

Prototype reimplementation of L^AT_EX 2_ε's block environments using templates

L^AT_EX Project*
v0.8w 2024-11-25

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 templates	16
6.4.1	Implementation of <code>blockenv</code> templates ...	16
6.4.2	Implementation of <code>para</code> templates ...	21
6.4.3	Implementation of <code>block</code> templates ...	21
6.4.4	Implementation of <code>list</code> templates ...	24
6.4.5	Implementation of <code>\item</code> template(s)	27
6.5	Tagging support commands	33
6.5.1	List tags	38
6.6	Tagging recipes	41
7	Implementation of document-level block environments	43
7.1	<code>Displayblock</code> environments	43
7.2	The <code>center</code> , <code>flushleft</code> , and <code>flushright</code> environments	43
7.3	<code>Display quote</code> environments	44
7.4	<code>Verbatim</code> environments	44
7.4.1	Helper commands for <code>verbatim</code>	44
7.5	Standard list environments	45
7.6	<code>verse</code> environment	45
7.7	Theorem-like environments	47
8	Instance declarations for environments	49
8.1	<code>Blockenv</code> instances	49
8.1.1	Basic instances	50
8.1.2	Center, <code>flushleft</code> , and <code>flushright</code> instances	51
8.1.3	<code>Blockquote</code> instances	52
8.1.4	The theorem instance	52
8.1.5	The <code>verbatim</code> instance	53
8.1.6	Standard list instances	53
8.2	<code>Block</code> instances	54
8.2.1	<code>Displayblock</code> instances	54
8.2.2	<code>Verbatim</code> instances	55
8.2.3	<code>Quote/quotationblock</code> instances	55
8.2.4	<code>Block</code> instances for the theorems	56
8.2.5	<code>Block</code> instances for the standard lists	56
8.3	List instances for the standard lists	56
8.4	<code>Item</code> instances	57
8.5	<code>Para</code> instances	58
A	Documentation from first prototype implementations	59
A.1	Open questions	59
A.2	Code cleanup	59
A.3	Tasks	60
B	Plan of attack of first prototype	61

1 Introduction

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

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

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

For example, a `quote` environment would make use of a (display) *block* and some *para* handling while a 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 name).

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

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

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

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

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

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

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

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

2.2.2 The block template ‘display’

Attributes:

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

`beginsep` (*skip*) Default: `\topsep`

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

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

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

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

`item-skip` (*skip*) The space in front of an item if the block is a list; if not the setting has no effect Default: `\itemsep`

`beginpenalty` (*integer*) Default: `\@beginparpenalty`

`endpenalty` (*integer*) Default: `\@endparpenalty`

`leftmargin` (*length*) Default: `\leftmargin`

`rightmargin` (*length*) Default: `\rightmargin`

`parindent` (*length*) Default: `0pt`

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

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

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

2.2.3 The para template ‘std’

Attributes:

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

2.2.4 The list template ‘std’

Attributes:

counter (<i>tokenlist</i>)	Counter name to be used in a numbered list or empty, if the list is unnumbered
item-label (<i>tokenlist</i>)	Label “string” for a fixed label or as generated from the current counter value
start (<i>integer</i>)	Start value for the counter if the list is numbered, otherwise irrelevant Default: <code>1</code>
resume (<i>boolean</i>)	Should a numbered list be resumed from the last instance? Default: <code>false</code>
item-instance (<i>instance</i>)	Instance of type <code>item</code> to be used to format the label string Default: <code>basic</code>
item-skip (<i>skip</i>)	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 (<i>length</i>)	Horizontal displacement of the item. Default: <code>0pt</code>
item-penalty (<i>integer</i>)	Penalty for breaking before an item (except the first) Default: <code>\@itempenalty</code>
label-width (<i>length</i>)	Width reserved for the formatted item label Default: <code>\labelwidth</code>
label-sep (<i>length</i>)	Horizontal separation between label and following text Default: <code>\labelsep</code>
legacy-support (<i>boolean</i>)	Is formatting the label via <code>\makelabel</code> supported? Default: <code>false</code>

2.2.5 The item template ‘std’

Attributes:

<code>counter-label</code> (<i>function1</i>) <i>unused</i>	Default: <code>\arabic{#1}</code>
<code>counter-ref</code> (<i>function1</i>) <i>unused</i>	Default: value from <code>counter-label</code>
<code>label-ref</code> (<i>function1</i>) <i>unused</i>	Default: <code>#1</code>
<code>label-autoref</code> (<i>function1</i>) <i>unused</i>	Default: <code>item #1</code>
<code>label-format</code> (<i>function1</i>) Formatting of the label, questionable the way it is used	Default: <code>#1</code>
<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>
      <Lbl> label </Lbl>
      <LBody>
        The text of the first item involving <text-unit> as necessary ...
      </LBody>
    </LI>
    <LI>
      The second item ...
    </LI>
    ... further items ...
  </itemize>
</text-unit>
```

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

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

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

```
with a paragraph break between them
```

```
\end{center}
followed by some more text.
```

will be tagged as follows:

```
<text-unit>
  <text>
    This is a paragraph with some
  </text>
  <text /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`

`\@setupverbinvisiblepace` A counterpart definition to the kernel command `\@setupverbinvisiblepace`, 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 redefined 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                 [\l@labblockdate\space v\l@labblockversion\space
5                 blockenv implementation]
6
7 \RequirePackage{latex-lab-kernel-changes}
8
9 \ExplSyntaxOn
10
11 \GeneralKernelChanges
12
13 \end{code}
```

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

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 the 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.

`_kernel_displayblock_doendpe:`

```

8 \def\@doendpe{\@endptrue
9 \def\par
10 {
11 \restorepar
12 \clubpenalty\@clubpenalty

```

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

```

13 \_kernel_displayblock_doendpe:

```

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

```

14 \endpefalse
15 \everypar{}
16 \par
17 }
18 \everypar{{\setbox\z@\lastbox}
19 \everypar{}
20 \endpefalse
21 }
22 }

```

By default we don't do any tagging:

```

23 \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, `block` (*objecttype*) and the template code is covered later.

```

para (objecttype)
list (objecttype) 24 \NewTemplateType{blockenv}{1}
item (objecttype) 25 \NewTemplateType{block}{1}
26 \NewTemplateType{para}{1}
27 \NewTemplateType{list}{1}
28 \NewTemplateType{item}{1}

```

`blockenv display` (*templ.*)

```

29 \DeclareTemplateInterface{blockenv}{display}{1}
30 {
31 env-name : tokenlist ,
32 tag-name : tokenlist ,
33 tag-class : tokenlist ,
34 tagging-recipe : tokenlist = standard,
35 level-increase : boolean = true ,
36 setup-code : tokenlist ,
37 block-instance : tokenlist = displayblock ,
38 para-instance : tokenlist ,
39 inner-level-counter : tokenlist,

```

```

40 max-inner-levels : tokenlist = 4,
41 inner-instance-type : tokenlist = list ,
42 inner-instance : tokenlist ,
43 para-flattened : boolean = false ,
44 final-code : tokenlist = \ignorespaces ,
45 }

```

block display (*templ.*)

```

46 \DeclareTemplateInterface{block}{display}{1}
47 {
48 heading : tokenlist = , % ??
49 beginsep : skip = \topsep ,
50 begin-par-skip : skip = \partopsep ,
51 par-skip : skip = \parsep ,
52 end-skip : skip = \KeyValue{beginsep} , % conflict with name below
53 end-par-skip : skip = \KeyValue{begin-par-skip} ,
54 item-skip : skip = \itemsep ,
55 beginpenalty : integer = \UseName{@beginparpenalty} ,
56 endpenalty : integer = \UseName{@endparpenalty} ,
57 leftmargin : length = \leftmargin ,
58 rightmargin : length = \rightmargin ,
59 parindent : length = Opt ,
60 % maybe add? (or more general for fonts and color)
61 % font : tokenlist
62 }

```

para std (*templ.*)

```

63 \DeclareTemplateInterface{para}{std}{1}
64 {
65 para-class : tokenlist = justify ,
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 }

```

list std (*templ.*)

```

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

```

```

88 }

item std (templ.)

89 \DeclareTemplateInterface{item}{std}{1}
90 {
91   counter-label : function{1} = \arabic{#1} ,
92   counter-ref   : function{1} = \KeyValue{counter-label} ,
93   label-ref     : function{1} = #1 ,
94   label-autoref : function{1} = item-#1 ,
95   label-format  : function{1} = #1 ,
96   label-strut   : boolean = false ,
97   label-align   : choice {left,center,right,parleft} = right ,
98   label-boxed   : boolean = true ,
99   next-line     : boolean = false ,
100  text-font     : tokenlist ,
101  compatibility : boolean = true ,
102 }

```

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:`

```

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

```

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

```

106 \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:.)

```

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

```

6.3.1 Debugging

`\g_block_debug_bool`

```

108 \bool_new:N \g_block_debug_bool

```

(End of definition for \g_block_debug_bool.)

`_block_debug:n`

`_block_debug_typeout:n`

```

109 \cs_new_eq:NN \_block_debug:n \use_none:n
110 \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:`

```

111 \cs_new_protected:Npn \block_debug_on:
112 {
113   \bool_gset_true:N \g_block_debug_bool
114   \_block_debug_gset:
115 }

```

```

116 \cs_new_protected:Npn \block_debug_off:
117 {
118   \bool_gset_false:N \g_block_debug_bool
119   \__block_debug_gset:
120 }
121 \cs_new_protected:Npn \__block_debug_gset:
122 {
123   \cs_gset_protected:Npx \__block_debug:n ##1
124   { \bool_if:NT \g_block_debug_bool {##1} }
125   \cs_gset_protected:Npx \__block_debug_typeof:n ##1
126   { \bool_if:NT \g_block_debug_bool { \typeof{<=>~ ##1} } }
127 }

```

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

`\DebugBlocksOn`
`\DebugBlocksOff`

```

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

```

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

6.4 Implementation of templates

6.4.1 Implementation of blockenv templates ...

`\g_block_nesting_depth_int`

L^AT_EX 2_ε already has a counter to record the nesting depth of blocks, but we want our own name because it isn't really tied to “lists” any more. However, `\@listdepth` is really part of the legacy interface (for example `minipage` alters it to point to a different counter) so that we are stuck with using at least indirectly for now and the following line makes this look like an L3 integer variable but internally expands to `\@listdepth`:

```

131 \cs_new:Npn \g_block_nesting_depth_int { \@listdepth } % a fake int
132 % for now

```

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

`blockenv display (templ.)`

```

133 \DeclareTemplateCode{blockenv}{display}{1}
134 {
135   env-name      = \l_block_env_name_tl ,
136   tag-name      = \l_block_tag_name_tl ,
137   tag-class     = \l_block_tag_class_tl ,
138   tagging-recipe = \l_block_tagging_recipe_tl ,
139   level-increase = \l_block_level_incr_bool ,
140   setup-code    = \l_block_setup_code_tl ,
141   block-instance = \l_block_block_instance_tl ,
142   para-instance = \l_block_para_instance_tl ,
143   para-flattened = \l_block_tag_para_flattened_bool ,
144   inner-level-counter = \l_block_inner_level_counter_tl ,
145   max-inner-levels = \l_block_max_inner_levels_tl ,
146   inner-instance-type = \l_block_inner_instance_type_tl ,
147   inner-instance = \l_block_inner_instance_tl ,

```



```

148 final-code      = \l__block_final_code_tl ,
149 }
150 {
151   \__block_debug_typeout:n{\l__block_env_name_tl -env-start}
152 %
153   \tl_if_empty:nF {#1} { \SetTemplateKeys{blockenv}{display}{#1} }
154 %

```

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>`.

```

155   \int_compare:nNnTF \l__tag_block_flattened_level_int > 0
156     {
157       \int_incr:N \l__tag_block_flattened_level_int
158     }
159     {
160       \bool_if:NT \l__tag_para_flattened_bool
161         {
162           \int_incr:N \l__tag_block_flattened_level_int
163         }
164     }
165 %
166 \tl_if_empty:NF \l__block_inner_level_counter_tl
167 {
168   \int_compare:nNnTF \l__block_inner_level_counter_tl >
169     { \l__block_max_inner_levels_tl - 1 }
170     { \@toodeep }
171     { \int_incr:N \l__block_inner_level_counter_tl } % not clean "o"
172 }

```

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.

```

173 \bool_if:NT \l__block_level_incr_bool
174 {
175   \int_compare:nNnTF \g_block_nesting_depth_int >
176     { \c@maxblocklevels - 1 }
177     { \@toodeep }
178     {
179       \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.

```

180       \use:c { @list \int_to_roman:n
181               { \g_block_nesting_depth_int } }
182     }
183 }

```

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

```

184 \tag_if_active:T
185   { \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.

```

186 \tl_clear:N \@itemlabel
187 \tl_clear:N \@listctr
188 \legacy_if_set_false:n { @nmbrrlist }

```

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

```

189 \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).

