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

LaTeX Project[*]

v0.8c 2023/05/16

Abstract

Contents

---

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

1 Introduction

The list implementation in LaTeX 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*) Opject type of the inner instance Default: `list`

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

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

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

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

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

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

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

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

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

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

The maximum number of *blockenv*s that can be nested into each other is 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:

basic This recipe does the following:

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

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

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

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

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

4 The Implementation

```
1 ⟨*package⟩
2 ⟨@@=block⟩

3 \ProvidesPackage {latex-lab-testphase-block-tagging}
4                 [\ltblocksdate\space \ltblocksversion\space
5                         blockenv implementation]
```

We make use of templates:
```
6 \RequirePackage{xtemplate}
```

Generell kernel changes, also loaded by the sec and toc code.
```
7 \RequirePackage{latex-lab-kernel-changes}

8 \ExplSyntaxOn

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

UFi:this variable(s) must be declared:

4.1 Handling \par after the end of the list

An empty line (or a \par) after a list has semantic meaning as it defines whether then following text is logically within the same paragraph as the list (no empty line) or whether it starts a new paragraph and the paragraph containing the list ends at the end of the list (empty line after the list). This is handled by 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.
```
11 \def\@doendpe{\@endpetrue
12   \def\par
13     {
14         \@restorepar
15         \clubpenalty\@clubpenalty
```

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

The standard \par command (\par_end:) acts on @endpe and attempts to close a still open text-unit and this would be wrong if it was already closed above. So we have to reset the switch to false first.
```
17         \@endpefalse
18         \everypar{}
19         \par
20     }
21   \everypar{{\setbox\z@\lastbox}
22             \everypar{}
23             \@endpefalse
24   }
25 }
```

11

By default we don't do any tagging:

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

> verify that this claim is actually correct!

The flag itself should be set globally not locally.

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

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

4.2 Object and template interfaces

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

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

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

blockenv display (*templ.*)

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

block display (*templ.*)

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

```
65 }
```

```
66 \DeclareTemplateInterface{para}{std}{1}
67 {
68   indent-width          : length = \parindent ,
69   start-skip            : skip = 0pt  ,
70   left-skip             : skip = 0pt ,
71   right-skip            : skip = 0pt ,
72   end-skip              : skip = \@flushglue ,
73   fixed-word-spaces     : boolean = false ,
74   final-hyphen-demerits : integer = 5000 ,
75   cr-cmd                : tokenlist = \@normalcr ,
76   para-class            : tokenlist = justify ,
77 }
```

```
78 \DeclareTemplateInterface{list}{std}{1}     % optional
79 {
80   counter        : tokenlist = ,
81   item-label     : tokenlist = ,
82   start          : integer = 1 ,
83   resume         : boolean = false ,
84   item-instance  : instance{item} = basic ,
85   item-skip      : skip = \itemsep ,
86   item-penalty   : integer = \UseName{@itempenalty} ,
87   item-indent    : length = 0pt ,          % was \itemindent
88   label-width    : length = \labelwidth ,
89   label-sep      : length = \labelsep ,
90   legacy-support  : boolean = false ,
91 }
```

```
92 \DeclareTemplateInterface{item}{std}{1}
93   {
94     counter-label : function{1} = \arabic{#1} ,
95     counter-ref   : function{1} = \KeyValue{counter-label} ,
96     label-ref     : function{1} = #1 ,
97     label-autoref : function{1} = item~#1 ,
98     label-format  : function{1} = #1 ,
99     label-strut   : boolean = false ,
100    label-align   : choice {left,center,right,parleft} = right ,
101    label-boxed   : boolean = true ,
102    next-line     : boolean = false ,
103    text-font     : tokenlist ,
104    compatibility : boolean = true ,
105   }
```

4.3   Useful helper commands

This section collects expl3 commands that will be useful.

13

`\__block_skip_set_to_last:N`
`\__block_skip_remove_last:`

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

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

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

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

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

`\tl_if_novalue:nTF`

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

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

`\tag_if_active:T`

can vanish one day

If tagging support is not loaded then we shouldn't try to execute any tagging related commands. Eventually this can go once the basic support is available in the kernel.

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

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

4.3.1 Debugging

`\g__block_debug_bool`

```
113 \bool_new:N \g__block_debug_bool
```

(*End definition for* `\g__block_debug_bool`.)

`\__block_debug:n`
`\__block_debug_typeout:n`

```
114 \cs_new_eq:NN \__block_debug:n \use_none:n
115 \cs_new_eq:NN \__block_debug_typeout:n \use_none:n
```

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

`\block_debug_on:`
`\block_debug_off:`
`\__block_debug_gset:`

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

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

\DebugBlocksOn
\DebugBlocksOff

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

135 \DebugBlocksOff
```

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

4.4   Implementation of the document-level block environments

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

4.4.1   Displayblock environments

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

displayblock (*env.*)

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

displayblockflattened (*env.*)

```
139 \NewDocumentEnvironment{displayblockflattened}{ !O{} }
140   { \UseInstance{blockenv}{displayblockflattened} {#1} }
141   { \endblockenv }
```

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

```
142 \AddToHook{begindocument/before}{
143   \RenewDocumentEnvironment{center} { !O{} }
144   { \UseInstance{blockenv}{center}{#1} }
145   { \endblockenv }

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

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

4.4.2 Display quote environments

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

4.4.3 Verbatim environments

```
161 \AddToHook{begindocument/before}{
162   \RenewDocumentEnvironment{verbatim}{ !O{} }
163     { \UseInstance{blockenv}{verbatim} {#1}
```

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

```
164        \@setupverbinvisiblespace\frenchspacing\@vobeyspaces
165        \@xverbatim
166     }
167     { \endblockenv }
168   \RenewDocumentEnvironment{verbatim*}{ !O{} }
169     { \UseInstance{blockenv}{verbatim} {#1}
170        \@setupverbvisiblespace\frenchspacing\@vobeyspaces
171        \@sxverbatim
172     }
173     { \endblockenv }
174 }
```

Helper commands for verbatim

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

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

16

```
188    \everypar \expandafter{\the\everypar \unpenalty}%
189    \tl_set:Nn \l__tag_para_main_tag_tl {codeline}
190    \tagtool{paratag=Code}%   oder faster: \tl_set:Nn\l__tag_para_tag_tl{Code}
191  }
192  ⟨@@=block⟩
```

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

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

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

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

4.4.4   Standard list environments

itemize (*env.*)   For the standard lists everything is managed by the blockenv instance.
enumerate (*env.*)
description (*env.*)
```
200  \AddToHook{begindocument/before}{
201    \RenewDocumentEnvironment{itemize}{!O{}}
202      { \UseInstance{blockenv}{itemize} {#1} }
203      { \endblockenv }
204    \RenewDocumentEnvironment{enumerate}{!O{}}
205      { \UseInstance{blockenv}{enumerate} {#1} }
206      { \endblockenv }
207    \RenewDocumentEnvironment{description}{!O{}}
208      { \UseInstance{blockenv}{description} {#1} }
209      { \endblockenv }
210  }
```

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

```
211  \AddToHook{begindocument/before}{
212    \RenewDocumentEnvironment{list}{O{} m m }
213      {
```

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.

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

Again something that should probably elsewhere: the rolemapping.

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

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

```
223 \tl_new:N \l__block_env_params_tl
```

(*End definition for* \l__block_env_params_tl.)

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

```
224 \cs_new:Npn \legacylistsetupcode {
```

Reset values to defaults:

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

By default a list environment is not numbered:

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

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

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

```
231     \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.

