

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

L^AT_EX Project*

v1.0i 2026-07-29

Abstract

Contents

1	Introduction	3
2	Template types and templates for blocks and lists	4
2.1	Template types	4
2.1.1	The template type ‘blockenv’	4
2.1.2	The template type ‘block’	5
2.1.3	The template type ‘para’	5
2.1.4	The template type ‘list’	5
2.1.5	The template type ‘captionedtext’	6
2.1.6	The template type ‘item’	6
2.1.7	The template type ‘thmstyle’	6
2.2	Templates	7
2.2.1	The blockenv template ‘std’	7
2.2.2	The block template ‘std’	9
2.2.3	The para template ‘std’	10
2.2.4	The list template ‘std’	11
2.2.5	The item template ‘std’	12
2.2.6	The captionedtext template ‘thmlike’	12
2.2.7	The captionedtext template ‘proof’	13
2.2.8	The thmstyle template ‘std’	13

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

3	Declaring standard display block environments and their instances	15
3.1	The <code>display</code> and <code>displayflattened</code> environments	16
3.1.1	Their <code>blockenv</code> instances	16
3.1.2	Their <code>block</code> instances	17
3.2	The <code>center</code> , <code>flushleft</code> , and <code>flushright</code> environments	17
3.2.1	Their <code>blockenv</code> instances	18
3.2.2	Their <code>block</code> instances	19
3.2.3	Their <code>para</code> instances	19
3.3	The <code>quote</code> and <code>quotation</code> environments	19
3.3.1	Their <code>blockenv</code> instances	19
3.3.2	Their <code>block</code> instances	20
3.4	The <code>verse</code> environment	20
3.4.1	Their <code>blockenv</code> instances	20
3.5	The <code>verbatim</code> , <code>verbatim*</code> and <code>alltt</code> environments	21
3.5.1	Their <code>blockenv</code> instances	21
3.5.2	Their <code>block</code> instances	23
3.6	The standard lists: <code>itemize</code> , <code>enumerate</code> , and <code>description</code>	24
3.6.1	Their <code>blockenv</code> instances	24
3.6.2	Their <code>block</code> instances	25
3.6.3	Their <code>list</code> instances	26
3.6.4	Their <code>item</code> instances	27
3.7	The legacy <code>list</code> and <code>trivlist</code> environments	27
3.7.1	Its <code>blockenv</code> instance	28
3.7.2	Its <code>list</code> instance	28
3.8	Theorem-like environments declared through <code>\newtheorem</code>	28
3.8.1	The <code>blockenv</code> instances they use	29
3.8.2	The <code>captionedtext</code> instances they use	29
3.8.3	The <code>thmstyle</code> instances they use	30
3.8.4	The <code>block</code> instances they use	31
3.9	The <code>proof</code> environment (from <code>amsthm</code>)	32
3.9.1	Block instances for the proofs	33
4	Declaring <code>para</code> instances	33
5	Advice on adjusting the layout of standard block environments	35
6	Tagging support	35
6.1	Paragraph tags	35
6.1.1	Tagging recipes	37
7	Tracing and debugging	38
8	New and redefined kernel command	39

9	The Implementation	40
9.0.1	Augmented <code>\SetKnownTemplateKeys</code>	40
9.0.2	Tracing templates and instances	41
9.0.3	Handling <code>\par</code> after the end of the list	41
9.0.4	Other useful <code>expl3</code> commands	44
9.1	Tracing and debugging interfaces	45
9.2	Template types and template interfaces	46
9.3	Implementation of templates	49
9.3.1	Some notes on the $\text{\LaTeX} 2_{\varepsilon}$ legacy switches	49
9.3.1.1	Original usage:	49
9.3.1.2	Repurpose:	50
9.3.2	Implementation of <code>blockenv</code> templates	51
9.3.3	Implementation of <code>para</code> templates	58
9.3.4	Implementation of <code>block</code> templates	59
9.3.5	Implementation of <code>list</code> templates	64
9.3.6	Implementation of <code>item</code> templates	68
9.3.7	Implementation of <code>captionedtext</code> and <code>thmstyle</code> templates	76
9.4	Tagging support commands	82
9.4.1	List tags	86
9.4.2	Tagging recipes	88
10	Support code for document-level block environments	91
10.1	Verbatim-like environments	91
10.1.1	Helper commands for <code>verbatim</code> and <code>verbatim*</code>	91
10.1.2	Helper commands for <code>alltt</code> and <code>alltt*</code>	92
10.1.3	Helper command for legacy <code>list</code> environment	93
10.2	Theorem-like environments	94
10.2.1	Declarations for theorem-like environments	94
10.2.2	Supporting QED in proofs	100
11	Support for other packages and classes	102
11.1	Replacement for <code>alltt</code>	102
11.2	Replacement for <code>amsthm</code>	102
11.3	Support for <code>amsart</code> and <code>amsbook</code> classes	102
11.4	Support for the <code>enumitem</code> interfaces	104
11.5	Support for the <code>doc</code> package	104
	Index	105

1 Introduction

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

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

The approach taking here is therefore to offer separate template 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 we have the template type `blockenv` that ties them together as necessary when we build document level environments.

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

Instead of a `list` instance to handle the inner structure of the environment one can use an instance of the type `captionedtext` to produce a display environment with an associated heading/caption, such as a theorem-like environment or a proof environment. Further possibilities (not yet implemented) are templates for producing boxed text or formal quotes like those produced by the `csquotes` package.

2 Template types and templates for blocks and lists

2.1 Template types

2.1.1 The template type ‘`blockenv`’

Arg: 1 key/value list to alter the default parameters of the template instances used by the particular `blockenv` environment

Arg: 2 Boolean to suppress a number in case this environment normally produces a numbered caption

Arg: 3 Caption/heading text in case this environment supports a caption (most don’t), otherwise `\NoValue`

Arg: 4 Sub-caption/heading text in case this environment supports a caption (most don’t), otherwise `\NoValue`

Semantics:

This template 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.

Arguments 2–4 are passed to the instance handling the inner structure, e.g., `list` or `captionedtext` which may or may not make use of it.

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 template type defines how it appears in a tagged PDF document, what tag names are used, how they are role-mapped and whether it adds additional attributes, etc.

2.1.2 The template 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.

2.1.3 The template type ‘para’

Arg: 1 key/value list to alter the default paragraph 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`.

2.1.4 The template type ‘list’

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

Arg: 2 Boolean to suppress a number in case this list environment also produces a numbered heading/caption

Arg: 3 Caption/heading text in case this environment supports a caption (lists normally don’t), otherwise `\NoValue`

Arg: 4 Sub-caption/heading text in case this environment supports a caption, otherwise `\NoValue`

Semantics:

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

Standard $\text{\LaTeX} 2_{\epsilon}$ lists have no heading/caption, so arguments 2–4 are ignored in the standard `list` template. But special lists, such as a list of ingredients for a cookbook, might so there might be other templates that make use of them in the future.

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

2.1.5 The template type ‘captionedtext’

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

Arg: 2 Boolean to suppress a number in case this environment also produces a numbered heading/caption

Arg: 3 Caption/heading text for this text block; if not given then `\NoValue`

Arg: 4 Sub-caption/heading text in case this environment supports a caption, otherwise `\NoValue`

Semantics:

Produces a text block with an associated caption/heading, e.g., a theorem-like environment. There may not be a user-supplied caption text—the caption may consist of a fixed text only like “Lemma”.

Handles the aspects related to the caption design and typically supports keys for adjusting the layout of the body text, e.g., its font, etc.

Note that this template type does not cover block-related aspects, e.g., the dimensions of the display block are handled there.

2.1.6 The template 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.

2.1.7 The template type ‘thmstyle’

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

Arg: 2 Boolean to suppress a number in case this environment also produces a numbered heading/caption

Arg: 3 Caption/heading text for this text block; if not given then `\NoValue`

Arg: 4 Sub-caption/heading text in case this environment supports a caption, otherwise `\NoValue`

Semantics:

A sub-type used as part of `captionedtext` when producing theorem-like environments. It does the bulk of the work and sets up most of the formatting. It has been separated out because many theorem-like environments use the same theorem layout and only differ in the fixed caption text they generate.

Not all templates of type `captionedtext` use `thmstyle` as an inner instance, e.g., proofs are implemented with a template that does everything necessary directly.

2.2 Templates

2.2.1 The `blockenv` template ‘std’

Attributes:

name (*tokenlist*) Name of the environment used in tracing and error messages.

tag-name (*tokenlist*) Name of the tag used for the block inside the PDF. If not explicitly given the name is defined by the `tagging-recipe`. Note that in case of `tagging-recipe=basic` no tag for the block is produced, so any key settings are ignored.
Default: `<empty>`

tag-attr-class (*tokenlist*) An explicit tag class attribute. Default: `<empty>`

tagging-recipe (*tokenlist*) Defines the way tagging is done. Currently the values `basic`, `standard`, `list`, and `standalone` are supported. The latter implies that the key `standalone` is set to true
Default: `standard`

standalone (*boolean*) If true, surrounds the environment implicitly with `\par` commands. Forced to true when `tagging-recipe` is `standalone`.
Default: `false`

transparent-level (*boolean*) Is this `blockenv` transparent for any blocks nested inside?
Default: `false`

early-legacy-code (*tokenlist*) Legacy setup code. This is executed after legacy defaults (from `\@listi`, `\@listii`, etc.) are used but before the block instance is called. At this point we may still be in horizontal mode! This means settings that alter paragraph parameters such as `\baselineskip` should **not** be used because they would affect preceding text! Use `late-legacy-code` for that instead in a display environment.
Default: `<empty>`

late-legacy-code (*tokenlist*) Legacy setup code. This is called after the `block` and `para` instances are processed but before the inner instance is called. At this point we are in vertical mode in a display environment and something like `\small` is not affecting preceding text.
Default: `<empty>`

block-instance (*tokenlist*) Part of the name of the `block` instance that is called. The full name has a `-<level>` appended.
Default: `std-display`

para-instance (*tokenlist*) Paragraph settings to use within the environment. If `<empty>` then the current (outer) values are retained. However, the **inner-instance** template might reset/overwrite some of the **para** values, e.g., **list** makes use of `\listparindent` to explicitly set the paragraph indentation for compatibility.
Default: `<empty>`

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*) Template type of the inner instance. Currently supported types are **list** and **captionedtext**.
Default: `<empty>`

inner-instance (*tokenlist*) Name of the inner instance (if any). If there is an **inner-level-counter** then the instance name gets `-<counter value>` appended.
Default: `<empty>`

tagging-suppress-paras (*boolean*) *describe* Default: `false`

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

Semantics & Comments: The **blockenv** type handles the overall setup for the document-level environments.

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_EX 2_ε name).

The internal block nesting level is stored (for historical reasons) in the `\@listdepth` counter and incremented by each block by one. The starting value at top-level (outside any block) is zero. A block environment with **transparent-level=true** also increments the level before it evaluates and sets its parameters but then decrements it again, just before it starts processing its body.

The template first checks that the block is not too deeply nested.

After the level was increased then corresponding `\@list...` macro to update the legacy defaults is called.

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

Afterwards it calls the appropriate **block** instance based on **block-instance** and current level, e.g., **std-display-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., `std-display-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 `transparent-level` is set to `true` then such an environment alters the nesting level only temporarily (while processing the `blockenv` template) and you can therefore nest those environments as often as you like (a typical example would be `flushleft` anywhere in the nesting hierarchy) as long as the level isn't already at `maxblocklevels`).

2.2.2 The block template ‘std’

Attributes:

- begin-vspace** (*skip*) Vertical space before the block. Default: `\topsep`
- begin-extra-vspace** (*skip*) Extra vertical space before the block if the block forms its own paragraph. Default: `\partopsep`
- begin-unchained-vspace** (*skip*) Vertical space before the block to use if this is a nested block, both blocks have items or captions, and these should not be chained; see description below. Default: `.5\topsep`
- para-vspace** (*skip*) The default for ordinary blocks is to use the `\parskip` from the outer galley. In lists and some other special blocks this is then changed. Default: `\parskip`
- end-vspace** (*skip*) Vertical space after the block. Default: value from **begin-vspace**
- end-extra-vspace** (*skip*) Extra vertical space after the block if the block forms its own paragraph. Default: value from **begin-extra-vspace**
- item-vspace** (*skip*) The space in front of an item if the block is a list; if not, the setting has no effect. Default: `\itemsep`
- begin-penalty** (*integer*) Penalty for breaking before the block. Default: `\@beginparpenalty`
- end-penalty** (*integer*) Penalty for breaking after the block. Default: `\@endparpenalty`
- item-penalty** (*integer*) Penalty for breaking before an item in the list (except the first). Default: `\@itempenalty`
- left-margin** (*length*) Space on the left of the block. Default: `\leftmargin`
- right-margin** (*length*) Space on the right of the block. Default: `\rightmargin`
- para-indent** (*length*) Paragraph indentation for paragraphs within the block. Default: `0pt`

Semantics & Comments: Sets up the main block parameters, e.g. its spacing before and after and the indentation on either side.

It also sets up some parameter defaults for the inner level, e.g., `item-penalty`, `item-vspace` and `para-indent`, which may get overwritten by inner instances that are called.

The vertical spacing before the block covers four different use cases: If there is a caption or an item waiting to be placed, and this item allows for “chaining”, and the new block also wants to place an item then no space is added (spacing was already added by the outer block). Instead, the items are chained and placed that the start of the block, i.e., producing a layout like the two nested `itemize` environments here:

- – A second-level item
 - Another ...
 More text for the first-level item
- Another first-level item

In that case there is also no vertical space after the block. If the items should not be chained (as specified by the setup of the outer block), then one gets a result like this one (using `itemize` environments inside `description` with different treatment of individual description `\items`):

An normal label • A second-level item

- Another ...

More text for the first-level item

An unchained label

- A second-level item
- Another ...