```

190 \__block_debug_typeout:n{use~ instance:~
191     \l__block_block_instance_tl - \int_use:N \g_block_nesting_depth_int }
192 \UseInstance{block}
193     { \l__block_block_instance_tl - \int_use:N
194       \g_block_nesting_depth_int }
195     {#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).

```

196 \tl_if_empty:NF \l__block_para_instance_tl
197 {
198     \__block_debug_typeout:n{ use~ para~ instance:~
199         \l__block_para_instance_tl }

```

For now we don't offer to alter instance parameters here so we pass an empty argument.

```

200     \UseInstance{para}{ \l__block_para_instance_tl } {}
201 }

```

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.

```

202 \tl_if_empty:NF \l__block_inner_instance_tl
203 {
204     \__block_debug_typeout:n{use~ instance:~ \l__block_inner_instance_tl
205         \tl_if_empty:NF \l__block_inner_level_counter_tl
206             { - \int_use:N \l__block_inner_level_counter_tl }}
207     \UseInstance{ \l__block_inner_instance_type_tl }
208         { \l__block_inner_instance_tl
209           \tl_if_empty:NF \l__block_inner_level_counter_tl
210             % not clean use "o"?
211             { - \int_use:N \l__block_inner_level_counter_tl }
212         }
213     {#1}
214 }

```

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.

```

215 \l__block_final_code_tl
216 }

```

`\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

```
217 \int_if_exist:NF \l__tag_block_flattened_level_int
218 {
219   \int_new:N \l__tag_block_flattened_level_int
220 }
```

(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.

```
221 \newcounter{maxblocklevels}
222 \setcounter{maxblocklevels}{6}
```

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

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

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

name is bad

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

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

```
225 \bool_if:NT \l__block_level_incr_bool
226   { \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.

```
227 \legacy_if:nT { @inlabel }
228   {
229     \mode_leave_vertical:
230     \legacy_if_gset_false:n { @inlabel }
231   }
```

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.

```
232 \__block_if_list:T { \legacy_if:nT { @newlist } { \noitemerr } }
233 \mode_if_horizontal:TF
234   { \__block_skip_remove_last: \__block_skip_remove_last: \par }
235   { \inmatherr{\end{\@currenenv}} }
```

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

```
236 \__kernel_displayblock_end:
```

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

```
237 \__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 L^AT_EX_{2 ϵ} list environment have been doing.

some redesign/extensions here?

```

238 %   \_block_debug_typeout:n{@noparlist =
239 %                                   \legacy_if:nTF { @noparlist }{true}{false}}
240 \legacy_if:nF { @noparlist }
241   {
242     \_block_skip_set_to_last:N \l_tmpa_skip
243     \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
244       {
245         \skip_vertical:n { - \l_tmpa_skip }
246         \skip_vertical:n { \l_tmpa_skip + \parskip - \outerparskip }
247       }
248     \addpenalty \@endparpenalty
249     \advspace \l_block_topsepadd_skip

```

L^AT_EX_{2 ϵ} triggered the paragraph handling after a list at this point here, i.e., only if the list didn't start a paragraph. One can make a case for that, but it can be somewhat surprising to the user and there is a good argument that even such a list could be followed 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

```

250 %   \legacy_if_gset_true:n { @endpe }
251   }

```

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.

decide

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.

```

252 \socket_use:n {tagsupport/block-endpe}
253 }

```

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

`_block_if_list:T`
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.

```

254 \cs_new:Npn \_block_if_list:T
255   { \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.

```

256 \cs_new:Npn \_kernel_displayblock_end: {
257   \_block_debug_typeout:n{\detokenize{\_kernel_displayblock_end:}}
258 }

```

(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`).

```

259 \socket_new:mn      {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.

```
260 \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

```
261             { \@endpetrue }
262 \socket_new_plug:nnn{tagsupport/block-endpe}{off}
263             { \@endpefalse }
264 \socket_assign_plug:nm{tagsupport/block-endpe}{on}
```

6.4.2 Implementation of para templates ...

`para std (templ.)`

```
265 \DeclareTemplateCode{para}{std}{1}
266 {
267   indent-width      = \parindent ,
268   start-skip        = \l__par_start_skip ,           % name??
269   left-skip         = \leftskip ,
270   right-skip        = \rightskip ,
271   end-skip          = \parfillskip ,
272   fixed-word-spaces = \l__par_fixed_word_spaces_bool , % name??
273   final-hyphen-demerits = \finalhyphendemerits ,
274   cr-command        = \\ ,
275   para-class        = \l__tag_para_attr_class_tl ,
276 }
277 {
278   \tl_if_empty:nF {#1} { \SetTemplateKeys{para}{std}{#1} }
279   \skip_set:Nn \@rightskip \rightskip
280 }
```

6.4.3 Implementation of block templates ...

`block display (templ.)` In contrast to the L^AT_EX 2_ε implementation we do not directly use `\listparindent` here but a private register of the template. The reason is that block template instances are also used outside of lists.

```
281 \DeclareTemplateCode{block}{display}{1}
282 {
283   heading           = \l__block_heading_tl ,
284   beginsep          = \topsep ,
285   begin-par-skip    = \partopsep ,
286   par-skip          = \parsep ,
287   end-skip          = \l__block_botsep_skip ,
288   end-par-skip      = \l__block_parbotsep_skip ,
289   item-skip         = \itemsep ,
290   beginpenalty      = \@beginparpenalty ,
291   endpenalty        = \@endparpenalty ,
292   rightmargin       = \rightmargin ,
293   leftmargin        = \leftmargin ,
294   parindent         = \l__block_parindent_dim ,
295 }
296 {
```

generalize heading usage
(or drop?)

```

297 \tl_if_empty:nF {#1} { \SetTemplateKeys{block}{display}{#1} }
298 \tl_if_blank:oF \l__block_heading_tl
299 { \mode_leave_vertical:
300   \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.

```

301 \legacy_if:nT { @noskipsec } { \mode_leave_vertical: }
302 \skip_set:Nn \l__block_topsepadd_skip { \topsep }
303 \mode_if_vertical:TF
304 {
305   \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.

```

306   \__kernel_displayblock_beginpar_vmode:
307 }
308 {

```

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`.

```

309   \__block_skip_remove_last: \__block_skip_remove_last:
310   \__kernel_displayblock_beginpar_hmode:w \par
311 }

```

Now we are back to legacy list implementation ...

```

312 \legacy_if:nTF { @inlabel }
313 {
314   \legacy_if_set_true:n { @noperitem }
315   \legacy_if_set_true:n { @noperlist }
316 }
317 {
318   \legacy_if:nT { @newlist } { \noitemerr }
319   \legacy_if_set_false:n { @noperlist }
320   \skip_set_eq:NN \l__block_effective_top_skip \l__block_topsepadd_skip
321 }
322 \skip_add:Nn \l__block_effective_top_skip { \parskip }

```

Next lines set some paragraph defaults, any of them may get overwritten if there is a `para`-instance specified on the `blockenv` instance.

```

323 \skip_zero:N \leftskip
324 \skip_set_eq:NN \rightskip \@rightskip
325 \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).

```

326 \int_zero:N \par@deathcycles
327 \@setpar
328 {
329   \legacy_if:nTF { @newlist }
330   {
331     \int_incr:N \par@deathcycles
332     \int_compare:nNnTF \par@deathcycles > { 1000 }
333     { \@noitemerr
334       { \para_end: }
335     }
336   }
337   {
338     { \para_end: }
339   }
340 }
341 \skip_set_eq:NN \@outerparskip \parskip
342 \skip_set_eq:NN \parskip \parsep
343 \dim_set_eq:NN \parindent \l_block_parindent_dim
344 \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
345 \dim_add:Nn \@totalleftmargin { \leftmargin }
346 \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.

```

347 \__kernel_displayblock_begin:

```

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

```

348 \legacy_if:nTF { @noparitem }
349 {
350   \legacy_if_set_false:n { @noparitem }
351   \hbox_gset:Nn \g_block_labels_box
352   {
353     \skip_horizontal:n { - \leftmargin }
354     \hbox_unpack_drop:N \g_block_labels_box
355     \skip_horizontal:n { \leftmargin }
356   }
357   \legacy_if:nF { @minipage } % Why this chunk of code?
358   {
359     \__block_skip_set_to_last:N \l_block_tmpa_skip
360     \skip_vertical:n { - \l_block_tmpa_skip }
361     \skip_vertical:n { \l_block_tmpa_skip +
362       \outerparskip - \parskip }
363   }
364 }
365 {
366   \legacy_if:nTF { @nobreak }
367   { \addvspace{\skip_eval:n{\@outerparskip-\parskip}} }
368   {
369     \addpenalty \@beginparpenalty

```

document 2e logic used here

```

370         \addvspace \l_block_effective_top_skip
371         \addvspace{-\parskip}
372     }
373 }
374 }

```

Extra keys to support enumitem conventions:

```

375 \keys_define:nn { template/block/display }
376 {
377   ,topsep      .skip_set:N = \topsep
378   ,partopsep   .skip_set:N = \partopsep
379   ,listparindent .skip_set:N = \listparindent
380 }

```

```

\__kernel_displayblock_begin:
\__kernel_displayblock_beginpar_hmode:w
\__kernel_displayblock_beginpar_vmode:

```

The internal kernel hooks for tagging.

```

381 \cs_new:Npn \__kernel_displayblock_begin: {
382   \__block_debug_typeout:n
383   {\detokenize{\__kernel_displayblock_begin:}}
384 }
385 \cs_new:Npn \__kernel_displayblock_beginpar_hmode:w {
386   \__block_debug_typeout:n
387   {\detokenize{\__kernel_displayblock_beginpar_hmode:w}}
388 }
389 \cs_new:Npn \__kernel_displayblock_beginpar_vmode: {
390   \__block_debug_typeout:n
391   {\detokenize{\__kernel_displayblock_beginpar_vmode:}}
392 }

```

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

6.4.4 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.

```

393 \tl_new:N \@itemlabel      % should have a top-level definition
394 \tl_new:N \@listctr       % should have a top-level definition

```

(End of definition for \@itemlabel and \@listctr. These functions are documented on page 12.)

```

\__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.

```

395 \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.

```

396 %\cs_new:Npn \__block_save_user_keys:n #1 {
397 % \tl_if_empty:nTF {#1}
398 %   { \cs_set_eq:NN \__block_evaluate_saved_user_keys:nn \use_none:nn }
399 %   {
400 %     \cs_set:Npe \__block_evaluate_saved_user_keys:nn ##1##2
401 %       { \SetTemplateKeys{##1}{##2}{ \exp_not:n{#1} } }
402 %   }
403 %}

```

(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.

```

404 \DeclareTemplateCode{list}{std}{1}
405 {
406   counter           = \l__block_counter_tl,
407   item-label        = \l__block_item_label_tl,
408   start             = \l__block_counter_start_int ,
409   resume            = \l__block_resume_bool ,
410   item-instance     = \__block_item_instance:n ,
411   item-skip         = \itemsep ,
412 % item-par-skip    = \parsep ,
413   item-penalty      = \@itempenalty ,
414   item-indent       = \itemindent ,
415   label-width       = \labelwidth ,
416   label-sep         = \labelsep ,
417   legacy-support    = \l__block_legacy_support_bool , % FMI questionable
418 }
419 {
420   \__block_debug_typeout:n{template:list:std}
421 %

```

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`.

```

422 \tl_if_empty:nTF {#1}
423   { \cs_set_eq:NN \__block_evaluate_saved_user_keys:nn \use_none:nn }
424   {
425     \SetTemplateKeys{list}{std}{#1}
426     \cs_set:Npe \__block_evaluate_saved_user_keys:nn ##1##2
427       { \SetTemplateKeys{##1}{##2}{ \exp_not:n{#1} } }
428   }

```

Has this list a counter name defined in the instance?

```

429 \tl_if_empty:NTF \l__block_counter_tl
430   {

```

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

```

431   \legacy_if:nT { @nمبرlist }
432   {

```

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).

```

433     \bool_if:NF \l__block_resume_bool
434     {
435         \int_gset:cn{ c@ \@listctr }
436         { \l__block_counter_start_int - 1 }
437     }
438 }
439 }

```

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.

```

440     {
441         \@nmbulisttrue
442         \tl_set_eq:NN \@listctr \l__block_counter_tl
443         \bool_if:NF \l__block_resume_bool
444         {
445             \int_gset:cn{ c@ \@listctr }
446             { \l__block_counter_start_int - 1 }
447         }
448     }

```

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.

```

449     \tl_if_empty:NF \l__block_item_label_tl
450     {
451         \tl_set_eq:NN \@itemlabel \l__block_item_label_tl
452     }

```

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).

```

453     \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).

```

454     \cs_set_eq:NN \__block_item_everypar: \__block_item_everypar_first:
455     \__block_debug_typeout:n{template:list:std~end}
456 }

```

Extra keys to support enumitem conventions:

```

457 \keys_define:nn { template/list/std }
458 {
459     ,nosep .code:n =
460     \dim_zero:N \itemsep
461     \dim_zero:N \parsep

```

Think about a better implementation at some point.

```

462   \dim_zero:N \topsep
463   \dim_zero:N \l_block_botsep_skip
464   \dim_zero:N \l_block_parbotsep_skip
465   ,midsep      .skip_set:N = \topsep
466 }

```

6.4.5 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.

```

467 \keys_define:nn { template/item/std }
468   { label .tl_set:N = \l_block_label_given_tl }
469 \DeclareTemplateCode{item}{std}{1}
470 {
471   counter-label = \l_block_counter_label:n ,
472   counter-ref   = \l_block_counter_ref:n ,
473   label-ref     = \l_block_label_ref:n ,
474   label-autoref = \l_block_label_autoref:n ,
475   label-format  = \l_block_label_format:n ,
476   label-strut   = \l_block_label_strut_bool ,
477   label-boxed   = \l_block_label_boxed_bool ,
478   next-line     = \l_block_next_line_bool ,
479   text-font     = \l_block_text_font_tl ,
480   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)

```

481   label-align   = {
482     left    = \tl_set:Nn \l_block_item_align_tl { \relax \hss } ,
483     center  = \tl_set:Nn \l_block_item_align_tl { \hss \hss } ,
484     right   = \tl_set:Nn \l_block_item_align_tl { \hss \relax } ,
485     parleft = \NOT_IMPLEMENTED ,
486   } ,
487 }

```

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.

```

488 {
489   \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.

```

490   \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).

