_structure:n { \PARALABEL }
1268            }
1269        }
1270 }
```

```
1271 \cs_new_protected:Npn \__block_start_para_structure:n #1 {
1272    \__block_debug_typeout:n
1273      { @endpe = \legacy_if:nTF { @endpe }{true}{false}
1274        \on@line }
1275    \legacy_if:nF { @endpe }
1276      {
1277        \bool_if:NF \l__tag_para_flattened_bool
1278          {
1279            \__tag_gincr_para_main_begin_int:
1280            \tag_struct_begin:n
1281              {
1282                tag=\l__tag_para_main_tag_tl,
1283                attribute-class=\l__tag_para_main_attr_class_tl,
1284              }
1285            \__tag_para_main_store_struct:
1286          }
1287      }
1288    \__tag_gincr_para_begin_int:
1289    \__block_debug_typeout:n{increment~ P \on@line }
1290    \tag_struct_begin:n
1291      {
1292        tag=\l__tag_para_tag_tl
1293        ,attribute-class=\l__tag_para_attr_class_tl
1294      }
1295    \__tag_check_para_begin_show:nn {green}{#1}
1296    \tag_mc_begin:n {}
1297 }
```

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

```
1298 \cs_new_protected:Npn \__block_start_para_structure_unconditionally:n #1 {
1299    \bool_if:NF \l__tag_para_flattened_bool
1300      {
1301        \__tag_gincr_para_main_begin_int:
1302        \tag_struct_begin:n
1303          {
1304            tag=\l__tag_para_main_tag_tl,
1305            attribute-class=\l__tag_para_main_attr_class_tl,
1306          }
1307        \__tag_para_main_store_struct:
1308      }
1309    \__tag_gincr_para_begin_int:
1310    \__block_debug_typeout:n{increment~ P \on@line }
1311    \tag_struct_begin:n
1312      {
1313        tag=\l__tag_para_tag_tl
1314        ,attribute-class=\l__tag_para_attr_class_tl
1315      }
```

```
1316        \__tag_check_para_begin_show:nn {green}{#1}
1317        \tag_mc_begin:n {}
1318    }

1319    \RemoveFromHook{para/end}[tagpdf]
1320    \AddToHook{para/end}
1321      {
1322        \bool_if:NT \l__tag_para_bool
1323          {
1324            \__tag_gincr_para_end_int:
1325            \__block_debug_typeout:n{increment~ /P \on@line }
1326            \tag_mc_end:
1327            \__tag_check_para_end_show:nn {red}{}
1328            \tag_struct_end:
1329            \bool_if:NF \l__tag_para_flattened_bool
1330              {
1331                \__tag_gincr_para_main_end_int:
1332                \tag_struct_end:
1333              }
1334          }
1335      }

1336    \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).

```
1337    \cs_set_protected:Npn \para_end: {
1338      \scan_stop:
1339      \mode_if_horizontal:TF {
1340        \mode_if_inner:F {
1341            \tex_unskip:D
1342            \hook_use:n{para/end}
1343            \@kernel@after@para@end
1344            \mode_if_horizontal:TF {
1345              \if_int_compare:w 11 = \tex_lastnodetype:D
1346                \tex_hskip:D \c_zero_dim
1347              \fi:
1348              \tex_par:D
1349              \hook_use:n{para/after}
1350              \@kernel@after@para@after
1351            }
1352            { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
1353        }
1354      }
1355      {
1356        \__kernel_endpe_vmode:        % should do nothing if no tagging
1357        \tex_par:D
1358      }
1359    }
1360    \cs_set_eq:NN \par       \para_end:
1361    \cs_set_eq:NN \__blockpar    \para_end:
1362    \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.

```
1363 \DeclareRobustCommand*\begin[1]{%
1364   \UseHook{env/#1/before}%
1365   \@ifundefined{#1}%
1366     {\def\reserved@a{\@latex@error{Environment~#1~undefined}\@eha}}%
1367     {\def\reserved@a{\def\@currenvir{#1}%
1368        \edef\@currenvline{\on@line}%
1369        \@execute@begin@hook{#1}%
1370        \csname #1\endcsname}}%
1371   \@ignorefalse
1372   \begingroup
1373     \__kernel_endpe_vmode:
1374     \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.

```
1375 \cs_new:Npn \__kernel_endpe_vmode: {
1376     \if@endpe \ifvmode
1377        \bool_if:NT \l__tag_para_bool
1378  {
1379    \bool_if:NF \l__tag_para_flattened_bool
1380      {
1381        \__tag_gincr_para_main_end_int:
1382        \tag_struct_end:
1383      }
1384    \@endpefalse
1385  }
1386     \fi \fi
1387 }
```

(*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

```
1388 \cs_set:Npn \__kernel_list_label_after: {
1389     \bool_if:NT \l__tag_para_bool
1390       {
1391         \__block_start_para_structure_unconditionally:n { LI- }
1392       }
1393 }
```

(*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.

```
1394 \cs_new:Npn \__block_inner_begin: {
1395   \tagstructbegin{tag=\l__block_tag_inner_tag_tl}
1396 }
```

(*End of definition for* __block_inner_begin:.)

