

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

\LaTeX Project*
v0.8v 2024-10-11

Abstract

Contents

1	Introduction	3
2	Object types and templates for blocks and lists	3
2.1	Object types	3
2.1.1	The object type ‘block’	3
2.1.2	The object type ‘para’	3
2.1.3	The object type ‘list’	4
2.1.4	The object type ‘item’	4
2.1.5	The object type ‘blockenv’	4
2.2	Templates	4
2.2.1	The <code>blockenv</code> template ‘display’	4
2.2.2	The <code>block</code> template ‘display’	6
2.2.3	The <code>para</code> template ‘std’	6
2.2.4	The <code>list</code> template ‘std’	7
2.2.5	The <code>item</code> template ‘std’	7
3	Tagging support	8
3.1	Paragraph tags	8
3.2	Tagging recipes	10
4	Debugging	11
5	New and redefined kernel command	11

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

6	The Implementation	12
6.1	Handling <code>\par</code> after the end of the list	12
6.2	Object and template interfaces	13
6.3	Useful helper commands	15
6.3.1	Debugging	15
6.4	Implementation of the document-level block environments	16
6.4.1	Displayblock environments	16
6.4.2	Display quote environments	17
6.4.3	Verbatim environments	17
6.4.4	Standard list environments	18
6.4.5	verse environment	18
6.4.6	Theorem-like environments	20
6.5	Implementation of templates	22
6.5.1	Implementation of blockenv templates	22
6.5.2	Implementation of para templates	27
6.5.3	Implementation of block templates	27
6.5.4	Implementation of list templates	30
6.5.5	Implementation of <code>\item</code> template(s)	33
6.6	Tagging recipes	39
6.7	Blockenv instances	41
6.7.1	Basic instances	41
6.7.2	Blockquote instances	42
6.7.3	Verbatim instances	44
6.7.4	Standard list instances	44
6.8	Block instances	45
6.8.1	Displayblock instances	45
6.8.2	Verbatim instances	46
6.8.3	Quote/quotationblock instances	46
6.8.4	Block instances for the standard lists	47
6.9	List instances for the standard lists	47
6.10	Item instances	48
6.11	Para instances	48
6.12	Tagging support	49
6.12.1	List tags	55
7	Documentation from first prototype implementations	57
7.1	Open questions	57
7.2	Code cleanup	57
7.3	Tasks	57
8	Plan of attack of first prototype	58
	Index	60

1 Introduction

The list implementation in $\text{\LaTeX} 2_{\epsilon}$ serves a dual purpose: it implements real lists such as `itemize` or `enumerate`, but it is also used as the basis for vertical blocks, i.e., to specify the vertical spacing and paragraph handling after such block, e.g., in environments like `center`, `quote`, `verbatim`, or in the theorem environments. They are all implemented as “trivial” lists with a single (hidden) item.

While this was convenient to get a consistent layout using a single implementation it is not adequate if it comes to interpreting the structure of a document, because environments based on `trivlist` should not advertise themselves as being a “list” — after all, from a semantic point of view they aren’t lists.

The approach taking here is therefore to offer separate object types: *block* (horizontally or vertically oriented data that needs some handling at the start and the end), *para* (that deals with different paragraph layouts), *list* (that handles list related parameters, and *item* (for item layouts and handling), to address the independent aspects and also offer the object type *blockenv* that ties them together as necessary.

For example, a `quote` environment would make use of a (display) *block* and some *para* handling while an standard `enumerate` would make use of a display *block*, a *list*, and an *item* and *para* instance. An inline list (like `enumerate*` from the `enumitem` package) would be using the same *list* instance but a different (horizontally oriented) *block*.

2 Object types and templates for blocks and lists

2.1 Object types

2.1.1 The object type ‘block’

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

Semantics:

Handle the layout aspects of a block of data. In case of a “display” block (i.e., vertically oriented) the spacing and page breaking as well as the handling if the block starts a paragraph or ends one, that is, if text is immediately following the block without being separated by an empty line, then this text is considered to be in the same paragraph as the block.

In case of a horizontally oriented block it covers any special handling at the start and end of the block, e.g., extra spacing, prohibiting or encouraging line breaks, and so forth.

2.1.2 The object type ‘para’

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

Semantics:

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

2.1.3 The object type ‘list’

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

Semantics:

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

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

2.1.4 The object type ‘item’

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

Semantics:

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

2.1.5 The object type ‘blockenv’

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

Semantics:

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

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

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

2.2 Templates

2.2.1 The blockenv template ‘display’

Attributes:

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

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

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

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

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

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

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

para-instance (*tokenlist*)

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

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

inner-instance-type (*tokenlist*) Object type of the inner instance Default: `list`

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

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

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

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

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

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

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

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

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

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

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

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

2.2.2 The block template ‘display’

Attributes:

<code>heading</code> (<i>tokenlist</i>)	<i>not really used yet</i>	
<code>beginsep</code> (<i>skip</i>)		Default: <code>\topsep</code>
<code>begin-par-skip</code> (<i>skip</i>)		Default: <code>\partopsep</code>
<code>par-skip</code> (<i>skip</i>)		Default: <code>\parsep</code>
<code>end-skip</code> (<i>skip</i>)		Default: value from <code>beginsep</code>
<code>end-par-skip</code> (<i>skip</i>)		Default: value from <code>begin-par-skip</code>
<code>item-skip</code> (<i>skip</i>)	The space in front of an item if the block is a list; if not the setting has no effect	Default: <code>\itemsep</code>
<code>beginpenalty</code> (<i>integer</i>)		Default: <code>\@beginparpenalty</code>
<code>endpenalty</code> (<i>integer</i>)		Default: <code>\@endparpenalty</code>
<code>leftmargin</code> (<i>length</i>)		Default: <code>\leftmargin</code>
<code>rightmargin</code> (<i>length</i>)		Default: <code>\rightmargin</code>
<code>parindent</code> (<i>length</i>)		Default: <code>\listparindent</code>

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

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

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

2.2.3 The para template ‘std’

Attributes:

<code>indent-width</code> (<i>length</i>)		Default: <code>\parindent</code>
<code>start-skip</code> (<i>skip</i>)		Default: <code>0pt</code>
<code>left-skip</code> (<i>skip</i>)		Default: <code>0pt</code>
<code>right-skip</code> (<i>skip</i>)		Default: <code>0pt</code>
<code>end-skip</code> (<i>skip</i>)		Default: <code>\@flushglue</code>
<code>fixed-word-spaces</code> (<i>boolean</i>)		Default: <code>false</code>
<code>final-hyphen-demerits</code> (<i>integer</i>)		Default: <code>5000</code>
<code>cr-cmd</code> (<i>tokenlist</i>)		Default: <code>\@normalcr</code>
<code>para-class</code> (<i>tokenlist</i>)		Default: <code>justify</code>

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. If not specified the value specified in the block template instance is used
- item-indent** (*length*) Horizontal displacement of the item. Default: Opt
- item-penalty** (*integer*) Penalty for breaking before an item (except the first)
Default: \@itempenalty
- label-width** (*length*) Width reserved for the formatted item label
Default: \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

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

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

3 Tagging support

3.1 Paragraph tags

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

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

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

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

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

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

```
<text-unit>
  <text>
    The paragraph text before the display element ...
  </text>
  <display element structure>
    Content of the display structure possibly involving inner <text-unit> tags
  </display element structure>
  <text>
```



```

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

```

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

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

```

<text-unit>
  <text>
    The intro text for the itemize environment ...
  </text>
  <itemize>
    <LI>
      <Lb1> label </Lb1>
      <LBody>
        The text of the first item involving <text-unit> as necessary ...
      </LBody>
    </LI>
    <LI>
      The second item ...
    </LI>
    ... further items ...
  </itemize>
</text-unit>

```

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

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

```

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

```

```

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

```

will be tagged as follows:

```

<text-unit>
  <text>
    This is a paragraph with some
  </text>
  <text /0 /Layout /TextAlign/Center>
    centered lines
  </text>
  <text /0 /Layout /TextAlign/Center>
    with a paragraph break between them

```

```

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

```

3.2 Tagging recipes

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

standalone This recipe does the following:

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

basic This recipe does the following:

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

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

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

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

- the inner structure is a list (`<L>`).
- Furthermore everything is set up so that we have list items (``) with suitable substructures (`<Lb1>` for the item labels and `<LBody>` for the item bodies).
- If the key `tag-name` is specified, this is used as the tag name for the whole list instead of `<L>`. Of course, it should then have a suitable rollmap.
- If the key `tag-class` is specified then this is used as the class attribute. Again, this requires a suitable setup on the outside.
- At the end of the environment the `</LBody>`, ``, and `</L>` (or the tag name used) are closed.
- Then the lookahead for an empty line is done as described previously.

4 Debugging

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

These commands enable/disable debugging messages.

5 New and redefined kernel command

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

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

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

`endblockenv` *to be documented*
`\g_block_nesting_depth_int`

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

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

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

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

`\para_end:` TODO: consider name, document

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

6 The Implementation

```
1 <*package>
2 <@@=block>
3 \ProvidesPackage {latex-lab-testphase-block}
4                 [\tlabblockdate\space v\tlabblockversion\space
5                 blockenv implementation]
6
7   General kernel changes, also loaded by the sec and toc code.
8 \RequirePackage{latex-lab-kernel-changes}
9 \ExplSyntaxOn
10 \tl_new:N \l__block_item_align_tl
11 \tl_new:N \l__block_legacy_env_params_tl
```

UFi: this variable(s) must be declared:

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

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

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

`\@doendpe` The original L^AT_EX_{2 ϵ} command is augmented to allow for tagging. TODO: use sockets for this and move to the kernel eventually.

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

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

```
15   \__kernel_displayblock_doendpe:
```

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

```

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

By default we don't do any tagging:

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

(End of definition for `\doendpe` and `__kernel_displayblock_doendpe:`. This function is documented on page 11.)

6.2 Object and template interfaces

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

```

list (objecttype) 26 \NewTemplateType{blockenv}{1}
item (objecttype) 27 \NewTemplateType{block}{1}
                    28 \NewTemplateType{para}{1}
                    29 \NewTemplateType{list}{1}
                    30 \NewTemplateType{item}{1}
```

`blockenv display` (*templ.*)

```

31 \DeclareTemplateInterface{blockenv}{display}{1}
32 {
33   env-name      : tokenlist ,
34   tag-name      : tokenlist ,
35   tag-class     : tokenlist ,
36   tagging-recipe : tokenlist = standard,
37   level-increase : boolean = true ,
38   setup-code    : tokenlist ,
39   block-instance : tokenlist = displayblock ,
40   para-instance : tokenlist ,
41   inner-level-counter : tokenlist,
42   max-inner-levels : tokenlist = 4,
43   inner-instance-type : tokenlist = list ,
44   inner-instance  : tokenlist ,
45   para-flattened : boolean = false ,
46   final-code     : tokenlist = \ignorespaces ,
47 }
```

`block display` (*templ.*)

```

48 \DeclareTemplateInterface{block}{display}{1}
49 {
50   heading      : tokenlist = ,           ???
51   beginsep     : skip = \topsep ,
```

```

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

```

para std (*templ.*)

```

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

```

list std (*templ.*)

```

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

```

item std (*templ.*)

```

90 \DeclareTemplateInterface{item}{std}{1}
91 {
92   counter-label : function{1} = \arabic{#1} ,
93   counter-ref   : function{1} = \KeyValue{counter-label} ,
94   label-ref     : function{1} = #1 ,
95   label-autoref : function{1} = item~#1 ,
96   label-format  : function{1} = #1 ,
97   label-strut   : boolean = false ,
98   label-align   : choice {left,center,right,parleft} = right ,
99   label-boxed   : boolean = true ,

```