```
491 \__block_evaluate_saved_user_keys:nn {item}{std}
492 \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[]`.

```
493 \tl_if_novalue:oTF \l__block_label_given_tl
494 {
```

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

fix

```
495 \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
496 \bool_if:NTF \l__block_item_compatibility_bool % not sure that
497 % conditional
498 % makes sense
499 { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}%
500 \itemlabel } } % TODO ?
501 { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}%
502 \__block_counter_label:n { \@listctr } } }
503 }
504 {
505 \__block_debug_typeout:n{item~ with~ optional}
506 \__block_make_label_box:n { \l__block_label_given_tl } }
507 \bool_if:nT
508 {
509 \l__block_label_boxed_bool
510 % TODO: is \linewidth correct?
511 && \dim_compare_p:n
512 { \box_wd:N \l__block_one_label_box <= \linewidth }
513 }
514 {
515 \dim_compare:nNnT
516 { \box_wd:N \l__block_one_label_box } < \labelwidth
517 {
518 \hbox_set_to_wd:Nnn \l__block_one_label_box { \labelwidth }
519 {
520 \exp_after:wN \use_i:nn \l__block_item_align_tl
```

FMi: $\text{\LaTeX} 2_{\epsilon}$ 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.

```
521 % TODO: customize?
522 % \hbox_unpack_drop:N \l__block_one_label_box
523 \box_use_drop:N \l__block_one_label_box
524 \exp_after:wN \use_ii:nn \l__block_item_align_tl
525 }
526 }
```

Add another box level to the label box:

```
527 \hbox_set:Nn \l__block_one_label_box
528 { \box_use_drop:N \l__block_one_label_box }
```

```

529     }
530     \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \labelwidth
531     { \bool_set_true:N \l__block_long_label_bool }
532     { \bool_set_false:N \l__block_long_label_bool }
533     \hbox_gset:Nn \g__block_labels_box
534     {
535         \hbox_unpack_drop:N \g__block_labels_box
536         \skip_horizontal:n { \itemindent - \labelsep - \labelwidth }
537         \hbox_unpack_drop:N \l__block_one_label_box
538         \skip_horizontal:n { \labelsep }
539         \bool_if:NT \l__block_next_line_bool
540         { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
541         % version of \newline inside an hbox that will be unpacked
542     }
543     % TODO??? FMI what's that?
544     % \skip_set_eq:NN \parsep \l__block_item_parsep_skip

```

The next setting is for compatibility: The list template sets `\listparindent` to zero and otherwise doesn't use it any more. However, in the second argument of a legacy `list` environment the user may have set it explicitly to some other value and whatever value it had was then used for `\parindent` within the list. Now we use its value only if it differs from zero but otherwise use whatever the template instances specify. This gives 99.9% compatibility for legacy documents. 100% for definitions using the `list` environment and a setting inside, but if the user used `\listparindent` within the document, e.g., inside a `verse` environment there there is one case in which the setting is ignored, i.e., when it was set back to zero. That's a rather unlikely scenario, but it is not impossible. However, I couldn't think of an approach that circumvents such boundary cases.

```

545     \dim_compare:nNnF \listparindent = {0pt}
546     { \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.

```

547     \cs_set_eq:NN \__block_item_everypar: \__block_item_everypar_std:
548     }

```

`\l__block_item_align_tl`

```

549 \tl_new:N \l__block_item_align_tl

```

(End of definition for `\l__block_item_align_tl`.)

`\l__block_one_label_box`
`\g__block_labels_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.

```

550 \box_new:N \l__block_one_label_box
551 \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`.

```

552 \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 `\makeLabel` in compatibility mode (used for the list environment).

```
553 \cs_new_protected:Npn \_block_make_label_box:n #1
554   {
555     \hbox_set:Nn \l_block_one_label_box
556     {
```

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

```
557     \_kernel_list_label_begin:
558     \_block_label_format:n
559     {
560       \bool_if:NT \l_block_label_strut_bool { \strut }
561       \bool_if:NTF \l_block_legacy_support_bool
562         \makeLabel
563         \use:n
564         {#1}
565     }
```

And what gets opened also needs closing:

```
566     \_kernel_list_label_end:
567   }
568 }
```

(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:
569 \cs_new_eq:NN \_kernel_list_label_begin: \prg_do_nothing:
570 \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).

```
571 \cs_new_eq:NN \_block_item_everypar: \prg_do_nothing:
572 \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.

```
573 \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.

```
574 \cs_new_protected:Npn \_block_item_everypar_std: {
575   \_block_debug_typeout:n{item~ everypar \on@line }
576   \legacy_if_set_false:n { @minipage }
577   \legacy_if_gset_false:n { @newlist }
```

```

578 \legacy_if:nT { @inlabel }
579 {
580   \legacy_if_gset_false:n { @inlabel }
581   \box_if_empty:NT \g_para_indent_box { \kern - \itemindent }
582   \para_omit_indent:
583   \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:

```

584   \__kernel_list_label_after:
585   \penalty \c_zero_int
586 }
587 \legacy_if:nTF { @nobreak }
588 {
589   \legacy_if_gset_false:n { @nobreak }
590   \int_set:Nn \clubpenalty { 10000 }
591 }
592 {
593   \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.

```

594   \cs_set_eq:NN \__block_item_everypar: \prg_do_nothing:
595 }
596 }

```

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

```

597 \cs_new:Npn \__block_item_everypar_first: {
598   \legacy_if:nT { @newlist } { \@noitemerr }
599 }

```

(End of definition for __block_item_everypar:, __block_item_everypar_std:, and __block_item_everypar_first:.)

```
\__kernel_list_label_after:
```

```
600 \cs_new_eq:NN \__kernel_list_label_after: \prg_do_nothing:
```

(End of definition for __kernel_list_label_after:.)

```
\l_block_tmpa_skip
```

```
601 \skip_new:N \l_block_tmpa_skip
```

(End of definition for \l_block_tmpa_skip.)

```
\l_block_topsepadd_skip
\l_block_effective_top_skip
```

Variables equivalent to 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.

```
602 \skip_new:N \l_block_topsepadd_skip
603 \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`.

```
604 \AddToHook{begindocument/before}{
605   \RenewDocumentCommand{\item}{={label}o }
606   {
607     \@inmatherr \item
```

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

```
608     \cs_if_free:NTF \__block_item_instance:n
609     {
610       \@latex@error{Lonely~\string\item--perhaps-a~missing-
611         list~environment}\@ehc
612     }
613     {
614       \legacy_if:NTF { @newlist }
615       {
616         \__kernel_list_item_begin:
```

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

```
617         \legacy_if_gset_false:n { @newlist }
618       }
619       { \__block_inter_item: }
```

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

```
620     \tl_if_novalue:NTF {#1}           % avoids reparsing label={ }
621     { \__block_item_instance:n { } }
622     { \__block_item_instance:n {#1} }
```

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

```
623     \legacy_if_gset_true:n { @inlabel }
624     \ignorespaces
625   }
626 }
627 }
```

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

`__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.

```
628 \cs_new_protected:Npn \__block_inter_item: {
629   \legacy_if:N { @inlabel }
630   { \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).

```
631   \mode_if_horizontal:T { \__block_skip_remove_last:
632     \__block_skip_remove_last: \par }
```

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

```
633   \__kernel_list_item_end:
634   \__kernel_list_item_begin:
```



```

635 \addpenalty \@itempenalty
636 \addvspace \itemsep
637 }

```

(End of definition for `_block_inter_item:`)

```

\_kernel_list_item_begin:
\_kernel_list_item_end:
638 \cs_new_eq:NN \_kernel_list_item_begin: \prg_do_nothing:
639 \cs_new_eq:NN \_kernel_list_item_end: \prg_do_nothing:

```

(End of definition for `_kernel_list_item_begin:` and `_kernel_list_item_end:`)

6.5 Tagging support commands

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

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

```

640 \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.

```

641 \cs_set:Npn \_block_beginpar_vmode: {
642     \_block_debug_typeout:n
643     { @endpe = \legacy_if:nTF { @endpe }{true}{false}
644     \on@line }
645 \legacy_if:nTF { @endpe }
646 {
647     \legacy_if_gset_false:n { @endpe }
648 }

```

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>`.

```

649 {
650     \int_compare:nNnT \l__tag_block_flattened_level_int < 2
651     {
652         \_tag_gincr_para_main_begin_int:
653         \tag_struct_begin:n
654         {
655             tag=\l__tag_para_main_tag_tl,
656             attribute-class=\l__tag_para_main_attr_class_tl,
657         }
658         \_tag_para_main_store_struct:
659     }
660 }
661 }

```

(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).

```

662 \cs_set:Npn \_block\_beginpar\_hmode:N #1
663   {
664     \tag\_mc\_end:
665     \_tag\_gincr\_para\_end\_int:
666     \_block\_debug\_typeout:n{increment~ /P \on@line }
667     \bool\_if:NT \l\_tag\_para\_show\_bool
668     { \tag\_mc\_begin:n{artifact}
669       \rlap{\color\_select:n{red}\tiny\ \int\_use:N\g\_tag\_para\_end\_int}
670       \tag\_mc\_end:
671     }
672     \tag\_struct\_end:
673     \tagpdfparaOff \par \tagpdfparaOn
674   }

```

(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.

```

675 \cs_set:Npn \_kernel\_displayblock\_doendpe: {
676   \bool\_if:NT \l\_tag\_para\_bool
677   {

```

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.

```

678     \legacy\_if:nT { @endpe }
679     {

```

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.

```

680       \_block\_debug\_typeout:n
681       { flattened= \bool\_if:NTF
682         \l\_tag\_para\_flattened\_bool
683         {true}{false}
684       \on@line }
685     \bool\_if:NF \l\_tag\_para\_flattened\_bool
686     {
687       \_block\_debug\_typeout:n{Structure-end~
688         \l\_tag\_para\_main\_tag\_tl\space
689         after~ displayblock \on@line }
690       \_tag\_gincr\_para\_main\_end\_int:
691       \tag\_struct\_end: %text-unit
692     }
693   }
694 }

```

```
695 }
```

(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).

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

```
697 \AddToHook{para/begin}[tagpdf]{
```

```
698   \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).

```
699     \legacy_if:nF { @inlabel }
```

```
700     {
```

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.

```
701     \_block_start_para_structure:n { \PARALABEL }
```

```
702   }
```

```
703 }
```

```
704 }
```

```
\_block_start_para_structure:n
```

```
705 \cs_new_protected:Npn \_block_start_para_structure:n #1 {
```

```
706   \_block_debug_typeof:n
```

```
707   { @endpe = \legacy_if:nTF { @endpe }{true}{false}
```

```
708   \on@line }
```

```
709   \legacy_if:nF { @endpe }
```

```
710   {
```

```
711     \bool_if:NF \l__tag_para_flattened_bool
```

```
712     {
```

```
713       \_tag_gincr_para_main_begin_int:
```

```
714       \tag_struct_begin:n
```

```
715       {
```

```
716         tag=\l__tag_para_main_tag_tl,
```

```
717         attribute-class=\l__tag_para_main_attr_class_tl,
```

```
718       }
```

```
719       \_tag_para_main_store_struct:
```

```
720     }
```

```
721   }
```

```
722   \_tag_gincr_para_begin_int:
```

```
723   \_block_debug_typeof:n{increment~ P \on@line }
```

```
724   \tag_struct_begin:n
```

```
725   {
```

```
726     tag=\l__tag_para_tag_tl
```

```
727     ,attribute-class=\l__tag_para_attr_class_tl
```

```
728   }
```

```

729   \_tag_check_para_begin_show:nn {green}{#1}
730   \tag_mc_begin:n {}
731 }

```

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

```

732 \cs_new_protected:Npn \_block_start_para_structure_unconditionally:n #1 {
733   \bool_if:NF \l__tag_para_flattened_bool
734   {
735     \_tag_gincr_para_main_begin_int:
736     \tag_struct_begin:n
737     {
738       tag=\l__tag_para_main_tag_tl,
739       attribute-class=\l__tag_para_main_attr_class_tl,
740     }
741     \_tag_para_main_store_struct:
742   }
743   \_tag_gincr_para_begin_int:
744   \_block_debug_typeout:n{increment~ P \on@line }
745   \tag_struct_begin:n
746   {
747     tag=\l__tag_para_tag_tl
748     ,attribute-class=\l__tag_para_attr_class_tl
749   }
750   \_tag_check_para_begin_show:nn {green}{#1}
751   \tag_mc_begin:n {}
752 }

753 \RemoveFromHook{para/end}[tagpdf]
754 \AddToHook{para/end}
755 {
756   \bool_if:NT \l__tag_para_bool
757   {
758     \_tag_gincr_para_end_int:
759     \_block_debug_typeout:n{increment~ /P \on@line }
760     \tag_mc_end:
761     \_tag_check_para_end_show:nn {red}{}
762     \tag_struct_end:
763     \bool_if:NF \l__tag_para_flattened_bool
764     {
765       \_tag_gincr_para_main_end_int:
766       \tag_struct_end:
767     }
768   }
769 }

770 \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).

```

771 \cs_set_protected:Npn \para_end: {

```

```

772 \scan_stop:
773 \mode_if_horizontal:TF {
774   \mode_if_inner:F {
775     \tex_unskip:D
776     \hook_use:n{para/end}
777     \@kernel@after@para@end
778     \mode_if_horizontal:TF {
779       \if_int_compare:w 11 = \tex_lastnodetype:D
780         \tex_hskip:D \c_zero_dim
781       \fi:
782       \tex_par:D
783       \hook_use:n{para/after}
784       \@kernel@after@para@after
785     }
786     { \msg_error:nmmn { hooks }{ para-mode }{end}{horizontal} }
787   }
788 }
789 {
790   \__kernel_endpe_vmode:      % should do nothing if no tagging
791   \tex_par:D
792 }
793 }
794 \cs_set_eq:NN \par      \para_end:
795 \cs_set_eq:NN \_blockpar \para_end:
796 \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.