More text for the first-level item

A normal label Another first-level item

If “unchaining” happens, as in the second item, then vertical spacing with the value of `begin-unchained-vspace` is used and at the end you get `end-vertical-space`.

Otherwise, if there is no item or caption waiting to be placed you get a vertical space of `begin-vspace` before the block and if the block is its own paragraph you additionally get `begin-extra-vspace` added to this.

Note that $\text{\LaTeX} 2_{\epsilon}$ always chained the list items, so the ability to prohibit this is a new functionality.

2.2.3 The para template ‘std’

Attributes:

para-indent (*length*) Default: `\parindent`

begin-hspace (*skip*) Horizontal skip added just in front of the indentation box if non-zero
 Default: 0pt

left-hspace (<i>skip</i>)	Default: 0pt
right-hspace (<i>skip</i>)	Default: 0pt
end-hspace (<i>skip</i>)	Default: \@flushglue
fixed-word-spaces (<i>boolean</i>)	Default: false
final-hyphen-demerits (<i>integer</i>)	Default: 5000
newline-cmd (<i>function(0)</i>) This defines the meaning of \\	Default: \@normalcr
para-attr-class (<i>tokenlist</i>)	Default: justify

Semantics & Comments: The **begin-hspace** (normally 0pt) is the counterpart of **end-hspace** (which is normally 0pt plus 1fil). It can be useful in special paragraph shapes. The skip is only inserted into the paragraph if it is non-zero. If it is made non-zero then paragraphs are always at least one line including a construct like `\noindent\par!`

The **para-attr-class** takes one of the attributes **justify**, **center**, **raggedright**, **raggedleft**. They are declared in `latex-lab-namespaces`.

TODO: to be further documented

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.	Default: <code><empty></code>
item-label (<i>tokenlist</i>) Label “string” for a fixed label or as generated from the current counter value.	Default: <code><empty></code>
start (<i>integer</i>) Start value for the counter if the list is numbered, otherwise irrelevant.	Default: 1
resume (<i>boolean</i>) Should a numbered list be resumed from the last instance? If true value of start is ignored.	Default: false
item-instance (<i>instance</i>) Instance of type item to be used to format the label string.	Default: basic
item-vspace (<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-penalty (<i>integer</i>) Penalty for breaking before an item (except the first). If not specified the value specified in the block template instance is used.	
item-indent (<i>length</i>) Horizontal displacement of the item.	Default: 0pt
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: false

Semantics & Comments: Sets up handling of list material, e.g., numbering (if any), layout of items and list elements, and tagging, if requested.

2.2.5 The `item` template ‘std’

Attributes:

counter-label (<i>function1</i>) <i>unused</i> .	Default: <code>\arabic{#1}</code>
counter-ref (<i>function1</i>) <i>unused</i> .	Default: value from <code>counter-label</code>
label-ref (<i>function1</i>) <i>unused</i> .	Default: <code>#1</code>
label-autoref (<i>function1</i>) <i>unused</i> .	Default: <code>item #1</code>
label-format (<i>function1</i>) Formatting of the label, questionable the way it is used. Default: <code>#1</code>	
label-strut (<i>boolean</i>) Add a <code>\strut</code> to the label?	Default: <code>false</code>
label-align (<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>
label-placement (<i>choice</i>) Placement of the label in relation to a directly following label (of a following inner list). Supported values are <code>chained</code> , <code>unchained</code> , and <code>standalone</code> .	Default: <code>chained</code>
label-boxed (<i>boolean</i>) Should the label be boxed?	Default: <code>true</code>
next-line (<i>boolean</i>)	Default: <code>false</code>
text-font (<i>tokenlist</i>) <i>unused</i> .	
compatibility (<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!

2.2.6 The `captionedtext` template ‘thmlike’

Attributes:

counter (<i>tokenlist</i>) Counter name to be used if the caption is numbered, otherwise empty.	Default: <code>\empty</code>
title (<i>tokenlist</i>) Fixed part of the caption, e.g., a theorem-like environment may want to specify “Lemma” here.	Default: <code>\empty</code>
style (<i>instance</i>) Instance of type <code>thmstyle</code> that actually implements the theorem-like environment.	Default: <code>plain</code>

fix

Semantics & Comments: The template combines the fixed `title` and a number (if present) with the caption text as specified on the document element, if one is given, e.g., “Theorem 1. (Fermat)”. See also the `proof` template, which handles this differently.

The bulk of the work is then outsourced to an instance of type `thmstyle`. As many such theorem-like environments share the same layout and only differ in the first caption string they use, there is this split for convenience.

2.2.7 The `captionedtext` template ‘proof’

Attributes:

title (*tokenlist*) Heading for the environment unless overwritten on document level.
The default value, i.e., `\proofname` resolves to “Proof” unless changed
Default: `\proofname`

punct (*tokenlist*) Punctuation following the heading. Default: `.`

caption-placement (*choice*) Supported values `chained`, `unchained`, and `standalone`
Default: `unchained`

caption-unbreakable (*boolean*) Can this caption have (implicit or explicit) line breaks?
Default: `false`

before-hspace (*skip*) Horizontal displacement of the heading. Default: `0pt`

after-hspace (*skip*) Space following the heading, only relevant if text follows on the same line. Default: `5pt`

caption-decls (*tokenlist*) Declarations that are applied to the whole caption, e.g., some font settings. Default: `\empty`

title-decls (*tokenlist*) Declarations that are applied only to the caption title.
Default: `\empty`

punct-decls (*tokenlist*) Declarations that are applied only to the caption punctuation.
Default: `\empty`

title-format (*function1*) Formatting applied to the `title` value. Default: `#1`

punct-format (*function1*) Formatting applied to the `punct` value. Default: `#1`

body-decls (*tokenlist*) Declarations that are applied to body of the environment, e.g., font settings. Default: `\empty`

Semantics & Comments: The “unnumbered?” argument (`#2`) is ignored, as proofs aren’t numbered. The template makes use of the caption argument (`#3`) but in contrast to theorem-like environments this template replaces the `title` key value with the content of this argument (if not `\NoValue`).

Typically there is only one layout for proofs so that there is no need to split the formatting over two templates as done for theorem-like environment. That’s the reason why the template has several layout customization parameters.

2.2.8 The `thmstyle` template ‘std’

Attributes:

- numbered** (*boolean*) Is this kind of environment numbered? Default: `true`
- separator** (*tokenlist*) Separation to be applied between elements of the heading, typically a space command of some sort.
Default: `\_`
- punct** (*tokenlist*) Punctuation following the heading. Default: `.`
- caption-placement** (*choice*) Supported values `chained`, `unchained`, and `standalone`
Default: `unchained`
- caption-unbreakable** (*boolean*) Can this caption have (implicit or explicit) line breaks?
Default: `false`
- before-hspace** (*skip*) Horizontal displacement of the heading. Default: `0pt`
- after-hspace** (*skip*) Space following the heading, only relevant if text follows on the same line. Default: `5pt`
- order** (*commalist*) Order of elements in the environment caption/heading. Supported values are `title`, `number`, `punct`, `separator`, and `note`.
Default: `title,separator,number,separator,note,punct`
- title-decls** (*tokenlist*) Declarations that are applied only to the caption title.
Default: `\empty`
- number-decls** (*tokenlist*) Declarations that are applied only to the caption number.
Default: `\empty`
- punct-decls** (*tokenlist*) Declarations that are applied only to the caption punctuation.
Default: `\empty`
- note-decls** (*tokenlist*) Declarations that are applied only to the caption note.
Default: `\empty`
- title-format** (*function1*) Formatting applied to the `title` value. Default: `#1`
- number-format** (*function1*) Formatting applied to the `number` value. Default: `#1`
- punct-format** (*tokenlist*) Formatting applied to the `punct` value. Default: `#1`
- note-format** (*function1*) Formatting applied to the `note` value. Default: `(#1)`
- body-decls** (*tokenlist*) Declarations that are applied to body of the environment, e.g., font settings. Default: `\empty`

Semantics & Comments: Numbering of the environment is suppressed unconditionally if the `numbered` is set to `false`. Otherwise the environment is numbered except when `#2` is `\BooleanTrue`, i.e., if the star form of the environment was used.

The caption of the environment can consist of a title, a number, a punctuation, some separators (typically spaces) and a note. Their order is defined by the key `order`. If a component is specified but has no value, e.g., no note or the numbering suppressed on an individual environment, then the component and any preceding separators are ignored.

Spaces between elements are uniform (as one can only specify a `separator` in the `order` key), but it is possible to use this several times in a row and adjust the `separator` key accordingly.

Alternatively, one can omit using `separator` in the `order` key and instead put all necessary spacing into the individual `...-format` keys. This approach is used, for example, if a theorem style is set up with `\newtheoremstyle` and its ninth argument contains a declaration such as

```
\thmname{#1}\thmnumber{ #2}\thmnote{ (#3)}
```

This is then translated to

```
order          = {title,number,punct,note} ,
title-format   = {#1} ,
number-format  = { #2} ,
note-format    = { (#3)} ,
```

when `\newtheoremstyle` sets up a new instance. The downside of this approach is that `\swapnumbers` would not work with such styles (because it would be necessary to transfer the space inside value for the `number-format` key to the value of `title-format`).

If you look closely you also see that in the `order` key a `punct` was added in the list even though it was not present originally. This is the way `\newtheoremstyle` worked and so we mimic that.

3 Declaring standard display block environments and their instances

Historically the L^AT_EX kernel has defined a number of block environments directly, e.g., `center` or lists like `itemize`, but left others to be set up by document classes. For now we declare all of them here, but in the future, some (or even all) might get moved to new class files.

`\SimpleBlockEnv` Most of the standard block environments have no need for a caption, so to simplify the setup we have added the command `\SimpleBlockEnv` that hides the arguments 2–4 required by a `blockenv` instance and gives them suitable values, i.e., `\BooleanFalse\NoValue\Novalue`. This way, a document level definition for the `center` environment will look like this:

```
\NewDocumentEnvironment{center} { !0{} }
{ \SimpleBlockEnv{center}{#1} } { \BlockEnvEnd }
```

instead of the more verbose

```

\NewDocumentEnvironment{center}{!0}{ }
{ \UseInstance{blockenv}{center}{#1} \BooleanFalse \NoValue \NoValue }
{ \BlockEnvEnd }

```

We use `!0{}` for the optional argument so that it is only recognized if it immediately follows `\begin{center}` without any spaces to avoid that a `[` at the start of the body text is misinterpreted as the opening bracket of the optional argument. This is only done for environments where this could be a problem.

This will then call the `center` instance of type `blockenv` that handles the rest.

`\BlockEnv` For the environments that make use of the other arguments, we offer `\BlockEnv` as syntactic sugar so that most environment declarations look similar. And we use `\BlockEnvEnd` in both cases to finish off.

```

1 (*class-code)

```

In the following sections we provide for all block environments the top-level definition and all instances that are used by it. Instances of type `block` are often reused across the environments, in which case we just provide cross-references. Note that this is a design decision, different classes may want to have adjusted settings for individual environments, in which case they would provide special `block` instances instead of reusing, say, the `std-display-⟨level⟩` instances.

3.1 The `display` and `displayflattened` environments

`displayblock (env.)` There are two basic block environments (`displayblock` and `displayblockflattened`) which are similar to $\text{\LaTeX 2}_{\epsilon}$'s `trivlist` except that they aren't degenerated lists and thus have no hidden `\item` inside.

```

2 \NewDocumentEnvironment{displayblock}{!0}{ }
3 { \SimpleBlockEnv{displayblock}{#1} } { \BlockEnvEnd }

4 \NewDocumentEnvironment{displayblockflattened}{!0}{ }
5 { \SimpleBlockEnv{displayblockflattened}{#1} } { \BlockEnvEnd }

```

3.1.1 Their `blockenv` instances

`blockenv displayblock (inst.)` This is like $\text{\LaTeX 2}_{\epsilon}$'s `trivlist`, i.e., it produces a vertical block with default setting, but doesn't put a list inside but uses a `<Div>` structure.

We list all keys, those with default values, commented out.

```

6 \DeclareInstance{blockenv}{displayblock}{std}
7 {
8   name                = displayblock
9   % ,tagging-recipe    = standard
10  % ,tag-name          =
11  % ,tag-attr-class    =
12  % ,transparent-level = true
13  % ,early-legacy-code =
14  % ,late-legacy-code  =
15  % ,block-instance    = std-display
16  % ,para-instance     =
17  % ,tagging-suppress-paras = false
18  % ,inner-instance    =
19  % ,inner-instance-type = % not relevant as there is no inner instance
20  % ,inner-level-counter = % not relevant as there is no inner instance

```

```

21 % ,max-inner-levels      = 4          % not relevant as there is no inner instance
22 % ,final-code             = \ignorespaces
23 }

```

The block uses the instance `std-display` which is shown below.