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

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

trivlist (*env.*)

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

4.4.5 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.

```
250 \RenewDocumentCommand \newtheorem { m O{#1} m o }
251 {
252   \expandafter\@ifdefinable\csname #1\endcsname
253     {
254       \str_if_eq:nnTF{#1}{#2}
255         {
256           \@definecounter {#2}
257           \IfNoValueTF {#4}
258             {  % @ynthm
259               \tl_gset:cx { the #2 }
260                 {
261                   \@thmcounter{#2}
262                 }
263             }
264             {  % @xnthm
265               \@newctr{#1}[#4]
266               \tl_gset:cx { the #2 }
267                 {
268                   \expandafter\noexpand\csname the#4\endcsname
269                   \@thmcountersep
270                   \@thmcounter{#2}
271                 }
272             }
273         }
274         {  % @othm
275           \@ifundefined{c@#2}
276             { \@nocounterr{#2} }
277             {
278               \tl_gset:cn { the #1 }
279                 { \UseName { the #2 } }
280             }
281         }
282       \global\@namedef{#1}    { \@thm{#2}{#3} }
283       \global\@namedef{end#1}{ \@endtheorem  }
284     }
285 }
```

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

`\@begintheorem`   The `\@thm` command expands to either `\@beginthorem` or `\@opargbegintheorem`. For
`\@opargbegintheorem`   the moment we stick with this as it will help with the transion. But instead of using a `trivlist` we use a blockenv and some tagging for the title (as a Caption).

```
286 \def\@begintheorem#1#2{
287   \UseInstance{blockenv}{theorem}{}
288   \tagpdfparaOff
```

```
289    \textbf{
290      \tag_struct_begin:n{tag=Caption}
291        \tag_mc_begin:n {}
292          #1\
293        \tag_mc_end:
294        \tag_struct_begin:n{tag=Lbl}
295          \tag_mc_begin:n {}
296            #2
297          \tag_mc_end:
298        \tag_struct_end:
299      \tag_struct_end:
300    }
301    \tagpdfparaOn
302    \__block_start_para_structure_unconditionally:n { \PARALABEL }
303    \itshape
304    \hskip\labelsep
305    \ignorespaces
306  }
307  \def\@opargbegintheorem#1#2#3{
308    \UseInstance{blockenv}{theorem}{}
309    \tagpdfparaOff
310    \textbf{
311      \tag_struct_begin:n{tag=Caption}
312        \tag_mc_begin:n {}
313          #1\
314        \tag_mc_end:
315        \tag_struct_begin:n{tag=Lbl}
316          \tag_mc_begin:n {}
317            #2
318          \tag_mc_end:
319        \tag_struct_end:
320        \tag_mc_begin:n {}
321          \ (#3)
322        \tag_mc_end:
323      \tag_struct_end:
324    }
325    \tagpdfparaOn
326    \__block_start_para_structure_unconditionally:n { \PARALABEL }
327    \itshape
328    \hskip\labelsep
329    \ignorespaces
330  }
331  \def\@endtheorem{\endblockenv}
```

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

4.5 Implementation of templates

4.5.1 Implementation of blockenv templates . . .

\g_block_nesting_depth_int 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`:

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

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

blockenv display (*templ.*)

```
334 \DeclareTemplateCode{blockenv}{display}{1}
335 {
336   env-name       = \l__block_env_name_tl ,
337   tag-name       = \l__block_tag_name_tl ,
338   tag-class      = \l__block_tag_class_tl ,
339   tagging-recipe = \l__block_tagging_recipe_tl ,
340   level-increase = \l__block_level_incr_bool ,
341   setup-code     = \l__block_setup_code_tl ,
342   block-instance = \l__block_block_instance_tl ,
343   para-instance  = \l__block_para_instance_tl ,
344   inner-level-counter = \l__block_inner_level_counter_tl ,
345   max-inner-levels    = \l__block_max_inner_levels_tl ,
346   inner-instance-type = \l__block_inner_instance_type_tl ,
347   inner-instance      = \l__block_inner_instance_tl ,
348   para-flattened = \l__tag_para_flattened_bool ,
349   final-code     = \l__block_final_code_tl ,
350 }
351 {
352   \__block_debug_typeout:n{\l__block_env_name_tl -env-start}
353 %
354   \tl_if_empty:nF {#1} { \SetTemplateKeys{blockenv}{display}{#1} }
355 %
```

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

```
356   \int_compare:nNnTF \l__block_flattened_level_int > 0
357       {
358         \int_incr:N \l__block_flattened_level_int
359       }
360       {
361         \bool_if:NT \l__tag_para_flattened_bool
362             {
363               \int_incr:N \l__block_flattened_level_int
364             }
365       }
366 %
367   \tl_if_empty:NF \l__block_inner_level_counter_tl
368       {
369         \int_compare:nNnTF  \l__block_inner_level_counter_tl >
370                             { \l__block_max_inner_levels_tl - 1 }
371             { \@toodeep }
```

```
372              { \int_incr:N \l__block_inner_level_counter_tl }  % not clean "o"?
373          }
```

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.

```
374      \bool_if:NT \l__block_level_incr_bool
375        {
376          \int_compare:nNnTF  \g_block_nesting_depth_int >
377                              { \c@maxblocklevels - 1 }
378            { \@toodeep }
379            {
380              \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.

```
381              \use:c { @list \int_to_roman:n { \g_block_nesting_depth_int } }
382            }
383        }
```

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

```
384      \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.

```
385      \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).

```
386      \__block_debug_typeout:n{use~ instance:~
387              \l__block_block_instance_tl - \int_use:N \g_block_nesting_depth_int }
388      \UseInstance{block}
389              { \l__block_block_instance_tl - \int_use:N
390                \g_block_nesting_depth_int }
391              {#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).

```
392      \tl_if_empty:NF \l__block_para_instance_tl
393        {
394          \__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.

```
395          \UseInstance{para}{ \l__block_para_instance_tl } {}
396        }
```

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

```
397      \tl_if_empty:NF \l__block_inner_instance_tl
398        {
399          \__block_debug_typeout:n{use~ instance:~ \l__block_inner_instance_tl
400                  \tl_if_empty:NF \l__block_inner_level_counter_tl
401                      { - \int_use:N \l__block_inner_level_counter_tl }}
402          \UseInstance{ \l__block_inner_instance_type_tl }
403                  { \l__block_inner_instance_tl
404                    \tl_if_empty:NF \l__block_inner_level_counter_tl
```

```
405                         { - \int_use:N \l__block_inner_level_counter_tl }  % not clean
406                                                            % use "o"?
407                    }
408                    {#1}
409        }
```

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.

```
410    \l__block_final_code_tl
411 }
```

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

```
412 \int_new:N \l__block_flattened_level_int
```

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

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

```
413 \newcounter{maxblocklevels}
414 \setcounter{maxblocklevels}{6}
```

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

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

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

name is bad

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

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

```
417    \bool_if:NT \l__block_level_incr_bool
418      { \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.

```
419    \legacy_if:nT { @inlabel }
420      {
421        \mode_leave_vertical:
422        \legacy_if_gset_false:n { @inlabel }
423      }
```

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)

```
424    \legacy_if:nT { @newlist }
425      {
426        \@noitemerr
427        \legacy_if_gset_false:n { @newlist }
428      }
429    \mode_if_horizontal:TF
430        { \__block_skip_remove_last: \__block_skip_remove_last: \par }
431        { \@inmatherr{\end{\@currenvir}} }
```

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

```
432   \__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?

```
433 %       \__block_debug_typeout:n{@noparlist =
434 %                        \legacy_if:nTF { @noparlist }{true}{false}}
435   \legacy_if:nF { @noparlist }
436     {
437       \__block_skip_set_to_last:N \l_tmpa_skip
438       \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
439         {
440           \skip_vertical:n { - \l_tmpa_skip }
441           \skip_vertical:n { \l_tmpa_skip + \parskip - \@outerparskip }
442         }
443       \addpenalty \@endparpenalty
444       \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 expanatory text that is part of the same paragraph and doesn't start a new one.

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

decide

```
445 %           \legacy_if_gset_true:n { @endpe }
446     }
```

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.

```
447     \bool_if:NTF \l__block_standalone_bool
```

It is possible that `@endpe` is true because a displayblock has just ended before we end the standalone displayblock and in that casevthere is no outer `<text-unit>` so we have to explicitly set `@endpe` back to false to prevent it from closing a structure later that isn't there.

```
448       { \legacy_if_gset_false:n { @endpe } }
449       { \legacy_if_gset_true:n  { @endpe } }
450 }
```

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

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

```
451 \cs_new:Npn \__kernel_displayblock_end: {
452   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_end:}}
453 }
```

(*End definition for* __kernel_displayblock_end:.)

\l__block_standalone_bool

```
454 \bool_new:N        \l__block_standalone_bool
455 \bool_set_false:N \l__block_standalone_bool
```

(*End definition for* \l__block_standalone_bool.)