```

100     next-line      : boolean = false ,
101     text-font      : tokenlist ,
102     compatibility  : boolean = true ,
103 }

```

6.3 Useful helper commands

This section collects `expl3` commands that will be useful.

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

```

104 \cs_new_protected:Npn \_block_skip_set_to_last:N #1 {
105   \skip_set:Nn #1 { \tex_lastskip:D }
106 }

```

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

```

107 \cs_new_eq:NN \_block_skip_remove_last: \tex_unskip:D

```

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

```

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

```

6.3.1 Debugging

`\g_block_debug_bool`

```

109 \bool_new:N \g_block_debug_bool

```

(End of definition for `\g_block_debug_bool.`)

`_block_debug:n`

`_block_debug_typeout:n`

```

110 \cs_new_eq:NN \_block_debug:n \use_none:n
111 \cs_new_eq:NN \_block_debug_typeout:n \use_none:n

```

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

`\block_debug_on:`

`\block_debug_off:`

`_block_debug_gset:`

```

112 \cs_new_protected:Npn \block_debug_on:
113 {
114   \bool_gset_true:N \g_block_debug_bool
115   \_block_debug_gset:
116 }
117 \cs_new_protected:Npn \block_debug_off:
118 {
119   \bool_gset_false:N \g_block_debug_bool
120   \_block_debug_gset:
121 }
122 \cs_new_protected:Npn \_block_debug_gset:
123 {
124   \cs_gset_protected:Npx \_block_debug:n ##1
125   { \bool_if:NT \g_block_debug_bool {##1} }
126   \cs_gset_protected:Npx \_block_debug_typeout:n ##1
127   { \bool_if:NT \g_block_debug_bool { \typeout{==>~ ##1} } }
128 }

```

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

```

\DebugBlocksOn
\DebugBlocksOff 129 \cs_new_protected:Npn \DebugBlocksOn { \block_debug_on: }
130 \cs_new_protected:Npn \DebugBlocksOff { \block_debug_off: }
131 \DebugBlocksOff

```

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

6.4 Implementation of the document-level block environments

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

6.4.1 Displayblock environments

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

`displayblock (env.)`

```

132 \NewDocumentEnvironment{displayblock}{!0{ } }
133 { \UseInstance{blockenv}{displayblock} {#1} }
134 { \endblockenv }

```

`displayblockflattened (env.)`

```

135 \NewDocumentEnvironment{displayblockflattened}{!0{ } }
136 { \UseInstance{blockenv}{displayblockflattened} {#1} }
137 { \endblockenv }

```

`center (env.)`

`flushleft (env.)`

`flushright (env.)`

```

138 \AddToHook{begindocument/before}{
139 \RenewDocumentEnvironment{center} {!0{ } }
140 { \UseInstance{blockenv}{center}{#1} }
141 { \endblockenv }

142 \RenewDocumentEnvironment{flushright} {!0{ } }
143 { \UseInstance{blockenv}{flushright}{#1} }
144 { \endblockenv }

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

```


6.4.2 Display quote environments

```
quote (env.)
quotation (env.) 149 \AddToHook{begindocument/before}{
150 \RenewDocumentEnvironment{quote}{ !0{ }
151 { \UseInstance{blockenv}{quote} {#1} }
152 { \endblockenv }
153 \RenewDocumentEnvironment{quotation}{ !0{ }
154 { \UseInstance{blockenv}{quotation} {#1} }
155 { \endblockenv }
156 }
```

6.4.3 Verbatim environments

```
verbatim (env.)
verbatim* (env.) 157 \AddToHook{begindocument/before}{
158 \RenewDocumentEnvironment{verbatim}{ !0{ }
159 { \UseInstance{blockenv}{verbatim} {#1} }
```

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

```
160 \setsetupverbinvisiblespace\frenchspacing\@vobeyspaces
161 \@xverbatim
162 }
163 { \endblockenv }
164 \RenewDocumentEnvironment{verbatim*}{ !0{ }
165 { \UseInstance{blockenv}{verbatim} {#1}
166 \setsetupverbvisiblespace\frenchspacing\@vobeyspaces
167 \@sxverbatim
168 }
169 { \endblockenv }
170 }
```

Helper commands for verbatim

`\legacyverbatimsetup`

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

```
171 <@@=
172 \def\legacyverbatimsetup{%
173 \language\l@nohyphenation
174 \@tempwafalse
175 \def\par{%
176 \if@tempswa
177 \leavevmode \null {\@@par}\penalty\interlinepenalty
178 \else
179 \@tempwatrue
180 \ifhmode{\@@par}\penalty\interlinepenalty\fi
181 \fi}%
182 \let\do\@makeother \dospecials
183 \obeylines \verbatim@font \@noligs
```

```

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

```

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

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

```

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

```

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

6.4.4 Standard list environments

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

```

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

200   \RenewDocumentEnvironment{enumerate}{!0{}}
201   { \UseInstance{blockenv}{enumerate} {#1} }
202   { \endblockenv }

203   \RenewDocumentEnvironment{description}{!0{}}
204   { \UseInstance{blockenv}{description} {#1} }
205   { \endblockenv }
206 }

```

6.4.5 verse environment

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

```

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

```

```

221     }
222     { \endblockenv }
223   }

```

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

```

224 \AddToHook{begindocument/before}{
225   \RenewDocumentEnvironment{list}{0{} m m }
226   {

```

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.

```

227     \tl_set:Nn \l__block_legacy_env_params_tl
228     {
229       \tl_set:Nn \@itemlabel {#2}
230       #3
231     }
232     \UseInstance{blockenv}{list} {#1}
233   }
234   { \endblockenv }
235 }

```

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

```

236 \cs_new:Npn \legacylistsetupcode {

```

Reset values to defaults:

```

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

```

By default a `list` environment is not numbered, but this happens already in the block template.

```

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

```

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

```

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

```

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

```

243   \l__block_legacy_env_params_tl

```

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

```

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

```

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

`trivlist (env.)`

```

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

```

6.4.6 Theorem-like environments

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

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

```

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

```

```

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

```

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

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

```

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

```

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

```

303 \def\hyper@nopatch@thm{}

```

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

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

```

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

```

```

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

```

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

6.5 Implementation of templates

6.5.1 Implementation of blockenv templates ...

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

```

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

```

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

`blockenv display (templ.)`

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

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

The default for `list` environments is that they have an empty label and are not numbered (something that is then overwriting by the setup of a specific list). We ensure this here even for non-lists, because we need a defined state that then can be overwriting by the legacy setup code for the `list` environment in `\l__block_setup_code_tl`. This is needed in case lists are nested as they otherwise would inherit outer values (and suddenly an `itemize` would start incrementing an outer `enumerate` counter, etc.

```
409 \tl_clear:N \@itemlabel
410 \tl_clear:N \@listctr
411 \legacy_if_set_false:n { @nmbriest }
```

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

```
412 \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).

```
413 \__block_debug_typeout:n{use~ instance:~
414 \l__block_block_instance_tl - \int_use:N \g_block_nesting_depth_int }
415 \UseInstance{block}
416 { \l__block_block_instance_tl - \int_use:N
417 \g_block_nesting_depth_int }
418 {#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).

```
419 \tl_if_empty:NF \l__block_para_instance_tl
420 {
421 \__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.

```
422 \UseInstance{para}{ \l__block_para_instance_tl } {}
423 }
```

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.

```
424 \tl_if_empty:NF \l__block_inner_instance_tl
425 {
426 \__block_debug_typeout:n{use~ instance:~ \l__block_inner_instance_tl
427 \tl_if_empty:NF \l__block_inner_level_counter_tl
428 { - \int_use:N \l__block_inner_level_counter_tl }}
429 \UseInstance{ \l__block_inner_instance_type_tl }
430 { \l__block_inner_instance_tl
431 \tl_if_empty:NF \l__block_inner_level_counter_tl
432 { - \int_use:N \l__block_inner_level_counter_tl } % not clean
```



```

433                                     % use "o"?
434                                     }
435                                     {#1}
436     }

```

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.

```

437     \l__block_final_code_tl
438 }

```

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

```

439 \int_if_exist:NF \l__tag_block_flattened_level_int
440 {
441     \int_new:N \l__tag_block_flattened_level_int
442 }

```

(End of definition for `\l__tag_block_flattened_level_int`.)

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

```

443 \newcounter{maxblocklevels}
444 \setcounter{maxblocklevels}{6}

```

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

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

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

```

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

```

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

```

447     \bool_if:NT \l__block_level_incr_bool
448     { \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.

```

449     \legacy_if:nT { @inlabel }
450     {
451         \mode_leave_vertical:
452         \legacy_if_gset_false:n { @inlabel }
453     }

```

If we are ending a list environment and we have not seen any `\item`, i.e., `@newlist` is still true, we raise an error. In basic a “displayblock” scenario `@newlist` will always be false, but if such an environment appears inside an outer list then `\noitemerr` could still be triggered and that is undesirable (as the missing item will be detected at the wrong point and again later, during the outer list processing). We therefore run it only if the current environment is a list.

```

454     \__block_if_list:T { \legacy_if:nT { @newlist } { \noitemerr } }
455     \mode_if_horizontal:TF
456     { \__block_skip_remove_last: \__block_skip_remove_last: \par }
457     { \@inmatherr{\end{\@currenenvir}} }

```

name is bad

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

```
458 \__kernel_displayblock_end:
```

Resetting the `@newlist` switch is also only done if the current environment is a list and not unconditionally.

```
459 \__block_if_list:T { \legacy_if_gset_false:n { @newlist } }
```

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 $\LaTeX 2_{\epsilon}$ list environment have been doing.

some redesign/extensions here?

```
460 % \__block_debug_typeout:n{@nparlist =
461 % \legacy_if:nTF { @nparlist }{true}{false}}
462 \legacy_if:nF { @nparlist }
463 {
464 \__block_skip_set_to_last:N \l_tmpa_skip
465 \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
466 {
467 \skip_vertical:n { - \l_tmpa_skip }
468 \skip_vertical:n { \l_tmpa_skip + \parskip - \@outerparskip }
469 }
470 \addpenalty \@endparpenalty
471 \addvspace \l__block_topsepadd_skip
```

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

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

```
472 % \legacy_if_gset_true:n { @endpe }
473 }
```

decide

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

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

```
474 \socket_use:n {taggsupport/block-endpe}
475 }
```

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

revisit

The following code may need some redesigning, as there is no good test for "is this environment a 'list' that has `\items`". For now this here does the trick well enough.

```
476 \cs_new:Npn \__block_if_list:T
477 { \tl_if_eq:NnT \l__block_block_instance_tl {list} }
```

(End of definition for `__block_if_list:T`.)

`__kernel_displayblock_end:`

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

```
478 \cs_new:Npn \__kernel_displayblock_end: {
479 \__block_debug_typeout:n{\detokenize{\__kernel_displayblock_end:}}
480 }
```

(End of definition for `__kernel_displayblock_end:.`)

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

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

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

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

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

```
483           { \@endpetrue }
484 \socket_new_plug:nnn{tagsupport/block-endpe}{off}
485           { \@endpefalse }
486 \socket_assign_plug:nn{tagsupport/block-endpe}{on}
```

6.5.2 Implementation of para templates ...

`para std` (*templ.*)

```
487 \DeclareTemplateCode{para}{std}{1}
488 {
489   indent-width      = \parindent ,
490   start-skip        = \l__par_start_skip ,           % name??
491   left-skip         = \leftskip ,
492   right-skip        = \rightskip ,
493   end-skip          = \parfillskip ,
494   fixed-word-spaces = \l__par_fixed_word_spaces_bool , % name??
495   final-hyphen-demerits = \finalhyphendemerits ,
496   cr-cmd            = \\ ,
497   para-class        = \l__tag_para_attr_class_tl ,
498 }
499 {
500   \tl_if_empty:nF {#1} { \SetTemplateKeys{para}{std}{#1} }
501   \skip_set:Nn \@rightskip \rightskip
502 }
```

6.5.3 Implementation of block templates ...

`block display` (*templ.*)

```
503 \DeclareTemplateCode{block}{display}{1}
504 {
505   heading           = \l__block_heading_tl ,
506   beginsep          = \topsep ,
507   begin-par-skip    = \partopsep ,
508   par-skip          = \parsep ,
509   end-skip          = \l__block_botsep_skip ,
510   end-par-skip      = \l__block_parbotsep_skip ,
511   item-skip         = \itemsep ,
512   beginpenalty      = \@beginparpenalty ,
513   endpenalty        = \@endparpenalty ,
514   rightmargin       = \rightmargin ,
515   leftmargin        = \leftmargin ,
516   parindent         = \listparindent ,
```

generalize heading usage
(or drop?)