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

```

24 \DeclareInstance{blockenv}{displayblockflattened}{std}
25 {
26   name                = displayblockflattened
27   ,tagging-recipe      = basic
28   ,tagging-suppress-paras = true
29   ,transparent-level   = true
30 }

```

3.1.2 Their block instances

We provide 6 nesting levels (as in $\text{\LaTeX} 2_{\epsilon}$). If you want to provide more you need to change the `maxblocklevels` counter, offer further `std-display-⟨level⟩` 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).

```

31 \setcounter{maxblocklevels}{6}

```

`block std-display-1 (inst.)` We show all keys here for reference, with those using their default values commented out:

```

block std-display-2 (inst.) 32 \DeclareInstance{block}{std-display-1}{std}
block std-display-3 (inst.) 33 {
block std-display-4 (inst.) 34 % ,begin-vspace      = \topsep
block std-display-5 (inst.) 35 % ,begin-extra-vspace = \partopsep
block std-display-6 (inst.) 36 % ,para-vspace       = \parskip
37 % ,end-vspace        = \KeyValue{begin-vspace}
38 % ,end-extra-vspace  = \KeyValue{begin-extra-vspace}
39 % ,item-vspace       = \itemsep
40 % ,begin-penalty     = \UseName{@beginparpenalty}
41 % ,end-penalty       = \UseName{@endparpenalty}
42 % ,left-margin       = Opt
43 % ,right-margin      = \rightmargin
44 % ,para-indent       = Opt
45 }
46 \DeclareInstanceCopy{block}{std-display-2}{std-display-1}
47 \DeclareInstanceCopy{block}{std-display-3}{std-display-1}
48 \DeclareInstanceCopy{block}{std-display-4}{std-display-1}
49 \DeclareInstanceCopy{block}{std-display-5}{std-display-1}
50 \DeclareInstanceCopy{block}{std-display-6}{std-display-1}

```

3.2 The center, flushleft, and flushright environments

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

`center (env.)` For now we redeclare various document environments as late as possible in order to make `flushleft (env.)` tagging work, even if classes have changed the definitions. Of course, this means that `flushright (env.)` such changes get lost.

```

51 \AddToHook{begindocument/before}[./legacy-core]{
52   \RenewDocumentEnvironment{center} { !0{} }
53   { \SimpleBlockEnv{center}{#1} } { \BlockEnvEnd }
54
55   \RenewDocumentEnvironment{flushright} { !0{} }
56   { \SimpleBlockEnv{flushright}{#1} } { \BlockEnvEnd }
57
58   \RenewDocumentEnvironment{flushleft} { !0{} }
59   { \SimpleBlockEnv{flushleft}{#1} } { \BlockEnvEnd }
60 }

```

3.2.1 Their blockenv instances

`blockenv center (inst.)` The `center` environment is defined through the `blockenv` instance `center` which makes use of the `block` instance `std-display-⟨level⟩` and the `para` instance `center`. The block nesting level is not incremented. With respect to tagging, text separated by `\par` commands (or empty lines) inside the environment is not tagged as separate paragraphs, i.e., the whole environment is considered to be part of an outer paragraph.

```

59 \DeclareInstance{blockenv}{center}{std}
60 {
61   name = center
62   ,tag-name =
63   ,tag-attr-class =
64   ,tagging-recipe = basic
65   ,tagging-suppress-paras = true
66   ,inner-level-counter =
67   ,transparent-level = true
68   ,early-legacy-code =
69   ,late-legacy-code =
70   ,block-instance = std-display
71   ,para-instance = center
72   ,inner-instance =
73 }

```

`blockenv flushleft (inst.)` Same as `center` except that we use the `para` instance `raggedright`.

```

74 %\DeclareInstance{blockenv}{flushleft}{std}
75 %{
76   % name = flushleft
77   % ,tag-name =
78   % ,tag-attr-class =
79   % ,tagging-recipe = basic
80   % ,tagging-suppress-paras = true
81   % ,inner-level-counter =
82   % ,transparent-level = true
83   % ,early-legacy-code =
84   % ,late-legacy-code =
85   % ,block-instance = std-display
86   % ,para-instance = raggedright
87   % ,inner-instance =
88   %}

```

Or more concise in the source and perhaps even faster in processing if only few keys are changed:

```

89 \DeclareInstanceCopy{blockenv}{flushleft}{center}
90 \EditInstance{blockenv}{flushleft}{
91     name = flushleft
92     ,para-instance = raggedright }

```

`blockenv flushright` (*inst.*) Same game for flushright.

```

93 \DeclareInstanceCopy{blockenv}{flushright}{center}
94 \EditInstance{blockenv}{flushright}{
95     name = flushright
96     ,para-instance = raggedleft }

```

3.2.2 Their block instances

They all use the block instances `std` which have already been set up in section 3.1.2.

3.2.3 Their para instances

Formatting of paragraphs is handled through the `para-instance` key which either refers to a instance of type `para` or is empty, in which case the handling of paragraphs is inherited. The predefined instances are discussed in section 4.

3.3 The quote and quotation environments

L^AT_EX₂ 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 paragraph setup is inherited. The block nesting level is incremented.

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

`quote` (*env.*) We can't use `\RenewDocumentEnvironment` for `quote` and other environments that `quotation` (*env.*) are class defined, because some classes aren't implementing them at all. So we use `\DeclareDocumentEnvironment` instead. This problem will vanish if all such definitions move in new versions of the classes instead.

```

97 \AddToHook{begindocument/before}[./legacy-quotes]{
98     \DeclareDocumentEnvironment{quote}{!0}{ }
99     { \SimpleBlockEnv{quote} {#1} } { \BlockEnvEnd }

100 \DeclareDocumentEnvironment{quotation}{!0}{ }
101     { \SimpleBlockEnv{quotation} {#1} } { \BlockEnvEnd }
102 }

```

3.3.1 Their blockenv instances

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

```

103 \DeclareInstance{blockenv}{quotation}{std}
104 {
105     name = quotation
106     ,tag-name = \UseStructureName{block/quotation}
107     ,tag-attr-class =
108     ,tagging-recipe = standard
109     ,inner-level-counter =

```

```

110 ,transparent-level = false
111 ,early-legacy-code =
112 ,late-legacy-code =
113 ,block-instance = quotation
114 ,inner-instance =
115 }

```

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

```

116 \DeclareInstance{blockenv}{quote}{std}
117 {
118     name = quote
119     ,tag-name = \UseStructureName{block/quote}
120     ,tag-attr-class =
121     ,tagging-recipe = standard
122     ,inner-level-counter =
123     ,transparent-level = false
124     ,early-legacy-code =
125     ,late-legacy-code =
126     ,block-instance = quote
127     ,inner-instance =
128 }

```

3.3.2 Their block instances

`block quote-1 (inst.)` Default layout is to indent equally from both sides. Note that settings here also affect `block quote-2 (inst.)` verse as it currently reuses the block instances!

```

block quote-3 (inst.) 129 \DeclareInstance{block}{quote-1}{std}
block quote-4 (inst.) 130 {
block quote-5 (inst.) 131     right-margin = \KeyValue{left-margin}
block quote-6 (inst.) 132     ,para-vspace = \parsep
133 }
134 \DeclareInstanceCopy{block}{quote-2}{quote-1}
135 \DeclareInstanceCopy{block}{quote-3}{quote-1}
136 \DeclareInstanceCopy{block}{quote-4}{quote-1}
137 \DeclareInstanceCopy{block}{quote-5}{quote-1}
138 \DeclareInstanceCopy{block}{quote-6}{quote-1}

```

`block quotation-1 (inst.)` Quotation additionally changes the para-indent.

```

block quotation-2 (inst.) 139 \DeclareInstance{block}{quotation-1}{std}
block quotation-3 (inst.) 140 { para-indent = 1.5em , right-margin = \KeyValue{left-margin} }
block quotation-4 (inst.) 141 \DeclareInstanceCopy{block}{quotation-2}{quotation-1}
block quotation-5 (inst.) 142 \DeclareInstanceCopy{block}{quotation-3}{quotation-1}
block quotation-6 (inst.) 143 \DeclareInstanceCopy{block}{quotation-4}{quotation-1}
144 \DeclareInstanceCopy{block}{quotation-5}{quotation-1}
145 \DeclareInstanceCopy{block}{quotation-6}{quotation-1}

```

3.4 The verse environment

The `verse` environment of L^AT_EX is intended for poetry. Not sure what that should mean with respect to tagging.

`verse (env.)` Implementation is like `quote` etc.

```

146 \AddToHook{begindocument/before}[./legacy]{
147   \DeclareDocumentEnvironment{verse}{!0{}}
148   { \SimpleBlockEnv{verse} {#1} } { \BlockEnvEnd }
149 }

```

3.4.1 Their blockenv instances

`blockenv verse (inst.)`

```

150 \DeclareInstance{blockenv}{verse}{std}
151 {
152   name = verse
153   ,tag-name = \UseStructureName{block/verse}
154   ,tag-attr-class =
155   ,tagging-recipe = standard
156   ,inner-level-counter =
157   ,transparent-level = false
158   ,early-legacy-code =
159   ,late-legacy-code =
160   ,block-instance = quote % reuse?
161   ,para-instance = verse
162   ,inner-instance =
163 }

```

The special indentation on continuation lines (the way L^AT_EX handled poetry) is done in the `para` instance `verse`, defined later on.

3.5 The `verbatim`, `verbatim*` and `alltt` environments

`verbatim (env.)` Here are the definitions for the `verbatim` environments. They look somewhat different
`verbatim* (env.)` than others (but this isn't the final definition). At the moment we use 2 optional arguments, the second is only there so that there is yet another scan even if one optional argument got detected. That then scans away the newline so that afterwards we can reinsert one via `\obeyedline`. A better solution will be to use a `c` specifier for grabbing the body, but that is for another day not Christmas Eve.

```

164 \AddToHook{begindocument/before}[./legacy-verbatim]{
165   \RenewDocumentEnvironment{verbatim}{={late-legacy-code} !o !o }
166   { \SimpleBlockEnv{verbatim} {#1} \obeyedline } { \BlockEnvEnd }

167   \RenewDocumentEnvironment{verbatim*}{={late-legacy-code} !o !o }
168   { \SimpleBlockEnv{verbatim*} {#1} \obeyedline } { \BlockEnvEnd }

```

`alltt (env.)` The `alltt` package implements a variation on `verbatim` handling where backslash and
`alltt* (env.)` braces retain their normal meanings. We also reimplement it using the template approach
 The `alltt*` variant didn't exist in the package, but it is trivial to set it up as well.

```

169 \NewDocumentEnvironment{alltt}{={late-legacy-code} !o }
170   { \SimpleBlockEnv{alltt} {#1} } { \BlockEnvEnd }
171 \NewDocumentEnvironment{alltt*}{={late-legacy-code} !o }
172   { \SimpleBlockEnv{alltt*} {#1} } { \BlockEnvEnd }
173 }

```

The parsing here should be adjusted as well, eventually.

3.5.1 Their blockenv instances

`blockenv verbatim` (*inst.*) The `verbatim` environment is defined through `blockenv` instance `verbatim` that makes use of the `block` instance `verbatim-⟨level⟩` and the `para` instance `justify`. The block nesting level is not incremented. Verbatim processing requires various `\catcode` changes, etc. and as a consequence a special parsing routine that grabs the whole environment while these catcodes are in force. This setup is done in the `final-code` key and its last action is to initiate the special parsing.

```

174 \DeclareInstance{blockenv}{verbatim}{std}
175 {
176   name                = verbatim
177   ,tag-name            = \UseStructureName{block/verbatim}
178   ,tag-attr-class      =
179   ,tagging-recipe       = standard
180   ,tagging-suppress-paras = true
181   ,inner-level-counter =
182   ,transparent-level    = true
183   ,early-legacy-code   =
184   ,late-legacy-code    =
185   ,block-instance      = verbatim
186   ,inner-instance      =
187   ,para-instance       = justify

```

Here is where `verbatim` and `verbatim*` technically differ: in the former we set up spaces to become nonbreakable spaces (if necessary followed by a `\pdfspacespace` in the pdfTeX engine) and in `verbatim*` we set it up to generate visible space chars.

```

188   ,final-code          = \legacyverbatimsetup{invisible}

```

Then we start the special scanning process to look for `\end{verbatim}` with special catcodes and grab everything in between. For `verbatim*` we use `\@sxverbatim` to look for `\end{verbatim*}` instead.¹

```

189                               \@sxverbatim
190 }

```

The role-mapping is `<verbatim>` to `<Code>` and `<codeline>` to `<Sub>` (which is role mapped to `<Span>` in pdf 1.7). Sub inside Code is allowed according the errata of ISO 32005. The paragraphs inside verbatim are flattened. Line numbers should be inside the `<codeline>` structure and be tagged either as `<Lb1>` or `<Artifact><Lb1>`.