4.5.2 Implementation of para templates . . .

```
456 \DeclareTemplateCode{para}{std}{1}
457 {
458   indent-width        = \parindent ,
459   start-skip          = \l__par_start_skip ,            % name??
460   left-skip           = \leftskip  ,
461   right-skip          = \rightskip  ,
462   end-skip            = \parfillskip ,
463   fixed-word-spaces   = \l__par_fixed_word_spaces_bool ,  % name??
464   final-hyphen-demerits = \finalhyphendemerits ,
465   cr-cmd              = \\ ,
466   para-class          = \l_tag_para_attr_class_tl ,
467 }
468 {
469   \tl_if_empty:nF {#1} { \SetTemplateKeys{para}{std}{#1} }
470   \skip_set:Nn \@rightskip \rightskip
471 }
```

4.5.3 Implementation of block templates . . .

```
472 \DeclareTemplateCode{block}{display}{1}
473 {
474   heading          = \l__block_heading_tl ,
475   beginsep         = \topsep ,
476   begin-par-skip   = \partopsep ,
477   par-skip         = \parsep ,
478   end-skip         = \l__block_botsep_skip ,
479   end-par-skip     = \l__block_parbotsep_skip ,
480   beginpenalty     = \@beginparpenalty ,
481   endpenalty       = \@endparpenalty ,
482   rightmargin      = \rightmargin ,
483   leftmargin       = \leftmargin ,
484   parindent        = \listparindent ,
485 }
486 {
487   \tl_if_empty:nF {#1} { \SetTemplateKeys{block}{display}{#1} }

488   \tl_if_blank:oF \l__block_heading_tl
489     { \mode_leave_vertical: \textbf{\l__block_heading_tl} } % TODO customize
```

generalize heading usage (or drop?)

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

```
490     \legacy_if:nT { @noskipsec } { \mode_leave_vertical: }
491     \skip_set:Nn \l__block_topsepadd_skip { \topsep }
492     \mode_if_vertical:TF
493       {
494         \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.

```
495              \__kernel_displayblock_beginpar_vmode:
496          }
497          {
```

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.

```
498              \__block_skip_remove_last: \__block_skip_remove_last:
499              \__kernel_displayblock_beginpar_hmode:w \par
500          }
```

Now we are back to legacy list implementation . . .

```
501        \legacy_if:nTF { @inlabel }
502          {
503            \legacy_if_set_true:n { @noparitem }
504            \legacy_if_set_true:n { @noparlist }
505          }
506          {
507            \legacy_if:nT { @newlist } { \@noitemerr }
508            \legacy_if_set_false:n { @noparlist }
509            \skip_set_eq:NN \l__block_effective_top_skip \l__block_topsepadd_skip
510          }
511        \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*.

```
512        \skip_zero:N \leftskip
513        \skip_set_eq:NN \rightskip \@rightskip
514        \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.

```
515        \int_zero:N \par@deathcycles
516        \@setpar
517          {
518            \legacy_if:nTF { @newlist }
519              {
520                \int_incr:N \par@deathcycles
521                \int_compare:nNnTF \par@deathcycles > { 1000 }
522                    { \@noitemerr
523                      { \para_end: }
524                    }
525              }
526              {
527                { \para_end: }
528              }
529          }
```

```
530    \skip_set_eq:NN \@outerparskip \parskip
531    \skip_set_eq:NN \parskip \parsep
532    \dim_set_eq:NN \parindent \listparindent
533    \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
534    \dim_add:Nn \@totalleftmargin { \leftmargin }
535    \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.

```
536    \__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`.

```
537    \legacy_if:nTF { @noparitem }
538      {
539        \legacy_if_set_false:n { @noparitem }
540        \hbox_gset:Nn \g__block_labels_box
541          {
542            \skip_horizontal:n { - \leftmargin }
543            \hbox_unpack_drop:N \g__block_labels_box
544            \skip_horizontal:n { \leftmargin }
545          }
546        \legacy_if:nF { @minipage } % Why this chunk of code?
547          {
548            \__block_skip_set_to_last:N \l__block_tmpa_skip
549            \skip_vertical:n { - \l__block_tmpa_skip }
550            \skip_vertical:n { \l__block_tmpa_skip + \@outerparskip - \parskip }
551          }
552      }
553      {
554        \legacy_if:nTF { @nobreak }
555          { \addvspace{\skip_eval:n{\@outerparskip-\parskip}} }
556          {
557            \addpenalty \@beginparpenalty
558            \addvspace \l__block_effective_top_skip
559            \addvspace{-\parskip}
560          }
561      }
562  }
```

document 2e logic used here

Extra keys to support enumitem conventions:

```
563  \keys_define:nn { template/block/display }
564  {
565    ,topsep        .skip_set:N = \topsep
566    ,partopsep     .skip_set:N = \partopsep
567    ,listparindent .skip_set:N = \listparindent
568  }
```

`\__kernel_displayblock_begin:`
`\__kernel_displayblock_beginpar_hmode:w`
`\__kernel_displayblock_beginpar_vmode:`

The internal kernel hooks for tagging.

```
569  \cs_new:Npn \__kernel_displayblock_begin: {
570    \__block_debug_typeout:n{\detokenize{__kernel_displayblock_begin:}}
571  }
```

27

```
572 \cs_new:Npn \__kernel_displayblock_beginpar_hmode:w {
573   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_beginpar_hmode:w}}
574 }
575 \cs_new:Npn \__kernel_displayblock_beginpar_vmode: {
576   \__block_debug_typeout:n{\detokenize{__kernel_displayblock_beginpar_vmode:}}
577 }
```

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

4.5.4 Implementation of list templates …

\@itemlabel   Both \@itemlabel and \@listctr from the 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.

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

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

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

```
580 \DeclareTemplateCode{list}{std}{1}
581 {
582   counter          = \l__block_counter_tl,
583   item-label       = \l__block_item_label_tl,
584   start            = \l__block_counter_start_int ,
585   resume           = \l__block_resume_bool ,
586   item-instance    = \__block_item_instance:n ,
587   item-skip        = \itemsep ,
588 %  item-par-skip   = \parsep ,
589   item-penalty     = \@itempenalty ,
590   item-indent      = \itemindent ,
591   label-width      = \labelwidth ,
592   label-sep        = \labelsep ,
593   legacy-support   = \l__block_legacy_support_bool , % FMi questionable
594 }
595 {
596   \__block_debug_typeout:n{template:list:std}
597 %
598   \tl_if_empty:nF {#1} { \SetTemplateKeys{list}{std}{#1} }
```

Has this list a counter name defined in the instance?

```
599   \tl_if_empty:NTF \l__block_counter_tl
600     {
```

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.

```
601     \tl_if_blank:oF \@listctr
602       {
```

28

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.

```
603          \bool_if:NF \l__block_resume_bool
604            {
605              \int_gset:cn{ c@ \@listctr }
606                  { \l__block_counter_start_int - 1 }
607            }
608        }
```

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

```
609        { \@nmbrlistfalse }
610      }
```

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.

```
611      {
612      \@nmbrlisttrue
613      \tl_set_eq:NN \@listctr \l__block_counter_tl
614      \bool_if:NF \l__block_resume_bool
615        {
616          \int_gset:cn{ c@ \@listctr }
617              { \l__block_counter_start_int - 1 }
618        }
619      }
```

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.

```
620    \tl_if_empty:NF \l__block_item_label_tl
621      {
622        \tl_set_eq:NN \@itemlabel \l__block_item_label_tl
623      }
```

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.

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

625    \__block_debug_typeout:n{template:list:std~end}
626 }
```

Extra keys to support enumitem conventions:

```
627 \keys_define:nn { template/list/std }
628 {
629   ,nosep  .code:n =
630     \dim_zero:N \itemsep
631     \dim_zero:N \parsep
632     \dim_zero:N \topsep
633     \dim_zero:N \l__block_botsep_skip
634     \dim_zero:N \l__block_parbotsep_skip
635   ,midsep    .skip_set:N = \topsep
636 }
```

4.5.5 Implementation of `\item` template(s)

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

```
637 \keys_define:nn { template/item/std }
638                 { label .tl_set:N = \l__block_label_given_tl }
639 \DeclareTemplateCode{item}{std}{1}
640   {
641     counter-label   = \__block_counter_label:n ,
642     counter-ref     = \__block_counter_ref:n ,

643     label-ref       = \__block_label_ref:n ,
644     label-autoref   = \__block_label_autoref:n ,
645     label-format    = \__block_label_format:n ,
646     label-strut     = \l__block_label_strut_bool ,
647     label-boxed     = \l__block_label_boxed_bool ,
648     next-line       = \l__block_next_line_bool ,
649     text-font       = \l__block_text_font_tl ,
650     compatibility   = \l__block_item_compatibility_bool ,
```

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

next set of key not yet used

complete

This probably needs a different implementation (and needs completing)

```
651     label-align     = {
652       left    = \tl_set:Nn \l__block_item_align_tl { \relax \hss } ,
653       center  = \tl_set:Nn \l__block_item_align_tl { \hss \hss } ,
654       right   = \tl_set:Nn \l__block_item_align_tl { \hss \relax } ,
655       parleft = \NOT_IMPLEMENTED ,
656     } ,
657   }
```

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.

```
658   {
659     \__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.