```

517 }
518 {
519   \tl_if_empty:nF {#1} { \SetTemplateKeys{block}{display}{#1} }
520   \tl_if_blank:oF \l__block_heading_tl
521   { \mode_leave_vertical: \textbf{\l__block_heading_tl} } % TODO customize

```

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

```

522   \legacy_if:nT { @noskipsec } { \mode_leave_vertical: }
523   \skip_set:Nn \l__block_topsepadd_skip { \topsep }
524   \mode_if_vertical:TF
525   {
526     \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.

```

527   \__kernel_displayblock_beginpar_vmode:
528   }
529   {

```

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`.

```

530     \__block_skip_remove_last: \__block_skip_remove_last:
531     \__kernel_displayblock_beginpar_hmode:w \par
532   }

```

Now we are back to legacy list implementation ...

```

533   \legacy_if:nTF { @inlabel }
534   {
535     \legacy_if_set_true:n { @noparitem }
536     \legacy_if_set_true:n { @noparlist }
537   }
538   {
539     \legacy_if:nT { @newlist } { \noitemerr }
540     \legacy_if_set_false:n { @noparlist }
541     \skip_set_eq:NN \l__block_effective_top_skip \l__block_topsepadd_skip
542   }
543   \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`.

```

544   \skip_zero:N \leftskip
545   \skip_set_eq:NN \rightskip \@rightskip
546   \skip_set_eq:NN \parfillskip \@flushglue

```

The next lines establish a parshape which is retained across paragraphs by executing `\para_end:` within a group and thus reestablishing the parshape for the next paragraph

again. In case a list got started `\par` is ignored until we have seen an `\item` (or we have executed `\par` one thousand times).

```

547 \int_zero:N \par@deathcycles
548 \@setpar
549 {
550   \legacy_if:nTF { @newlist }
551   {
552     \int_incr:N \par@deathcycles
553     \int_compare:nNnTF \par@deathcycles > { 1000 }
554     { \@noitemerr
555       { \para_end: }
556     }
557   }
558   {
559     { \para_end: }
560   }
561 }
562 \skip_set_eq:NN \@outerparskip \parskip
563 \skip_set_eq:NN \parskip \parsep
564 \dim_set_eq:NN \parindent \listparindent
565 \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
566 \dim_add:Nn \@totalleftmargin { \leftmargin }
567 \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.

```

568 \__kernel_displayblock_begin:

```

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

```

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

```

document 2e logic used here

```

589         \addpenalty \@beginparpenalty
590         \addvspace \l_block_effective_top_skip
591         \addvspace{-\parskip}
592     }
593 }
594 }

```

Extra keys to support enumitem conventions:

```

595 \keys_define:nn { template/block/display }
596 {
597   ,topsep          .skip_set:N = \topsep
598   ,partopsep      .skip_set:N = \partopsep
599   ,listparindent  .skip_set:N = \listparindent
600 }

```

```

\__kernel_displayblock_begin:
\__kernel_displayblock_beginpar_hmode:w
\__kernel_displayblock_beginpar_vmode:

```

The internal kernel hooks for tagging.

```

601 \cs_new:Npn \__kernel_displayblock_begin: {
602   \__block_debug_typeout:n{\detokenize{\__kernel_displayblock_begin:}}
603 }
604 \cs_new:Npn \__kernel_displayblock_beginpar_hmode:w {
605   \__block_debug_typeout:n{\detokenize{\__kernel_displayblock_beginpar_hmode:w}}
606 }
607 \cs_new:Npn \__kernel_displayblock_beginpar_vmode: {
608   \__block_debug_typeout:n{\detokenize{\__kernel_displayblock_beginpar_vmode:}}
609 }

```

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

6.5.4 Implementation of list templates ...

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

```

610 \tl_new:N \@itemlabel      % should have a top-level definition
611 \tl_new:N \@listctr       % should have a top-level definition

```

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

```
\__block_evaluate_saved_user_keys:nn
```

Keys set on individual list environments may be intended to alter the behavior of the template instance that defines the `\item` command. If meant to alter only a single `\item` command one would specify them in the optional argument of the `\item`, but if they should alter all items the right place would be the list environment. For this reason we need to store the values and then set them inside the `\item` template code using `\SetTemplateKeys` in the appropriate context (template type and template name). This is done in `__block_evaluate_saved_user_keys:nn`. The context is provided in the two arguments (because different list environments may use different `\item` instances based on different templates. By default the command does nothing because most environments do not have user key settings.

```
612 \cs_new_eq:NN \__block_evaluate_saved_user_keys:nn \use_none:nn
```

Maybe something like this should become a public function, but for now this is a one-off for the `\item` command and therefore coded inline and internal to the block code.

```

613 %\cs_new:Npn \__block_save_user_keys:n #1 {
614 % \tl_if_empty:NTF {#1}
615 %   { \cs_set_eq:NN \__block_evaluate_saved_user_keys:nn \use_none:nn }
616 %   {
617 %     \cs_set:Npe \__block_evaluate_saved_user_keys:nn ##1##2
618 %       { \SetTemplateKeys{##1}{##2}{ \exp_not:n{#1} } }
619 %   }
620 %}

```

(End of definition for `__block_evaluate_saved_user_keys:nn`.)

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

```

621 \DeclareTemplateCode{list}{std}{1}
622 {
623   counter           = \l__block_counter_tl,
624   item-label        = \l__block_item_label_tl,
625   start             = \l__block_counter_start_int ,
626   resume            = \l__block_resume_bool ,
627   item-instance     = \__block_item_instance:n ,
628   item-skip         = \itemsep ,
629 % item-par-skip     = \parsep ,
630   item-penalty      = \@itempenalty ,
631   item-indent       = \itemindent ,
632   label-width       = \labelwidth ,
633   label-sep         = \labelsep ,
634   legacy-support    = \l__block_legacy_support_bool , % FMI questionable
635 }
636 {
637   \__block_debug_typeout:n{template:list:std}
638 %

```

We start by looking at the user supplied keys in #1. If there aren't any we reset `__block_evaluate_saved_user_keys:nn` to do nothing. Otherwise we evaluate and set the keys in the context of the current list template. In addition we prepare `__block_evaluate_saved_user_keys:nn` for execution in the template for `\item`.

```

639 \tl_if_empty:NTF {#1}
640   { \cs_set_eq:NN \__block_evaluate_saved_user_keys:nn \use_none:nn }
641   {
642     \SetTemplateKeys{list}{std}{#1}
643     \cs_set:Npe \__block_evaluate_saved_user_keys:nn ##1##2
644       { \SetTemplateKeys{##1}{##2}{ \exp_not:n{#1} } }
645   }

```

Has this list a counter name defined in the instance?

```

646 \tl_if_empty:NTF \l__block_counter_tl
647   {

```

If not we check if `\@nmbrlist` is true which may be the case in legacy environments that used `\usecounter` in the argument to the `list` environment.

```

648   \legacy_if:nT { @nmbrlist }
649   {

```

In that case we only check if we should resume a previous list (`\@listctr` should be set in that case through the legacy method as well so we should be able to use it).

```

650     \bool_if:NF \l__block_resume_bool
651     {
652         \int_gset:cn{ c@ \@listctr }
653         { \l__block_counter_start_int - 1 }
654     }
655 }
656 }

```

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.

```

657     {
658         \@nmbulisttrue
659         \tl_set_eq:NN \@listctr \l__block_counter_tl
660         \bool_if:NF \l__block_resume_bool
661         {
662             \int_gset:cn{ c@ \@listctr }
663             { \l__block_counter_start_int - 1 }
664         }
665     }

```

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.

```

666     \tl_if_empty:NF \l__block_item_label_tl
667     {
668         \tl_set_eq:NN \@itemlabel \l__block_item_label_tl
669     }

```

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

```

670     \legacy_if_gset_true:n { @newlist }

```

If we encounter horizontal material before the first `\item` we do want a `\@noitemerr` straight away, because afterwards we end up with tagging structure faults whose cause is the missing `\item`. So we setup up `__block_item_everypar`: to test for this; when the first `\item` is encountered this will get reset. This is only relevant for vertical lists, when dealing with inline lists one would need to test for something else to identify that there is horizontal material between the start of the list and the first `\item` (maybe some `\spacefactor` trick could be used then, or the material is boxed first and the width is inspected as suggested by Joseph).

```

671     \cs_set_eq:NN \__block_item_everypar: \__block_item_everypar_first:
672     \__block_debug_typeout:n{template:list:std~end}
673 }

```

Extra keys to support enumitem conventions:

```

674 \keys_define:nn { template/list/std }
675 {
676     ,nosep .code:n =
677     \dim_zero:N \itemsep
678     \dim_zero:N \parsep

```

Think about a better implementation at some point.


```

679   \dim_zero:N \topsep
680   \dim_zero:N \l_block_botsep_skip
681   \dim_zero:N \l_block_parbotsep_skip
682 ,midsep .skip_set:N = \topsep
683 }

```

6.5.5 Implementation of \item template(s)

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

```

684 \keys_define:nn { template/item/std }
685   { label .tl_set:N = \l_block_label_given_tl }
686 \DeclareTemplateCode{item}{std}{1}
687 {
688   counter-label = \l_block_counter_label:n ,
689   counter-ref   = \l_block_counter_ref:n ,
690   label-ref     = \l_block_label_ref:n ,
691   label-autoref = \l_block_label_autoref:n ,
692   label-format  = \l_block_label_format:n ,
693   label-strut   = \l_block_label_strut_bool ,
694   label-boxed   = \l_block_label_boxed_bool ,
695   next-line     = \l_block_next_line_bool ,
696   text-font     = \l_block_text_font_tl ,
697   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)

```

698   label-align   = {
699     left   = \tl_set:Nn \l_block_item_align_tl { \relax \hss } ,
700     center = \tl_set:Nn \l_block_item_align_tl { \hss \hss } ,
701     right  = \tl_set:Nn \l_block_item_align_tl { \hss \relax } ,
702     parleft = \NOT_IMPLEMENTED ,
703   } ,
704 }

```

Then typeset the label at its natural width by applying `\l_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.