`blockenv verbatim*` (*inst.*) The implementation of `verbatim*` is similar using the `blockenv` instance `verbatim*`. Its `final-code` sets up visible spaces and a slightly different parsing that grabs everything up to `\end{verbatim*}`. Otherwise the setup is identical.

```

191 \DeclareInstance{blockenv}{verbatim*}{std}
192 {
193   name                = verbatim
194   ,tag-name            = \UseStructureName{block/verbatim}
195   ,tag-attr-class      =
196   ,tagging-recipe       = standard
197   ,tagging-suppress-paras = true
198   ,inner-level-counter =
199   ,transparent-level    = true

```

¹Perhaps there should be some other command names for this?

```

200 ,early-legacy-code      =
201 ,late-legacy-code       =
202 ,block-instance         = verbatim
203 ,inner-instance         =
204 ,para-instance          = justify
205 ,final-code             = \legacyverbatimsetup{visible}
206                          \@sxverbatim
207 }

```

`blockenv alltt (inst.)` The implementation of the `alltt` environment from the `alltt` is more or less identical as well. We just need a slightly different final code to keep backslash and braces functional.

```

208 \DeclareInstance{blockenv}{alltt}{std}
209 {
210   name                = alltt
211   ,tag-name            = \UseStructureName{block/verbatim} % private tag instead?
212   ,tag-attr-class     =
213   ,tagging-recipe      = standard
214   ,tagging-suppress-paras = true
215   ,inner-level-counter =
216   ,transparent-level   = true
217   ,early-legacy-code  =
218   ,late-legacy-code   =
219   ,block-instance     = verbatim
220   ,inner-instance     =
221   ,para-instance      = justify

```

Now set up the special environment settings with most characters verbatim. We don't even have to scan ahead for the `\end{alltt}` because backslash and braces still have their normal meaning.

```

222   ,final-code          = \legacyallttsetup {invisible}
223 }

```

`blockenv alltt* (inst.)` The `alltt*` variant didn't exist in the `alltt` package, but it is trivial to set it up as well.

```

224 \DeclareInstance{blockenv}{alltt*}{std}
225 {
226   name                = alltt*
227   ,tag-name            = \UseStructureName{block/verbatim} % private tag instead?
228   ,tag-attr-class     =
229   ,tagging-recipe      = standard
230   ,tagging-suppress-paras = true
231   ,inner-level-counter =
232   ,transparent-level   = true
233   ,early-legacy-code  =
234   ,late-legacy-code   =
235   ,block-instance     = verbatim
236   ,inner-instance     =
237   ,para-instance      = justify
238   ,final-code          = \legacyallttsetup {visible}
239 }

```

3.5.2 Their block instances

`block verbatim-1` (*inst.*) Verbatim instances have their own levels so that one can specify specific indentations or vertical separations between lines.

```

block verbatim-3 (inst.) 240 \DeclareInstance{block}{verbatim-1}{std}
block verbatim-4 (inst.) 241 {
block verbatim-5 (inst.) 242     ,left-margin      = 0pt
block verbatim-6 (inst.) 243     ,para-vspace     = 0pt
                        244 }

245 \DeclareInstanceCopy{block}{verbatim-2}{verbatim-1}
246 \DeclareInstanceCopy{block}{verbatim-3}{verbatim-1}
247 \DeclareInstanceCopy{block}{verbatim-4}{verbatim-1}
248 \DeclareInstanceCopy{block}{verbatim-5}{verbatim-1}
249 \DeclareInstanceCopy{block}{verbatim-6}{verbatim-1}

```

3.6 The standard lists: `itemize`, `enumerate`, and `description`

`itemize` (*env.*) For the standard lists everything is managed by the `blockenv` instances. For the list we do not require that the optional argument directly follows without spaces as there can be no conflict with any following text (only `\item` or an empty line or the optional argument would be possible).

```

250 \AddToHook{begindocument/before}[./legacy-lists]{
251   \RenewDocumentEnvironment{itemize}{0}{} {
252     { \SimpleBlockEnv{itemize} {#1} } { \BlockEnvEnd }

253   \RenewDocumentEnvironment{enumerate}{0}{} {
254     { \SimpleBlockEnv{enumerate} {#1} } { \BlockEnvEnd }

255   \DeclareDocumentEnvironment{description}{0}{} {
256     { \SimpleBlockEnv{description} {#1} } { \BlockEnvEnd }
257 }

```

3.6.1 Their `blockenv` instances

`blockenv itemize` (*inst.*) The `itemize` environment is defined through the `blockenv` instance `itemize` which makes use of the `block` instance `list-⟨level⟩`, and an inner instance `itemize-⟨inner-level⟩` of type `list`. The paragraph setup is inherited.² The `⟨inner-level⟩` is determined through `\@itemdepth`. The block nesting level and the inner list nesting level are incremented.

```

258 \DeclareInstance{blockenv}{itemize}{std}
259 {
260   ,name                = itemize
261   ,tag-name            = \UseStructureName{block/itemize}
262   ,tag-attr-class      = itemize
263   ,tagging-recipe      = list
264   ,inner-level-counter = \@itemdepth
265   ,transparent-level   = false
266   ,max-inner-levels    = 4
267   ,early-legacy-code  =

```

²In the L^AT_EX 2_ε implementation justified paragraphs were forced, even if the whole document was set in ragged text. If this slightly strange behavior is desired then one has to set the `para-instance` key to `justify`.

```

268 ,late-legacy-code =
269 ,block-instance   = std-list
270 ,inner-instance-type = list
271 ,inner-instance    = itemize
272 ,para-instance     =
273 }

```

`blockenv enumerate (inst.)` The `enumerate` environment is similar to `itemize` but uses the `blockenv` instance `enumerate`, the block instance `list-⟨level⟩`, and the inner instance `enumerate-⟨inner-level⟩`. The `⟨inner-level⟩` is determined through `\@enumdepth`.

```

274 \DeclareInstance{blockenv}{enumerate}{std}
275 {
276   name = enumerate
277   ,tag-name = \UseStructureName{block/enumerate}
278   ,tag-attr-class = enumerate
279   ,tagging-recipe = list
280   ,transparent-level = false
281   ,max-inner-levels = 4
282   ,early-legacy-code =
283   ,late-legacy-code =
284   ,early-legacy-code =
285   ,block-instance = std-list
286   ,inner-level-counter = \@enumdepth
287   ,inner-instance-type = list
288   ,inner-instance = enumerate
289 }

```

`blockenv description (inst.)` The `description` environment uses the `blockenv` instance `description`, the block instance `list-⟨level⟩`, and the inner instance `description`.

In $\text{\LaTeX 2}_\epsilon$ that was no dependency on the nesting level, i.e., the environment has the same appearance on all nesting levels, but there is actually no good reason for disallowing layout adjustments on each level. All we have to do for this is to provide a value `inner-level-counter` and allocate a counter register for that.

We allow for 6 levels.

```

290 \DeclareInstance{blockenv}{description}{std}
291 {
292   name = description
293   ,tag-name = \UseStructureName{block/description}
294   ,tag-attr-class = description
295   ,tagging-recipe = list
296   ,inner-level-counter = \@descriptiondepth
297   ,transparent-level = false
298   ,max-inner-levels = 6
299   ,early-legacy-code =
300   ,late-legacy-code =
301   ,block-instance = std-list
302   ,inner-instance-type = list
303   ,inner-instance = description
304 }

```

`\@descriptiondepth` Counting nested description environments.

```

305 \newcount\@descriptiondepth

```

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

3.6.2 Their block instances

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

```
block std-list-4 (inst.) 306 \DeclareInstance{block}{std-list-1}{std}{
block std-list-5 (inst.) 307 %   begin-vspace          = \topsep
block std-list-6 (inst.) 308 %   ,begin-extra-vspace = \partopsep
```

This is the only one we have to explicitly set for lists if the default setup is wanted.

```
309 ,para-vspace          = \parsep
310 %   ,end-vspace        = \KeyValue{begin-vspace}
311 %   ,end-extra-vspace  = \KeyValue{begin-extra-vspace}
312 %   ,item-vspace       = \itemsep
313 %   ,begin-penalty      = \UseName{@beginparpenalty}
314 %   ,end-penalty        = \UseName{@endparpenalty}
315 %   ,left-margin        = \leftmargin
316 %   ,right-margin       = \rightmargin
317 %   ,para-indent        = 0pt
318 }

319 \DeclareInstanceCopy{block}{std-list-2}{std-list-1}
320 \DeclareInstanceCopy{block}{std-list-3}{std-list-2}
321 \DeclareInstanceCopy{block}{std-list-4}{std-list-3}
322 \DeclareInstanceCopy{block}{std-list-5}{std-list-4}
323 \DeclareInstanceCopy{block}{std-list-6}{std-list-5}
```

If the legacy `\list<romannumeral>` is not used in a modern class then, of course, these instances all need to set up the different parameters explicitly. The new implementation of the standard classes (will) show that approach.

3.6.3 Their list instances

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
`list itemize-2 (inst.)` definitions rather than defining the `item-label` code in the instance to ensure that old
`list itemize-3 (inst.)` documents still work.

```
list itemize-4 (inst.) 324 \DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi  }
325 \DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii }
326 \DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
327 \DeclareInstance{list}{itemize-4}{std}{ item-label = \labelitemiv  }
```

`list enumerate-1 (inst.)` `enumerate` environments are similar, except that we also have to say which counter to
`list enumerate-2 (inst.)` use on each level.

```
list enumerate-3 (inst.) 328 \DeclareInstance{list}{enumerate-1}{std}
list enumerate-4 (inst.) 329 { item-label = \labelenumi ,   counter = enumi   }
330 \DeclareInstance{list}{enumerate-2}{std}
331 { item-label = \labelenumii ,  counter = enumii  }
332 \DeclareInstance{list}{enumerate-3}{std}
333 { item-label = \labelenumiii , counter = enumiii }
334 \DeclareInstance{list}{enumerate-4}{std}
335 { item-label = \labelenumiv ,  counter = enumiv  }
```

`list description (inst.)` In $\text{\LaTeX} 2_{\epsilon}$ the `description` lists used only a single list instance with only one key not using the default. But in this implementation we allow for customization of every level, even though we aren't making use of it by default.

Doing that means that you can only use a limited number of nested environments (while in $\text{\LaTeX} 2_{\epsilon}$ one could have arbitrary nestings, so let's hope 6 levels are enough).

TODO: consider reusing the last level if you run out of specified levels rather than using `max-inner-levels`.

```

336 \DeclareInstance{list}{description-1}{std} { item-instance = description }

337 \DeclareInstanceCopy{list}{description-2}{description-1}
338 \DeclareInstanceCopy{list}{description-3}{description-1}
339 \DeclareInstanceCopy{list}{description-4}{description-1}
340 \DeclareInstanceCopy{list}{description-5}{description-1}
341 \DeclareInstanceCopy{list}{description-6}{description-1}

```

3.6.4 Their item instances

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

```

342 \DeclareInstance{item}{basic}{std}
343     { label-align = right }

344 \DeclareInstance{item}{description}{std}
345 {
346     ,label-format = \normalfont\bfseries #1
347     ,label-align  = left
348 }

```

3.7 The legacy list and trivlist environments

In $\text{\LaTeX} 2_{\epsilon}$, `trivlist` was used to define various display environments that aren't really lists at all. To support such legacy definitions (even though they should be updated to achieve proper tagging) we continue to support and implement it as a `list` environment with a few hardwired settings mimicking the original behavior.

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

```

349 \AddToHook{begindocument/before}[./legacy]{
350     \RenewDocumentEnvironment{list}{0}{m m }
351     {

```

We do this by storing them away and then call the list instance. Inside this instance the `early-legacy-code` key contains `\legacylistsetup` which makes use of the stored values.

```

352     \tl_set:Nn \l_@@_legacy_env_params_tl
353     {
354         \tl_set:Nn \@itemlabel {#2}
355         #3
356     }

```

maybe we should simply implement it as a `displayblock` instance (at least when doing tagging) - decide

The $\text{\LaTeX} 2_{\epsilon}$ lists don't support captions so we use `\SimpleBlockEnv`.

```

357     \SimpleBlockEnv{list} {#1}
358   }
359   { \BlockEnvEnd }
360 }

```

`trivlist (env.)` $\text{\LaTeX} 2_{\epsilon}$ defined `trivlist` as an implementation of `list` (or rather the other way around).

Replace with code not using `\list`

```

361 \AddToHook{begindocument/before}[./legacy]{
362   \RenewDocumentEnvironment{trivlist}{!0}{ }
363     { \list[#1]{ }
364       {
365         \dim_zero:N \leftmargin
366         \dim_zero:N \labelwidth
367         \cs_set_eq:NN \makelabel \use:n
368       }
369     }
370   { \BlockEnvEnd }
371 }

```

3.7.1 Its `blockenv` instance

`blockenv list (inst.)` The generic `list` environment of $\text{\LaTeX} 2_{\epsilon}$ is modeled with a `blockenv` instance named `list`, a block instance named `std-list-⟨level⟩`, and an inner instance named `legacy` (with no dependency on the nesting level). This environment has two arguments and customization of the layout is expected to be directly set in the second argument. For this reason this `legacy` instance is something that shouldn't be changed (all that is attempted to provide a way to support legacy setups).

To set up the default settings (as they were used in $\text{\LaTeX} 2_{\epsilon}$) the `early-legacy-code` key gets `\legacylistsetup` assigned that contains the necessary code to set up these defaults. Changing the `blockenv` is therefore not recommended for the legacy `list` environment.

```

372 \DeclareInstance{blockenv}{list}{std}
373 {
374   ,name = list
375   ,tag-name = \UseStructureName{block/list}
376   ,tag-attr-class =
377   ,tagging-recipe = list
378   ,transparent-level = false
379   ,early-legacy-code = \legacylistsetup
380   ,late-legacy-code =
381   ,block-instance = std-list
382   ,inner-level-counter =
383   ,inner-instance-type = list
384   ,inner-instance = legacy
385 }

```

3.7.2 Its `list` instance

`list legacy (inst.)` For the legacy `list` environment there is only one instance which is reused on all levels. This is done this way 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.

```

386 \DeclareInstance{list}{legacy}{std} {
387   ,item-instance = basic
388   ,legacy-support = true
389 }

```

3.8 Theorem-like environments declared through `\newtheorem`

In standard $\text{\LaTeX}$ theorem-like environments are not defined directly, but with the help of a `\newtheorem` declaration. That allows specifying the typeset environment title, e.g., “Lemma”, and the counter to use to number the environments, e.g., they could be all numbered individually or one could number them using the same counter as some other theorem-like environment.

This was first augmented by the `theorem` package which implemented the idea of a `\theoremstyle`; this is now considered obsolete. Michael Downes from the AMS improved on these early ideas and wrote the `amsthm` package, which offered more functionality including a `\newtheoremstyle` declaration and for the document level a `\swapnumbers` and an `proof` environment. It also provided star-forms for `\newtheorem` (to define an unnumbered environment) and allowed to use star-forms of the theorem-like environments to suppress numbering on an individual instance in the document.

This new implementation based on templates, is supposed to cover the functionality of `amsthm` including its declarations so that documents that use `amsthm` explicitly or implicitly via their class should continue to work seamlessly.

For other packages that provide theorem-like environments we have to see if they could be easily remodeled using the new implementation or if there is a need for extended templates.

Assuming declarations such as

```

% \swapnumbers           % <- commented out
\theoremstyle{definition}
\newtheorem{axiom}[def]{Axiom}

```

in a document, then the following instances of type `blockenv` and `captionedtext` are declared by `\newtheorem`.