```
660     \tl_set_eq:NN \l__block_label_given_tl \c_novalue_tl
661     \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[]`.

```
662     \tl_if_novalue:oTF \l__block_label_given_tl
663       {
```

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

`fix` wrong.

```
664          \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
665          \bool_if:NTF \l__block_item_compatibility_bool    % not sure that conditional
666                                                     % makes sense
667            { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}\@itemlabel } } % TODO ?
668            { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}\__block_counter_label:n
669          }
670          {
671            \__block_debug_typeout:n{item~ with~ optional}
672            \__block_make_label_box:n { \l__block_label_given_tl } }
673        \bool_if:nT
674          {
675            \l__block_label_boxed_bool
676            && \dim_compare_p:n { \box_wd:N \l__block_one_label_box <= \linewidth } % TODO: is \
677          }
678          {
679            \dim_compare:nNnT
680              { \box_wd:N \l__block_one_label_box } < \labelwidth
681              {
682                \hbox_set_to_wd:Nnn \l__block_one_label_box { \labelwidth }
683                  {
684                    \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.

```
685 %                    \hbox_unpack_drop:N \l__block_one_label_box    %TODO: customize?
686                    \box_use_drop:N \l__block_one_label_box

687                    \exp_after:wN \use_ii:nn \l__block_item_align_tl
688                  }
689              }
690          \hbox_set:Nn \l__block_one_label_box
691                  { \box_use_drop:N \l__block_one_label_box }
692          }
693      \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \labelwidth
694        { \bool_set_true:N \l__block_long_label_bool }
695        { \bool_set_false:N \l__block_long_label_bool }
696      \hbox_gset:Nn \g__block_labels_box
697        {
698          \hbox_unpack_drop:N \g__block_labels_box
699          \skip_horizontal:n { \itemindent - \labelsep - \labelwidth }
700          \hbox_unpack_drop:N \l__block_one_label_box
701          \skip_horizontal:n { \labelsep }
702          \bool_if:NT \l__block_next_line_bool
703            { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
704          % version of \newline inside an hbox that will be unpacked
705        }
706      % \skip_set_eq:NN \parsep \l__block_item_parsep_skip TODO??? FMi
707                                          % what's that?
708      \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.

```
709      \cs_set_eq:NN \__block_item_everypar: \__block_item_everypar_std:
710    }
```

\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.

```
711 \box_new:N \l__block_one_label_box
712 \box_new:N \g__block_labels_box
```

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

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

```
713 \bool_new:N \l__block_long_label_bool
```

(*End definition for* \l__block_long_label_bool.)

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

```
714 \cs_new_protected:Npn \__block_make_label_box:n #1
715    {
716      \hbox_set:Nn \l__block_one_label_box
717        {
```

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

```
718          \__kernel_list_label_begin:
719          \__block_label_format:n
720            {
721              \bool_if:NT \l__block_label_strut_bool { \strut }
722              \bool_if:NTF \l__block_legacy_support_bool
723                            \makelabel
724                            \use:n
725                  {#1}
726            }
```

And what gets opened also needs closing:

```
727          \__kernel_list_label_end:
728        }
729    }
```

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

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

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

`\__block_item_everypar:`
`\__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.

```
732 \cs_new_eq:NN \__block_item_everypar: \prg_do_nothing:
```

```
733 \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!

```
734 \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.

```
735 \cs_new_protected:Npn \__block_item_everypar_std: {
736     \__block_debug_typeout:n{item~ everypar \on@line }
737     \legacy_if_set_false:n { @minipage }
738     \legacy_if_gset_false:n { @newlist }
739     \legacy_if:nT { @inlabel }
740         {
741             \legacy_if_gset_false:n { @inlabel }
742             \box_if_empty:NT \g_para_indent_box { \kern - \itemindent }
743             \para_omit_indent:

744             \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:

```
745                 \__kernel_list_label_after:

746                 \penalty \c_zero_int
747         }
748     \legacy_if:nTF { @nobreak }
749         {
750             \legacy_if_gset_false:n { @nobreak }
751             \int_set:Nn \clubpenalty { 10000 }
752         }
753         {
754             \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.

```
755             \cs_set_eq:NN \__block_item_everypar: \prg_do_nothing:
756         }
757 }
```

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

`\__kernel_list_label_after:`

```
758 \cs_new_eq:NN \__kernel_list_label_after: \prg_do_nothing:
```

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

`\l__block_tmpa_skip`

```
759 \skip_new:N \l__block_tmpa_skip
```

33

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

`\l__block_topsepadd_skip`
`\l__block_effective_top_skip`

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

```
760 \skip_new:N \l__block_topsepadd_skip
761 \skip_new:N \l__block_effective_top_skip
```

(End definition for `\l__block_topsepadd_skip` *and* `\l__block_effective_top_skip`*.)*

`\item`

Here we already have all the building blocks. Complain in math mode. 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`.

```
762 \AddToHook{begindocument/before}{
763   \RenewDocumentCommand{\item}{ ={label}o }
764     {
765       \@inmatherr \item
```

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

```
766       \tl_if_empty:oTF \__block_item_instance:n %%FMi?
767         { \msg_error:nnn { __block } { item-in-nonlist } { \item[{#1}] } }
768         {
769           \legacy_if:nTF { @newlist }
770             { \__kernel_list_item_begin: }
771             { \__block_inter_item:            }
```

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

```
772           \tl_if_novalue:nTF {#1}          % avoids reparsing label={}
773             { \__block_item_instance:n { } }
774             { \__block_item_instance:n {#1} }
```

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

```
775           \legacy_if_gset_true:n { @inlabel }
776           \ignorespaces
777         }
778     }
779 }
```

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

`\__block_inter_item:`

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

```
780 \cs_new_protected:Npn \__block_inter_item: {
781   \legacy_if:nT { @inlabel }
782                 { \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).

```
783   \mode_if_horizontal:T { \__block_skip_remove_last:
784                           \__block_skip_remove_last: \par }
```

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

```
785   \__kernel_list_item_end:
786   \__kernel_list_item_begin:

787   \addpenalty \@itempenalty
788   \addvspace \itemsep
789 }
```

(*End definition for* __block_inter_item:.)

__kernel_list_item_begin:
__kernel_list_item_end:

```
790 \cs_new_eq:NN \__kernel_list_item_begin: \prg_do_nothing:
791 \cs_new_eq:NN \__kernel_list_item_end:   \prg_do_nothing:
```

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

4.6   Tagging recipes

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

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

```
792 \cs_new:Npn \__block_recipe_basic: {
793   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
794                                         \__block_beginpar_hmode:N
795   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
796                                         \__block_beginpar_vmode:
797   \let \__kernel_displayblock_begin:        \prg_do_nothing:
798   \let \__kernel_displayblock_end:          \prg_do_nothing:
799 }
```

(*End 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.

```
800 \cs_new:Npn \__block_recipe_standalone: {
801   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
802                                         \prg_do_nothing:
803   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
804                                         \prg_do_nothing:
805   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
806   \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:

807   \bool_set_true:N \l__block_standalone_bool

808   \tl_if_empty:NTF \l__block_tag_name_tl
809     { \tl_set:Nn    \l__block_tag_inner_tag_tl {Sect}             }
810     { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
811 }
```

(*End 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 the inner structure (`Figure` or explicit one) but leave the `text-unit` open to be either continued or closed due to a following `\par`.