```

797 \DeclareRobustCommand*\begin[1]{%
798   \UseHook{env/#1/before}%
799   \ifundefined{#1}%
800     {\def\reserved@a{\@latex@error{Environment-#1~undefined}\@eha}}%
801     {\def\reserved@a{\def@currentenv{#1}%
802       \edef@currentvline{\on@line}%
803       \@execute@begin@hook{#1}%
804       \csname #1\endcsname}}%
805   \@ignorefalse
806   \begingroup
807   \__kernel_endpe_vmode:
808   \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`.

```

809 \cs_new:Npn \__kernel_endpe_vmode: {
810   \if@endpe \ifvmode
811     \bool_if:NT \l__tag_para_bool
812   {
813     \bool_if:NF \l__tag_para_flattened_bool
814     {
815       \__tag_gincr_para_main_end_int:
816       \tag_struct_end:

```

```

817     }
818     \@endpfalse
819   }
820   \fi \fi
821 }

```

(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

```

822 \cs_new:Npn \_kernel_list_label_after: {
823   \bool_if:NT \l__tag_para_bool
824   {
825     \_block_start_para_structure_unconditionally:n { LI- }
826   }
827 }

```

(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.

```

828 \cs_new:Npn \_block_inner_begin: {
829   \tagstructbegin{tag=\l__block_tag_inner_tag_tl}
830 }

```

(End of definition for _block_inner_begin:.)

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

```

831 \cs_new:Npn \_block_inner_end: {
832   \_block_debug_typeout:n{block-end \on@line}
833   \legacy_if:nT { @endpe }
834   {
835     \_tag_gincr_para_main_end_int:
836     \_block_debug_typeout:n{close~ /text-unit \on@line}
837     \tagstructend
838   }
839   \tagstructend      % end inner structure
840 }

```

(End of definition for _block_inner_end:.)

6.5.1 List tags

```

841 \tl_new:N \l__tag_L_tag_tl
842 \tl_set:Nn \l__tag_L_tag_tl {L}
843
844 \tl_new:N \l__tag_L_attr_class_tl
845 \tl_set:Nn \l__tag_L_attr_class_tl {list}
846
847 \tag_if_active:T
848 {
849   \tagpdfsetup
850   {
851     role/new-attribute = {itemize}
852                         {/O /List /ListNumbering/Unordered},
853     role/new-attribute = {enumerate}
854                         {/O /List /ListNumbering/Ordered},

```

```

854         role/new-attribute = {description}
855                             {/O /List /ListNumbering/Description},

```

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

```

856         % default if unknown
857         role/new-attribute = {list}{/O /List /ListNumbering/Unordered},
858     }
859 }
860 \def\LItag{LI}

```

`_block_list_begin:` Start a list ...

```

861 \cs\_set:Npn \_block\_list\_begin: {
862   \tagstructbegin
863   {
864     tag=\l\_tag\_L\_tag\_tl
865     ,attribute-class=\l\_tag\_L\_attr\_class\_tl
866   }
867 }

```

(End of definition for _block_list_begin:.)

`_block_list_item_begin:` Start tagging a list item.

```

868 \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.

```

869 \cs\_set:Npn \_kernel\_list\_label\_begin: {
870   %
871   % FMI: this needs a different logic to decide when to make the label
872   %   an artifact (after cleaning up the \item code ), therefore
873   %   disabled for now
874   % \tl\_if\_empty:oTF \@itemlabel
875   %   {
876   %     \tag\_mc\_begin:n {artifact}
877   %   }
878   %   {
879     \tagstructbegin{tag=Lbl}
880     \tagmcbegin{tag=Lbl}
881   %   }
882 }

```

(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.

```

883 \cs\_set:Npn \_kernel\_list\_label\_end: {
884   \tagmccend % end mc-Lbl or artifact
885   % FMI: unconditionally for now
886   % \tl\_if\_empty:oF \@itemlabel
887   \tagstructend % end Lbl
888   \tagstructbegin{tag=LBody}
889 }
890 \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 `` but also a `<text>` in the special case that the item material ends in a list (identifiable via `@endpe`).

```
891 \cs_set:Npn \_block\_list\_item\_end: {
892   \legacy\_if:nT { @endpe }
893   {
894     \_tag\_gincr\_para\_main\_end\_int:
895     \tagstructend                % text-unit
896 %   \_block\_debug\_typeout:n{Structure-end- P- at- item-end \on@line }
897   }
898   \tagstructend \tagstructend   % end LBody, LI
899 }
```

(End of definition for `_block_list_item_end:`.)

`_block_list_end:` Finally, at the list end we have to close the open `<LBody>`, ``, `<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 `` 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.

```
900 \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.

```
901   \legacy\_if:nF { @newlist }
902   {
903     \legacy\_if:nT { @endpe }
904     {
905       \_tag\_gincr\_para\_main\_end\_int:
906       \tagstructend                % text-unit
907       \_block\_debug\_typeout:n{Structure-end- P- at- list-end \on@line }
908     }
909     \tagstructend\tagstructend   % end LBody, LI
910   }
911   \tagstructend                % end L
912 }
```

(End of definition for `_block_list_end:`.)

End of tagging related declarations.

```
913 }
```

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

```
914 {
915   \cs_new:Npn \_block\_start\_para\_structure\_unconditionally:n #1 {}
916 }
```


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.

```

917 \cs_new:Npn \_block_recipe_basic: {
918   \cs_set_eq:NN \_kernel_displayblock_beginpar_hmode:w
919                                     \_block_beginpar_hmode:N
920   \cs_set_eq:NN \_kernel_displayblock_beginpar_vmode:
921                                     \_block_beginpar_vmode:
922   \let \_kernel_displayblock_begin:  \prg_do_nothing:
923   \let \_kernel_displayblock_end:    \prg_do_nothing:

```

End environment `\par` handling:

```

924   \socket_assign_plug:nn{tagssupport/block-endpe}{on}
925 }

```

(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.

```

926 \cs_new:Npn \_block_recipe_standalone: {
927   \cs_set_eq:NN \_kernel_displayblock_beginpar_hmode:w
928                                     \prg_do_nothing:
929   \cs_set_eq:NN \_kernel_displayblock_beginpar_vmode:
930                                     \prg_do_nothing:
931   \cs_set_eq:NN \_kernel_displayblock_begin: \_block_inner_begin:
932   \cs_set_eq:NN \_kernel_displayblock_end:  \_block_inner_end:

```

End environment `\par` handling:

```

933   \socket_assign_plug:nn{tagssupport/block-endpe}{off}
934   \tl_if_empty:NTF \l__block_tag_name_tl
935     { \tl_set:Nn \l__block_tag_inner_tag_tl {Sect} }
936     { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
937 }

```

(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`.

```

938 \cs_new:Npn \__block_recipe_standard:
939 {
940   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
941                                     \__block_beginpar_hmode:N
942   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
943                                     \__block_beginpar_vmode:
944   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
945   \cs_set_eq:NN \__kernel_displayblock_end: \__block_inner_end:

```

End environment `\par` handling:

```

946   \socket_assign_plug:nn{tagssupport/block-endpe}{on}
947   \tl_if_empty:NTF \l__block_tag_name_tl
948     { \tl_set:Nn \l__block_tag_inner_tag_tl {Figure} }
949     { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
950 }

```

(End of definition for `__block_recipe_standard:.`)

`\l__block_tag_inner_tag_tl`

```

951 \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.

```

952 \cs_new:Npn \__block_recipe_list:
953 {
954   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
955                                     \__block_beginpar_hmode:N
956   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
957                                     \__block_beginpar_vmode:
958   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_list_begin:
959   \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)).

```

960   \cs_set_eq:NN \__kernel_list_item_begin: \__block_list_item_begin:
961   \cs_set_eq:NN \__kernel_list_item_end: \__block_list_item_end:

```

End environment `\par` handling:

```

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

```

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

```

963 \tl_if_empty:NTF \l__block_tag_name_tl
964   { \tl_set:Nn \l__tag_L_tag_tl {L} }
965   { \tl_set_eq:NN \l__tag_L_tag_tl \l__block_tag_name_tl }
966 \tl_if_empty:NTF \l__block_tag_class_tl
967   { \tl_set:Nn \l__tag_L_attr_class_tl {} }
968   { \tl_set_eq:NN \l__tag_L_attr_class_tl \l__block_tag_class_tl }
969 }

```

(End of definition for `__block_recipe_list:.`)

7 Implementation of document-level block environments

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

7.1 Displayblock environments

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

`displayblock` (*env.*)

```

970 \NewDocumentEnvironment{displayblock}{!O{}}
971   { \UseInstance{blockenv}{displayblock} {#1} }
972   { \endblockenv }

```

`displayblockflattened` (*env.*)

```

973 \NewDocumentEnvironment{displayblockflattened}{!O{}}
974   { \UseInstance{blockenv}{displayblockflattened} {#1} }
975   { \endblockenv }

```

7.2 The center, flushleft, and flushright environments

`center` (*env.*)

`flushleft` (*env.*)

`flushright` (*env.*)

```

976 \AddToHook{begindocument/before}{
977   \RenewDocumentEnvironment{center} {!O{}}
978   { \UseInstance{blockenv}{center} {#1} }
979   { \endblockenv }

980   \RenewDocumentEnvironment{flushright} {!O{}}
981   { \UseInstance{blockenv}{flushright} {#1} }
982   { \endblockenv }

983   \RenewDocumentEnvironment{flushleft} {!O{}}
984   { \UseInstance{blockenv}{flushleft} {#1} }
985   { \endblockenv }
986 }

```

7.3 Display quote environments

```
quote (env.)
quotation (env.) 987 \AddToHook{begindocument/before}{
988   \RenewDocumentEnvironment{quote}{!O{ }
989     { \UseInstance{blockenv}{quote} {#1} }
990     { \endblockenv }
991   \RenewDocumentEnvironment{quotation}{!O{ }
992     { \UseInstance{blockenv}{quotation} {#1} }
993     { \endblockenv }
994 }
```

7.4 Verbatim environments

```
verbatim (env.)
verbatim* (env.) 995 \AddToHook{begindocument/before}{
996   \RenewDocumentEnvironment{verbatim}{!O{ }
997     { \UseInstance{blockenv}{verbatim} {#1}

```

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

```
998     \@setupverbinvisible\space\@frenchspacing\@vobeyspaces
999     \@xverbatim
1000   }
1001   { \endblockenv }
1002 \RenewDocumentEnvironment{verbatim*}{!O{ }
1003   { \UseInstance{blockenv}{verbatim} {#1}
1004     \@setupverbvisible\space\@frenchspacing\@vobeyspaces
1005     \@sxverbatim
1006   }
1007   { \endblockenv }
1008 }
```

7.4.1 Helper commands for `verbatim`

`\legacyverbatimsetup` This code resembles the $\text{\LaTeX} 2_{\epsilon}$ `verbatim` implementation with a slight twist: in $\text{\LaTeX} 2_{\epsilon}$ 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.

```
1009 <@@=
1010 \def\legacyverbatimsetup{%
1011   \language\l@nohyphenation
1012   \@tempwafalse
1013   \def\par{%
1014     \if@tempswa
1015       \leavevmode \null {\@@par}\penalty\interlinepenalty
1016     \else
1017       \@tempwatruer
1018       \ifhmode{\@@par}\penalty\interlinepenalty\fi
1019     \fi}%
1020   \let\do\@makeoother \dospecials
```

```

1021 \obeylines \verbatim@font \@noligs
1022 \everypar \expandafter{\the\everypar \unpenalty}%
1023 \tl_set:Nn \l__tag_para_main_tag_tl {codeline}
1024 \tagtool{paratag=Code}% oder faster: \tl_set:Nn\l__tag_para_tag_tl{Code}
1025 }
1026 <@@=block>

```

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

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

```

1027 \newcommand\@setupverbinvisiblespace{}
1028 \tag_if_active:T {
1029   \bool_if:NF\g__tag_mode_lua_bool
1030   {
1031     \renewcommand\@setupverbinvisiblespace
1032       {\def\xobeysp{\nobreakspace\pdffakespace}}
1033   }
1034 }

```

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

7.5 Standard list environments

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

```

enumeration (env.) 1035 \AddToHook{begindocument/before}{
description (env.) 1036 \RenewDocumentEnvironment{itemize}{!0{}}
1037   { \UseInstance{blockenv}{itemize} {#1} }
1038   { \endblockenv }
1039 \RenewDocumentEnvironment{enumerate}{!0{}}
1040   { \UseInstance{blockenv}{enumerate} {#1} }
1041   { \endblockenv }
1042 \RenewDocumentEnvironment{description}{!0{}}
1043   { \UseInstance{blockenv}{description} {#1} }
1044   { \endblockenv }
1045 }

```

7.6 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.

```

1046 \AddToHook{begindocument/before}{
1047 \RenewDocumentEnvironment{verse}{!0{}}
1048   {
1049     \let\\@centercr
1050     \UseInstance{blockenv}{list}
1051     {
1052       item-indent=-1.5em,
1053       parindent=-1.5em,
1054       item-skip=0pt,
1055       rightmargin=\leftmargin,
1056       leftmargin=\leftmargin+1.5em,

```

```

1057         #1
1058     }
1059     \item\relax
1060 }
1061 { \endblockenv }
1062 }

```

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

```

1063 \AddToHook{begindocument/before}{
1064   \RenewDocumentEnvironment{list}{0{} m m }
1065   {

```

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.

```

1066     \tl_set:Nn \l__block_legacy_env_params_tl
1067     {
1068         \tl_set:Nn \@itemlabel {#2}
1069         #3
1070     }
1071     \UseInstance{blockenv}{list} {#1}
1072 }
1073 { \endblockenv }
1074 }

```

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

```

1075 \cs_new:Npn \legacylistsetupcode {

```

Reset values to defaults:

```

1076     \dim_zero:N \listparindent
1077     \dim_zero:N \rightmargin
1078     \dim_zero:N \itemindent

```

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

```

1079 % \tl_set:Nn \@listctr {}
1080 % \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.

```

1081 \let\makelabel\@mklab % TODO: customize

```

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

```

1082 \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.