3.8.1 The `blockenv` instances they use

Given the above input `\newtheorem` defines the following `blockenv` instance:

```

\DeclareInstance{blockenv}{axiom}{std}
{
  name                = theorem-like
  ,tag-name           = \UseStructureName{block/theorem-like}
  ,tagging-recipe     = standalone
  ,standalone         = true
  ,transparent-level  = true
  ,block-instance:e   = thm-
                      \IfInstanceExistsTF{block}
                      { thm-definition-1 }
                      { definition } { plain }
}

```

```

,inner-instance-type = captionedtext
,inner-instance      = axiom
,para-instance       = justify
}

```

The setting for `block-instance` means that it checks if a `block` instance with the name `thm-definition-1` exists. If so then the value `thm-definition` is used, otherwise `thm-plain` is used which is always defined, i.e., if the `theoremstyle` does not specify any special vertical spacing the `block` instance from the `plain` style is reused.

What varies from `blockenv` instance to instance are the values for `block-instance` and `inner-instance`.

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

3.8.2 The `captionedtext` instances they use

The instance of type `captionedtext` is also defined by `\newtheorem` and in this case it looks like this:

```

\DeclareInstance{captionedtext}{axiom}{thmlike}
{
  ,counter = def
  ,title   = Axiom           % <-- that the title provided to \newtheorem
  ,style   = definition      % <-- that's the used \theoremstyle
}

```

If we uncomment the `\swapnumbers` line in the example above then we get

```

,style   = definition-swap

```

in the `captionedtext` instance instead.

3.8.3 The `thmstyle` instances they use

New theorem styles can be declared with `\newtheoremstyle` which then generates an instance of type `thmstyle`. Alternatively, it is, of course, possible to declare the instances directly (which gives you a bit more flexibility). A few such styles are predeclared, matching what is offered by `amsthm`. These are shown below.

`thmstyle plain (inst.)` The main style used for many theorem-like environments, i.e., the one you get if no special `\theoremstyle` has been specified.

```

390 \DeclareInstance{thmstyle}{plain}{std}
391 {
392   ,caption-placement = unchained
393   ,numbered          = true
394   ,separator         = \
395   ,punct             = .
396   ,before-hspace     = 0pt
397   ,after-hspace      = 5pt plus 1pt minus 1pt
398   ,order             = {title, separator, number, separator, note, punct}
399   ,caption-decls     = \bfseries
400   ,title-decls       =
401   ,number-decls      = \upshape

```

```

402     ,punct-decls      =
403     ,note-decls       = \upshape\mdseries
404     ,title-format     = #1
405     ,number-format    = #1
406     ,punct-format     = #1
407     ,note-format      = (#1)
408     ,body-decls       = \itshape
409 }

```

`thmstyle remark (inst.)` The `remark` is like `plain` with two changes:

```

410 \DeclareInstanceCopy{thmstyle}{remark}{plain}
411 \EditInstance{thmstyle}{remark}
412 {
413     ,caption-decls = \itshape
414     ,body-decls    = \normalfont
415 }

```

`thmstyle definition (inst.)` The `definition` is like `plain` with only a difference in the font used for the body:

```

416 \DeclareInstanceCopy{thmstyle}{definition}{plain}
417 \EditInstance{thmstyle}{definition}
418 {
419     ,body-decls = \normalfont
420 }

```

`thmstyle legacy2e (inst.)` Vanilla L^AT_EX 2_ε (without `amsthm` loaded) had a slightly different default. We provide this under the name `legacy2e`. It doesn't use a punctuation after the number and it has slightly different vertical spacing (defined by `thm-legacy2e-1` below). Thus, to reprocess an old document for tagging that uses `\newtheorem` without loading `amsthm` one has to set `\theoremstyle{legacy2e}` to avoid layout changes. How such a compatibility setting is automated is not yet decided.

```

421 \DeclareInstanceCopy{thmstyle}{legacy2e}{plain}
422 \EditInstance{thmstyle}{legacy2e}{ punct = }

```

3.8.4 The block instances they use

`block thm-plain-1 (inst.)` Theorems do not support nesting, so in theory we have only one to set up. There are, however, documents that put theorem-like environments inside of lists or other block environments. While that is in most case somewhat dubious, it can make sense, for example, in `description` lists. So we support it by providing `thm-plain` instances for levels 1 and 2. If somebody really nests them further down, then more such instances need to be declared.

The L^AT_EX default reused the general value of `\parindent` and `\parskip` and, of course, they start at the outer margin.

```

423 \DeclareInstance{block}{thm-plain-1}{std}
424 {
425     ,begin-extra-vspace = 0pt
426     ,left-margin        = 0pt
427     ,para-indent        = \parindent
428     ,para-vspace        = \parskip
429 }
430 \DeclareInstanceCopy{block}{thm-plain-2}{thm-plain-1}

```

`block thm-remark-1` (*inst.*) The `\thmstyle` for “remarks” is defined by `amsthm` to use less vertical spacing. It therefore needs its own `block` instance.

```

431 \DeclareInstance{block}{thm-remark-1}{std}
432 {
433   ,begin-vspace      = 0.5\topsep
434   ,begin-extra-vspace = 0pt
435   ,left-margin       = 0pt
436   ,para-indent       = \parindent
437   ,para-vspace       = \parskip
438 }
439 \DeclareInstanceCopy{block}{thm-remark-2}{thm-remark-1}

```

`block thm-legacy2e-1` (*inst.*) These are like the plain ones but without resetting `begin-extra-vspace` to zero.

```

440 \DeclareInstance{block}{thm-legacy2e-1}{std}
441 {
442   ,left-margin      = 0pt
443   ,para-indent      = \parindent
444   ,para-vspace      = \parskip
445 }
446 \DeclareInstanceCopy{block}{thm-legacy2e-2}{thm-legacy2e-1}

```

3.9 The proof environment (from `amsthm`)

`proof` (*env.*) The `proof` environment expects one optional argument holding an alternative title for the proof. We parse this optional argument as an implicit key/value argument, so that it is possible to interpret it either as the value for the key `note` or as a key/value list that holds special key settings for this particular environment instance. The result is analyzed by `\ParseLaTeXeTheoremlike` which then calls a `blockenv` instance with the name `proof`.

In addition we have to set up handling of QED symbols using `\pushQED` and `\popQED` using the logic already defined in `amsthm`. Details on all this is given in the code section of this module but normally this top-level declaration doesn’t require any changes. Conceptually, it would be better to disallow spaces before the optional argument here. But `amsthm` never did this, so for compatibility we aren’t doing this either.

```

447 \NewDocumentEnvironment{proof}{={note}o }
448 { \pushQED{\qed}%
449   \ParseLaTeXeTheoremlike {proof} \BooleanTrue {#1} }
450 { \popQED \BlockEnvEnd }

```

`blockenv proof` (*inst.*) A proof uses its own `proofblock` instance of type `block` for vertical spacing. As the proof has a heading we use a `captionedtext` instance with name `proof` as the inner instance and the paragraphs of the proof are justified.

```

451 \DeclareInstance{blockenv}{proof}{std}
452 {
453   name           = proof
454   ,tag-name      = \UseStructureName{block/proof}
455   ,tag-attr-class =
456   ,tagging-recipe = standalone
457   ,standalone    = true
458   ,inner-level-counter =

```

```

459 ,transparent-level      = true
460 ,early-legacy-code      =
461 ,late-legacy-code       =
462 ,block-instance         = proof
463 ,inner-instance-type     = captionedtext
464 ,inner-instance          = proof
465 ,para-instance           = justify
466 }

```

`captionedtext proof (inst.)` We use a special `captionedtext` template to set up the proof because proofs are not numbered and the argument to a proof environment has a somewhat different semantic meaning than that of theorem-like environments. If the title ends in a punctuation we should not add an additional one, thus we add it with `\AddPunctuation`.

```

467 \DeclareInstance{captionedtext}{proof}{proof}
468 {
469   ,title           = \proofname      % default value
470   ,punct           = .
471   ,before-hspace   = 0pt
472   ,after-hspace    = 5pt plus 1pt minus 1pt
473   ,caption-decls   = \itshape
474   ,title-format    = #1
475   ,punct-format    = \AddPunctuation{#1}
476   ,body-decls      = \normalfont
477 }

```

3.9.1 Block instances for the proofs

`block proof-1 (inst.)` Blocks for proofs are pretty normal (the values are taken from the `amsthm` implementation):

```

478 \DeclareInstance{block}{proof-1}{std}
479 {
480   ,begin-vspace     = 6pt plus 6pt
481   ,left-margin      = 0pt
482   ,para-indent      = \parindent
483   ,para-vspace      = \parskip
484 }
485 \DeclareInstanceCopy{block}{proof-2}{proof-1}

```

4 Declaring para instances

Display block environments often require special paragraph settings and therefore have a `para-instance` key to specify and appropriate instance. Here are the standard instances that are predefined for this purpose.

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

```

486 \DeclareInstance{para}{justify}{std}
487 {
488   % ,para-attr-class    = justify
489   % ,para-indent        = \parindent
490   % ,begin-hspace       = 0pt
491   % ,left-hspace        = \z@skip

```

```

492 % ,right-hspace          = \z@skip
493 % ,end-hspace             = \@flushglue
494 % ,final-hyphen-demerits = 5000
495 % ,newline-cmd           = \@normalcr
496 }

```

`para center` (*inst.*) Centering a paragraph means putting stretchable glue on both sides.

```

497 \DeclareInstance{para}{center}{std}
498 {
499   ,para-attr-class      = center
500   ,para-indent          = Opt
501   % ,begin-hspace       = Opt
502   ,left-hspace          = \@flushglue
503   ,right-hspace         = \@flushglue
504   ,end-hspace           = \z@skip
505   ,final-hyphen-demerits = 0
506   ,newline-cmd          = \@centercr
507 }

```

`para raggedright` (*inst.*) This is the plain T_EX version of ragged right, which basically means no hyphenation unless a word is truly longer than a line. This implements `flushleft`.

```

508 \DeclareInstance{para}{raggedright}{std}
509 {
510   ,para-attr-class      = raggedright
511   ,para-indent          = Opt
512   % ,begin-hspace       = Opt
513   ,left-hspace          = \z@skip
514   ,right-hspace         = \@flushglue
515   ,end-hspace           = \parfillskip
516   ,final-hyphen-demerits = 0
517   ,newline-cmd          = \@centercr
518 }

```

`para raggedleft` (*inst.*) This here is for `flushright`.

```

519 \DeclareInstance{para}{raggedleft}{std}
520 {
521   ,para-attr-class      = raggedleft
522   ,para-indent          = Opt
523   % ,begin-hspace       = Opt
524   ,left-hspace          = \@flushglue
525   ,right-hspace         = \z@skip
526   ,end-hspace           = \z@skip
527   ,final-hyphen-demerits = 0
528   ,newline-cmd          = \@centercr
529 }

```

`\centering` These instances are also used to implement declarations for direct use in documents or
`\raggedleft` in user definitions.

```

\raggedright 530 \DeclareRobustCommand\centering {\UseInstance{para}{center}}{}
\justifying  531 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}}{}
532 \DeclareRobustCommand\raggedright {\UseInstance{para}{raggedright}}{}
533 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}}{}

```

L^AT_EX’s default is to typeset paragraphs justified.

```
534 \justifying
```

(End of definition for `\centering` and others.)

`para verse` (*inst.*) For the `verse` environment we use a special `para` instance. If the right hand side should be ragged then a different `right-hspace` is needed.

```
535 \DeclareInstance{para}{verse}{std}
536 {
537   para-attr-class      = justify ,
538   para-indent          = 0pt ,
539   begin-hspace         = -1.5em ,
540   left-hspace          = 1.5em ,
541   right-hspace         = 0pt ,
542   end-hspace           = \@flushglue ,
543   final-hyphen-demerits = 0 ,
544   newline-cmd          = \@centercr ,
545 }
546 \</class-code>
```

5 Advice on adjusting the layout of standard block environments

to document

6 Tagging support

6.1 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 life, 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 `<semantic-para>` and use `<text-block>` (role-mapped to `<P>`) only for (portions of) the actual paragraph text in a way that the `<text-block>`s are not nested. As a result we have for a simple paragraph the structures

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

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

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

```
<semantic-para>
  <text-block>
    The paragraph text before the display element ...
  </text-block>
  <display element structure>
    Content of the display structure possibly involving inner <semantic-para> tags
  </display element structure>
  <text-block>
    ... continuing the outer paragraph text
  </text-block>
</semantic-para>
```

In other words such a display block is always embedded in a `<semantic-para>` structure, possibly preceded by a `<text-block>...</text-block>` 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:

```
<semantic-para>
  <text-block>
    The intro text for the itemize environment ...
  </text-block>
  <itemize>
    <LI>
      <itemlabel> label </itemlabel>
      <itembody>
        The text of the first item involving <semantic-para> as necessary ...
      </itembody>
    </LI>
    <LI>
      The second item ...
    </LI>
    ... further items ...
  </itemize>
</semantic-para>
```

The `<itemize>` is role-mapped to `<L>`.

For some display blocks, such as centered text, we use a simpler strategy. Such blocks still ensure that they are inside a `<semantic-para>` structure but their body uses simple `<text-block>` blocks and not `<semantic-para><text-block>` 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:

```

<semantic-para>
  <text-block>
    This is a paragraph with some
  </text-block>
  <text /0 /Layout /TextAlign/Center>
    centered lines
  </text-block>
  <text /0 /Layout /TextAlign/Center>
    with a paragraph break between them
  </text-block>
  <text-block>
    followed by some more text.
</semantic-para>

```

The semantic-para structures are added by using the tagging sockets `para/semantic/begin` and `para/semantic/end` declared in `l1tagging.dtx`. They can be disabled by assigning these sockets the plug `noop`.

6.1.1 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:

noop This recipe does not add tagging structures and also does not set the endpe switch. The recipe is meant for environments like `adjustwidth` that only want to change the layout, and which can contain, e.g., sectioning commands.

standalone This recipe does the following:

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

basic This recipe does the following:

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

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

- it starts an inner tagging structure (i.e., which is therefore a child of the outer `<semantic-para>`).
- By default this structure is a `<Div>` unless overwritten by the key `tag-name`. If that key is used, a suitable role-map 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 `<semantic-para>` level from the outside.
- Then the lookahead for an empty line is done as described previously.