```
812 \cs_new:Npn \__block_recipe_standard:
813 {
814   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
815                                       \__block_beginpar_hmode:N
816   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
817                                       \__block_beginpar_vmode:
818   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
819   \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:
820   \tl_if_empty:NTF \l__block_tag_name_tl
821     { \tl_set:Nn    \l__block_tag_inner_tag_tl {Figure}           }
822     { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
823 }
```

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

`\l__block_tag_inner_tag_tl`

```
824 \tl_new:N \l__block_tag_inner_tag_tl
```

(*End definition for* `\l__block_tag_inner_tag_tl`.)

`\__block_recipe_list:` The `list` recipe does the following.

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

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

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

```
825 \cs_new:Npn \__block_recipe_list:
826 {
827   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
828                                       \__block_beginpar_hmode:N
829   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
830                                       \__block_beginpar_vmode:
831   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_list_begin:
832   \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 `\item`s, 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)).

```
833   \cs_set_eq:NN \__kernel_list_item_begin:    \__block_list_item_begin:
834   \cs_set_eq:NN \__kernel_list_item_end:      \__block_list_item_end:
```

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

```
835    \tl_if_empty:NTF \l__block_tag_name_tl
836      { \tl_set:Nn    \l__tag_L_tag_tl {L}                  }
837      { \tl_set_eq:NN \l__tag_L_tag_tl \l__block_tag_name_tl }
838    \tl_if_empty:NTF \l__block_tag_class_tl
839      { \tl_set:Nn    \l__tag_L_attr_class_tl {}               }
840      { \tl_set_eq:NN \l__tag_L_attr_class_tl \l__block_tag_class_tl }
841  }
```

(*End definition for* __block_recipe_list:.)

4.7 Blockenv instances

4.7.1 Basic instances

blockenv displayblock (*inst.*)

```
842  \DeclareInstance{blockenv}{displayblock}{display}
843  {
844    env-name       = displayblock,
845    tag-name       = ,
846    tag-class      = ,
847    tagging-recipe = standard,
848    inner-level-counter  = ,
849    level-increase = false,
850    setup-code     = ,
851    block-instance = displayblock ,
852    inner-instance = ,
853  }
```

nv displayblockflattened (*inst.*)

```
854  \DeclareInstance{blockenv}{displayblockflattened}{display}
855  {
856    env-name       = displayblockflattened,
857    tag-name       = ,
858    tag-class      = ,
859    tagging-recipe = basic,
860    inner-level-counter  = ,
861    level-increase = false,
862    setup-code     = ,
863    block-instance = displayblock ,
864    para-flattened = true ,
865    inner-instance = ,
866  }
```

blockenv center (*inst.*)

```
867  \DeclareInstance{blockenv}{center}{display}
868  {
869    env-name       = center,
870    tag-name       = ,
871    tag-class      = ,
872    tagging-recipe = basic,
873    inner-level-counter  = ,
874    level-increase = false,
```

```
875   setup-code      = ,
876   block-instance = displayblock ,
877   para-flattened = true ,
878   para-instance  = center ,
879   inner-instance = ,
880 }
```

blockenv flushleft (*inst.*)

```
881 \DeclareInstance{blockenv}{flushleft}{display}
882 {
883   env-name        = flushleft,
884   tag-name        = ,
885   tag-class       = ,
886   tagging-recipe = basic,
887   inner-level-counter  = ,
888   level-increase = false,
889   setup-code      = ,
890   block-instance = displayblock ,
891   para-flattened = true ,
892   para-instance  = raggedright ,
893   inner-instance = ,
894 }
```

blockenv flushright (*inst.*)

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

4.7.2   Blockquote instances

blockenv quotation (*inst.*)

```
909 \tag_if_active:T {
910   \tagpdfsetup{add-new-tag={tag=quote,role=BlockQuote}}
911   \tagpdfsetup{add-new-tag={tag=quotation,role=BlockQuote}}
912 }
913 \DeclareInstance{blockenv}{quotation}{display}
914 {
915   env-name        = quotation,
916   tag-name        = quotation,
917   tag-class       = ,
918   tagging-recipe = standard,
919   inner-level-counter  = ,
```

38

```
920    level-increase = true,
921    setup-code     = ,
922    block-instance = quotationblock ,
923    inner-instance =   ,
924 }
```

```
925 \DeclareInstance{blockenv}{quote}{display}
926 {
927    env-name        = quote,
928    tag-name        = quote,
929    tag-class       = ,
930    tagging-recipe = standard,
931    inner-level-counter  = ,
932    level-increase = true,
933    setup-code     = ,
934    block-instance = quoteblock ,
935    inner-instance =   ,
936 }
```

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

```
937 % \DeclareInstance{blockenv}{quotation}{display}
938 % {
939 %    env-name        = quotation,
940 %    tag-name        = ,
941 %    tag-class       = ,
942 %    tagging-recipe = blockquote,
943 %    inner-level-counter  = ,
944 %    level-increase = true,
945 %    setup-code     = \setlength\rightmargin{\leftmargin}
946 %                      \setlength\parsep{1.5em} ,
947 %    block-instance = displayblock ,
948 %    inner-instance =   ,
949 % }
```

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

```
962 \DeclareInstance{blockenv}{theorem}{display}
963 {
964    env-name        = theorem-like,
965    tag-name        = theorem-like,
966    tag-class       = ,
```

39

```
967    tagging-recipe = standalone,
968    inner-level-counter  = ,
969    level-increase = false,
970    setup-code       = ,
971    block-instance = displayblock ,
972 %  inner-instance-type = innerblock ,
973 %  inner-instance = theorem,
974 }
```

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

```
975 \tag_if_active:T {
976    \tagpdfsetup{add-new-tag={tag=theorem-like,role=Sect}}
977 }
```

4.7.3   Verbatim instances

```
978 \tag_if_active:T {
979    \tagpdfsetup{add-new-tag={tag=verbatim,role=P}}
980    \tagpdfsetup{add-new-tag={tag=codeline,role=Sub}}
```

Possible alternative for PDF 1.7:

```
981 %  \tagpdfsetup{add-new-tag={tag=verbatim,role=Div}}
982 %  \tagpdfsetup{add-new-tag={tag=codeline,role=P}}
983 }
984 \DeclareInstance{blockenv}{verbatim}{display}
985 {
986    env-name        = verbatim,
987    tag-name        = verbatim,
988    tag-class       = ,
989    tagging-recipe = standard,
990    inner-level-counter  = ,
991    level-increase = false,
992    setup-code       = ,
993    block-instance = verbatimblock ,
994    inner-instance = ,
995    final-code       = \legacyverbatimsetup ,
996 }
```

4.7.4   Standard list instances

```
997  \DeclareInstance{blockenv}{itemize}{display}
998  {
999     env-name        = itemize,
1000    tag-name        = itemize,
1001    tag-class       = itemize,
1002    tagging-recipe = list,
1003    inner-level-counter  = \@itemdepth,
1004    level-increase = true,
1005    max-inner-levels = 4,
1006    setup-code       = ,
```

```
1007    block-instance = list ,
1008    inner-instance = itemize ,
1009 }
```

blockenv enumerate (*inst.*)

```
1010 \DeclareInstance{blockenv}{enumerate}{display}
1011 {
1012   env-name            = enumerate,
1013   tag-name            = enumerate,
1014   tag-class           = enumerate,
1015   tagging-recipe      = list,
1016   level-increase      = true,
1017   setup-code          = ,
1018   block-instance      = list ,
1019   inner-level-counter = \@enumdepth,
1020   max-inner-levels    = 4,
1021   inner-instance      = enum ,
1022 }
```

blockenv description (*inst.*)

```
1023
1024 \DeclareInstance{blockenv}{description}{display}
1025 {
1026   env-name          = description,
1027   tag-name          = description,
1028   tag-class         = description,
1029   tagging-recipe    = list,
1030   inner-level-counter  = ,
1031   level-increase = true,
1032   setup-code        = ,
1033   block-instance = list ,
1034   inner-instance = description ,
1035 }
```

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

```
1036 \DeclareInstance{blockenv}{list}{display}
1037 {
1038   env-name          = list,
1039   tag-name          = list,
1040   tag-class         = ,
1041   tagging-recipe = list,
1042   level-increase = true,
1043   setup-code        = \legacylistsetupcode ,
1044   block-instance = list ,
1045   inner-level-counter  = ,
1046   inner-instance = legacy ,
1047 }
```