```

705 {
706   \l_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.

```

707   \tl_set_eq:NN \l_block_label_given_tl \c_novalue_tl

```

First we evaluate and set any keys specified on the list environment by calling `__block_evaluate_saved_user_keys:nn`. Then we do the same with all keys specified on this `\item` command (which may overwrite one or the other setting just made).

```
708 \__block_evaluate_saved_user_keys:nn {item}{std}
709 \tl_if_empty:nF{#1}{ \SetTemplateKeys{item}{std}{#1} }
```

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

```
710 \tl_if_novalue:oTF \l__block_label_given_tl
711 {
```

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

```
712 \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
713 \bool_if:NTF \l__block_item_compatibility_bool % not sure that conditional
714 % makes sense
715 { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{\@itemlabel} } } % TODO ?
716 { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{\__block_counter_label:n} } }
717 }
718 {
719 \__block_debug_typeout:n{item~ with~ optional}
720 \__block_make_label_box:n { \l__block_label_given_tl } }
721 \bool_if:nT
722 {
723 \l__block_label_boxed_bool
724 && \dim_compare_p:n { \box_wd:N \l__block_one_label_box <= \linewidth } % TODO: is \
725 }
726 {
727 \dim_compare:nNnT
728 { \box_wd:N \l__block_one_label_box } < \linewidth
729 {
730 \hbox_set_to_wd:Nnn \l__block_one_label_box { \linewidth }
731 {
732 \exp_after:wN \use_i:nn \l__block_item_align_tl
```

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

```
733 % \hbox_unpack_drop:N \l__block_one_label_box %TODO: customize?
734 \box_use_drop:N \l__block_one_label_box
735 \exp_after:wN \use_ii:nn \l__block_item_align_tl
736 }
737 }
```

Add another box level to the label box:

```
738 \hbox_set:Nn \l__block_one_label_box
739 { \box_use_drop:N \l__block_one_label_box }
740 }
741 \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \linewidth
742 { \bool_set_true:N \l__block_long_label_bool }
743 { \bool_set_false:N \l__block_long_label_bool }
744 \hbox_gset:Nn \g__block_labels_box
745 {
```

fix

```

746     \hbox_unpack_drop:N \g__block_labels_box
747     \skip_horizontal:n { \itemindent - \labelsep - \labelwidth }
748     \hbox_unpack_drop:N \l__block_one_label_box
749     \skip_horizontal:n { \labelsep }
750     \bool_if:NT \l__block_next_line_bool
751     { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
752     % version of \newline inside an hbox that will be unpacked
753   }
754   % \skip_set_eq:NN \parsep \l__block_item_parsep_skip TODO??? FMI
755                                     % what's that?
756   \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.

```

757   \cs_set_eq:NN \__block_item_everypar: \__block_item_everypar_std:
758 }

```

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

```

759 \box_new:N \l__block_one_label_box
760 \box_new:N \g__block_labels_box

```

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

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

```

761 \bool_new:N \l__block_long_label_bool

```

(End of definition for `\l__block_long_label_bool`.)

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

```

762 \cs_new_protected:Npn \__block_make_label_box:n #1
763 {
764   \hbox_set:Nn \l__block_one_label_box
765   {

```

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

```

766     \__kernel_list_label_begin:
767     \__block_label_format:n
768     {
769       \bool_if:NT \l__block_label_strut_bool { \strut }
770       \bool_if:NTF \l__block_legacy_support_bool
771         \makeatlabel
772         \use:n
773         {#1}
774     }

```

And what gets opened also needs closing:

```
775     \_kernel_list_label_end:
776     }
777 }
```

(End of definition for `_block_make_label_box:n` and `_block_label_format:e`.)

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

```
\_kernel_list_label_end: 778 \cs_new_eq:NN \_kernel_list_label_begin: \prg_do_nothing:
779 \cs_new_eq:NN \_kernel_list_label_end: \prg_do_nothing:
```

(End of definition for `_kernel_list_label_begin:` and `_kernel_list_label_end:.`)

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

```
780 \cs_new_eq:NN \_block_item_everypar: \prg_do_nothing:
781 \AddToHook{para/begin}[items]{\_block_item_everypar:}
```

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

By the end of the day both should probably move into the kernel hook instead or, better, into sockets.

```
782 \DeclareHookRule{para/begin}[items]{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.

```
783 \cs_new_protected:Npn \_block_item_everypar_std: {
784   \_block_debug_typeout:n{item~ everypar \on@line }
785   \legacy_if_set_false:n { @minipage }
786   \legacy_if_gset_false:n { @newlist }
787   \legacy_if:nT { @inlabel }
788   {
789     \legacy_if_gset_false:n { @inlabel }
790     \box_if_empty:NT \g_para_indent_box { \kern - \itemindent }
791     \para_omit_indent:
792     \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:

```
793   \_kernel_list_label_after:
794   \penalty \c_zero_int
795   }
796   \legacy_if:nTF { @nobreak }
797   {
798     \legacy_if_gset_false:n { @nobreak }
799     \int_set:Nn \clubpenalty { 10000 }
800   }
801   {
802     \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.

```
803     \cs_set_eq:NN \_block\_item\_everypar: \prg\_do\_nothing:
804     }
805 }
```

This is the definition of `_block_item_everypar`: before the first `\item` is encountered.

```
806 \cs_new:Npn \_block\_item\_everypar\_first: {
807   \legacy\_if:nT { @newlist } { \@noitemerr }
808 }
```

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

`_kernel_list_label_after`:

```
809 \cs_new_eq:NN \_kernel\_list\_label\_after: \prg\_do\_nothing:
```

(End of definition for `_kernel_list_label_after`:.)

`\l_block_tmpa_skip`

```
810 \skip_new:N \l\_block\_tmpa\_skip
```

(End of definition for `\l_block_tmpa_skip`:.)

`\l_block_topsepadd_skip`

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

```
811 \skip_new:N \l\_block\_topsepadd\_skip
812 \skip_new:N \l\_block\_effective\_top\_skip
```

(End of definition for `\l_block_topsepadd_skip` and `\l_block_effective_top_skip`:.)

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

```
813 \AddToHook{begindocument/before}{
814   \RenewDocumentCommand{\item}{={label}o }
815   {
816     \@inmatherr \item
```

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

```
817   \cs_if_free:NTF \_block\_item\_instance:n
818   {
819     \@latex@error{Lonely~\string\item--perhaps-a~missing~
820     list~environment}\@ehc
821   }
822   {
823     \legacy\_if:nTF { @newlist }
824     {
825       \_kernel\_list\_item\_begin:
```

The first item of a list also has to change the `@newlist` switch.

```
826         \legacy_if_gset_false:n { @newlist }
827     }
828     { \__block_inter_item: }
```

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

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

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

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

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

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

```
837 \cs_new_protected:Npn \__block_inter_item: {
838     \legacy_if:nT { @inlabel }
839     { \indent \par } % case of \item\item
```

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

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

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

```
842     \__kernel_list_item_end:
843     \__kernel_list_item_begin:
844     \addpenalty \@itempenalty
845     \addvspace \itemsep
846 }
```

(End of definition for `__block_inter_item:`.)

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

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

6.6 Tagging recipes

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

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

```

849 \cs_new:Npn \__block_recipe_basic: {
850   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
851                                     \__block_beginpar_hmode:N
852   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
853                                     \__block_beginpar_vmode:
854   \let \__kernel_displayblock_begin: \prg_do_nothing:
855   \let \__kernel_displayblock_end:  \prg_do_nothing:

```

End environment `\par` handling:

```

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

```

(End of definition for `__block_recipe_basic:.`)

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

```

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

```

End environment `\par` handling:

```

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

```

(End of definition for `__block_recipe_standalone:.`)

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

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

```

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

```

End environment `\par` handling:

```

878   \socket_assign_plug:nn{tag-support/block-endpe}{on}
879   \tl_if_empty:NTF \l__block_tag_name_tl
880     { \tl_set:Nn \l__block_tag_inner_tag_tl {Figure}          }
881     { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
882 }

```

(End of definition for `__block_recipe_standard:.`)

`\l__block_tag_inner_tag_tl`

```

883 \tl_new:N \l__block_tag_inner_tag_tl

```

(End of definition for `\l__block_tag_inner_tag_tl.`)

`__block_recipe_list:` The list recipe does the following.

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

```

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

```

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

```

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

```

End environment `\par` handling:

```

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

```


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

```

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

```

(End of definition for `__block_recipe_list:.`)

6.7 Blockenv instances

6.7.1 Basic instances

`blockenv displayblock` (*inst.*)

```

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

```

`blockenv displayblockflattened` (*inst.*)

```

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

```

`blockenv center` (*inst.*)

```

927 \DeclareInstance{blockenv}{center}{display}
928 {
929   env-name      = center,
930   tag-name      = ,
931   tag-class     = ,
932   tagging-recipe = basic,
933   inner-level-counter = ,
934   level-increase = false,

```

```

935 setup-code      = ,
936 block-instance = displayblock ,
937 para-flattened = true ,
938 para-instance  = center ,
939 inner-instance = ,
940 }

```

`blockenv flushleft` (*inst.*)

```

941 \DeclareInstance{blockenv}{flushleft}{display}
942 {
943   env-name      = flushleft,
944   tag-name      = ,
945   tag-class     = ,
946   tagging-recipe = basic,
947   inner-level-counter = ,
948   level-increase = false,
949   setup-code    = ,
950   block-instance = displayblock ,
951   para-flattened = true ,
952   para-instance = raggedright ,
953   inner-instance = ,
954 }

```

`blockenv flushright` (*inst.*)

```

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

```

6.7.2 Blockquote instances

`blockenv quotation` (*inst.*)

```

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

```

blockenv quote (*inst.*)

```
981 \DeclareInstance{blockenv}{quote}{display}
982 {
983   env-name      = quote,
984   tag-name      = quote,
985   tag-class     = ,
986   tagging-recipe = standard,
987   inner-level-counter = ,
988   level-increase = true,
989   setup-code    = ,
990   block-instance = quoteblock ,
991   inner-instance = ,
992 }
```

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

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

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

```

1029 % inner-instance = theorem,
1030 }

```

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

6.7.3 Verbatim instances

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

```

1031 \DeclareInstance{blockenv}{verbatim}{display}
1032 {
1033   env-name      = verbatim,
1034   tag-name      = verbatim,
1035   tag-class     = ,
1036   tagging-recipe = standard,
1037   inner-level-counter = ,
1038   level-increase = false,
1039   setup-code    = ,
1040   block-instance = verbatimblock ,
1041   inner-instance = ,
1042   final-code    = \legacyverbatimsetup ,
1043   para-instance = justify
1044 }

```

6.7.4 Standard list instances

`blockenv itemize (inst.)`

```

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

```

`blockenv enumerate (inst.)`

```

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

```

```

1068 max-inner-levels = 4,
1069 inner-instance   = enum ,
1070 }

```

blockenv description (*inst.*)

```

1071
1072 \DeclareInstance{blockenv}{description}{display}
1073 {
1074   env-name       = description,
1075   tag-name       = description,
1076   tag-class      = description,
1077   tagging-recipe = list,
1078   inner-level-counter = ,
1079   level-increase = true,
1080   setup-code     = ,
1081   block-instance = list ,
1082   inner-instance = description ,
1083 }

```

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

```

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

```

6.8 Block instances

6.8.1 Displayblock instances

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

```

1096 \setcounter{maxblocklevels}{6}

```

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

```

block displayblock-1 (inst.)
block displayblock-2 (inst.)
block displayblock-3 (inst.) 1097 \DeclareInstance{block}{displayblock-0}{display}
block displayblock-4 (inst.) 1098 {
block displayblock-5 (inst.) 1099   leftmargin       = Opt ,
block displayblock-6 (inst.) 1100   parindent       = Opt ,
1101 }

```

```

1102 \DeclareInstanceCopy{block}{displayblock-1}{displayblock-0}
1103 \DeclareInstanceCopy{block}{displayblock-2}{displayblock-0}
1104 \DeclareInstanceCopy{block}{displayblock-3}{displayblock-0}
1105 \DeclareInstanceCopy{block}{displayblock-4}{displayblock-0}
1106 \DeclareInstanceCopy{block}{displayblock-5}{displayblock-0}
1107 \DeclareInstanceCopy{block}{displayblock-6}{displayblock-0}

```

6.8.2 Verbatim instances

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

```

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