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

- the inner structure is a list (`<L>`).
- Furthermore everything is set up so that we have list items (`<LI>`) with suitable substructures (`<itemlabel>` for the item labels and `<itembody>` 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 role-map.
- If the key `tag-attr-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 `</itembody>`, `</LI>`, and `</L>` (or the tag name used) are closed.
- Then the lookahead for an empty line is done as described previously.

7 Tracing and debugging

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

These commands enable/disable debugging messages for blocks. They also enable/disable debugging of templates (e.g., call `\DebugTemplatesOn` or `\DebugTemplatesOff`).

The data that is produced is rather verbose and largely guided (so far) by what seemed helpful while developing the code. This needs some cleanup at a later stage. At the moment, if you have the following simple document

```

1      \DocumentMetadata{tagging=on, lang=en}
2
3      \documentclass{article}
4
5      \DebugBlocksOn
6
7      \begin{document}
8          \begin{itemize}[item-vspace=3pt]
9              \item          A normal item
10             \item[\textbf{+}] A special item
11         \end{itemize}
12     \end{document}

```

then you will get the following information on the screen and in the .log file:

```

[Template] ==> Use 'blockenv' instance: itemize on input line 8
[Template] ==>   template: 'std'; arguments: |item-vspace=3pt|\BooleanFalse |\NoValue |\NoValue |
[Template] ==> Use 'block' instance: std-list-1 on input line 8
[Template] ==>   template: 'std'; argument: |item-vspace={3pt}|
[Blocks] ==> @endpe=false on input line 8
[Template] ==> Use 'list' instance: itemize-1 on input line 8
[Template] ==>   template: 'std'; arguments: ||\BooleanFalse |\NoValue |\NoValue |
[Blocks] ==> Set first block everypar on input line 8
[Blocks] ==> template:list:std end

[Template] ==> Use 'item' instance: basic on input line 9
[Template] ==>   template: 'std'; argument: ||
[Blocks] ==> Set item block everypar on input line 9
[Blocks] ==> ... in item block everypar on input line 9
[Blocks] ==> increment P on input line 9
[Blocks] ==> Set noop block everypar on input line 9

[Template] ==> Use 'item' instance: basic on input line 10
[Template] ==>   template: 'std'; argument: |label={\textbf {+}}|
[Blocks] ==> item with optional
[Blocks] ==> Set item block everypar on input line 10
[Blocks] ==> ... in item block everypar on input line 10
[Blocks] ==> increment P on input line 10
[Blocks] ==> Set noop block everypar on input line 10

[Blocks] ==> blockenv common ending on input line 11

[Blocks] ==> flattened=false on input line 12
[Blocks] ==> Structure-end semantic-para after displayblock on input line 12

```

8 New and redefined kernel command

<code>\SimpleBlockEnv</code>	<i>to be documented</i>
<code>\BlockEnv</code>	
<code>\BlockEnvEnd</code>	
<code>\g_block_nesting_depth_int</code>	

<code>\legacyverbatimsetup</code>	<i>to be documented</i>
<code>\legacyallttsetup</code>	
<code>\legacylistsetup</code>	
<code>\@setupverbinvisiblespace</code>	A counterpart definition to the kernel command <code>\@setupverbinvisiblespace</code> , needed as we need to handle real space chars in verbatim.
<code>\newtheorem</code>	Reimplemented to fit the template approach. <code>\newtheoremstyle</code> was defined by <code>amsthm</code> .
<code>\newtheoremstyle</code>	
<code>\@nthm</code>	These are no longer used (to be removed).
<code>\@xnthm</code>	
<code>\@ynthm</code>	
<code>\@thm</code>	
<code>\@xthm</code>	
<code>\@ythm</code>	
<code>\@othm</code>	
<code>\@begintheorem</code>	
<code>\@opargbegintheorem</code>	
<code>\@endtheorem</code>	
<code>\item</code>	The <code>\item</code> is redefined.
<code>\@itemlabel</code>	
<code>\c@maxblocklevels</code>	A counter to increase or decrease the number of supported level. If increased, one needs to supply additional level instances.
<code>\begin</code>	The <code>\begin</code> is slightly redefined to handle <code>\@doendpe</code> better. TODO: move to kernel
<code>\@doendpe</code>	The original $\text{\LaTeX 2}_{\epsilon}$ command is augmented to allow for tagging.
<code>\para_end:</code>	TODO: consider name, document
<code>para/begin</code>	The <code>para/begin</code> hook is enhanced to support list ends

9 The Implementation

```

547 <*package-start>
548 <@@=block>

549 \ProvidesPackage {latex-lab-testphase-block}
550             [\tlabblockdate\space v\tlabblockversion\space
551             blockenv implementation]

552 \ExplSyntaxOn

```

9.0.1 Augmented \SetKnownTemplateKeys

\SetKnownTemplateKeys A key/val list passed to `\SetKnownTemplateKeys` can either be empty (in which we do not want to start up the parsing machinery) or it could be `\NoValue` in which we do not want to do that either. The latter can happen, for example, with `verbatim` where we define the optional argument with `={late-legacy-code} !o` so that people can write `\begin{verbatim}[\small]` a syntax promoted by the TUGboat class.

```

553 \cs_set_protected:Npn \SetKnownTemplateKeys #1#2#3
554 {

```

An “empty” argument (or rather one that is empty after one expansion) is the most likely case so we test for this first.

```

555     \tl_if_empty:oTF {#3}
556     {
557         \tl_set_eq:NN \UnusedTemplateKeys \c_empty_tl
558     }
559     {
560         \tl_if_novalue:nTF {#3}
561         {
562             \tl_set_eq:NN \UnusedTemplateKeys \c_empty_tl
563         }
564         {
565             \keys_set_known:noN { template / #1 / #2 } {#3}
566             \UnusedTemplateKeys
567         }
568     }
569 }

```

(End of definition for \SetKnownTemplateKeys. This function is documented on page ??.)

\SetTemplateKeys Same kind of extension for `\SetTemplateKeys`:

```

570 \cs_set_protected:Npn \SetTemplateKeys #1#2#3
571 {
572     \tl_if_empty:oF {#3}
573     {
574         \tl_if_novalue:nF {#3}
575         {
576             \keys_set:no { template / #1 / #2 } {#3}
577         }
578     }
579 }

```

(End of definition for \SetTemplateKeys. This function is documented on page ??.)

9.0.2 Tracing templates and instances

`\template_debug_typeout:n` I guess that tracing macro is needed in several modules, so should become public (or at least kernel).

```
580 \cs_new_protected:Npn \template_debug_typeout:n { \__template_debug_typeout:n }
```

(End of definition for `\template_debug_typeout:n`. This function is documented on page ??.)

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

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

For the reimplementaion 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: move to the kernel eventually.

```
581 \def\@doendpe{\@endpetrue
582   \def\par
583     {
```

If we are processing a $\$$ math display and we encounter a real `\par` after it, we need to add a `\parskip` when tagging is done, because the one added by T_EX is always canceled by the processing in `\__math_tag_dollardollar_display_end:` in that case. This is signaled by the global legacy switch `@domathendpe` which is set to true in that case. Once the skip is applied we set it to false. If there is no `\par` at all, it will be reset in `\everypar` when the next paragraph starts.

But we first have to restore `\par`, otherwise `\skip_vertical:n` might trigger `\par` in horizontal mode and then loop because it never returns to vertical mode.

```
584   \@restorepar
585   \clubpenalty\@clubpenalty

586   \if@domathendpe
587     \skip_vertical:n { \tex_parskip:D }
588     \@domathendpefalse
589   \fi
```

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

```
590   \tag_socket_use:n {\@doendpe}
```

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

```
591   \@endpefalse
592   \everypar{}
593   \par
```

```

594     }
595     \everypar{{\setbox\z@\lastbox}
596               \everypar{}
597               \@endpefalse

```

Not sure what is faster: testing for the status of the switch or setting it unconditionally to false (globally), probably roughly the same, so we set it always:

```

598 %           \ifdomathendpe
599 %           \@domathendpefalse
600 %           \fi
601 }
602 }

```

(End of definition for \@doendpe. This function is documented on page 40.)

\@endpefalse Some extra debugging lines, but commented out for now.

```

\@endpetrue
603 \def\@endpefalse{
604   \_block_debug_endpe_typeout:n { @endpe~ :=~ \if@endpe true \else false \fi -> false \on@li
605   \global\let\if@endpe\iffalse
606 }
607 \def\@endpetrue {%
608   \_block_debug_endpe_typeout:n { @endpe~ :=~ \if@endpe true \else false \fi -> true \on@li
609   \global\let\if@endpe\iftrue
610   \ifnum\currentgrouptype =14 % semi-simple group
611     \aftergroup\propagate@doendpe
612   \else
613     \ifnum\currentgrouptype =\z@ % no group: top-level
614     \else
615       \ifnum\currentgrouptype =\@ne % simple group
616       \aftergroup\propagate@doendpe
617     \fi
618   \fi
619   \fi
620 }

```

For checking the @endpe logic, we have something we can turn into a typeout in tests:

```

621 \cs_new_eq:NN \_block_debug_endpe_typeout:n \use_none:n

```

(End of definition for \@endpefalse and \@endpetrue. These functions are documented on page ??.)

tagsupport/@doendpe (socket) The socket is used in the \@doendpe TODO: if this goes into the kernel, the name should probably be different.

```

622 \socket_if_exist:nF{ tagsupport/@doendpe }
623 {
624   \NewTaggingSocket {@doendpe}{0}
625 }

```

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

```

626 \NewTaggingSocketPlug {@doendpe}{default}
627 {
628   \bool_if:NT \l__tag_para_bool
629   {

```

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 the socket code is called while `@endpe` is already handled and then we should not attempt to close a `<semantic-para>` structure). So we need to check for this.

```
630 \legacy_if:nTF { @endpe }
631 {
632 \__block_debug_endpe_typeout:n {==>~ handle~ @endpe}
```

If the display block currently ending was “flattened” (i.e., uses simplified paragraphs that are not tagged by a combination of `<semantic-para>` followed by `<text-block>`, but simply with a `<text-block>`), then we don’t have to do anything, because the `<text-block>` is already closed.

```
633 \__block_debug_typeout:n
634 { flattened= \bool_if:NTF
635 \l__tag_para_flattened_bool
636 {true}{false}
637 \on@line }
638 \bool_if:NF \l__tag_para_flattened_bool
639 {
640 \UseTaggingSocket{para/semantic/end}
641 {
642 \__block_debug_typeout:n{Structure-end~
643 \l__tag_para_main_tag_tl\space
644 after~ displayblock \on@line }
645 }
646 }
647 }
648 {
649 \__block_debug_endpe_typeout:n{==>~ @endpe~ already~ handled}
650 }
651 }
652 }

653 \AssignTaggingSocketPlug{@doendpe}{default}
```

```
\if@domathendpe Signal that special paragraph handling after a math display is required.
\@domathendpefalse
\@domathendpetrue
654 \newif\if@domathendpe
655 \def\@domathendpefalse{\global\let\if@domathendpe\iffalse}
656 \def\@domathendpetrue {\global\let\if@domathendpe\iftrue}
```

(End of definition for `\if@domathendpe`, `\@domathendpefalse`, and `\@domathendpetrue`.)

There is a general bug in the para handling: when the output routine is triggered the current setting of `@endpe` affects what happens in the OR. But it shouldn’t, so we need to reset its value (which is global) and set it back after the OR has finished. This is what the following code does (final implementation should probably not involve a normal

hook):

```
657 \AddToHook{build/column/before}{%
658 \if@endpe \endpefalse \aftergroup\@endpetrue \fi
659 }
```

fix in kernel

9.0.4 Other useful expl3 commands

This section collects expl3 commands that will be useful in the code here and possibly generally.

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

```
660 \cs_new_protected:Npn \_block_skip_set_to_last:N #1 {
661   \skip_set:Nn #1 { \tex_lastskip:D }
662 }
```

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

```
663 \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:.)

check

Not sure this is still necessary (or even correct) after the move to `\NoValue`.

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

(End of definition for \tl_if_novalue:oTF. This function is documented on page ??.)

9.1 Tracing and debugging interfaces

This follows the same convention as in other modules, but eventually that should be given a better implementation.

refactor at some stage

`\g_block_debug_bool` Boolean to indicate if we want to get debugging info from commands and templates handling block displays.

```
665 \bool_new:N \g_block_debug_bool
```

(End of definition for \g_block_debug_bool.)

`\_block_debug:n` Put debugging info in the code, displayed or not displayed depending on the value in `\g_block_debug_bool`.

```
666 \cs_new_eq:NN \_block_debug:n \use_none:n
667 \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:` Changing the debugging status.

`\block_debug_off:`

```
\_block_debug_gset:
668 \cs_new_protected:Npn \block_debug_on:
669 {
670   \bool_gset_true:N \g_block_debug_bool
671   \_block_debug_gset:
672 }
```

```
673 \cs_new_protected:Npn \block_debug_off:
```

```
674 {
675   \bool_gset_false:N \g_block_debug_bool
676   \_block_debug_gset:
677 }
```

```

678 \cs_new_protected:Npn \__block_debug_gset:
679 {
680   \cs_gset_protected:Npe \__block_debug:n ##1
681   { \bool_if:NT \g__block_debug_bool {##1} }
682   \cs_gset_protected:Npe \__block_debug_typeof:n ##1
683   { \bool_if:NT \g__block_debug_bool
684     { \iow_term:e { ^~J [Blocks]~ ==>~ ##1} } }
685 }

```

(End of definition for `\block_debug_on:`, `\block_debug_off:`, and `\__block_debug_gset:`. These functions are documented on page 38.)