4.8   Block instances

4.8.1   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).

```
1048 \setcounter{maxblocklevels}{6}
```

block displayblock-0 (*inst.*) Here we need level zero as well in case a flattened displayblock (like the center env) it is
block displayblock-1 (*inst.*) used on top-level.
block displayblock-2 (*inst.*)
block displayblock-3 (*inst.*)
```
1049 \DeclareInstance{block}{displayblock-0}{display}
1050   {
1051     leftmargin       = 0pt ,
1052     parindent        = 0pt ,
1053   }
```
block displayblock-4 (*inst.*)
block displayblock-5 (*inst.*)
block displayblock-6 (*inst.*)

```
1054 \DeclareInstanceCopy{block}{displayblock-1}{displayblock-0}
1055 \DeclareInstanceCopy{block}{displayblock-2}{displayblock-0}
1056 \DeclareInstanceCopy{block}{displayblock-3}{displayblock-0}
1057 \DeclareInstanceCopy{block}{displayblock-4}{displayblock-0}
1058 \DeclareInstanceCopy{block}{displayblock-5}{displayblock-0}
1059 \DeclareInstanceCopy{block}{displayblock-6}{displayblock-0}
```

4.8.2 Verbatim instances

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

block verbatimblock-0 (*inst.*)
block verbatimblock-1 (*inst.*)
block verbatimblock-2 (*inst.*)
block verbatimblock-3 (*inst.*)
block verbatimblock-4 (*inst.*)
block verbatimblock-5 (*inst.*)
block verbatimblock-6 (*inst.*)
```
1060 \DeclareInstance{block}{verbatimblock-0}{display}
1061   {
1062     leftmargin       = 0pt ,
1063     parindent        = 0pt ,
1064     par-skip         = 0pt ,
1065   }
```

```
1066 \DeclareInstanceCopy{block}{verbatimblock-1}{verbatimblock-0}
1067 \DeclareInstanceCopy{block}{verbatimblock-2}{verbatimblock-0}
1068 \DeclareInstanceCopy{block}{verbatimblock-3}{verbatimblock-0}
1069 \DeclareInstanceCopy{block}{verbatimblock-4}{verbatimblock-0}
1070 \DeclareInstanceCopy{block}{verbatimblock-5}{verbatimblock-0}
1071 \DeclareInstanceCopy{block}{verbatimblock-6}{verbatimblock-0}
```

4.8.3 Quote/quotationblock instances

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

block quoteblock-1 (*inst.*) Default layout is to indent equaly from both side.
block quoteblock-2 (*inst.*)
block quoteblock-3 (*inst.*)
```
1072 \DeclareInstance{block}{quoteblock-1}{display}
1073   { rightmargin = \KeyValue{leftmargin} }
```
block quoteblock-4 (*inst.*)
block quoteblock-5 (*inst.*)
block quoteblock-6 (*inst.*)
```
1074 \DeclareInstanceCopy{block}{quoteblock-2}{quoteblock-1}
1075 \DeclareInstanceCopy{block}{quoteblock-3}{quoteblock-1}
1076 \DeclareInstanceCopy{block}{quoteblock-4}{quoteblock-1}
1077 \DeclareInstanceCopy{block}{quoteblock-5}{quoteblock-1}
1078 \DeclareInstanceCopy{block}{quoteblock-6}{quoteblock-1}
```

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

```
1079 \DeclareInstance{block}{quotationblock-1}{display}
```
block quotationblock-2 (*inst.*)
```
1080   { parindent = 1.5em , rightmargin = \KeyValue{leftmargin} }
```
block quotationblock-3 (*inst.*)
block quotationblock-4 (*inst.*)
```
1081 \DeclareInstanceCopy{block}{quotationblock-2}{quotationblock-1}
```
block quotationblock-5 (*inst.*)
```
1082 \DeclareInstanceCopy{block}{quotationblock-3}{quotationblock-1}
```
block quotationblock-6 (*inst.*)
```
1083 \DeclareInstanceCopy{block}{quotationblock-4}{quotationblock-1}
1084 \DeclareInstanceCopy{block}{quotationblock-5}{quotationblock-1}
1085 \DeclareInstanceCopy{block}{quotationblock-6}{quotationblock-1}
```

4.8.4   Block instances for the standard lists

block list-1 (*inst.*) The block instances for the various list environments use the same underlying instance
block list-2 (*inst.*) (well by default) and nothing nothing needs to be set up specifically (because that is
block list-3 (*inst.*) already done in the legacy \list⟨*romannumeral*⟩ unless a different layout is wanted.
block list-4 (*inst.*)
```
1086 \DeclareInstance{block}{list-1}{display}{
```
block list-5 (*inst.*)
```
1087 %   heading           = ,
```
block list-6 (*inst.*)
```
1088 %   beginsep          = \topsep ,
1089 %   begin-par-skip    = \partopsep ,
1090 %   par-skip          = \parsep ,
1091 %   end-skip          = \KeyValue{beginsep} ,
1092 %   end-par-skip      = \KeyValue{begin-par-skip} ,
1093 %   beginpenalty      = \UseName{@beginparpenalty} ,
1094 %   endpenalty        = \UseName{@endparpenalty} ,
1095 %   leftmargin        = \leftmargin ,
1096 %   rightmargin       = \rightmargin ,
1097 %   parindent         = \listparindent ,
1098 }
1099 \DeclareInstance{block}{list-2}{display}{}
1100 \DeclareInstance{block}{list-3}{display}{}
1101 \DeclareInstance{block}{list-4}{display}{}
1102 \DeclareInstance{block}{list-5}{display}{}
1103 \DeclareInstance{block}{list-6}{display}{}
```

4.9   List instances for the standard lists

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

list itemize-1 (*inst.*) For `itemize` environments this is all we need to do and we refer back to the external
list itemize-2 (*inst.*) definitions rather than defining the `item-label` code in the instance to ensure that old
list itemize-3 (*inst.*) documents still work.
list itemize-4 (*inst.*)
```
1104 \DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi }
1105 \DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii }
1106 \DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
1107 \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.*)
```
1108 \DeclareInstance{list}{enum-1}{std}
```
list enumerate-4 (*inst.*)
```
1109   { item-label = \labelenumi ,   counter = enumi }
1110 \DeclareInstance{list}{enum-2}{std}
1111   { item-label = \labelenumii ,   counter = enumii }
```

```
1112 \DeclareInstance{list}{enum-3}{std}
1113   { item-label = \labelenumiii , counter = enumiii }
1114 \DeclareInstance{list}{enum-4}{std}
1115   { 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

```
1116 \DeclareInstance{list}{legacy}{std} {
1117   item-instance = basic ,
1118   legacy-support = true ,
1119 }
```

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

```
1120 \DeclareInstance{list}{description}{std} { item-instance = description }
```

4.10   Item instances

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

```
1121 \DeclareInstance{item}{basic}{std}
1122   {
1123     label-align = right ,
1124   }
1125 \DeclareInstance{item}{description}{std}
1126   {
1127     label-format = \normalfont\bfseries #1 ,
1128   }
```