```

6.8.3 Quote/quotationblock instances

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

```

block quoteblock-1 (inst.) Default layout is to indent equally from both sides.
block quoteblock-2 (inst.) 1120 \DeclareInstance{block}{quoteblock-1}{display}
block quoteblock-3 (inst.) 1121 { rightmargin = \KeyValue{leftmargin} }
block quoteblock-4 (inst.) 1122 \DeclareInstanceCopy{block}{quoteblock-2}{quoteblock-1}
block quoteblock-5 (inst.) 1123 \DeclareInstanceCopy{block}{quoteblock-3}{quoteblock-1}
block quoteblock-6 (inst.) 1124 \DeclareInstanceCopy{block}{quoteblock-4}{quoteblock-1}
1125 \DeclareInstanceCopy{block}{quoteblock-5}{quoteblock-1}
1126 \DeclareInstanceCopy{block}{quoteblock-6}{quoteblock-1}

block quotationblock-1 (inst.) Quotation additionally changes the parindent.
block quotationblock-2 (inst.) 1127 \DeclareInstance{block}{quotationblock-1}{display}
block quotationblock-3 (inst.) 1128 { parindent = 1.5em , rightmargin = \KeyValue{leftmargin} }
block quotationblock-4 (inst.) 1129 \DeclareInstanceCopy{block}{quotationblock-2}{quotationblock-1}
block quotationblock-5 (inst.) 1130 \DeclareInstanceCopy{block}{quotationblock-3}{quotationblock-1}
block quotationblock-6 (inst.) 1131 \DeclareInstanceCopy{block}{quotationblock-4}{quotationblock-1}
1132 \DeclareInstanceCopy{block}{quotationblock-5}{quotationblock-1}
1133 \DeclareInstanceCopy{block}{quotationblock-6}{quotationblock-1}

```

6.8.4 Block instances for the standard lists

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

```

block list-4 (inst.) 1134 \DeclareInstance{block}{list-1}{display}{
block list-5 (inst.) 1135 % heading = ,
block list-6 (inst.) 1136 % beginsep = \topsep ,
1137 % begin-par-skip = \partopsep ,
1138 % par-skip = \parsep ,
1139 % end-skip = \KeyValue{beginsep} ,
1140 % end-par-skip = \KeyValue{begin-par-skip} ,
1141 % beginpenalty = \UseName{@beginparpenalty} ,
1142 % endpenalty = \UseName{@endparpenalty} ,
1143 % leftmargin = \leftmargin ,
1144 % rightmargin = \rightmargin ,
1145 % parindent = \listparindent ,
1146 }
1147 \DeclareInstance{block}{list-2}{display}{}
1148 \DeclareInstance{block}{list-3}{display}{}
1149 \DeclareInstance{block}{list-4}{display}{}
1150 \DeclareInstance{block}{list-5}{display}{}
1151 \DeclareInstance{block}{list-6}{display}{}

```

6.9 List instances for the standard lists

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

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

```

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

```

`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

```

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

```

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

```

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

```

6.10 Item instances

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

```

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

```

6.11 Para instances

```

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

```

`para center (inst.)`

```

1188 \DeclareInstance{para}{center}{std}
1189 {
1190   indent-width      = Opt ,
1191   start-skip        = Opt ,
1192   left-skip         = \@flushglue ,
1193   right-skip        = \@flushglue ,
1194   end-skip          = \z@skip ,
1195   final-hyphen-demerits = 0 ,
1196   cr-command        = \@centercr ,
1197   para-class        = center ,
1198 }
1199 \DeclareInstance{para}{raggedright}{std}
1200 {
1201   indent-width      = Opt ,
1202   start-skip        = Opt ,

```



```

1203 left-skip          = \z@skip ,
1204 right-skip         = \@flushglue ,
1205 end-skip           = \z@skip ,
1206 final-hyphen-demerits = 0 ,
1207 cr-cmd             = \@centercr ,
1208 para-class         = raggedright ,
1209 }

1210 \DeclareInstance{para}{raggedleft}{std}
1211 {
1212   indent-width      = Opt ,
1213   start-skip        = Opt ,
1214   left-skip         = \@flushglue ,
1215   right-skip        = \z@skip ,
1216   end-skip          = \z@skip ,
1217   final-hyphen-demerits = 0 ,
1218   cr-cmd            = \@centercr ,
1219   para-class        = raggedleft ,
1220 }

1221 \DeclareInstance{para}{justify}{std}
1222 {
1223   % indent-width      = Opt ,
1224   start-skip         = Opt ,
1225   left-skip          = \z@skip ,
1226   right-skip         = \z@skip ,
1227   end-skip           = \@flushglue ,
1228   final-hyphen-demerits = 5000 ,
1229   cr-cmd             = \@normalcr ,
1230   para-class         = justify ,
1231 }

1232 \DeclareRobustCommand\centering {\UseInstance{para}{center}{}}
1233 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}{}}
1234 \DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}{}}
1235 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}{}}
1236
1237 \justifying

```

6.12 Tagging support

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

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

```

1238 \tag_if_active:TF {

```

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

```

1239   \cs_set:Npn \_block_beginpar_vmode: {
1240     \_block_debug_typeout:n
1241     { @endpe = \legacy_if:nTF { @endpe }{true}{false}

```

```

1242         \on@line }
1243     \legacy_if:nTF { @endpe }
1244     {
1245         \legacy_if_gset_false:n { @endpe }
1246     }

```

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

```

1247     {
1248         \int_compare:nNnT \l__tag_block_flattened_level_int < 2
1249         {
1250             \__tag_gincr_para_main_begin_int:
1251             \tag_struct_begin:n
1252             {
1253                 tag=\l__tag_para_main_tag_tl,
1254                 attribute-class=\l__tag_para_main_attr_class_tl,
1255             }
1256             \__tag_para_main_store_struct:
1257         }
1258     }
1259 }

```

(End of definition for `__block_beginpar_vmode:`)

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

```

1260 \cs_set:Npn \__block_beginpar_hmode:N #1
1261 {
1262     \tag_mc_end:
1263     \__tag_gincr_para_end_int:
1264     \__block_debug_typeout:n{increment~ /P \on@line }
1265     \bool_if:NT \l__tag_para_show_bool
1266     { \tag_mc_begin:n{artifact}
1267       \rlap{\color_select:n{red}\tiny\ \int_use:N\g__tag_para_end_int}
1268     }
1269     \tag_struct_end:
1270     \tagpdfpara0ff \par \tagpdfpara0n
1271 }
1272 }

```

(End of definition for `__block_beginpar_hmode:N`)

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

```

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

```

Given that restoring `\par` through the legacy L^AT_EX 2_ε method can take a few iterations (for example, in case of nested lists, e.g., `... \end{itemize} \item ... \par` it can happen that `__kernel_displayblock_doendpe:` is called while `@endpe` is already handled

and then we should not attempt to close a `text-unit` structure). So we need to check for this.

```

1276     \legacy_if:nT { @endpe }
1277     {
If the display block currently ending was “flattened” (i.e., uses simplified paragraphs that
are not tagged by a combination of text-unit followed by <text>, but simply with a
<text>), then we don’t have to do anything, because the <text> is already closed.
1278         \__block_debug_typeof:n
1279         { flattened= \bool_if:NTF
1280             \l__tag_para_flattened_bool {true}{false}
1281             \on@line }
1282     \bool_if:NF \l__tag_para_flattened_bool
1283     {
1284         \__block_debug_typeof:n{Structure-end~
1285             \l__tag_para_main_tag_tl\space after~ displayblock \on@line }
1286         \__tag_gincr_para_main_end_int:
1287         \tag_struct_end: %text-unit
1288     }
1289 }
1290 }
1291 }

```

(End of definition for `__kernel_displayblock_doendpe:`.)

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

```

1292 \RemoveFromHook{para/begin}[tagpdf]
1293 \AddToHook{para/begin}[tagpdf]{
1294     \bool_if:NT \l__tag_para_bool {

```

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

```

1295     \legacy_if:nF { @inlabel }
1296     {

```

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

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

```

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

```

```

\__block_start_para_structure:n 1301 \cs_new_protected:Npn \__block_start_para_structure:n #1 {
1302     \__block_debug_typeof:n
1303     { @endpe = \legacy_if:nTF { @endpe }{true}{false}

```

```

1304     \on@line }
1305 \legacy_if:nF { @endpe }
1306 {
1307     \bool_if:NF \l__tag_para_flattened_bool
1308     {
1309         \__tag_gincr_para_main_begin_int:
1310         \tag_struct_begin:n
1311         {
1312             tag=\l__tag_para_main_tag_tl,
1313             attribute-class=\l__tag_para_main_attr_class_tl,
1314         }
1315         \__tag_para_main_store_struct:
1316     }
1317 }
1318 \__tag_gincr_para_begin_int:
1319 \__block_debug_typeout:n{increment~ P \on@line }
1320 \tag_struct_begin:n
1321 {
1322     tag=\l__tag_para_tag_tl
1323     ,attribute-class=\l__tag_para_attr_class_tl
1324 }
1325 \__tag_check_para_begin_show:nn {green}{#1}
1326 \tag_mc_begin:n {}
1327 }

```

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

```

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

1349 \RemoveFromHook{para/end}[tagpdf]
1350 \AddToHook{para/end}
1351 {
1352     \bool_if:NT \l__tag_para_bool
1353     {
1354         \__tag_gincr_para_end_int:

```

```

1355     \_block_debug_timeout:n{increment~ /P \on@line }
1356     \tag_mc_end:
1357     \_tag_check_para_end_show:nn {red}{ }
1358     \tag_struct_end:
1359     \bool_if:NF \l__tag_para_flattened_bool
1360     {
1361         \_tag_gincr_para_main_end_int:
1362         \tag_struct_end:
1363     }
1364 }
1365 }
1366 \def\PARALABEL{NP-}

```

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

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

```

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

```

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

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

```

1393 \DeclareRobustCommand*\begin[1]{%
1394     \UseHook{env/#1/before}%
1395     \@ifundefined{#1}%
1396         {\def\reserved@a{\@latex@error{Environment~#1~undefined}\@eha}}%
1397         {\def\reserved@a{\def\@currentenv{#1}%

```

```

1398     \edef\@currentvline{\on@line}%
1399     \@execute@begin@hook{#1}%
1400     \csname #1\endcsname}}%
1401 \ignorefalse
1402 \begingroup
1403   \__kernel_endpe_vmode:
1404   \reserved@a}

```

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

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

```

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

```

(End of definition for `__kernel_endpe_vmode:.`)

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

```

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

```

(End of definition for `__kernel_list_label_after:.`)

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

```

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

```

(End of definition for `__block_inner_begin:.`)

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

```

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

```

```

1435 \tagstructend          % end inner structure
1436 }

```

(End of definition for `_block_inner_end:`.)

6.12.1 List tags

```

1437 \tl_new:N \l__tag_L_tag_tl
1438 \tl_set:Nn \l__tag_L_tag_tl {L}
1439
1440 \tl_new:N \l__tag_L_attr_class_tl
1441 \tl_set:Nn \l__tag_L_attr_class_tl {list}
1442
1443 \tag_if_active:T
1444 {
1445   \tagpdfsetup
1446   {
1447     role/new-attribute = {itemize}{/O /List /ListNumbering/Unordered},
1448     role/new-attribute = {enumerate}{/O /List /ListNumbering/Ordered},
1449     role/new-attribute = {description}{/O /List /ListNumbering/Description},
1450
1451     % default if unknown
1452     role/new-attribute = {list}{/O /List /ListNumbering/Unordered},
1453   }
1454 }
1455
1456 \def\LItag{LI}

```

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

`_block_list_begin:` Start a list ...

```

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

```

(End of definition for `_block_list_begin:`.)

`_block_list_item_begin:` Start tagging a list item.

```

1461 \cs_set:Npn \_block\_list\_item\_begin: { \tagstructbegin{tag=\LItag} }

```

(End of definition for `_block_list_item_begin:`.)