```

1083 \legacy_if_nTF { @nbrlist }
1084 { \tl_set:Nn \l__tag_L_attr_class_tl {enumerate} } % numbered list
1085 { \tl_if_empty:NTF \@itemlabel

```

```

1086         { \tl_set:Nn \l__tag_L_attr_class_tl {list}      } % no label
1087         { \tl_set:Nn \l__tag_L_attr_class_tl {itemize} } % unnumbered,
1088                                         % unordered
1089     }
1090 }

```

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

`\l_block_legacy_env_params_tl`

```

1091 \tl_new:N\l_block_legacy_env_params_tl

```

(End of definition for `\l_block_legacy_env_params_tl`.)

Replace with code not using `\list`

```

1092 \AddToHook{begindocument/before}{
1093   \RenewDocumentEnvironment{trivlist}{!O{}}{
1094     { \list[#1]{
1095       {
1096         \dim_zero:N \leftmargin
1097         \dim_zero:N \labelwidth
1098         \cs_set_eq:NN \makelabel \use:n
1099       }
1100     }
1101   } \endblockenv }
1102 }

```

7.7 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.

```

1103 \RenewDocumentCommand \newtheorem { m O{#1} m o }
1104 {
1105   \expandafter\@ifdefinable\csname #1\endcsname
1106   {
1107     \str_if_eq:nnTF{#1}{#2}
1108     {
1109       \definecounter {#2}
1110       \IfNoValueTF {#4}
1111       { % @ynthm
1112         \tl_gset:ce { the #2 }
1113         {
1114           \@thmcounter{#2}
1115         }
1116       }
1117       { % @xnthm
1118         \@newctr{#1}[#4]
1119         \tl_gset:ce { the #2 }
1120         {
1121           \expandafter\noexpand\csname the#4\endcsname

```

```

1122         \@thmcountersep
1123         \@thmcounter{#2}
1124     }
1125 }
1126 }
1127 { % @othm
1128     \@ifundefined{c@#2}
1129     { \@nocounterr{#2} }
1130     {
1131         \tl_gset:cn { the #1 }
1132         { \UseName { the #2 } }
1133     }
1134 }
1135 \global\@namedef{#1} { \@thm{#2}{#3} }
1136 \global\@namedef{end#1}{ \@endtheorem }
1137 }
1138 }

```

(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.

```

1139 \tl_new:N \l__block_thm_current_counter_tl
1140 \def\@thm#1#2{%
1141     \@kernel@refstepcounter{#1}
1142     \tl_set:Nn \l__block_thm_current_counter_tl{#1}
1143     \@ifnextchar[{\@ythm{#1}{#2}}{\@xthm{#1}{#2}}}]

```

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

```

1144 \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.

```

1145 \def\@begintheorem#1#2{
1146     \UseInstance{blockenv}{theorem}{-}
1147     \tagpdfparaOff
1148
1149     \noindent
1150     \MakeLinkTarget{\l__block_thm_current_counter_tl}
1151     \tag_struct_begin:n{tag=Caption}
1152     \group_begin:
1153     \bfseries
1154     \tag_mc_begin:n {#1}
1155     \tag_mc_end:
1156     \tag_struct_begin:n{tag=Lbl}

```



```

1157     \tag_mc_begin:n {}
1158     #2
1159     \tag_mc_end:
1160     \tag_struct_end:
1161     \group_end:
1162     \tag_struct_end:
1163     \tagpdfparaOn
1164     \__block_start_para_structure_unconditionally:n { \PARALABEL }
1165     \itshape
1166     \hskip\labelsep
1167     \ignorespaces
1168   }
1169   \def\@opargbegintheorem#1#2#3{
1170     \UseInstance{blockenv}{theorem}{#3}
1171     \tagpdfparaOff

1172     \noindent
1173     \MakeLinkTarget{\l_block_thm_current_counter_tl}
1174     \tag_struct_begin:n{tag=Caption}
1175     \group_begin:
1176     \bfseries
1177     \tag_mc_begin:n {}
1178     #1\
1179     \tag_mc_end:
1180     \tag_struct_begin:n{tag=Lbl}
1181     \tag_mc_begin:n {}
1182     #2
1183     \tag_mc_end:
1184     \tag_struct_end:
1185     \tag_mc_begin:n {}
1186     \ (#3)
1187     \tag_mc_end:
1188     \group_end:
1189     \tag_struct_end:
1190     \tagpdfparaOn
1191     \__block_start_para_structure_unconditionally:n { \PARALABEL }
1192     \itshape
1193     \hskip\labelsep
1194     \ignorespaces
1195   }
1196   \def\@endtheorem{\endblockenv}

```

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

8 Instance declarations for environments

8.1 Blockenv instances

The blockenv instances handle overall setup for the document-level environments, i.e.,

- name of the environment for debugging purposes (`env-name`)

- how tagging should be performed (`tagging-recipe`, `tag-name`, `tag-class`)
- does this environment changes the block level if nested (`level-increase`)
- any special setup code; normally not used (`setup-code`)
- what kind of block instance should be used (`block-instance`)
- what kind of para instance should be used; empty means inherit from the outside (`para-instance`)
- are inner paragraphs real paragraphs (default) or are they just `<text>` structures and part of an outer paragraph (`para-flattened`)
- what kind of inner instance should be used, if any (`inner-instance`, `inner-instance-type`)
- the counter name of the inner level, if any (`inner-level-counter`)
- supported nesting depth of the inner level, if relevant (`max-inner-levels`)
- extra code executed at the end, by default `\ignorespaces` (`final-code`)

The `blockenv displayblock` instance below shows the full set with those using default values being commented out.

8.1.1 Basic instances

`blockenv displayblock (inst.)` This is like L^AT_EX 2_ε's `trivlist`, i.e., it produces a vertical block with default setting, but doesn't put a list inside but uses a `<Figure>` structure.

```

1197 \DeclareInstance{blockenv}{displayblock}{display}
1198 {
1199   env-name      = displayblock,
1200   % tagging-recipe = standard,
1201   % tag-name      = ,
1202   % tag-class     = ,
1203   level-increase = false,
1204   % setup-code   = ,
1205   % block-instance = displayblock ,
1206   % para-instance = ,
1207   % para-flattened = false ,
1208   % inner-instance = ,
1209   % inner-instance-type = list , % not relevant as there is no inner instance
1210   % inner-level-counter = ,      % not relevant as there is no inner instance
1211   % max-inner-levels = 4,        % not relevant as there is no inner instance
1212   % final-code = \ignorespaces ,
1213 }

```

`blockenv displayblockflattened (inst.)` This flattens inner paragraphs without any surrounding tag structure by using the `basic` tagging recipe.

```

1214 \DeclareInstance{blockenv}{displayblockflattened}{display}
1215 {
1216   env-name      = displayblockflattened,
1217   tag-name      = ,
1218   tag-class     = ,

```

```

1219 tagging-recipe = basic,
1220 inner-level-counter = ,
1221 level-increase = false,
1222 setup-code = ,
1223 block-instance = displayblock ,
1224 para-flattened = true ,
1225 inner-instance = ,
1226 }

```

8.1.2 Center, flushleft, and flushright instances

All three environments use the `displayblock` instance as block instance. They only differ in the choice of para instance.

`blockenv center` (*inst.*) The center environment without using a list internally.

```

1227 \DeclareInstance{blockenv}{center}{display}
1228 {
1229   env-name      = center,
1230   tag-name      = ,
1231   tag-class     = ,
1232   tagging-recipe = basic,
1233   inner-level-counter = ,
1234   level-increase = false,
1235   setup-code    = ,
1236   block-instance = displayblock ,
1237   para-flattened = true ,
1238   para-instance = center ,
1239   inner-instance = ,
1240 }

```

`blockenv flushleft` (*inst.*) The flushleft environment without using a list internally.

```

1241 \DeclareInstance{blockenv}{flushleft}{display}
1242 {
1243   env-name      = flushleft,
1244   tag-name      = ,
1245   tag-class     = ,
1246   tagging-recipe = basic,
1247   inner-level-counter = ,
1248   level-increase = false,
1249   setup-code    = ,
1250   block-instance = displayblock ,
1251   para-flattened = true ,
1252   para-instance = raggedright ,
1253   inner-instance = ,
1254 }

```

`blockenv flushright` (*inst.*) The flushright environment without using a list internally.

```

1255 \DeclareInstance{blockenv}{flushright}{display}
1256 {
1257   env-name      = flushleft,
1258   tag-name      = ,
1259   tag-class     = ,
1260   tagging-recipe = basic,
1261   inner-level-counter = ,

```

```

1262 level-increase = false,
1263 setup-code     = ,
1264 block-instance = displayblock ,
1265 para-flattened = true ,
1266 para-instance  = raggedleft ,
1267 inner-instance = ,
1268 }

```

8.1.3 Blockquote instances

L^AT_EX 2_ε has two environments for quoting: `quote` and `quotation`. By default they differ only in indentation of inner paragraphs. This is handled by using separate block instances.

The tag names are both roll-mapped to `<BlockQuote>`.

`blockenv quotation` (*inst.*) For the quotation environment:

```

1269 \DeclareInstance{blockenv}{quotation}{display}
1270 {
1271   env-name       = quotation,
1272   tag-name       = quotation,
1273   tag-class      = ,
1274   tagging-recipe = standard,
1275   inner-level-counter = ,
1276   level-increase = true,
1277   setup-code     = ,
1278   block-instance = quotationblock ,
1279   inner-instance = ,
1280 }

```

`blockenv quote` (*inst.*) For the quote environment:

```

1281 \DeclareInstance{blockenv}{quote}{display}
1282 {
1283   env-name       = quote,
1284   tag-name       = quote,
1285   tag-class      = ,
1286   tagging-recipe = standard,
1287   inner-level-counter = ,
1288   level-increase = true,
1289   setup-code     = ,
1290   block-instance = quoteblock ,
1291   inner-instance = ,
1292 }

```

8.1.4 The theorem instance

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

```

1293 \DeclareInstance{blockenv}{theorem}{display}
1294 {
1295   env-name       = theorem-like,
1296   tag-name       = theorem-like,
1297   tag-class      = ,
1298   tagging-recipe = standalone,
1299   inner-level-counter = ,

```

```

1300 level-increase = false,
1301 setup-code     = ,
1302 block-instance = theoremblock ,
1303 }

```

8.1.5 The verbatim instance

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

```

1304 \DeclareInstance{blockenv}{verbatim}{display}
1305 {
1306   env-name      = verbatim,
1307   tag-name      = verbatim,
1308   tag-class     = ,
1309   tagging-recipe = standard,
1310   inner-level-counter = ,
1311   level-increase = false,
1312   setup-code    = ,
1313   block-instance = verbatimblock ,
1314   inner-instance = ,
1315   final-code    = \legacyverbatimsetup ,
1316   para-instance = justify
1317 }

```

8.1.6 Standard list instances

`blockenv itemize` (*inst.*) For the `itemize` environment:

```

1318 \DeclareInstance{blockenv}{itemize}{display}
1319 {
1320   env-name      = itemize,
1321   tag-name      = itemize,
1322   tag-class     = itemize,
1323   tagging-recipe = list,
1324   inner-level-counter = \@itemdepth,
1325   level-increase = true,
1326   max-inner-levels = 4,
1327   setup-code    = ,
1328   block-instance = list ,
1329   inner-instance = itemize ,
1330 }

```

`blockenv enumerate` (*inst.*) For the `enumerate` environment:

```

1331 \DeclareInstance{blockenv}{enumerate}{display}
1332 {
1333   env-name      = enumerate,
1334   tag-name      = enumerate,
1335   tag-class     = enumerate,
1336   tagging-recipe = list,
1337   level-increase = true,
1338   setup-code    = ,
1339   block-instance = list ,
1340   inner-level-counter = \@enumdepth,
1341   max-inner-levels = 4,

```

```

1342 inner-instance      = enum ,
1343 }

```

`blockenv description` (*inst.*) For the description environment:

```

1344
1345 \DeclareInstance{blockenv}{description}{display}
1346 {
1347   env-name          = description,
1348   tag-name          = description,
1349   tag-class         = description,
1350   tagging-recipe    = list,
1351   inner-level-counter = ,
1352   level-increase    = true,
1353   setup-code        = ,
1354   block-instance    = list ,
1355   inner-instance    = description ,
1356 }

```

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

```

1357 \DeclareInstance{blockenv}{list}{display}
1358 {
1359   env-name          = list,
1360   tag-name          = list,
1361   tag-class         = ,
1362   tagging-recipe    = list,
1363   level-increase    = true,
1364   setup-code        = \legacylistsetupcode ,
1365   block-instance    = list ,
1366   inner-level-counter = ,
1367   inner-instance    = legacy ,
1368 }

```

8.2 Block instances

Below, we declare the different block instances for the document-level environments. Some, such as the `displayblock` ones are shared between different environments (as long as one doesn't need to adjust individual values), other environments have their own instances (for precisely that reason).