`\DebugBlocksOn` If we are debugging blocks we also want to know about template instances, so we turn
`\DebugBlocksOff` the debugging for templates as well (for now).

```

686 \cs_new_protected:Npn \DebugBlocksOn { \block_debug_on: \DebugTemplatesOn }
687 \cs_new_protected:Npn \DebugBlocksOff { \block_debug_off: \DebugTemplatesOff }
688 \DebugBlocksOff

```

(End of definition for `\DebugBlocksOn` and `\DebugBlocksOff`. These functions are documented on page 38.)

`\DebugSwitchesOn` This debugs the use of legacy switches (so perhaps better called `\DebugLegacySwitchesOn`)
`\DebugSwitchesOff` but so far it was just a quick debugging aid while I was trying to understand. It needs
some further thoughts and is probably not necessary at all in the end.

```

689 \cs_new_protected:Npn \DebugSwitchesOn {
690   \AddToHookWithArguments{cmd/legacy_if_gset_false:n/before}[debug]
691   { \typeout{[Switch] ==>~ ##1~==false~(global)}}
692   \AddToHookWithArguments{cmd/legacy_if_gset_true:n/before}[debug]
693   { \typeout{[Switch] ==>~ ##1~==true~(global)}}
694   \AddToHookWithArguments{cmd/legacy_if_set_false:n/before}[debug]
695   { \typeout{[Switch] ==>~ ##1~==false}}
696   \AddToHookWithArguments{cmd/legacy_if_set_true:n/before}[debug]
697   { \typeout{[Switch] ==>~ ##1~==true}}
698 }
699 \cs_new_protected:Npn \DebugSwitchesOff {
700   \RemoveFromHook{cmd/legacy_if_gset_false:n/before}[debug]
701   \RemoveFromHook{cmd/legacy_if_gset_true:n/before}[debug]
702   \RemoveFromHook{cmd/legacy_if_set_false:n/before}[debug]
703   \RemoveFromHook{cmd/legacy_if_set_true:n/before}[debug]
704 }
705 %\DebugSwitchesOn
706 %\DebugSwitchesOff

```

(End of definition for `\DebugSwitchesOn` and `\DebugSwitchesOff`. These functions are documented on page ??.)

9.2 Template types and template interfaces

This section is devoted to the template interfaces, and the template code is covered later.

`blockenv (type)` All template types expect a first key-value argument used to tweak template parameters
`list (type)` at a specific point in the document for a single environment or command. The template
`captionedtext (type)` types `blockenv`, `list`, `captionedtext`, and `thmstyle` take three more arguments which
`thmstyle (type)` are a boolean for suppressing numbering, a possible caption, and a possible sub-caption.

```

block (type)
item (type)
para (type)
707 \NewTemplateType{blockenv}{4}
708 \NewTemplateType{list}{4}
709 \NewTemplateType{captionedtext}{4}
710 \NewTemplateType{thmstyle}{4}

711 \NewTemplateType{block}{1}
712 \NewTemplateType{item}{1}
713 \NewTemplateType{para}{1}

```

`blockenv std (templ.)`

```

714 \DeclareTemplateInterface{blockenv}{std}{4}
715 {
716   name                : tokenlist ,

```

If not explicitly set then `tag-name` and `tag-attr-class` are set by the `tagging-recipe`. However, we have to default both to `(empty)` so that nested blocks do not inherit from the outer level.

```

717 ,tag-name              : tokenlist =
718 ,tag-attr-class        : tokenlist =
719 ,tagging-recipe        : tokenlist = standard
720 ,transparent-level     : boolean   = false
721 ,early-legacy-code     : tokenlist =
722 ,late-legacy-code      : tokenlist =
723 ,block-instance        : tokenlist = std-display

```

Paragraph instance is normally inherited so no default.

```

724 ,para-instance         : tokenlist
725 ,inner-level-counter   : tokenlist
726 ,max-inner-levels      : tokenlist = 4
727 ,inner-instance-type   : tokenlist =
728 ,inner-instance        : tokenlist =
729 ,tagging-suppress-paras : boolean = false
730 ,final-code            : tokenlist = \ignorespaces
731 ,standalone            : boolean = false
732 }

```

`block std (templ.)`

```

733 \DeclareTemplateInterface{block}{std}{1}
734 {
735   ,begin-vspace         : skip = \topsep
736   ,begin-extra-vspace   : skip = \partopsep
737   ,begin-unchained-vspace : skip = .5\topsep
738   ,para-vspace          : skip = \parskip
739   ,end-vspace           : skip = \KeyValue{begin-vspace}
740   ,end-extra-vspace     : skip = \KeyValue{begin-extra-vspace}
741   ,item-vspace          : skip = \itemsep
742   ,begin-penalty        : integer = \UseName{@beginparpenalty}
743   ,end-penalty          : integer = \UseName{@endparpenalty}
744   ,item-penalty         : integer = \UseName{@itempenalty}

```

```

745 ,left-margin      : length = \leftmargin
746 ,right-margin     : length = \rightmargin
747 ,para-indent      : length = Opt
748 }

```

para std (*templ.*)

```

749 \DeclareTemplateInterface{para}{std}{1}
750 {
751   ,para-attr-class : tokenlist = justify
752   ,para-indent     : length = \parindent
753   ,begin-hspace    : skip = Opt
754   ,left-hspace     : skip = Opt
755   ,right-hspace    : skip = Opt
756   ,end-hspace      : skip = \@flushglue
757   ,fixed-word-spaces : boolean = false
758   ,final-hyphen-demerits : integer = 5000
759   ,newline-cmd     : function(0) = \@normalcr
760 }

```

list std (*templ.*)

```

761 \DeclareTemplateInterface{list}{std}{4}
762 {
763   ,counter      : tokenlist =
764   ,item-label   : tokenlist =
765   ,start        : integer = 1
766   ,resume       : boolean = false
767   ,item-instance : instance{item} = basic
768   ,item-vspace  : skip = \itemsep
769   ,item-penalty : integer = \UseName{@itempenalty}
770   ,item-indent  : length = \itemindent
771   ,label-width  : length = \labelwidth
772   ,label-sep    : length = \labelsep
773   ,legacy-support : boolean = false
774 }

```

item std (*templ.*)

```

775 \DeclareTemplateInterface{item}{std}{1}
776 {
777   ,counter-label : function{1} = \arabic{#1}
778   ,counter-ref   : function{1} = \KeyValue{counter-label}
779   ,label-ref     : function{1} = #1
780   ,label-autoref : function{1} = item~#1
781   ,label-format  : function{1} = #1
782   ,label-strut   : boolean = false
783   ,label-align   : choice {left,center,right,parleft} = right
784   ,label-boxed   : boolean = true
785   ,next-line     : boolean = false      % <- review viz standalone below
786   ,text-font     : tokenlist
787   ,compatibility : boolean = true
788   ,label-placement : choice {chained,unchained,standalone} = chained ,
789 }

```

`captionedtext thmlike (templ.)` The `captionedtext thmlike` template for theorem-like environments has only three keys because it delegates most of the work to the `thmstyle` template specified in the key `style`.

```

790 \DeclareTemplateInterface{captionedtext}{thmlike}{4}
791 {
792   ,counter      : tokenlist =
793   ,title        : tokenlist =      % <- bad name?
794   ,style        : instance{thmstyle} = plain
795 }

```

`captionedtext proof (templ.)` In contrast, the `captionedtext proof` template implements all of the `proof` environment without any delegation and therefore shows several keys for customizing the layout (similar to those seen with `thmstyle std`).

```

796 \DeclareTemplateInterface{captionedtext}{proof}{4}
797 {
798   ,title          : tokenlist = \proofname
799   ,punct          : tokenlist = .
800   ,caption-placement : choice {chained,unchained,standalone} = unchained
801   ,caption-unbreakable : boolean = false
802   ,before-hspace    : skip = 0pt
803   ,after-hspace     : skip = 5pt
804   ,caption-decls    : tokenlist =
805   ,title-decls      : tokenlist =
806   ,punct-decls      : tokenlist =
807   ,title-format     : function{1} = #1
808   ,punct-format     : function{1} = #1
809   ,body-decls       : tokenlist =
810 }

```

`\proofname` The `amsthm` package supported redefinition of `\proofname` as a means to alter default caption for proofs. We continue to support this as long as the instance declaration doesn't overwrite it.

```

811 \newcommand\proofname{Proof}

```

(End of definition for \proofname. This function is documented on page ??.)

`thmstyle std (templ.)`

```

812 \DeclareTemplateInterface{thmstyle}{std}{4}
813 {
814   ,numbered       : boolean = true
815   ,separator       : tokenlist = \
816   ,punct          : tokenlist = .
817   ,caption-placement : choice {chained,unchained,standalone} = unchained
818   ,caption-unbreakable : boolean = false
819   ,before-hspace    : skip = 0pt
820   ,after-hspace     : skip = 5pt
821   ,order           : commalist = { title, separator, number, punct, separator, note }
822   ,caption-decls    : tokenlist =
823   ,title-decls      : tokenlist =
824   ,number-decls     : tokenlist =
825   ,punct-decls      : tokenlist =
826   ,note-decls       : tokenlist =
827   ,title-format     : function{1} = #1
828   ,number-format    : function{1} = #1

```

```

829 ,punct-format      : function{1} = #1
830 ,note-format       : function{1} = (#1)
831 ,body-decls        : tokenlist  =
832 }

```

9.3 Implementation of templates

9.3.1 Some notes on the L^AT_EX 2_ε legacy switches

L^AT_EX 2_ε used a number of switches to manage its list environments and everything that was based on them.

For the reimplementation I made some notes about the original usage and how this got changed (while keeping the names for now).

Some of these switches really need to keep their names, e.g., `@nobreak` or `minipage`, because they are used all over the place. Others can probably be replaced with L3 booleans which makes things faster and cleaner, but for now I kept them too.

9.3.1.1 Original usage:

```

833 %
834 % @newlist (global):  signal that we are at the start of a list
835 %
836 % -> true   at the start of a list before the first item when control
837 %           is returned to document
838 % -> false  in everypar setting the first item
839 % -> false  at end of list if still true (after generating an error)
840 %
841 % -> tests: at list start setting @noparitemtrue and @noparlisttrue
842 %
843 %
844 % @inlabel (global): signaling that some item label waits to be typeset
845 %
846 % -> true   in \@item
847 % -> false  at list end
848 % -> false  in everypar after label has been typeset
849 % -> false  in \newpage after \leavevmode to typeset item label
850 %           (probably not needed)
851 %
852 % -> tests: at list start setting @noparitemtrue and @noparlisttrue
853 % -> tests: at list end to ensure that dangling label is typeset
854 % -> tests: in \@item to output a dangling item label by switching to hmode
855 % -> tests: in \everypar to output a dangling item label
856 % -> tests: in \newpage to output a dangling item label before the page is ended
857 % -> tests: in tagging hook {para/begin}{kernel}
858 %
859 %
860 % @noparlist (local):
861 %
862 % -> true   at start of list if already @inlabel=true
863 % -> false  at start of list otherwise
864 %
865 % -> tests: in \endtrivlist. If true suppress vertical spacing after the list
866 %
867 %

```

```

868 % @noparitem (local):
869 %
870 % -> true   at start of list if already @inlabel=true
871 % -> false
872 %

```

9.3.1.2 Repurpose:

```

873 %
874 % Interpret legacy switches as follows (keeping the names for now)
875 %
876 % @newlist   -> signals that we are at the start of a new block with a caption or
877 %              at the start of a list block expecting an item next
878 %
879 %              In other words this is now really start of a block
880 %              with inner structure.
881 %
882 % @noparlist -> signals that we are on a new block with @inlabel already true, i.e.,
883 %              and this placement should happen horizontally
884 %
885 % @inlabel   -> Signals that we have at least one item or caption waiting to be typeset
886 %              inside the label box
887 %
888 % @noparitem -> dropped (handled directly)
889 %

```

9.3.2 Implementation of blockenv templates

So far there is only one, but who knows ... — however, the majority will be vertically oriented blocks, so we make this the *std*.