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

```

1462 \cs_set:Npn \_kernel\_list\_label\_begin: {
1463   %
1464   % FMi: this needs a different logic to decide when to make the label
1465   %   an artifact (after cleaning up the \item code ), therefore
1466   %   disabled for now
1467   % \tl_if_empty:oTF \@itemlabel
1468   % {
1469   %   \tag_mc_begin:n {artifact}
1470   % }
1471   % {

```

```

1472     \tagstructbegin{tag=Lbl}
1473     \tagmcbegin{tag=Lbl}
1474 %     }
1475 }

```

(End of definition for _kernel_list_label_begin:.)

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

```

1476 \cs_set:Npn \_kernel\_list\_label\_end: {
1477   \tagmcbend                                % end mc-Lbl or artifact
1478 % Fmi: unconditionally for now
1479 % \tl_if_empty:oF \@itemlabel
1480     \tagstructend % end Lbl
1481   \tagstructbegin{tag=LBody}
1482 }
1483 \def\LBody{LBody}

```

(End of definition for _kernel_list_label_end:.)

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

```

1484 \cs_set:Npn \_block\_list\_item\_end: {
1485   \legacy_if:nT { @endpe }
1486   {
1487     \_tag_gincr_para_main_end_int:
1488     \tagstructend                                % text-unit
1489 %     \_block_debug_typeout:n{Structure-end~ P~ at~ item-end \on@line }
1490   }
1491   \tagstructend \tagstructend % end LBody, LI
1492 }

```

(End of definition for _block_list_item_end:.)

`_block_list_end:` Finally, at the list end we have to close the open LBody, LI, L, and possibly a <text> if the last item ends with a list. However, if the user forgot to add an \item then there will be no LI and LBody open, so we check for the status of @newlist. The corresponding no-item error was generated earlier outside the tagging code.

One could argue that it doesn’t matter if the tagging is wrong after a \@noitemerr was issued. However, there is one case where it isn’t an error: In the thebibliography environment (which is internally a list) it is often the case that documents start out with an empty environment, not containing any \bibitems. For that reason \@noitemerr is redefined inside that environment to only produce a warning; hence we have to produce correct tag structures in that case.

```

1493 \cs_set:Npn \_block\_list\_end: {

```

If @newlist is true (i.e., when we have an error or warning situation) there is not much to close.

```

1494   \legacy_if:nF { @newlist }
1495   {
1496     \legacy_if:nT { @endpe }
1497     {
1498       \_tag_gincr_para_main_end_int:

```



```

1499         \tagstructend                               % text-unit
1500         \_block_debug_typeof:n{Structure-end- P~ at~ list-end \on@line }
1501     }
1502     \tagstructend\tagstructend % end LBody, LI
1503 }
1504 \tagstructend                               % end L
1505 }

```

(End of definition for `_block_list_end:`.)

End of tagging related declarations.

```
1506 }
```

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

```

1507 {
1508   \cs_new:Npn \_block_start_para_structure_unconditionally:n #1 {}
1509 }
1510 </package>
1511 <*latex-lab>
1512 \ProvidesFile{block-latex-lab-testphase.ltx}
1513     [\ltlabblockdate\space v\ltlabblockversion\space
1514         blockenv implementation]
1515 \RequirePackage{latex-lab-testphase-block}
1516 </latex-lab>

```

7 Documentation from first prototype implementations

7.1 Open questions

- Existing questions — moved to issues —

7.2 Code cleanup

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

7.3 Tasks

- Change author to LaTeX Team once it's nice enough to deserve that label.
- Reproducing exactly the standard layouts and examples in the `enumitem` documentation.
- Hooks, but do not duplicate those that already exist as environment hooks. Hence, mostly around items.

- Customization and interaction with LDB:
 - Allow arbitrary nesting depth with automatically defined styles for labels, counters etc.
 - Adapt everything to font size! (e.g. footnotes).
 - How to model the inheritance from `trivlist` to `list` to `enumerate`?
- Add key–value settings mimicking `enumitem`'s ability to set any four of five horizontal parameters and deduce the fifth by `\leftmargin + \itemindent = \labelindent + \labelwidth + \labelsep`.
- Provide good ways to customize how overlong labels are dealt with.
- Use the `.aux` file.
 - Implement the `\ref` styles that `enumitem` provides.
 - Reverse enumerations, important in publication lists and the like. Somehow avoid needing 3 compilations for references to reverse enumerations to settle?
 - Ability to calculate `\labelwidth` from the label contents. Share calculated parameters between multiple environments (cf. `resume` option).
- Related to grabbing the whole list environment, and input syntax variations:
 - Other layouts: `tabular` (see `listliketab` vs `typed-checklist`), `multicolumn` and horizontally numbered (see `tasks`), inline lists, `runin` lists in the easy case where there is no intervening `\par`.
 - Formatting the item text in a box or similar (requires grabbing the whole list).
 - Filtering which items to show: hide certain items according to criteria (useful together with `list reuse`), see `typed-checklist`.
 - Shorthands `\iitem` for automatic nested lists, or `\1`, `\2` etc from `outlines`.
 - Support markdown input like `asciilist`.
- Check interaction with `babel` options such as `french` or `accadian` (see `FrenchItemizeSpacing`)
- RTL and vertical typesetting.

8 Plan of attack of first prototype

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

- It has a certain layout in the page, with vertical and horizontal spacing around it. This type of layout is shared with environments such as `quote`, `flushright`, or `tabbing`. This common layout is implemented in L^AT_EX 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 L^AT_EX 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 template² that omits the `heading` parameter and whose default for `item-instance` is non-empty.

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

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

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

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

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

- Label formatting: `label-format` function, `label-strut` boolean.
- Label alignment (`label-align`, `label-boxed`, `next-line`).
- Content parameters: `text-font`.
- A `compatibility` boolean that controls for instance whether `\makelabel` is used.

document class
customizations

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

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

Index

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

Symbols	
<code>\</code>	210, 496
<code>_</code>	313, 337, 345, 1267
Numbers	
<code>\1</code>	58
<code>\2</code>	58
A	
<code>\addpenalty</code>	470, 589, 844
<code>\AddToHook</code>	138, 149, 157, 196, 207, 224, 251, 781, 813, 1293, 1350
<code>\addvspace</code>	471, 587, 590, 591, 845
<code>\arabic</code>	7, 92
B	
<code>\begin</code>	12, 54, 1393
<code>\begingroup</code>	1402
<code>\bfseries</code>	21, 311, 335, 1175
<code>\bibitem</code>	56
<code>block</code> (objecttype)	26
block commands:	
<code>\block_debug_off:</code> ..	11, 112, 117, 130
<code>\block_debug_on:</code> ..	11, 112, 112, 129
<code>\g_block_nesting_depth_int</code>	11, 356, 400, 404, 405, 414, 417, 448
<code>block display</code> (template)	48, 503
<code>block displayblock-0</code> (instance) ...	1097
<code>block displayblock-1</code> (instance) ...	1097
<code>block displayblock-2</code> (instance) ...	1097
<code>block displayblock-3</code> (instance) ...	1097
<code>block displayblock-4</code> (instance) ...	1097
<code>block displayblock-5</code> (instance) ...	1097
<code>block displayblock-6</code> (instance) ...	1097
block internal commands:	
<code>__block_beginpar_hmode:N</code>	851, 873, 887, 1260, 1260
<code>__block_beginpar_vmode:</code>	853, 875, 889, 1239, 1239
<code>\l_block_block_instance_tl</code>	366, 414, 416, 477
<code>\l_block_botsep_skip</code>	509, 680
<code>__block_counter_label:n</code>	688, 716
<code>__block_counter_ref:n</code>	689
<code>\l_block_counter_start_int</code>	625, 653, 663
<code>\l_block_counter_tl</code> ..	623, 646, 659
<code>__block_debug:n</code>	110, 110, 124
<code>\g_block_debug_bool</code>	109, 114, 119, 125, 127
<code>__block_debug_gset:</code> ..	112, 115, 120, 122
<code>__block_debug_timeout:n</code>	110, 111, 126, 376, 413, 421, 426, 446, 460, 479, 602, 605, 608, 637, 672, 706, 719, 784, 1240, 1264, 1278, 1284, 1302, 1319, 1340, 1355, 1428, 1432, 1489, 1500
<code>\l_block_effective_top_skip</code> ..	541, 543, 590, 811

⁴This should be made easily extendible to deeper levels.

<code>\l_block_env_name_tl</code>	360, 376	<code>\l_block_long_label_bool</code>	
<code>_block_evaluate_saved_user_</code>			
<code>keys:nn</code>	30, 31,		
34, 612, 612, 615, 617, 640, 643, 708		<code>_block_make_label_box:n</code>	
<code>\l_block_final_code_tl</code>	25, 373, 437		33, 715, 716, 720, 762, 762
<code>\l_block_heading_tl</code>	505, 520, 521	<code>\l_block_max_inner_levels_tl</code>	
<code>_block_if_list:TF</code> 454, 459, 476, 476			369, 394
<code>_block_inner_begin:</code>		<code>\l_block_next_line_bool</code>	695, 750
	863, 876, 1424, 1424	<code>\l_block_one_label_box</code>	
<code>_block_inner_end:</code>			35, 724, 728, 730,
	864, 877, 1427, 1427		733, 734, 738, 739, 741, 748, 759, 764
<code>\l_block_inner_instance_tl</code>		<code>\l_block_para_instance_tl</code>	
	371, 424, 426, 430		367, 419, 421, 422
<code>\l_block_inner_instance_type_tl</code>		<code>\l_block_parbotsep_skip</code>	510, 681
	370, 429	<code>_block_recipe_basic:</code>	849, 849
<code>\l_block_inner_level_counter_tl</code>		<code>_block_recipe_list:</code>	884, 884
	368, 391, 393, 396, 427, 428, 431, 432	<code>_block_recipe_standalone:</code>	858, 858
<code>_block_inter_item:</code> 37, 828, 837, 837		<code>_block_recipe_standard:</code>	870, 870
<code>\l_block_item_align_tl</code>		<code>\l_block_resume_bool</code>	626, 650, 660
	8, 699, 700, 701, 732, 735	<code>_block_save_user_keys:n</code>	613
<code>\l_block_item_compatibility_</code>		<code>\l_block_setup_code_tl</code>	24, 365, 412
<code>bool</code>	697, 713	<code>_block_skip_remove_last:</code>	
<code>_block_item_everypar:</code>	32,		104, 107, 456, 530, 840, 841
35-37, 671, 757, 780, 780, 781, 803		<code>_block_skip_set_to_last:N</code>	
<code>_block_item_everypar_first:</code>			104, 104, 464, 580
	671, 780, 806	<code>_block_start_para_structure:n</code>	
<code>_block_item_everypar_std:</code>			1297, 1301, 1301
	757, 780, 783	<code>_block_start_para_structure_</code>	
<code>_block_item_instance:n</code>		<code>unconditionally:n</code>	
	37, 627, 817, 830, 831		323, 350, 1328, 1421, 1508
<code>\l_block_item_label_tl</code> 624, 666, 668		<code>\l_block_tag_class_tl</code>	362, 898, 900
<code>\l_block_item_parsep_skip</code>	754	<code>\l_block_tag_inner_tag_tl</code>	
<code>_block_label_autoref:n</code>	691		867, 868, 880, 881, 883, 1425
<code>\l_block_label_boxed_bool</code>	694, 723	<code>\l_block_tag_name_tl</code>	
<code>_block_label_format:n</code>			361, 866, 868, 879, 881, 895, 897
	35, 692, 762, 767	<code>\l_block_tagging_recipe_tl</code> 363, 408	
<code>\l_block_label_given_tl</code>		<code>\l_block_text_font_tl</code>	696
	33, 34, 685, 707, 710, 720	<code>\l_block_thm_current_counter_tl</code>	
<code>_block_label_ref:n</code>	690		298, 301, 308, 332
<code>\l_block_label_strut_bool</code>	693, 769	<code>\l_block_tmpa_skip</code> 580, 581, 582, 810	
<code>\g_block_labels_box</code>		<code>\l_block_topsepadd_skip</code>	
	33, 35, 572, 575, 744, 746, 759, 792		26, 471, 523, 526, 541, 811
<code>\l_block_legacy_env_params_tl</code>	19, 9, 227, 243	<code>block list-1 (instance)</code>	1134
<code>\l_block_legacy_support_bool</code>	634, 770	<code>block list-2 (instance)</code>	1134
<code>\l_block_level_incr_bool</code>		<code>block list-3 (instance)</code>	1134
	364, 398, 447	<code>block list-4 (instance)</code>	1134
<code>_block_list_begin:</code>	890, 1454, 1454	<code>block list-5 (instance)</code>	1134
<code>_block_list_end:</code>	891, 1493, 1493	<code>block list-6 (instance)</code>	1134
<code>_block_list_item_begin:</code>		<code>block quotationblock-1 (instance)</code>	1127
	892, 1461, 1461	<code>block quotationblock-2 (instance)</code>	1127
<code>_block_list_item_end:</code>		<code>block quotationblock-3 (instance)</code>	1127
	893, 1484, 1484	<code>block quotationblock-4 (instance)</code>	1127
		<code>block quotationblock-5 (instance)</code>	1127
		<code>block quotationblock-6 (instance)</code>	1127
		<code>block quoteblock-1 (instance)</code>	1120