8.2.1 Displayblock instances

We provide 6 nesting levels (as in L^AT_EX 2_ε). 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).

```

1369 \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.*) We show all keys here for reference, with those using their default values commented out:

`block displayblock-2` (*inst.*)

`block displayblock-3` (*inst.*)

`block displayblock-4` (*inst.*)

`block displayblock-5` (*inst.*)

`block displayblock-6` (*inst.*)

```

1370 \DeclareInstance{block}{displayblock-0}{display}
1371 {
1372 % heading = , %??
1373 % beginsep = \topsep ,
1374 % begin-par-skip = \partopsep ,
1375 % par-skip = \parsep ,
1376 % end-skip = \KeyValue{beginsep} ,
1377 % end-par-skip = \KeyValue{begin-par-skip} ,
1378 % item-skip = \itemsep ,
1379 % beginpenalty = \UseName{@beginparpenalty} ,
1380 % endpenalty = \UseName{@endparpenalty} ,
1381 leftmargin = Opt ,
1382 rightmargin = \rightmargin ,
1383 % parindent = Opt ,
1384 }

1385 \DeclareInstanceCopy{block}{displayblock-1}{displayblock-0}
1386 \DeclareInstanceCopy{block}{displayblock-2}{displayblock-0}
1387 \DeclareInstanceCopy{block}{displayblock-3}{displayblock-0}
1388 \DeclareInstanceCopy{block}{displayblock-4}{displayblock-0}
1389 \DeclareInstanceCopy{block}{displayblock-5}{displayblock-0}
1390 \DeclareInstanceCopy{block}{displayblock-6}{displayblock-0}

```

8.2.2 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.) 1391 \DeclareInstance{block}{verbatimblock-0}{display}
block verbatimblock-2 (inst.) 1392 {
block verbatimblock-3 (inst.) 1393 leftmargin = Opt ,
block verbatimblock-4 (inst.) 1394 par-skip = Opt ,
block verbatimblock-5 (inst.) 1395 }
block verbatimblock-6 (inst.) 1396 \DeclareInstanceCopy{block}{verbatimblock-1}{verbatimblock-0}
1397 \DeclareInstanceCopy{block}{verbatimblock-2}{verbatimblock-0}
1398 \DeclareInstanceCopy{block}{verbatimblock-3}{verbatimblock-0}
1399 \DeclareInstanceCopy{block}{verbatimblock-4}{verbatimblock-0}
1400 \DeclareInstanceCopy{block}{verbatimblock-5}{verbatimblock-0}
1401 \DeclareInstanceCopy{block}{verbatimblock-6}{verbatimblock-0}

```

8.2.3 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.) 1402 \DeclareInstance{block}{quoteblock-1}{display}
block quoteblock-3 (inst.) 1403 { rightmargin = \KeyValue{leftmargin} }
block quoteblock-4 (inst.) 1404 \DeclareInstanceCopy{block}{quoteblock-2}{quoteblock-1}
block quoteblock-5 (inst.) 1405 \DeclareInstanceCopy{block}{quoteblock-3}{quoteblock-1}
block quoteblock-6 (inst.) 1406 \DeclareInstanceCopy{block}{quoteblock-4}{quoteblock-1}
1407 \DeclareInstanceCopy{block}{quoteblock-5}{quoteblock-1}
1408 \DeclareInstanceCopy{block}{quoteblock-6}{quoteblock-1}

```

```

block quotationblock-1 (inst.) Quotation additionally changes the parindent.
block quotationblock-2 (inst.) 1409 \DeclareInstance{block}{quotationblock-1}{display}
block quotationblock-3 (inst.) 1410 { parindent = 1.5em , rightmargin = \KeyValue{leftmargin} }
block quotationblock-4 (inst.) 1411 \DeclareInstanceCopy{block}{quotationblock-2}{quotationblock-1}
block quotationblock-5 (inst.) 1412 \DeclareInstanceCopy{block}{quotationblock-3}{quotationblock-1}
block quotationblock-6 (inst.) 1413 \DeclareInstanceCopy{block}{quotationblock-4}{quotationblock-1}
1414 \DeclareInstanceCopy{block}{quotationblock-5}{quotationblock-1}
1415 \DeclareInstanceCopy{block}{quotationblock-6}{quotationblock-1}

```

8.2.4 Block instances for the theorems

block theoremblock-0 (inst.) Theorems do not support nesting, so we have only one to set up. The L^AT_EX default reused the general value of `\parindent` and `\parskip` and, of course, they start at the outer margin.

```

1416 \DeclareInstance{block}{theoremblock-0}{display}
1417 {
1418   leftmargin      = Opt ,
1419   parindent       = \parindent ,
1420   par-skip        = \parskip ,
1421 }

```

8.2.5 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
block list-3 (inst.) done in the legacy `\list{romannumeral}` unless a different layout is wanted.

```

block list-4 (inst.) 1422 \DeclareInstance{block}{list-1}{display}{
block list-5 (inst.) 1423 % heading          = , % unused, might vanish
block list-6 (inst.) 1424 % beginsep         = \topsep ,
1425 % begin-par-skip   = \partopsep ,
1426 % par-skip         = \parsep ,
1427 % end-skip         = \KeyValue{beginsep} ,
1428 % end-par-skip     = \KeyValue{begin-par-skip} ,
1429 % beginpenalty     = \UseName{@beginparpenalty} ,
1430 % endpenalty       = \UseName{@endparpenalty} ,
1431 % leftmargin       = \leftmargin ,
1432 % rightmargin      = \rightmargin ,
1433 % parindent        = Opt ,
1434 }
1435 \DeclareInstance{block}{list-2}{display}{}
1436 \DeclareInstance{block}{list-3}{display}{}
1437 \DeclareInstance{block}{list-4}{display}{}
1438 \DeclareInstance{block}{list-5}{display}{}
1439 \DeclareInstance{block}{list-6}{display}{}

```

8.3 List instances for the standard lists

For all list instances we have to say what kind of label we want (`item-label`) 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 definitions rather than defining the `item-label` code in the instance to ensure that old documents still work.

```
list itemize-2 (inst.)
list itemize-3 (inst.)
list itemize-4 (inst.)
1440 \DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi }
1441 \DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii }
1442 \DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
1443 \DeclareInstance{list}{itemize-4}{std}{ item-label = \labelitemiv }
```

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

```
list enumerate-2 (inst.)
list enumerate-3 (inst.)
list enumerate-4 (inst.)
1444 \DeclareInstance{list}{enum-1}{std}
1445   { item-label = \labelenumi , counter = enumi }
1446 \DeclareInstance{list}{enum-2}{std}
1447   { item-label = \labelenumii , counter = enumii }
1448 \DeclareInstance{list}{enum-3}{std}
1449   { item-label = \labelenumiii , counter = enumiii }
1450 \DeclareInstance{list}{enum-4}{std}
1451   { 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

```
1452 \DeclareInstance{list}{legacy}{std} {
1453   item-instance = basic ,
1454   legacy-support = true ,
1455 }
```

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

```
1456 \DeclareInstance{list}{description}{std} { item-instance = description }
```

8.4 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).

```
1457 \DeclareInstance{item}{basic}{std}
1458   { label-align = right }
1459 \DeclareInstance{item}{description}{std}
1460   {
1461     label-format = \normalfont\bfseries #1 ,
1462     label-align = left
1463   }
```

8.5 Para instances

```
1464 \tag_if_active:T
1465 {
1466   \tagpdfsetup
1467   {
1468     role/new-attribute = {justify}    {/O /Layout /TextAlign/Justify},
1469     role/new-attribute = {center}    {/O /Layout /TextAlign/Center},
1470     role/new-attribute = {raggedright}{/O /Layout /TextAlign/Start},
1471     role/new-attribute = {raggedleft}{/O /Layout /TextAlign/End},
1472   }
1473 }
```

`para center` (*inst.*)

```
1474 \DeclareInstance{para}{center}{std}
1475 {
1476   para-class          = center ,
1477   indent-width        = Opt ,
1478   % start-skip        = Opt ,
1479   left-skip           = \@flushglue ,
1480   right-skip          = \@flushglue ,
1481   end-skip            = \@skip ,
1482   final-hyphen-demerits = 0 ,
1483   cr-cmd              = \@centercr ,
1484 }
```

`para raggedright` (*inst.*)

```
1485 \DeclareInstance{para}{raggedright}{std}
1486 {
1487   para-class          = raggedright ,
1488   indent-width        = Opt ,
1489   % start-skip        = Opt ,
1490   left-skip           = \@skip ,
1491   right-skip          = \@flushglue ,
1492   end-skip            = \@skip ,
1493   final-hyphen-demerits = 0 ,
1494   cr-cmd              = \@centercr ,
1495 }
```

`para raggedleft` (*inst.*)

```
1496 \DeclareInstance{para}{raggedleft}{std}
1497 {
1498   para-class          = raggedleft ,
1499   indent-width        = Opt ,
1500   % start-skip        = Opt ,
1501   left-skip           = \@flushglue ,
1502   right-skip          = \@skip ,
1503   end-skip            = \@skip ,
1504   final-hyphen-demerits = 0 ,
1505   cr-cmd              = \@centercr ,
1506 }
```

`para justify` (*inst.*) Justifying is exactly what the default values do, so the instance hasn't any special setup.

```
1507 \DeclareInstance{para}{justify}{std}
```

```

1508 {
1509 % para-class           = justify ,
1510 % indent-width         = \parindent ,
1511 % start-skip           = Opt ,
1512 % left-skip            = \z@skip ,
1513 % right-skip           = \z@skip ,
1514 % end-skip             = \@flushglue ,
1515 % final-hyphen-demerits = 5000 ,
1516 % cr-cmd               = \@normalcr ,
1517 }

\centering
\raggedleft
\raggedright
\justifying
1518 \DeclareRobustCommand\centering {\UseInstance{para}{center}{}}
1519 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}{}}
1520 \DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}{}}
1521 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}{}}
1522 \justifying

(End of definition for \centering and others.)

1523 </package>
1524 <*latex-lab>
1525 \ProvidesFile{block-latex-lab-testphase.ltx}
1526 [\tllabblockdate\space v\tllabblockversion\space
1527 blockenv implementation]
1528 \RequirePackage{latex-lab-testphase-block}
1529 </latex-lab>

```

A Documentation from first prototype implementations

A.1 Open questions

- Existing questions — moved to issues —

A.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.

A.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 $\backslash\text{leftmargin} + \backslash\text{itemindent} = \backslash\text{labelindent} + \backslash\text{labelwidth} + \backslash\text{labelsep}$.
- 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.

B 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`.

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?

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

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

- Counter name (`counter`), shared with the parent *list block* template, but needed for incrementing.
- Label construction: a function `counter-label` that produces the label from the counter name, used if `\item` is given without argument.
- References: a function `counter-ref` for how the label should be referred to when it is constructed from the counter, `label-ref` and `label-autoref` used when `\item` has an optional argument.
- Label formatting: `label-format` function, `label-strut` boolean.
- Label alignment (`label-align`, `label-boxed`, `next-line`).
- Content parameters: `text-font`.
- A `compatibility` boolean that controls for instance whether `\makelabel` is used.

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

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

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	B
<code>\</code> 274, 1049	<code>\begin</code> <u>12</u> , <u>797</u>
<code>_</code> 669, 1154, 1178, 1186	<code>\begingroup</code> <u>806</u>
	<code>\bfseries</code> 1152, 1176, 1461
	<code>block</code> (objecttype) <u>24</u>
	block commands:
	<code>\block_debug_off:</code> .. <u>11</u> , <u>111</u> , 116, 129
	<code>\block_debug_on:</code> ... <u>11</u> , <u>111</u> , 111, 128
	<code>\g_block_nesting_depth_int</code>
	. <u>11</u> , <u>131</u> , 175, 179, 181, 191, 194, 226
	<code>block display</code> (template) <u>46</u> , <u>281</u>
	<code>block displayblock-0</code> (instance) ... <u>1370</u>
	<code>block displayblock-1</code> (instance) ... <u>1370</u>
	<code>block displayblock-2</code> (instance) ... <u>1370</u>
Numbers	
<code>\1</code> 60	
<code>\2</code> 60	
A	
<code>\addpenalty</code> 248, 369, 635	
<code>\AddToHook</code> 572, 604, 697, 754,	
976, 987, 995, 1035, 1046, 1063, 1092	
<code>\addvspace</code> 249, 367, 370, 371, 636	
<code>\arabic</code> 91	

⁴This should be made easily extendible to deeper levels.