4.11   Para instances

```
1129 \tag_if_active:T
1130 {
1131   \tagpdfsetup
1132     {
1133       newattribute = {justify}    {/O /Layout /TextAlign/Justify},
1134       newattribute = {center}     {/O /Layout /TextAlign/Center},
1135       newattribute = {raggedright}{/O /Layout /TextAlign/Start},
1136       newattribute = {raggedleft} {/O /Layout /TextAlign/End},
1137     }
1138 }
```

para center (*inst.*)

```
1139 \DeclareInstance{para}{center}{std}
1140 {
1141   indent-width         = 0pt ,
1142   start-skip           = 0pt ,
1143   left-skip            = \@flushglue ,
1144   right-skip           = \@flushglue ,
```

```
1145    end-skip             = \z@skip ,
1146    final-hyphen-demerits = 0 ,
1147    cr-cmd               = \@centercr ,
1148    para-class           = center ,
1149 }
1150 \DeclareInstance{para}{raggedright}{std}
1151 {
1152    indent-width         = 0pt ,
1153    start-skip           = 0pt ,
1154    left-skip            = \z@skip ,
1155    right-skip           = \@flushglue ,
1156    end-skip             = \z@skip ,
1157    final-hyphen-demerits = 0 ,
1158    cr-cmd               = \@centercr ,
1159    para-class           = raggedright ,
1160 }
1161 \DeclareInstance{para}{raggedleft}{std}
1162 {
1163    indent-width         = 0pt ,
1164    start-skip           = 0pt ,
1165    left-skip            = \@flushglue ,
1166    right-skip           = \z@skip ,
1167    end-skip             = \z@skip ,
1168    final-hyphen-demerits = 0 ,
1169    cr-cmd               = \@centercr ,
1170    para-class           = raggedleft ,
1171 }
1172 \DeclareInstance{para}{justify}{std}
1173 {
1174 %  indent-width          = 0pt ,
1175    start-skip           = 0pt ,
1176    left-skip            = \z@skip ,
1177    right-skip           = \z@skip ,
1178    end-skip             = \@flushglue ,
1179    final-hyphen-demerits =  5000 ,
1180    cr-cmd               = \@normalcr ,
1181    para-class           = justify ,
1182 }
1183 \DeclareRobustCommand\centering  {\UseInstance{para}{center}{}}
1184 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}{}}
1185 \DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}{}}
1186 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}{}}
1187
1188 \justifying
```

4.12 Tagging support

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

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

```
1189 \tag_if_active:T {
```

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

```
1190    \cs_set:Npn \__block_beginpar_vmode: {
1191        \__block_debug_typeout:n
1192            { @endpe = \legacy_if:nTF { @endpe }{true}{false}
1193            \on@line }
1194    \legacy_if:nTF { @endpe }
1195        {
1196            \legacy_if_gset_false:n { @endpe }
1197        }
```

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.

```
1198        {
1199            \int_compare:nNnT \l__block_flattened_level_int < 2
1200                {
1201                    \int_gincr:N \g__tag_para_main_begin_int
1202                    \tagstructbegin{tag=\l__tag_para_main_tag_tl}
1203                }
1204        }
1205    }
```

(*End definition for* __block_beginpar_vmode:.)

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

```
1206    \cs_set:Npn \__block_beginpar_hmode:N #1
1207        {
1208            \tag_mc_end:
1209            \int_gincr:N \g__tag_para_end_int
1210            \__block_debug_typeout:n{increment~ /P \on@line }
1211            \bool_if:NT \l__tag_para_show_bool
1212                { \tag_mc_begin:n{artifact}
1213                    \rlap{\color_select:n{red}\tiny\ \int_use:N\g__tag_para_end_int}
1214                    \tag_mc_end:
1215                }
1216            \tag_struct_end:
1217            \tagpdfparaOff \par \tagpdfparaOn
1218        }
```

(*End definition for* __block_beginpar_hmode:N.)

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

```
1219    \cs_set:Npn \__kernel_displayblock_doendpe: {
1220        \bool_if:NT \l__tag_para_bool
1221            {
```

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

```
1222            \legacy_if:nT { @endpe }
1223              {
```

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.

```
1224                \__block_debug_typeout:n
1225                  { flattened= \bool_if:NTF
1226                                  \l__tag_para_flattened_bool {true}{false}
1227                    \on@line }
1228                \bool_if:NF \l__tag_para_flattened_bool
1229                  {
1230                    \__block_debug_typeout:n{Structure-end~
1231                      \l__tag_para_main_tag_tl\space after~ displayblock \on@line }
1232                    \int_gincr:N \g__tag_para_main_end_int
1233                    \tag_struct_end: %text-unit
1234                  }
1235              }
1236          }
1237 }
```

(*End definition for* `\__kernel_displayblock_doendpe:`.)

para/begin   Paragraph tagging is mainly done using the paragraph hooks (will get moved eventually). The default hook setting is not good enough when lists get suppported: 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).

```
1238 \RemoveFromHook{para/begin}[tagpdf]
```

```
1239 \AddToHook{para/begin}[tagpdf]{
1240   \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).

```
1241        \legacy_if:nF { @inlabel }
1242          {
```

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.

```
1243            \__block_start_para_structure:n { \PARALABEL }
1244          }
1245      }
1246 }
```

```
\__block_start_para_structure:n    1247  \cs_new_protected:Npn \__block_start_para_structure:n #1 {
                                   1248    \__block_debug_typeout:n
                                   1249      { @endpe = \legacy_if:nTF { @endpe }{true}{false}
                                   1250        \on@line }
                                   1251    \legacy_if:nF { @endpe }
                                   1252      {
                                   1253        \bool_if:NF \l__tag_para_flattened_bool
                                   1254          {
                                   1255            \int_gincr:N \g__tag_para_main_begin_int
                                   1256            \tag_struct_begin:n{tag=\l__tag_para_main_tag_tl}
                                   1257          }
                                   1258      }
                                   1259    \int_gincr:N \g__tag_para_begin_int
                                   1260    \__block_debug_typeout:n{increment~ P \on@line }
                                   1261    \tag_struct_begin:n
                                   1262      {
                                   1263        tag=\l__tag_para_tag_tl
                                   1264        ,attribute-class=\l_tag_para_attr_class_tl
                                   1265      }
                                   1266    \__tag_check_para_begin_show:nn {green}{#1}
                                   1267    \tag_mc_begin:n {}
                                   1268  }
```

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

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

                                   1285  \tag_if_active:T {
                                   1286  %  \tagpdfsetup{add-new-tag={tag=text-unit,role=Part}}
                                   1287  }

                                   1288  \RemoveFromHook{para/end}[tagpdf]
                                   1289  \AddToHook{para/end}
                                   1290    {
                                   1291      \bool_if:NT \l__tag_para_bool
                                   1292        {
                                   1293          \int_gincr:N \g__tag_para_end_int
                                   1294          \__block_debug_typeout:n{increment~ /P \on@line }
                                   1295          \tag_mc_end:
```

```
1296                \__tag_check_para_end_show:nn {red}{}
1297                \tag_struct_end:
1298                \bool_if:NF \l__tag_para_flattened_bool
1299                 {
1300                    \int_gincr:N \g__tag_para_main_end_int
1301                    \tag_struct_end:
1302                 }
1303            }
1304        }
1305    \def\PARALABEL{NP-}
```

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

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

```
1306    \cs_set_protected:Npn \para_end: {
1307        \scan_stop:
1308        \mode_if_horizontal:TF {
1309            \mode_if_inner:F {
1310                \tex_unskip:D
1311                \hook_use:n{para/end}
1312                \@kernel@after@para@end
1313                \mode_if_horizontal:TF {
1314                    \if_int_compare:w 11 = \tex_lastnodetype:D
1315                        \tex_hskip:D \c_zero_dim
1316                    \fi:
1317                    \tex_par:D
1318                    \hook_use:n{para/after}
1319                    \@kernel@after@para@after
1320                }
1321                { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
1322            }
1323        }
1324        {
1325            \__kernel_endpe_vmode:        % should do nothing if no tagging
1326            \tex_par:D
1327        }
1328    }
1329    \cs_set_eq:NN \par       \para_end:
1330    \cs_set_eq:NN \__blockpar    \para_end:
1331    \cs_set_eq:NN \endgraf \para_end:
```

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

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

```
1332    \DeclareRobustCommand*\begin[1]{%
1333        \UseHook{env/#1/before}%
1334        \@ifundefined{#1}%
1335            {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
1336            {\def\reserved@a{\def\@currenvir{#1}%
1337                \edef\@currenvline{\on@line}%
1338                \@execute@begin@hook{#1}%
```

```
1339          \csname #1\endcsname}}%
1340     \@ignorefalse
1341     \begingroup
1342        \__kernel_endpe_vmode:
1343        \reserved@a}
```

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

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

```
1344 \cs_new:Npn \__kernel_endpe_vmode: {
1345     \if@endpe \ifvmode
1346        \bool_if:NT \l__tag_para_bool
1347 {
1348   \bool_if:NF \l__tag_para_flattened_bool
1349   {
1350       \int_gincr:N \g__tag_para_main_end_int
1351       \tag_struct_end:
1352   }
1353   \@endpefalse
1354 }
1355     \fi \fi
1356 }
```

(*End definition for* __kernel_endpe_vmode:.)

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

```
1357 \cs_set:Npn \__kernel_list_label_after: {
1358     \bool_if:NT \l__tag_para_bool
1359        {
1360          \__block_start_para_structure_unconditionally:n { LI- }
1361        }
1362 }
```

(*End definition for* __kernel_list_label_after:.)