<code>displayblock (env.)</code>	132		
<code>displayblockflattened (env.)</code>	135		
<code>\do</code>	182		
<code>\dospecials</code>	182		
E			
<code>\edef</code>	1398		
<code>\else</code>	178		
<code>\end</code>	12 , 457		
<code>\endblockenv</code>	16 , 134 , 137 , 141 , 144 , 147 , 152 , 155 , 163 , 169 , 199 , 202 , 205 , 222 , 234 , 260 , 355 , 445		
<code>endblockenv</code>	11		
<code>\endcsname</code>	264 , 280 , 1400		
<code>\endgraf</code>	1392		
<code>enumerate (env.)</code>	196		
environments:			
<code>center</code>	138		
<code>description</code>	196		
<code>displayblock</code>	132		
<code>displayblockflattened</code>	135		
<code>enumerate</code>	196		
<code>flushleft</code>	138		
<code>flushright</code>	138		
<code>itemize</code>	196		
<code>list</code>	224		
<code>quotation</code>	149		
<code>quote</code>	149		
<code>trivlist</code>	251		
<code>verbatim</code>	157		
<code>verbatim*</code>	157		
<code>verse</code>	207		
<code>\everypar</code>	17 , 20 , 21 , 184		
exp commands:			
<code>\exp_after:wN</code>	732 , 735		
<code>\exp_not:n</code>	618 , 644		
<code>\expandafter</code>	184 , 264 , 280		
<code>\ExplSyntaxOn</code>	7		
F			
<code>\fi</code>	180 , 181 , 1416		
fi commands:			
<code>\fi:</code>	1377		
<code>\finalhyphendemerits</code>	495		
<code>flushleft (env.)</code>	138		
<code>flushright (env.)</code>	138		
<code>\frenchspacing</code>	160 , 166		
G			
<code>\global</code>	294 , 295		
group commands:			
<code>\group_begin:</code>	310 , 334		
<code>\group_end:</code>	320 , 347		
H			
hbox commands:			
<code>\hbox_gset:Nn</code>	572 , 744		
<code>\hbox_set:Nn</code>	738 , 764		
<code>\hbox_set_to_wd:Nnn</code>	730		
<code>\hbox_unpack_drop:N</code>	575 , 733 , 746 , 748		
<code>\hfil</code>	751		
hook commands:			
<code>\hook_use:n</code>	1372 , 1379		
Hooks:			
<code>para/begin</code>	35 , 36		
<code>\hskip</code>	325 , 352		
<code>\hss</code>	699 , 700 , 701		
I			
if commands:			
<code>\if_int_compare:w</code>	1375		
<code>\ifhmode</code>	180		
<code>\IfNoValueTF</code>	269		
<code>\ifvmode</code>	1406		
<code>\ignorespaces</code>	5 , 25 , 46 , 326 , 353 , 833		
<code>\indent</code>	839		
instances:			
<code>block displayblock-0</code>	1097		
<code>block displayblock-1</code>	1097		
<code>block displayblock-2</code>	1097		
<code>block displayblock-3</code>	1097		
<code>block displayblock-4</code>	1097		
<code>block displayblock-5</code>	1097		
<code>block displayblock-6</code>	1097		
<code>block list-1</code>	1134		
<code>block list-2</code>	1134		
<code>block list-3</code>	1134		
<code>block list-4</code>	1134		
<code>block list-5</code>	1134		
<code>block list-6</code>	1134		
<code>block quotationblock-1</code>	1127		
<code>block quotationblock-2</code>	1127		
<code>block quotationblock-3</code>	1127		
<code>block quotationblock-4</code>	1127		
<code>block quotationblock-5</code>	1127		
<code>block quotationblock-6</code>	1127		
<code>block quoteblock-1</code>	1120		
<code>block quoteblock-2</code>	1120		
<code>block quoteblock-3</code>	1120		
<code>block quoteblock-4</code>	1120		
<code>block quoteblock-5</code>	1120		
<code>block quoteblock-6</code>	1120		
<code>block verbatimblock-0</code>	1108		
<code>block verbatimblock-1</code>	1108		
<code>block verbatimblock-2</code>	1108		
<code>block verbatimblock-3</code>	1108		
<code>block verbatimblock-4</code>	1108		
<code>block verbatimblock-5</code>	1108		

block verbatimblock-6	1108		
blockenv center	927		
blockenv description	1071		
blockenv displayblock	902		
blockenv displayblockflattened	914		
blockenv enumerate	1058		
blockenv flushleft	941		
blockenv flushright	955		
blockenv itemize	1045		
blockenv list	1084		
blockenv quotation	969		
blockenv quote	981		
blockenv verbatim	1031		
item basic	1169		
item description	1169		
list description	1168		
list enumerate-1	1156		
list enumerate-2	1156		
list enumerate-3	1156		
list enumerate-4	1156		
list itemize-1	1152		
list itemize-2	1152		
list itemize-3	1152		
list itemize-4	1152		
list legacy	1164		
para center	1188		
int commands:			
\int_compare:nNnTF	380, 393, 400, 553, 1248		
\int_gdecr:N	448		
\int_gincr:N	404		
\int_gset:Nn	652, 662		
\int_if_exist:NTF	439		
\int_incr:N	382, 387, 396, 552		
\int_new:N	441		
\int_set:Nn	799		
\int_set_eq:NN	802		
\int_to_roman:n	405		
\int_use:N	414, 416, 428, 432, 1267		
\int_zero:N	547		
\c_zero_int	794		
\interlinepenalty	177, 180		
item (objecttype)	26		
\item	11, 16, 25, 26, 29-35, 37, 40, 56, 58, 220, 813, 839, 1465		
item basic (instance)	1169		
item description (instance)	1169		
item std (template)	90, 684		
\itemindent	18, 58, 85, 239, 631, 747, 790		
itemize (env.)	196		
\itemsep	6, 56, 83, 511, 628, 677, 845		
\itshape	324, 351		
		J	
		\justifying	1235, 1237
		K	
		\kern	790
		kernel internal commands:	
		_kernel_displayblock_begin:	39, 568, 601, 601, 854, 863, 876, 890
		_kernel_displayblock_beginpar_hmode:w	531, 601, 604, 850, 859, 872, 886
		_kernel_displayblock_beginpar_vmode:	527, 601, 607, 852, 861, 874, 888
		_kernel_displayblock_doendpe:	50, 10, 15, 25, 1273, 1273
		_kernel_displayblock_end:	39, 458, 478, 478, 855, 864, 877, 891
		_kernel_endpe_vmode:	1386, 1403, 1405, 1405
		_kernel_list_item_begin:	825, 843, 847, 847, 892
		_kernel_list_item_end:	842, 847, 848, 893
		_kernel_list_label_after:	793, 809, 809, 1418, 1418
		_kernel_list_label_begin:	766, 778, 778, 1462, 1462
		_kernel_list_label_end:	775, 778, 779, 1476, 1476
		keys commands:	
		\keys_define:nm	33, 595, 674, 684
		\KeyValue	54, 55, 93, 1121, 1128, 1139, 1140
		L	
		\labelenumi	1157
		\labelenumii	1159
		\labelenumiii	1161
		\labelenumiv	1163
		\labelindent	58
		\labelitemi	1152
		\labelitemii	1153
		\labelitemiii	1154
		\labelitemiv	1155
		\labelsep	7, 58, 87, 325, 352, 633, 747, 749
		\labelwidth	7, 58, 86, 256, 632, 728, 730, 741, 747
		\language	173
		\lastbox	20
		\LBody	1481, 1483
		\leavevmode	177
		\leftmargin	6, 58, 59, 216, 217, 255, 515, 565, 566, 574, 576, 1001, 1014, 1143
		\leftskip	17, 491, 544

legacy commands:	
\legacy_if:nTF	244, 449, 454, 461, 462, 522, 533, 539, 550, 569, 578, 586, 648, 787, 796, 807, 823, 838, 1241, 1243, 1276, 1295, 1303, 1305, 1429, 1485, 1494, 1496
\legacy_if_gset_false:n	452, 459, 786, 789, 798, 826, 1245
\legacy_if_gset_true:n	27, 472, 670, 832
\legacy_if_set_false:n	241, 411, 540, 571, 785
\legacy_if_set_true:n	535, 536
\legacylistsetupcode	11, 19, 236, 1091
\legacyverbatimsetup	11, 171, 1042
\let	182, 210, 242, 854, 855
\linewidth	565, 567, 724
list (env.)	224
list (objecttype)	26
\list	58, 253
list description (instance)	1168
list enumerate-1 (instance)	1156
list enumerate-2 (instance)	1156
list enumerate-3 (instance)	1156
list enumerate-4 (instance)	1156
list itemize-1 (instance)	1152
list itemize-2 (instance)	1152
list itemize-3 (instance)	1152
list itemize-4 (instance)	1152
list legacy (instance)	1164
list std (template)	76, 621
\list<romannumeral>	45, 47
\listparindent	6, 61, 237, 516, 564, 599, 756, 1145
\LItag	1453, 1461
\ltxlabblockdate	4, 1513
\ltxlabblockversion	4, 1513
M	
\makelabel	7, 19, 47, 242, 257, 771
\MakeLinkTarget	308, 332, 715, 716
mode commands:	
\mode_if_horizontal:TF	455, 840, 1369, 1374
\mode_if_inner:TF	1370
\mode_if_vertical:TF	524
\mode_leave_vertical:	307, 331, 451, 521, 522
msg commands:	
\msg_error:nnnn	1382
N	
\newcommand	189
\newcounter	443
\NewDocumentEnvironment	132, 135
\newline	752
\NewTemplateType	26, 27, 28, 29, 30
\newtheorem	11, 20, 262
\nobreak	751
\nobreakspace	193
\noexpand	280
\noitemerr	25
\normalfont	1175
NOT commands:	
\NOT_IMPLEMENTED	702
\null	177
O	
\obeylines	183
object types:	
block	26
blockenv	26
item	26
list	26
para	26
off (plug)	26, 27, 482
on (plug)	26, 27, 482
P	
\par	10, 12, 13, 26–29, 32, 38–40, 50, 53, 11, 18, 175, 456, 531, 839, 841, 1271, 1390
par commands:	
\par_end:	13
par internal commands:	
\l_par_fixed_word_spaces_bool	494
\l_par_start_skip	490
para (objecttype)	26
para center (instance)	1188
para commands:	
\para_end:	12, 28, 54, 555, 559, 1367, 1367, 1390, 1391, 1392
\g_para_indent_box	790
\para_omit_indent:	791
para std (template)	64, 487
para/begin (hook)	35, 36
para/begin	12, 1292
\PARALABEL	323, 350, 1297, 1366
\parfillskip	493, 546
\parindent	6, 66, 489, 564, 756
\parsep	6, 53, 508, 563, 629, 678, 754, 1002, 1138
\parskip	29, 468, 543, 562, 563, 582, 587, 591
\partopsep	6, 52, 507, 526, 598, 1137
\pdfakespace	18, 193
\penalty	177, 180, 794
Plugs:	
off	26, 27, 482
on	26, 27, 482