block displayblock-3 (instance) ...	1370	_block_item_everypar_std: ...	
block displayblock-4 (instance) ...	1370	547 , 571 , 574
block displayblock-5 (instance) ...	1370	_block_item_instance:n	
block displayblock-6 (instance) ...	1370	32 , 410 , 608 , 621 , 622
block internal commands:		\\l_block_item_label_tl	407 , 449 , 451
_block_beginpar_hmode:N		\\l_block_item_parsep_skip	544
.....	662 , 662 , 919 , 941 , 955	_block_label_autoref:n	474
_block_beginpar_vmode:		\\l_block_label_boxed_bool	477 , 509
.....	641 , 641 , 921 , 943 , 957	_block_label_format:n	
\\l_block_block_instance_tl	30 , 475 , 553 , 558
.....	141 , 191 , 193 , 255	\\l_block_label_given_tl	
\\l_block_botsep_skip	287 , 463	27 , 28 , 468 , 490 , 493 , 506
_block_counter_label:n ...	471 , 502	_block_label_ref:n	473
_block_counter_ref:n	472	\\l_block_label_strut_bool	476 , 560
\\l_block_counter_start_int		\\g_block_labels_box	
.....	408 , 436 , 446	..	27 , 29 , 351 , 533 , 535 , 550 , 583
\\l_block_counter_tl ..	406 , 429 , 442	\\l_block_legacy_env_params_tl ..	
_block_debug:n	109 , 109 , 123	46 , 1066 , 1082 , 1091
\\g_block_debug_bool		\\l_block_legacy_support_bool	
.....	108 , 113 , 118 , 124 , 126	417 , 561
_block_debug_gset: 111 , 114 , 119 , 121		\\l_block_level_incr_bool	
_block_debug_timeout:n ...	109 ,	139 , 173 , 225
110 , 125 , 151 , 190 , 198 , 204 , 224 ,		_block_list_begin: ..	861 , 861 , 958
238 , 257 , 382 , 386 , 390 , 420 , 455 ,		_block_list_end:	900 , 900 , 959
489 , 505 , 575 , 642 , 666 , 680 , 687 ,		_block_list_item_begin:	
706 , 723 , 744 , 759 , 832 , 836 , 896 , 907		868 , 868 , 960
\\l_block_effective_top_skip ...		_block_list_item_end: 891 , 891 , 961	
.....	320 , 322 , 370 , 602	\\l_block_long_label_bool	
\\l_block_env_name_tl	135 , 151	531 , 532 , 540 , 552
_block_evaluate_saved_user_-		_block_make_label_box:n	
keys:nn	24 , 25 ,	27 , 499 , 501 , 506 , 553 , 553
28 , 395 , 395 , 398 , 400 , 423 , 426 , 491		\\l_block_max_inner_levels_tl ...	
\\l_block_final_code_tl ..	18 , 148 , 215	145 , 169
\\l_block_heading_tl ..	283 , 298 , 300	\\l_block_next_line_bool ...	478 , 539
_block_if_list:TF	232 , 237 , 254 , 254	\\l_block_one_label_box	
_block_inner_begin:	29 , 512 , 516 , 518 ,
.....	828 , 828 , 931 , 944	522 , 523 , 527 , 528 , 530 , 537 , 550 , 555
_block_inner_end: 831 , 831 , 932 , 945		\\l_block_para_instance_tl	
\\l_block_inner_instance_tl	142 , 196 , 199 , 200
.....	147 , 202 , 204 , 208	\\l_block_parbotsep_skip ...	288 , 464
\\l_block_inner_instance_type_tl		\\l_block_parindent_dim ...	294 , 343
.....	146 , 207	_block_recipe_basic:	917 , 917
\\l_block_inner_level_counter_tl		_block_recipe_list:	952 , 952
.....	144 , 166 , 168 , 171 , 205 , 206 , 209 , 211	_block_recipe_standalone: 926 , 926	
_block_inter_item: 32 , 619 , 628 , 628		_block_recipe_standard: ..	938 , 938
\\l_block_item_align_tl		\\l_block_resume_bool ..	409 , 433 , 443
.....	482 , 483 , 484 , 520 , 524 , 549	_block_save_user_keys:n	396
\\l_block_item_compatibility_-		\\l_block_setup_code_tl ..	17 , 140 , 189
bool	480 , 496	_block_skip_remove_last:	
_block_item_everypar:	26 ,	103 , 106 , 234 , 309 , 631 , 632
29-31 , 454 , 547 , 571 , 571 , 572 , 594		_block_skip_set_to_last:N	
_block_item_everypar_first:	103 , 103 , 242 , 359
.....	454 , 571 , 597	_block_start_para_structure:n ..	
		701 , 705 , 705

<code>\DeclareInstance</code>	1197, 1214, 1227, 1241, 1255, 1269, 1281, 1293, 1304, 1318, 1331, 1345, 1357, 1370, 1391, 1402, 1409, 1416, 1422, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1446, 1448, 1450, 1452, 1456, 1457, 1459, 1474, 1485, 1496, 1507	
<code>\DeclareInstanceCopy</code>	1385, 1386, 1387, 1388, 1389, 1390, 1396, 1397, 1398, 1399, 1400, 1401, 1404, 1405, 1406, 1407, 1408, 1411, 1412, 1413, 1414, 1415	
<code>\DeclareRobustCommand</code>	797, 1518, 1519, 1520, 1521	
<code>\DeclareTemplateCode</code>	133, 265, 281, 404, 469	
<code>\DeclareTemplateInterface</code>	29, 46, 63, 75, 89	
<code>\def</code>	8, 9, 770, 800, 801, 860, 890, 1010, 1013, 1032, 1140, 1144, 1145, 1169, 1196	
description (env.)	1035	
<code>\detokenize</code>	257, 383, 387, 391	
dim commands:		
<code>\dim_add:Nn</code>	344, 345	
<code>\dim_compare:nNnTF</code>	243, 515, 530, 545	
<code>\dim_compare_p:n</code>	511	
<code>\dim_set_eq:NN</code>	343, 546	
<code>\dim_zero:N</code>	460, 461, 462, 463, 464, 1076, 1077, 1078, 1096, 1097	
<code>\c_zero_dim</code>	243, 780	
displayblock (env.)	970	
displayblockflattened (env.)	973	
<code>\do</code>	1020	
<code>\dospecials</code>	1020	
E		
<code>\edef</code>	802	
<code>\else</code>	1016	
<code>\end</code>	235	
<code>\endblockenv</code>	223, 972, 975, 979, 982, 985, 990, 993, 1001, 1007, 1038, 1041, 1044, 1061, 1073, 1101, 1196	
endblockenv	11	
<code>\endcsname</code>	804, 1105, 1121	
<code>\endgraf</code>	796	
enumerate (env.)	1035	
environments:		
center	976	
description	1035	
displayblock	970	
displayblockflattened	973	
enumerate	1035	
flushleft	976	
flushright	976	
itemize	1035	
list	1063	
quotation	987	
quote	987	
trivlist	1092	
verbatim	995	
verbatim*	995	
verse	1046	
<code>\everypar</code>	15, 18, 19, 1022	
exp commands:		
<code>\exp_after:wN</code>	520, 524	
<code>\exp_not:n</code>	401, 427	
<code>\expandafter</code>	1022, 1105, 1121	
<code>\ExplSyntaxOn</code>	7	
F		
<code>\fi</code>	820, 1018, 1019	
fi commands:		
<code>\fi:</code>	781	
<code>\finalhyphendemerits</code>	273	
flushleft (env.)	976	
flushright (env.)	976	
<code>\frenchspacing</code>	998, 1004	
G		
<code>\global</code>	1135, 1136	
group commands:		
<code>\group_begin:</code>	1151, 1175	
<code>\group_end:</code>	1161, 1188	
H		
hbox commands:		
<code>\hbox_gset:Nn</code>	351, 533	
<code>\hbox_set:Nn</code>	527, 555	
<code>\hbox_set_to_wd:Nnn</code>	518	
<code>\hbox_unpack_drop:N</code>	354, 522, 535, 537	
<code>\hfil</code>	540	
hook commands:		
<code>\hook_use:n</code>	776, 783	
Hooks:		
para/begin	29, 30	
<code>\hskip</code>	1166, 1193	
<code>\hss</code>	482, 483, 484	
I		
if commands:		
<code>\if_int_compare:w</code>	779	
<code>\ifhmode</code>	1018	
<code>\IfNoValueTF</code>	1110	
<code>\ifvmode</code>	810	
<code>\ignorespaces</code>	44, 624, 1167, 1194, 1212	
<code>\indent</code>	630	
instances:		
block displayblock-0	1370	
block displayblock-1	1370	

block displayblock-2	1370	list itemize-4	1440
block displayblock-3	1370	list legacy	1452
block displayblock-4	1370	para center	1474
block displayblock-5	1370	para justify	1507
block displayblock-6	1370	para raggedleft	1496
block list-1	1422	para raggedright	1485
block list-2	1422	int commands:	
block list-3	1422	\int_compare:nNnTF	
block list-4	1422	155, 168, 175, 332, 650
block list-5	1422	\int_gdecr:N	226
block list-6	1422	\int_gincr:N	179
block quotationblock-1	1409	\int_gset:Nn	435, 445
block quotationblock-2	1409	\int_if_exist:NTF	217
block quotationblock-3	1409	\int_incr:N	157, 162, 171, 331
block quotationblock-4	1409	\int_new:N	219
block quotationblock-5	1409	\int_set:Nn	590
block quotationblock-6	1409	\int_set_eq:NN	593
block quoteblock-1	1402	\int_to_roman:n	180
block quoteblock-2	1402	\int_use:N	191, 193, 206, 211, 669
block quoteblock-3	1402	\int_zero:N	326
block quoteblock-4	1402	\c_zero_int	585
block quoteblock-5	1402	\interlinepenalty	1015, 1018
block quoteblock-6	1402	item (objecttype)	24
block theoremblock-0	1416	\item	12, 604, 630, 872, 1059
block verbatimblock-0	1391	item basic (instance)	1457
block verbatimblock-1	1391	item description (instance)	1457
block verbatimblock-2	1391	item std (template)	89, 467
block verbatimblock-3	1391	\itemindent	84, 414, 536, 581, 1078
block verbatimblock-4	1391	itemize (env.)	1035
block verbatimblock-5	1391	\itemsep	54, 82, 289, 411, 460, 636, 1378
block verbatimblock-6	1391	\itshape	1165, 1192
blockenv center	1227		
blockenv description	1344		
blockenv displayblock	1197		
blockenv displayblockflattened	1214		
blockenv enumerate	1331		
blockenv flushleft	1241		
blockenv flushright	1255		
blockenv itemize	1318		
blockenv list	1357		
blockenv quotation	1269		
blockenv quote	1281		
blockenv theorem	1293		
blockenv verbatim	1304		
item basic	1457		
item description	1457		
list description	1456		
list enumerate-1	1444		
list enumerate-2	1444		
list enumerate-3	1444		
list enumerate-4	1444		
list itemize-1	1440		
list itemize-2	1440		
list itemize-3	1440		
		J	
		\justifying	1518
		K	
		\kern	581
		kernel internal commands:	
		__kernel_displayblock_begin:	
		41, 347, 381, 381, 922, 931, 944, 958
		__kernel_displayblock_beginpar_	
		hmode:w	
		310, 381, 385, 918, 927, 940, 954
		__kernel_displayblock_beginpar_	
		vmode:	
		306, 381, 389, 920, 929, 942, 956
		__kernel_displayblock_doendpe:	
		34, 8, 13, 23, 675, 675
		__kernel_displayblock_end:	
		41, 236, 256, 256, 923, 932, 945, 959
		__kernel_endpe_vmode:	
		790, 807, 809, 809
		__kernel_list_item_begin:	
		616, 634, 638, 638, 960

<code>__kernel_list_item_end:</code>	<code>list enumerate-3 (instance)</code>	1444
..... 633 , 638 , 639 , 961	<code>list enumerate-4 (instance)</code>	1444
<code>__kernel_list_label_after:</code>	<code>list itemize-1 (instance)</code>	1440
..... 584 , 600 , 600 , 822 , 822	<code>list itemize-2 (instance)</code>	1440
<code>__kernel_list_label_begin:</code>	<code>list itemize-3 (instance)</code>	1440
..... 557 , 569 , 569 , 869 , 869	<code>list itemize-4 (instance)</code>	1440
<code>__kernel_list_label_end:</code>	<code>list legacy (instance)</code>	1452
..... 566 , 569 , 570 , 883 , 883	<code>list std (template)</code>	75 , 404
keys commands:	<code>\listparindent</code>	379 , 545 , 546 , 1076
<code>\keys_define:mn</code> ... 27 , 375 , 457 , 467	<code>\LItag</code>	860 , 868
<code>\KeyValue</code>	<code>\ltablblockdate</code>	4 , 1526
92 , 1376 , 1377 , 1403 , 1410 , 1427 , 1428	<code>\ltablblockversion</code>	4 , 1526
L		
<code>\labelenumi</code>	1445	
<code>\labelenumii</code>	1447	
<code>\labelenumiii</code>	1449	
<code>\labelenumiv</code>	1451	
<code>\labelitemi</code>	1440	
<code>\labelitemii</code>	1441	
<code>\labelitemiii</code>	1442	
<code>\labelitemiv</code>	1443	
<code>\labelsep</code> ... 86 , 416 , 536 , 538 , 1166 , 1193		
<code>\labelwidth</code> 85 , 415 , 516 , 518 , 530 , 536 , 1097		
<code>\language</code>	1011	
<code>\lastbox</code>	18	
<code>\LBody</code>	888 , 890	
<code>\leavevmode</code>	1015	
<code>\leftmargin</code>	57 , 293 , 344 ,	
345 , 353 , 355 , 1055 , 1056 , 1096 , 1431		
<code>\leftskip</code>	269 , 323	
legacy commands:		
<code>\legacy_if:nTF</code> 227 , 232 , 239 , 240 , 301 , 312 ,	
..... 318 , 329 , 348 , 357 , 366 , 431 , 578 ,		
..... 587 , 598 , 614 , 629 , 643 , 645 , 678 ,		
..... 699 , 707 , 709 , 833 , 892 , 901 , 903 , 1083		
<code>\legacy_if_gset_false:n</code> 230 , 237 , 577 , 580 , 589 , 617 , 647	
<code>\legacy_if_gset_true:n</code> 21 , 250 , 453 , 623	
<code>\legacy_if_set_false:n</code> 188 , 319 , 350 , 576 , 1080	
<code>\legacy_if_set_true:n</code>	314 , 315	
<code>\legacylistsetupcode</code>	11 , 1075 , 1364	
<code>\legacyverbatimsetup</code>	11 , 1009 , 1315	
<code>\let</code>	922 , 923 , 1020 , 1049 , 1081	
<code>\linewidth</code>	344 , 346 , 510 , 512	
<code>list (env.)</code>	1063	
<code>list (objecttype)</code>	24	
<code>\list</code>	1094	
<code>list description (instance)</code>	1456	
<code>list enumerate-1 (instance)</code>	1444	
<code>list enumerate-2 (instance)</code>	1444	
M		
<code>\makelabel</code>	562 , 1081 , 1098	
<code>\MakeLinkTarget</code>	499 , 501 , 1149 , 1173	
mode commands:		
<code>\mode_if_horizontal:TF</code> 233 , 631 , 773 , 778	
<code>\mode_if_inner:TF</code>	774	
<code>\mode_if_vertical:TF</code>	303	
<code>\mode_leave_vertical:</code> ..	229 , 299 , 301	
msg commands:		
<code>\msg_error:nnnn</code>	786	
N		
<code>\newcommand</code>	1027	
<code>\newcounter</code>	221	
<code>\NewDocumentEnvironment</code>	970 , 973	
<code>\newline</code>	541	
<code>\NewTemplateType</code>	24 , 25 , 26 , 27 , 28	
<code>\newtheorem</code>	11 , 1103	
<code>\nobreak</code>	540	
<code>\nobreakspace</code>	1032	
<code>\noexpand</code>	1121	
<code>\noindent</code>	1148 , 1172	
<code>\normalfont</code>	1461	
NOT commands:		
<code>\NOT_IMPLEMENTED</code>	485	
<code>\null</code>	1015	
O		
<code>\obeylines</code>	1021	
object types:		
<code>block</code>	24	
<code>blockenv</code>	24	
<code>item</code>	24	
<code>list</code>	24	
<code>para</code>	24	
<code>off (plug)</code>	20 , 20 , 260	
<code>on (plug)</code>	20 , 20 , 260	
P		
<code>\par</code> 9 , 16 , 234 , 310 , 630 , 632 , 673 , 794 , 1013		