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

```
1363 \cs_new:Npn \__block_inner_begin: {
1364     \tagstructbegin{tag=\l__block_tag_inner_tag_tl}
1365 }
```

(*End definition for* __block_inner_begin:.)

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

```
1366 \cs_new:Npn \__block_inner_end: {
1367     \__block_debug_typeout:n{block-end \on@line}
1368     \legacy_if:nT { @endpe }
1369        {
1370          \int_gincr:N \g__tag_para_main_end_int
1371          \__block_debug_typeout:n{close~ /text-unit \on@line}
1372          \tagstructend
1373        }
1374     \tagstructend          % end inner structure
1375 }
```

(*End definition for* __block_inner_end:.)

50

4.12.1 List tags

```
1376 \tl_new:N  \l__tag_L_tag_tl
1377 \tl_set:Nn \l__tag_L_tag_tl {L}
1378
1379 \tl_new:N\l__tag_L_attr_class_tl
1380 \tl_set:Nn \l__tag_L_attr_class_tl {list}
1381 \tag_if_active:T
1382 {
1383   \tagpdfsetup
1384       {
1385         % default if unknown
1386         newattribute = {list}{/O /List /ListNumbering/None},
1387         newattribute = {itemize}{/O /List /ListNumbering/Unordered},
1388         newattribute = {enumerate}{/O /List /ListNumbering/Ordered},
1389         newattribute = {description}{/O /List /ListNumbering/Description},
1390       }
1391 }
1392 \def\LItag{LI}
```

__block_list_begin: Start a list . . .

```
1393 \cs_set:Npn \__block_list_begin: {
1394   \tagstructbegin
1395       {
1396          tag=\l__tag_L_tag_tl
1397         ,attribute-class=\l__tag_L_attr_class_tl
1398       }
1399 }
```

(*End definition for* __block_list_begin:.)

__block_list_item_begin: Start tagging a list item.

```
1400 \cs_set:Npn \__block_list_item_begin: { \tagstructbegin{tag=\LItag} }
```

(*End definition for* __block_list_item_begin:.)

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

```
1401 \cs_set:Npn \__kernel_list_label_begin: {
1402 %
1403 % FMi: this needs a different logic to decide when to make the label
1404 %    an artifact (after cleaning up the the \item code  ), therefore
1405 %    disabled for now
1406 %  \tl_if_empty:oTF \@itemlabel
1407 %     {
1408 %        \tag_mc_begin:n {artifact}
1409 %     }
1410 %     {
1411       \tagstructbegin{tag=Lbl}
1412       \tagmcbegin{tag=Lbl}
1413 %     }
1414 }
```

(*End definition for* __kernel_list_label_begin:.)

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

```
1415 \cs_set:Npn \__kernel_list_label_end: {
1416   \tagmcend                                      % end mc-Lbl or artifact
1417 % FMi: unconditionally for now
1418 %   \tl_if_empty:oF \@itemlabel
1419           \tagstructend    % end   Lbl
1420   \tagstructbegin{tag=\LBody}
1421 }
1422 \def\LBody{LBody}
```

(*End definition for* __kernel_list_label_end:*.*)

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

```
1423 \cs_set:Npn \__block_list_item_end: {
1424   \legacy_if:nT { @endpe }
1425     {
1426       \int_gincr:N \g__tag_para_main_end_int
1427       \tagstructend                              % text-unit
1428 %       \__block_debug_typeout:n{Structure-end~ P~ at~ item-end \on@line }
1429     }
1430   \tagstructend \tagstructend   % end LBody, LI
1431 }
```

(*End definition for* __block_list_item_end:*.*)

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

```
1432 \cs_set:Npn \__block_list_end: {
1433   \legacy_if:nT { @endpe }
1434     {
1435       \int_gincr:N \g__tag_para_main_end_int
1436       \tagstructend                              % text-unit
1437       \__block_debug_typeout:n{Structure-end~ P~ at~ list-end \on@line }
1438     }
1439   \tagstructend\tagstructend  % end LBody, LI
1440   \tagstructend              % end L
1441 }
```

(*End definition for* __block_list_end:*.*)

End of tagging related declarations.

```
1442 }
```

```
1443 ⟨/package⟩
```

```
1444 ⟨∗latex-lab⟩
1445 \ProvidesFile{block-tagging-latex-lab-testphase.ltx}
1446         [\ltblocksdate\space \ltblocksversion\space
1447                         blockenv implementation]
1448 \RequirePackage{latex-lab-testphase-block-tagging}
1449 ⟨/latex-lab⟩
```

5 Documentation from first prototype implementations

5.1 Open questions

- Existing questions — moved to issues —

5.2 Code cleanup

- Actually implement what's announced.

- Encapsulate most uses of `\legacy_if...` into commands with `expl3` syntax: we cannot rename these booleans for compatibility reasons but we can make the code cleaner nevertheless. — made issue —

- The `\topsep` and `\partopsep` business is tricky to reproduce exactly (see `\@topsepadd` and `\@topsep`) because of how it accumulates when lists are nested immediately.

5.3 Tasks

- Change author to LaTeX Team once it's nice enough to deserve that label.

- Reproducing exactly the standard layouts and examples in the `enumitem` documentation.

- Hooks, but do not duplicate those that already exist as environment hooks. Hence, mostly around items.

- Customization and interaction with LDB:

  - Allow arbitrary nesting depth with automatically defined styles for labels, counters etc.
  - Adapt everything to font size! (e.g. footnotes).
  - How to model the inheritance from trivlist to list to enumerate?

- Add key–value settings mimicking `enumitem`'s ability to set any four of five horizontal parameters and deduce the fifth by `\leftmargin` + `\itemindent` = `\labelindent` + `\labelwidth` + `\labelsep`.

- Provide good ways to customize how overlong labels are dealt with.

- Use the `.aux` file.

  - Implement the `\ref` styles that `enumitem` provides.
  - Reverse enumerations, important in publication lists and the like. Somehow avoid needing 3 compilations for references to reverse enumerations to settle?
  - Ability to calculate `\labelwidth` from the label contents. Share calculated parameters between multiple environments (cf. `resume` option).

- Related to grabbing the whole list environment, and input syntax variations:

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

- Check interaction with babel options such as french or accadian (see FrenchItem-izeSpacing)

- RTL and vertical typesetting.

6 Plan of attack of first prototype

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

- It has a certain layout in the page, with vertical and horizontal spacing around it. This type of layout is shared with environments such as quote, flushright, or tabbing. This common layout is implemented in LaTeX 2_ε through \trivlist (or \list).

- It defines how each \item should be typeset: how to construct the label, in particular the counter name, and how to format the content of the item.

This suggests defining two object types, *block* and *item* covering these two aspects.[1] While the *item* type will perhaps have a single template, one could typeset a *block* object in several ways, for instance the standard LaTeX 2_ε way or a fancy colored box.

The *general block* template should receive the following parameters. The *plain block* template is a restricted template that freezes all item-related parameters to dummy values (counter, start, resume, label-width, label-sep and all item-*). The *list block* template is a restricted template[2] that omits the heading parameter and whose default for item-instance is non-empty.

- Structural parameters: the heading to place before, counter name, start value, whether to resume a previous list, and the item-instance (an *item* instance) to use when typesetting items.

- 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

---

[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.

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

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

- Structural parameters: `counter`, `start`, `resume`, `item-instance`.

- Spacing and penalties: `beginpenalty`, `beginsep`, `item-penalty`, `item-skip`, `endpenalty`, `end-skip`.

- Horizontal spacing: `label-width`, `label-sep`.

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

- Counter name (`counter`), shared with the parent *list block* template, but needed for incrementing.

- Label construction: a function `counter-label` that produces the label from the counter name, used if `\item` is given without argument.

- References: a function `counter-ref` for how the label should be referred to when it is constructed from the counter, `label-ref` and `label-autoref` used when `\item` has an optional argument.

- Label formatting: `label-format` function, `label-strut` boolean.

- Label alignment (`label-align`, `label-boxed`, `next-line`).

- Content parameters: `text-font`.

- A `compatibility` boolean that controls for instance whether `\makelabel` is used.

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.

---

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

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

Index

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

59

61

63