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

```
1397 \cs_new:Npn \__block_inner_end: {
1398   \__block_debug_typeout:n{block-end \on@line}
1399   \legacy_if:nT { @endpe }
1400     {
1401       \__tag_gincr_para_main_end_int:
1402       \__block_debug_typeout:n{close~ /text-unit \on@line}
1403       \tagstructend
1404     }
1405   \tagstructend        % end inner structure
1406 }
```

(*End of definition for* __block_inner_end:.)

6.12.1 List tags

```
1407 \tl_new:N  \l__tag_L_tag_tl
1408 \tl_set:Nn \l__tag_L_tag_tl {L}
1409
1410 \tl_new:N\l__tag_L_attr_class_tl
1411 \tl_set:Nn \l__tag_L_attr_class_tl {list}
1412 \tag_if_active:T
1413 {
1414   \tagpdfsetup
1415     {
1416       role/new-attribute = {itemize}{/O /List /ListNumbering/Unordered},
1417       role/new-attribute = {enumerate}{/O /List /ListNumbering/Ordered},
1418       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.

```
1419       % default if unknown
1420       role/new-attribute = {list}{/O /List /ListNumbering/Unordered},
1421     }
1422 }
1423 \def\LItag{LI}
```

__block_list_begin: Start a list ...

```
1424 \cs_set:Npn \__block_list_begin: {
1425   \tagstructbegin
1426     {
1427       tag=\l__tag_L_tag_tl
1428       ,attribute-class=\l__tag_L_attr_class_tl
1429     }
1430 }
```

(*End of definition for* __block_list_begin:.)

__block_list_item_begin: Start tagging a list item.

```
1431 \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.

```
1432 \cs_set:Npn \__kernel_list_label_begin: {
1433 %
1434 % FMi: this needs a different logic to decide when to make the label
1435 %     an artifact (after cleaning up the \item code  ), therefore
1436 %     disabled for now
1437 %  \tl_if_empty:oTF \@itemlabel
1438 %    {
1439 %       \tag_mc_begin:n {artifact}
1440 %    }
1441 %    {
1442       \tagstructbegin{tag=Lbl}
1443       \tagmcbegin{tag=Lbl}
1444 %    }
1445 }
```

(*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.

```
1446 \cs_set:Npn \__kernel_list_label_end: {
1447   \tagmcend                                   % end mc-Lbl or artifact
1448 % FMi: unconditionally for now
1449 %  \tl_if_empty:oF \@itemlabel
1450         \tagstructend   % end   Lbl
1451   \tagstructbegin{tag=LBody}
1452 }
1453 \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`).

```
1454 \cs_set:Npn \__block_list_item_end: {
1455   \legacy_if:nT { @endpe }
1456     {
1457        \__tag_gincr_para_main_end_int:
1458        \tagstructend                              % text-unit
1459 %        \__block_debug_typeout:n{Structure-end~ P~ at~ item-end \on@line }
1460     }
1461   \tagstructend \tagstructend   % end LBody, LI
1462 }
```

(*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.

```
1463 \cs_set:Npn \__block_list_end: {
1464   \legacy_if:nT { @endpe }
1465     {
1466        \__tag_gincr_para_main_end_int:
1467        \tagstructend                              % text-unit
1468        \__block_debug_typeout:n{Structure-end~ P~ at~ list-end \on@line }
```

```
1469        }
1470    \tagstructend\tagstructend   % end LBody, LI
1471    \tagstructend                % end L
1472 }
```

(*End of definition for* __block_list_end:.)

 End of tagging related declarations.

```
1473 }
```

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

```
1474 {
1475    \cs_new:Npn \__block_start_para_structure_unconditionally:n #1 {}
1476 }
1477 ⟨/package⟩
1478 ⟨*latex-lab⟩
1479 \ProvidesFile{block-latex-lab-testphase.ltx}
1480          [\ltlabblockdate\space v\ltlabblockversion\space
1481                          blockenv implementation]
1482 \RequirePackage{latex-lab-testphase-block}
1483 ⟨/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.

55

- – 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$_\varepsilon$ 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.

- 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`.

document class customizations   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.

---

[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.

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

- 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.

document class customizations

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.

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.

---

[4]This should be made easily extendible to deeper levels.

62