par commands:	
\par_end:	13
par internal commands:	
\l_par_fixed_word_spaces_bool	272
\l__par_start_skip	268
para (objecttype)	24
para center (instance)	1474
para commands:	
\para_end:	12, 22, 37, 334, 338, 771, 771, 794, 795, 796
\g_para_indent_box	581
\para_omit_indent:	582
para justify (instance)	1507
para raggedleft (instance)	1496
para raggedright (instance)	1485
para std (template)	63, 265
para/begin (hook)	29, 30
para/begin	12, 696
\PARALABEL	701, 770, 1164, 1191
\parfillskip	271, 325
\parindent	66, 267, 343, 546, 1419, 1510
\parsep	51, 286, 342, 412, 461, 544, 1375, 1426
\parskip	246, 322, 341, 342, 362, 367, 371, 1420
\partopsep	50, 285, 305, 378, 1374, 1425
\pdffakepace	1032
\penalty	585, 1015, 1018
Plugs:	
off	20, 20, 260
on	20, 20, 260
prg commands:	
\prg_do_nothing:	23, 569, 570, 571, 594, 600, 638, 639, 922, 923, 928, 930
\ProvidesFile	1525
\ProvidesPackage	3
Q	
quotation (env.)	987
quote (env.)	987
R	
\raggedleft	1518
\raggedright	1518
\relax	482, 484, 1059
\RemoveFromHook	696, 753
\renewcommand	1031
\RenewDocumentCommand	605, 1103
\RenewDocumentEnvironment	977, 980, 983, 988, 991, 996, 1002, 1036, 1039, 1042, 1047, 1064, 1093
\RequirePackage	6, 1528
\rightmargin	58, 292, 344, 1077, 1382, 1432
\rightskip	270, 279, 324
\rlap	669
	S
scan commands:	
\scan_stop:	772
\setbox	18
\setcounter	222, 1369
\SetTemplateKeys	153, 278, 297, 401, 425, 427, 492
skip commands:	
\skip_add:Nn	305, 322
\skip_eval:n	367
\skip_horizontal:n	353, 355, 536, 538
\skip_new:N	601, 602, 603
\skip_set:Nn	104, 279, 302
\skip_set_eq:NN	320, 324, 325, 341, 342, 544
\skip_vertical:n	245, 246, 360, 361
\skip_zero:N	323
\l_tmpa_skip	242, 243, 245, 246
socket commands:	
\socket_assign_plug:nn	264, 924, 933, 946, 962
\socket_new:nn	259
\socket_new_plug:nnn	260, 262
\socket_use:n	252
Sockets:	
tagsupport/block-endpe	259
\space	4, 688, 1526
str commands:	
\str_if_eq:nnTF	1107
\string	610
\strut	560
	T
tag commands:	
\tag_if_active:TF	184, 640, 846, 1028, 1464
\tag_mc_begin:n	668, 730, 751, 876, 1153, 1157, 1177, 1181, 1185
\tag_mc_end:	664, 670, 760, 1155, 1159, 1179, 1183, 1187
\tag_struct_begin:n	653, 714, 724, 736, 745, 1150, 1156, 1174, 1180
\tag_struct_end:	672, 691, 762, 766, 816, 1160, 1162, 1184, 1189
tag internal commands:	
\l__tag_block_flattened_level_	
int	17, 155, 157, 162, 217, 650
__tag_check_para_begin_show:nn	729, 750
__tag_check_para_end_show:nn	761
__tag_gincr_para_begin_int:	722, 743
__tag_gincr_para_end_int:	665, 758
__tag_gincr_para_main_begin_	
int:	652, 713, 735

<code>__tag_gincr_para_main_end_int:</code> .	<code>\@enumdepth</code>	1340
.	<code>\@execute@begin@hook</code>	803
<code>\l__tag_L_attr_class_tl</code>	<code>\@flushglue</code>	6,
845, 865, 967, 968, 1084, 1086, 1087	70, 325, 1479, 1480, 1491, 1501, 1514	
<code>\l__tag_L_tag_tl</code> 841, 842, 864, 964, 965	<code>\@ifdefinable</code>	1105
<code>\g__tag_mode_lua_bool</code>	<code>\@ifnextchar</code>	1143
.	<code>\@ifundefined</code>	799, 1128
.	<code>\@ignorefalse</code>	805
.	<code>\@inmatherr</code>	235, 607
.	<code>\@itemdepth</code>	1324
.	<code>\@itemlabel</code>	12, 24, 26,
.	186, 393, 451, 500, 874, 886, 1068, 1085	
.	<code>\@itempenalty</code>	7, 413, 635
.	<code>\@kernel@after@para@after</code>	784
.	<code>\@kernel@after@para@end</code>	777
.	<code>\@kernel@refstepcounter</code>	495, 1141
.	<code>\@labels</code>	29
.	<code>\@latex@error</code>	610, 800
.	<code>\@list...</code>	5
.	<code>\@listctr</code>	24, 26, 187, 393,
.	435, 442, 445, 495, 499, 501, 502, 1079	
.	<code>\@listdepth</code>	5, 16, 131
.	<code>\@listi</code>	5
.	<code>\@listii</code>	5
.	<code>\@listvi</code>	5
.	<code>\@makeother</code>	1020
.	<code>\@mklab</code>	1081
.	<code>\@namedef</code>	1135, 1136
.	<code>\@newctr</code>	1118
.	<code>\@nmbrrlist</code>	25
.	<code>\@nmbrrlisttrue</code>	441
.	<code>\@nocounterr</code>	1129
.	<code>\@noitemerr</code>	26, 40, 232, 318, 333, 598
.	<code>\@noligs</code>	1021
.	<code>\@normalcr</code>	6, 73, 1516
.	<code>\@opargbegintheorem</code>	48, 1145
.	<code>\@outerparskip</code>	246, 341, 362, 367
.	<code>\@restorepar</code>	11
.	<code>\@rightskip</code>	279, 324
.	<code>\@setpar</code>	327
.	<code>\@setupverbinvisible</code>	
.	11, 998, 1027
.	<code>\@setupverbvisible</code>	1004
.	<code>\@sxverbatim</code>	1005
.	<code>\@tempwafalse</code>	1012
.	<code>\@tempwatru</code>	1017
.	<code>\@thm</code>	11, 48, 1135, 1139
.	<code>\@thmcounter</code>	1114, 1123
.	<code>\@thmcountersep</code>	1122
.	<code>\@toodeep</code>	170, 177
.	<code>\@topsep</code>	31, 59
.	<code>\@topsepadd</code>	31, 59
.	<code>\@totalleftmargin</code>	44, 345, 346
.	<code>\@vobeyspaces</code>	998, 1004
<code>\tagmcbegin</code>		880
<code>\tagmchend</code>		884
<code>\tagpdfparaOff</code>		673, 1147, 1171
<code>\tagpdfparaOn</code>		673, 1163, 1190
<code>\tagpdfsetup</code>		848, 1466
<code>\tagstructbegin</code>		829, 862, 868, 879, 888
<code>\tagstructend</code>		
.		837, 839, 887, 895, 898, 906, 909, 911
<code>tagssupport/block-endpe (socket)</code>		259
<code>\tagtool</code>		1024
templates:		
<code>block display</code>		46, 281
<code>blockenv display</code>		29, 133
<code>item std</code>		89, 467
<code>list std</code>		75, 404
<code>para std</code>		63, 265
\TeX and $\LaTeX 2_\epsilon$ commands:		
<code>\@@par</code>		1015, 1018
<code>\@beginparpenalty</code>		6, 290, 369
<code>\@begintheorem</code>		11, 48, 1145
<code>\@beginthorem</code>		48
<code>\@centercr</code>		1049, 1483, 1494, 1505
<code>\@clubpenalty</code>		12, 593
<code>\@currentenv</code>		235, 801
<code>\@currentvline</code>		802
<code>\@definecounter</code>		1109
<code>\@doendpe</code>		11, 12, 8
<code>\@eha</code>		800
<code>\@ehc</code>		611
<code>\@endparpenalty</code>		6, 248, 291
<code>\@endpefalse</code>		14, 20, 263, 818
<code>\@endpetrue</code>		8, 261
<code>\@endtheorem</code>		1136, 1196

<code>\@xobeysp</code>	1032	<code>\textbf</code>	48
<code>\@xthm</code>	1143	<code>\topsep</code>	6, 59
<code>\@xverbatim</code>	999	<code>\trivlist</code>	61
<code>\@ythm</code>	1143	<code>\usecounter</code>	25
<code>\arabic</code>	7	<code>\UseInstance</code>	32
<code>\begin</code>	12, 37	<code>\verbatim@font</code>	1021
<code>\bfseries</code>	48	<code>\z@</code>	18
<code>\bibitem</code>	40	<code>\z@skip</code>	1481, 1490, 1492, 1502, 1503, 1512, 1513
<code>\g_block_nesting_depth_int</code>	61	tex commands:	
<code>\c@maxblocklevels</code>	12, 176, 221	<code>\tex_hskip:D</code>	780
<code>\end</code>	12	<code>\tex_lastnodetype:D</code>	779
<code>\endblockenv</code>	43	<code>\tex_lastskip:D</code>	104
<code>\everypar</code>	32	<code>\tex_par:D</code>	782, 791
<code>\hyper@nopatch@thm</code>	1144	<code>\tex_parshape:D</code>	346
<code>\if@endpe</code>	810	<code>\tex_unskip:D</code>	106, 775
<code>\if@tempswa</code>	1014	<code>\textbf</code>	300
<code>\ignorespaces</code>	5, 18, 50	<code>\the</code>	1022
<code>\iitem</code>	60	<code>\tiny</code>	669
<code>\item</code>	12, 19, 20, 22–29, 31, 40, 42, 43, 61, 62	tl commands:	
<code>\itemindent</code>	45, 60	<code>\c_novalue_tl</code>	27, 28, 490
<code>\itemsep</code>	6	<code>\tl_clear:N</code>	186, 187
<code>\l@nohyphenation</code>	1011	<code>\tl_gset:Nn</code>	1112, 1119, 1131
<code>\labelindent</code>	60	<code>\tl_if_blank:nTF</code>	298, 495
<code>\labelsep</code>	7, 60	<code>\tl_if_empty:NTF</code>	166, 196, 202, 205, 209, 429, 449, 934, 947, 963, 966, 1085
<code>\labelwidth</code>	7, 27, 29, 60	<code>\tl_if_empty:nTF</code>	153, 278, 297, 397, 422, 492, 874, 886
<code>\leftmargin</code>	6, 60	<code>\tl_if_eq:NnTF</code>	255
<code>\leftskip</code>	44	<code>\tl_if_novalue:nTF</code>	107, 493, 620
<code>\legacylistsetupcode</code>	46	<code>\tl_new:N</code>	393, 394, 549, 841, 844, 951, 1091, 1139
<code>\list</code>	47, 61	<code>\tl_set:Nn</code>	482, 483, 484, 842, 845, 935, 948, 964, 967, 1023, 1024, 1066, 1068, 1079, 1084, 1086, 1087, 1142
<code>\list(romannumeral)</code>	54, 56	<code>\tl_set_eq:NN</code>	442, 451, 490, 936, 949, 965, 968
<code>\listparindent</code>	21, 29	<code>\topsep</code>	49, 284, 302, 377, 462, 465, 1373, 1424
<code>\makelabel</code>	7, 30, 46, 57, 62	<code>\trivlist (env.)</code>	1092
<code>\newline</code>	27	<code>\typeout</code>	126
<code>\newtheorem</code>	47		
<code>\noitemerr</code>	19		
<code>\on@line</code>	224, 575, 644, 666, 684, 689, 708, 723, 744, 759, 802, 832, 836, 896, 907		
<code>\par</code>	10, 12, 13, 20, 22, 26, 32, 34, 36, 41, 42, 60		
<code>\par@deathcycles</code>	326, 331, 332		
<code>\parindent</code>	6, 29, 56		
<code>\parsep</code>	6		
<code>\parskip</code>	23, 56		
<code>\partopsep</code>	6, 59		
<code>\pdffakepace</code>	45		
<code>\ref</code>	60		
<code>\refstepcounter</code>	48		
<code>\reserved@a</code>	800, 801, 808		
<code>\rightmargin</code>	6		
<code>\SetTemplateKeys</code>	24		
<code>\spacefactor</code>	26		
<code>\strut</code>	8, 30		

U

		V	
1003, 1037, 1040, 1043, 1050, 1071, 1146, 1170, 1518, 1519, 1520, 1521	\UseName	verbatim (env.)	<u>995</u>
55,		verbatim* (env.)	<u>995</u>
56, 83, 1132, 1379, 1380, 1429, 1430		verse (env.)	<u>1046</u>