prg commands:		\space	4, 1285, 1513
\prg_do_nothing:	25, 778, 779, 780, 803, 809, 847, 848, 854, 855, 860, 862	\spacefactor	32
\ProvidesFile	1512	str commands:	
\ProvidesPackage	3	\str_if_eq:nnTF	266
		\string	819
		\strut	8, 769
Q			
quotation (env.)	149	T	
quote (env.)	149	tag commands:	
R			
\raggedleft	1233	\tag_if_active:TF	190, 408, 1178, 1238, 1442
\raggedright	1234	\tag_mc_begin:n	312, 316, 336, 340, 344, 1266, 1326, 1347, 1469
\refstepcounter	21	\tag_mc_end:	314, 318, 338, 342, 346, 1262, 1268, 1356
\relax	220, 699, 701	\tag_struct_begin:n	309, 315, 333, 339, 1251, 1310, 1320, 1332, 1341
\RemoveFromHook	1292, 1349	\tag_struct_end:	319, 321, 343, 348, 1270, 1287, 1358, 1362, 1412
\renewcommand	193	tag internal commands:	
\RenewDocumentCommand	262, 814	\l__tag_block_flattened_level_ int	23, 380, 382, 387, 439, 1248
\RenewDocumentEnvironment	139, 142, 145, 150, 153, 158, 164, 197, 200, 203, 208, 225, 252	__tag_check_para_begin_show:nn	1325, 1346
\RequirePackage	6, 1515	__tag_check_para_end_show:nn	1357
\rightmargin	6, 60, 238, 514, 565, 1001, 1014, 1144	__tag_gincr_para_begin_int:	1318, 1339
\rightskip	492, 501, 545	__tag_gincr_para_end_int:	1263, 1354
\rlap	1267	__tag_gincr_para_main_begin_ int:	1250, 1309, 1331
		__tag_gincr_para_main_end_int:	1286, 1361, 1411, 1431, 1487, 1498
S			
scan commands:		\l__tag_L_attr_class_tl	245, 247, 248, 899, 900, 1440, 1441, 1458
\scan_stop:	1368	\l__tag_L_tag_tl	896, 897, 1437, 1438, 1457
\setbox	20	\g__tag_mode_lua_bool	191
\setcounter	444, 1096	\l__tag_para_attr_class_tl	497, 1323, 1344
\setlength	1001, 1002, 1014	\l__tag_para_bool	1274, 1294, 1352, 1407, 1419
\SetTemplateKeys	30, 378, 500, 519, 618, 642, 644, 709	\g__tag_para_end_int	1267
skip commands:		\l__tag_para_flattened_bool	372, 385, 1280, 1282, 1307, 1329, 1359, 1409
\skip_add:Nn	526, 543	\l__tag_para_main_attr_class_tl	1254, 1313, 1335
\skip_eval:n	587	__tag_para_main_store_struct:	1256, 1315, 1337
\skip_horizontal:n	574, 576, 747, 749	\l__tag_para_main_tag_tl	185, 1253, 1285, 1312, 1334
\skip_new:N	810, 811, 812	\l__tag_para_show_bool	1265
\skip_set:Nn	105, 501, 523	\l__tag_para_tag_tl	186, 1322, 1343
\skip_set_eq:NN	541, 545, 546, 562, 563, 754	\tagmcbegin	1473
\skip_vertical:n	467, 468, 581, 582	\tagmccend	1477
\skip_zero:N	544		
\l_tmpa_skip	464, 465, 467, 468		
socket commands:			
\socket_assign_plug:nn	486, 856, 865, 878, 894		
\socket_new:nn	481		
\socket_new_plug:nnn	482, 484		
\socket_use:n	474		
Sockets:			
tagsupport/block-endpe	481		

<code>\tagpdfparaOff</code>	306, 330, 1271	<code>\@listii</code>	5
<code>\tagpdfparaOn</code>	322, 349, 1271	<code>\@listvi</code>	5
<code>\tagpdfsetup</code>	1180, 1444	<code>\@makeother</code>	182
<code>\tagstructbegin</code>	1425, 1455, 1461, 1472, 1481	<code>\@mklab</code>	242
<code>\tagstructend</code>	1433, 1435, 1480, 1488, 1491, 1499, 1502, 1504	<code>\@namedef</code>	294, 295
<code>tagsupport/block-endpe (socket)</code>	481	<code>\@newctr</code>	277
<code>\tagtool</code>	186	<code>\@nmbrlist</code>	31
templates:		<code>\@nmbrlisttrue</code>	658
<code>block display</code>	48, 503	<code>\@nocounterr</code>	288
<code>blockenv display</code>	31, 358	<code>\@noitemerr</code>	32, 56, 454, 539, 554, 807
<code>item std</code>	90, 684	<code>\@noligs</code>	183
<code>list std</code>	76, 621	<code>\@normalcr</code>	6, 73, 1229
<code>para std</code>	64, 487	<code>\@opargbegintheorem</code>	21, 304
\TeX and $\LaTeX 2_{\epsilon}$ commands:		<code>\@outerparskip</code>	468, 562, 582, 587
<code>\@@par</code>	177, 180	<code>\@restorepar</code>	13
<code>\@beginparpenalty</code>	6, 512, 589	<code>\@rightskip</code>	501, 545
<code>\@begintheorem</code>	11, 21, 304	<code>\@setpar</code>	548
<code>\@beginthorem</code>	21	<code>\@setupverbinvisiblespace</code>	11, 160, 189
<code>\@centercr</code>	210, 1196, 1207, 1218	<code>\@setupverbvisiblespace</code>	166
<code>\@clubpenalty</code>	14, 802	<code>\@sxverbatim</code>	167
<code>\@currentenv</code>	457, 1397	<code>\@tempswafalse</code>	174
<code>\@currentvline</code>	1398	<code>\@tempswatru</code>	179
<code>\@definecounter</code>	268	<code>\@thm</code>	11, 21, 294, 298
<code>\@doendpe</code>	11, 12, 10	<code>\@thmcounter</code>	273, 282
<code>\@eha</code>	1396	<code>\@thmcountersep</code>	281
<code>\@ehc</code>	820	<code>\@toodeep</code>	395, 402
<code>\@endparpenalty</code>	6, 470, 513	<code>\@topsep</code>	37, 57
<code>\@endpefalse</code>	16, 22, 485, 1414	<code>\@topsepadd</code>	37, 57
<code>\@endpetrue</code>	10, 483	<code>\@totalleftmargin</code>	17, 566, 567
<code>\@endtheorem</code>	295, 355	<code>\@vobeyspaces</code>	160, 166
<code>\@enumdepth</code>	1067	<code>\@xobeysp</code>	193
<code>\@execute@begin@hook</code>	1399	<code>\@xthm</code>	302
<code>\@flushglue</code>	6, 70, 546, 1192, 1193, 1204, 1214, 1227	<code>\@xverbatim</code>	161
<code>\@ifdefinable</code>	264	<code>\@ythm</code>	302
<code>\@ifnextchar</code>	302	<code>_g_block_nesting_depth_int</code>	59
<code>\@ifundefined</code>	287, 1395	<code>_c_maxblocklevels</code>	12, 401, 443
<code>\@ignorefalse</code>	1401	<code>\everypar</code>	37, 38
<code>\@inmatherr</code>	457, 816	<code>\hyper@nopatch@thm</code>	303
<code>\@itemdepth</code>	1051	<code>\if@endpe</code>	1406
<code>\@itemlabel</code>	11, 30, 32, 229, 246, 409, 610, 668, 715, 1467, 1479	<code>\if@tempswa</code>	176
<code>\@itempenalty</code>	7, 630, 844	<code>\iitem</code>	58
<code>\@kernel@after@para@after</code>	1380	<code>\item</code>	35, 59
<code>\@kernel@after@para@end</code>	1373	<code>\l@nohyphenation</code>	173
<code>\@kernel@refstepcounter</code>	300, 712	<code>\labelwidth</code>	33, 35, 58
<code>\@labels</code>	35	<code>\makelabel</code>	35, 60
<code>\@latex@error</code>	819, 1396	<code>\newline</code>	33
<code>\@list...</code>	5	<code>\on@line</code>	446, 784, 1242, 1264, 1281, 1285, 1304, 1319, 1340, 1355, 1398, 1428, 1432, 1489, 1500
<code>\@listctr</code>	30, 32, 240, 410, 610, 652, 659, 662, 712, 715, 716	<code>\par</code>	38, 58
<code>\@listdepth</code>	5, 22, 356	<code>\par@deathcycles</code>	547, 552, 553
<code>\@listi</code>	5	<code>\partopsep</code>	57
		<code>\ref</code>	58
		<code>\reserved@a</code>	1396, 1397, 1404

<code>\strut</code>	35	<code>\tl_set:Nn</code>	185, 186, 227, 229, 240, 245, 247, 248, 301, 699, 700, 701, 867, 880, 896, 899, 1438, 1441
<code>\topsep</code>	57	<code>\tl_set_eq:NN</code>	659, 668, 707, 868, 881, 897, 900
<code>\verbatim@font</code>	183	<code>\topsep</code>	6, 51, 506, 523, 597, 679, 682, 1136
<code>\z@</code>	20	<code>trivlist (env.)</code>	<u>251</u>
<code>\z@skip</code>	1194, 1203, 1205, 1215, 1216, 1225, 1226	<code>\trivlist</code>	58
tex commands:		<code>\typeout</code>	127
<code>\tex_hskip:D</code>	1376	U	
<code>\tex_lastnodetype:D</code>	1375	<code>\unpenalty</code>	184
<code>\tex_lastskip:D</code>	105	use commands:	
<code>\tex_par:D</code>	1378, 1387	<code>\use:N</code>	405, 408
<code>\tex_parshape:D</code>	567	<code>\use:n</code>	257, 772
<code>\tex_unskip:D</code>	107, 1371	<code>\use_i:nn</code>	732
<code>\textbf</code>	21, 521	<code>\use_ii:nn</code>	735
<code>\the</code>	184	<code>\use_none:n</code>	110, 111
<code>\tiny</code>	1267	<code>\use_none:nn</code>	612, 615, 640
tl commands:		<code>\usecounter</code>	31
<code>\c_novalue_tl</code>	33, 34, 707	<code>\UseHook</code>	1394
<code>\tl_clear:N</code>	409, 410	<code>\UseInstance</code>	37, 133, 136, 140, 143, 146, 151, 154, 159, 165, 198, 201, 204, 211, 232, 305, 329, 415, 422, 429, 1232, 1233, 1234, 1235
<code>\tl_gset:Nn</code>	271, 278, 290	<code>\UseName</code>	57, 58, 84, 291, 1141, 1142
<code>\tl_if_blank:nTF</code>	520, 712	V	
<code>\tl_if_empty:NTF</code>	246, 391, 419, 424, 427, 431, 646, 666, 866, 879, 895, 898	<code>verbatim (env.)</code>	<u>157</u>
<code>\tl_if_empty:nTF</code>	378, 500, 519, 614, 639, 709, 1467, 1479	<code>verbatim* (env.)</code>	<u>157</u>
<code>\tl_if_eq:NnTF</code>	477	<code>verse (env.)</code>	<u>207</u>
<code>\tl_if_novalue:nTF</code>	108, 710, 829		
<code>\tl_new:N</code>	8, 9, 298, 610, 611, 883, 1437, 1440		