`blockenv std (templ.)`

```

890 \DeclareTemplateCode{blockenv}{std}{4}
891 {
892   name                = \l__block_env_name_tl
893   ,tag-name           = \l__block_tag_name_tl
894   ,tag-attr-class     = \l__block_tag_class_tl
895   ,tagging-recipe     = \l__block_tagging_recipe_tl
896   ,transparent-level  = \l__block_transparent_level_bool
897   ,early-legacy-code  = \l__block_early_legacy_code_tl
898   ,late-legacy-code   = \l__block_late_legacy_code_tl
899   ,block-instance     = \l__block_block_instance_tl
900   ,para-instance     = \l__block_para_instance_tl
901   ,tagging-suppress-paras = \l__tag_para_flattened_bool
902   ,inner-level-counter = \l__block_inner_level_counter_tl
903   ,max-inner-levels   = \l__block_max_inner_levels_tl
904   ,inner-instance-type = \l__block_inner_instance_type_tl
905   ,inner-instance     = \l__block_inner_instance_tl
906   ,final-code         = \l__block_final_code_tl
907   ,standalone         = \l__block_standalone_bool
908 }
909 {
910   \template_debug_typeout:n{~\space template:~ 'std;~
911     arguments:~ \exp_not:n{|#1|#2|#3|#4|}}

```

For special legacy support we start with a hook that receives the `name` value as its argument (so that it contains code that is only executed for a specific name). Before calling this hook we also set `\UnusedTemplateKeys` to empty. In the hook code this macro can then be filled with key/value settings (for example, from `\setlist`) to be fed below into `\SetKnownTemplateKeys`.

```

912 \tl_set_eq:NN \UnusedTemplateKeys \c_empty_tl
913 \exp_args:Nno \hook_use:nnw {blockenv} 1 \l__block_env_name_tl

```

To the `\UnusedTemplateKeys` (empty or filled) we append any keys specified in the first argument to the template unless `#1` is empty or contains just `\NoValue`. This allows keys set on the document level to overwrite those set by `\setlist` and ultimately also those set in the instance.

```

914 \tl_if_empty:oF {#1}
915 {
916   \tl_if_novalue:nF {#1}
917   {

```

We expand `#1` once because the keys might be hidden in a user macro.

```

918     \tl_set:Ne \UnusedTemplateKeys
919     { \exp_not:o \UnusedTemplateKeys \exp_not:o { #1 } }
920   }
921 }

```

Finally the key list stored in `\UnusedTemplateKeys` is then fed to `\SetKnownTemplateKeys` for evaluation. `\SetKnownTemplateKeys` applies all keys that apply to the `blockenv` template and saves those that don't back in `\UnusedTemplateKeys` to be applied to inner instances below. This is why we had to combine all keys and evaluate them in one go.

```

922 \SetKnownTemplateKeys {blockenv} {std} \UnusedTemplateKeys

```

The tagging recipe `standalone` implies that the `standalone` key is set to `true`.

```

923 \tl_if_eq:NnT \l__block_tagging_recipe_tl { standalone }
924 { \bool_set_true:N \l__block_standalone_bool }

```

If this is a standalone environment we ensure that there is an implicit `\par` before and after:

```

925 \bool_if:NTF \l__block_standalone_bool
926 {
927   \__block_debug_endpe_typeout:n {===>~ standalone = true}

```

We can't simply issue a `\par` because the current environment may have set `tagging-suppress-paras` in which case a `\par` would not close the outer semantic para structure. We therefore temporarily set this boolean to false.

```

928   \bool_set_eq:NN \l__block_flattened_bool \l__tag_para_flattened_bool
929   \bool_set_false:N \l__tag_para_flattened_bool
930   \par
931   \bool_set_eq:NN \l__tag_para_flattened_bool \l__block_flattened_bool
932 }
933 {
934   \__block_debug_endpe_typeout:n {===>~ standalone = true}
935 }

```

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 `<semantic-para>` tag, while for any value above 1 we have to omit the `<semantic-para>`.

```

936 \int_compare:nNnTF \l__tag_block_flattened_level_int > 0
937 {
938   \int_incr:N \l__tag_block_flattened_level_int
939 }
940 {
941   \bool_if:NT \l__tag_para_flattened_bool
942   {
943     \int_incr:N \l__tag_block_flattened_level_int
944   }
945 }

946 \tl_if_empty:NF \l__block_inner_level_counter_tl
947 {
948   \int_compare:nNnTF \l__block_inner_level_counter_tl >
949     { \l__block_max_inner_levels_tl - 1 }
950     { \@toodeep }
951     { \int_incr:N \l__block_inner_level_counter_tl } % not clean "o"?
952 }

```

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.

```

953 \int_compare:nNnTF \g_block_nesting_depth_int >
954   { \c@maxblocklevels - 1 }
955   { \@toodeep }
956   {
957     \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.

If have a transparent level (e.g., something like a `center` environment) we omit setting the legacy defaults, because that is the way $\text{\LaTeX} 2_\epsilon$ lists worked as well.

```

958 \bool_if:NF \l__block_transparent_level_bool
959 {
960   \use:c { @list
961     \int_to_roman:n { \g_block_nesting_depth_int } }
962 }
963 }

```

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

```

964 \UseTaggingSocket{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 overwritten 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 overwritten by the legacy setup code for the `list` environment in `\l__block_early_legacy_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.

```

965 \tl_clear:N \@itemlabel
966 \tl_clear:N \@listctr
967 \legacy_if_set_false:n { @nmbrlist }

```

Then run the legacy setup code if any is given in the instance. We may still be in horizontal mode at this point so anything affecting paragraph parameters would alter preceding text!

```

968 \l_block_early_legacy_code_tl

```

Next call a block instance at the appropriate level passing it any remaining key/value from the optional document-level argument (i.e., those now stored in `\UnusedTemplateKeys`).

```

969 \exp_args:Nee \UseInstance {block}
970     { \l_block_block_instance_tl - \int_use:N
971       \g_block_nesting_depth_int }
972     \UnusedTemplateKeys

```

After this instance has been processed, any remaining unused keys are stored in `\UnusedTemplateKeys` and we can make use of this data later as long as we do not call another instance that also does unused key processing and overwrites it. But this is what happens below, so we better save its current value for now.

```

973 \tl_set_eq:NN \l_block_unused_blockenv_keys_tl \UnusedTemplateKeys

```

After the block instance call the para and then inner (list) instance if either or both are specified (which may not be the case).

```

974 \tl_if_empty:NF \l_block_para_instance_tl
975 {

```

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

```

976     \exp_args:Nee \UseInstance {para}{ \l_block_para_instance_tl } {}
977 }

```

At this point we are in vertical mode in a display environment and we execute legacy setup code that should not appear in horizontal mode.

```

978 \l_block_late_legacy_code_tl

```

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.

```

979 \tl_if_empty:NF \l_block_inner_instance_tl
980 {

```

We expand the first two arguments so that we get proper names for template type and instance, because `\UseInstance` is not doing that for us in the right way.

```

981     \exp_args:Nee
982     \UseInstance{ \l_block_inner_instance_type_tl }
983     { \l_block_inner_instance_tl
984       \tl_if_empty:NF \l_block_inner_level_counter_tl
985         % not clean use "o"?
986         { - \int_use:N \l_block_inner_level_counter_tl }
987     }
988     \l_block_unused_blockenv_keys_tl
989     #2      % <-- \BooleanTrue or False

```

```

990             { #3 }      % <-- \NoValue or content
991             { #4 }      % <-- \NoValue or content

```

Again the instance may have processed a few keys from the so far unused keys, so we update `\l__block_unused_blockenv_keys_tl` to match the new reality.

```

992     \tl_set_eq:NN \l__block_unused_blockenv_keys_tl \UnusedTemplateKeys
993 }

```

At this point, the `\l__block_unused_blockenv_keys_tl` token list should either be empty or it should contain only keys that are suitable for the item template, but right now there is no code to test that can test the latter; it would help probably if we have an interface for this.

For now we handle that when the first item is encountered, but that isn't really clean.

```

994 % \tl_if_empty:NF \l__block_unused_blockenv_keys_tl
995 % {
996 %     % check if only item template keys remain
997 % }

```

If this is supposed to be a transparent block environment then we have to decrement the nesting level again so that nested environments think nothing is there.

```

998 \bool_if:NT \l__block_transparent_level_bool
999 { \int_gdecr:N \g_block_nesting_depth_int }

```

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.

```

1000 \l__block_final_code_tl
1001 }

```

`\l__block_flattened_bool` Temp variable for standalone handling.

```

1002 \bool_new:N \l__block_flattened_bool

```

(End of definition for `\l__block_flattened_bool`.)

blockenv (*hook*) For legacy support we want a hook with one argument (the name of the `blockenv` instance). This way code could be added that is conditional based on which instance is executed.

```

1003 \NewHookWithArguments{blockenv}{1}

```

\BlockEnv To simplify the environment declarations later we provide two simple commands that invoke a `blockenv` instance. The matching counterpart to these commands is `\BlockEnvEnd` (defined below) that carries out all necessary action when a block environment ends.

\SimpleBlockEnv

```

1004 \cs_new_protected:Npn \BlockEnv          % #1#2#3#4 implicit
1005 { \UseInstance{blockenv} }

```

This here is the most common one that hides arguments 2–4 when they aren't needed, e.g., in a `center` environment.

```

1006 \cs_new_protected:Npn \SimpleBlockEnv #1#2
1007 { \UseInstance{blockenv}{#1}{#2} \BooleanFalse \NoValue \NoValue }

```

(End of definition for `\BlockEnv` and `\SimpleBlockEnv`. These functions are documented on page 39.)

\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`:

```
1008 \cs_new_protected:Npn \g_block_nesting_depth_int { \@listdepth } % a fake int
1009 % for now
```

(End of definition for `\g_block_nesting_depth_int`. This function is documented on page 39.)

\l_block_unused_blockenv_keys_tl The token list that holds key values we haven't yet used while we are processing the instances in a block environment.

```
1010 \tl_new:N \l_block_unused_blockenv_keys_tl
```

(End of definition for `\l_block_unused_blockenv_keys_tl`.)

\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

```
1011 \int_if_exist:NF \l_tag_block_flattened_level_int
1012 {
1013   \int_new:N \l_tag_block_flattened_level_int
1014 }
```

(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.

```
1015 \newcounter{maxblocklevels}
1016 \setcounter{maxblocklevels}{6}
```

(End of definition for `\c@maxblocklevels`. This function is documented on page 40.)

\BlockEnvEnd 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.

```
1017 \cs_new_protected:Npn \BlockEnvEnd {
1018   \__block_debug_typeout:n{blockenv~ common~ ending \on@line}
```

If this block is not a transparent one we have to decrement the level now again, otherwise that had happened earlier:

```
1019   \bool_if:NF \l_block_transparent_level_bool
1020     { \int_gdecr:N \g_block_nesting_depth_int }
```

If the `@inlabel` switch is true, i.e., if there is a caption or an item waiting to be placed we move to horizontal mode to get them typeset.

```
1021   \legacy_if:nT { @inlabel }
1022   {
1023     \mode_leave_vertical:
1024     \legacy_if_gset_false:n { @inlabel }
1025   }
```

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.

```

1026 \__block_if_list:TF
1027 { \legacy_if:nT { @newlist } { \noitemerr } }

```

If we aren’t in a list we check if there are still unused keys and in that case generate an error. In lists that is already done at each `\item` command.

```

1028 {
1029   \tl_if_empty:NF \UnusedTemplateKeys
1030   {
1031     \msg_error:nneo { block } { unknown-keys }
1032     { \l__block_env_name_tl \space environment}
1033     \UnusedTemplateKeys
1034   }
1035 }

1036 \mode_if_horizontal:TF
1037 { \__block_skip_remove_last: \__block_skip_remove_last: \par }
1038 { \inmatherr{\end{\@currenvir}} }

```

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

```

1039 \__kernel_displayblock_end:

```

Resetting the `@newlist` switch is also only done if the current environment is a list.

```

1040 \__block_if_list:T { \legacy_if_gset_false:n { @newlist } }

```

There is a possibility that the `@nobreak` switch is still true so we set it back just in case.

```

1041 \legacy_if_gset_false:n { @nobreak }

```

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 environments have been doing.

```

1042 % \__block_debug_typeout:n{@nparlist =
1043 % \legacy_if:nTF { @nparlist }{true}{false}}
1044 \legacy_if:nF { @nparlist }
1045 {
1046   \__block_skip_set_to_last:N \l_tmpa_skip
1047   \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
1048   {
1049     \skip_vertical:n { - \l_tmpa_skip }
1050     \skip_vertical:n { \l_tmpa_skip + \parskip - \@outerparskip }
1051   }
1052   \addpenalty \@endparpenalty
1053   \addvspace \l__block_effective_bot_skip

```

some redesign/extensions here?

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

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.

```
1054 % \legacy_if_gset_true:n { @endpe }
1055 }
```

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

Finally, we have the code that does the `\par` handling after the block. Normally, we continue the semantic paragraph, i.e., set `@endpetrue` but there are some exceptions: If this is a standalone environment we need to issue a `\par`, but that has to happen after the environment in case the current `blockenv` has a special definition for `\par`.

```
1056 \bool_if:NT \l_block_standalone_bool { \aftergroup \par }
```

If the `tagging-recipe` is `noop` we do not set `@endpe` at all so its value is whatever it was set to in the body of the environment (which may have ended in a list, for example).

```
1057 \tl_if_eq:NnF \l_block_tagging_recipe_tl { noop }
1058 {
```

If the `tagging-recipe` is `standalone` we set it to false. This is necessary because the `\par` is put in with `\aftergroup` and there is a subtle racing condition with the way the `@endpe` setting propagates out of groups (also using `\aftergroup`. As a result it would be again true after this implicit empty line resulting in havoc.

```
1059 \tl_if_eq:NnTF \l_block_tagging_recipe_tl { standalone }
1060 \endpefalse
```

For all other recipes we can safely set it to true.

```
1061 \endpetrue
1062 }
1063 }
```

(End of definition for `\BlockEnvEnd`. This function is documented on page 39.)

```
\__block_if_list:T
\__block_if_list:TF
```

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 as `\items` are only supported if the `inner-instance-type` is `list`.

```
1064 \cs_new:Npn \__block_if_list:T
1065 { \tl_if_eq:NnT \l_block_inner_instance_type_tl {list} }
1066 \cs_new:Npn \__block_if_list:TF
1067 { \tl_if_eq:NnTF \l_block_inner_instance_type_tl {list} }
```

(End of definition for `\__block_if_list:T` and `\__block_if_list:TF`.)

```
\__kernel_displayblock_end:
```

The kernel hook for tagging at the end of the block. Without tagging it executes the `item/after` hook if inside a list, with active tagging it is overwritten by other commands in the recipes.

```
1068 \cs_new_protected:Npn \__kernel_displayblock_end: {
1069 \__block_if_list:T\hook_use:n{item/after}}
1070 \__block_debug_typeout:n{\detokenize{\__kernel_displayblock_end:}}
1071 }
```

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

9.3.3 Implementation of para templates

para std (*templ.*)

```
1072 \DeclareTemplateCode{para}{std}{1}
1073 {
1074   ,para-indent          = \parindent
```

The next parameter needs integrating in the basic paragraph handling (not done yet) and it should therefore probably a public name like the rest.

```
1075   ,begin-hspace         = \l_para_begin_skip
1076   ,left-hspace          = \leftskip
1077   ,right-hspace         = \rightskip
1078   ,end-hspace           = \parfillskip
```

Next isn't yet implemented (and the variable name is wrong).

```
1079   ,fixed-word-spaces    = \l__par_fixed_word_spaces_bool % name??
1080   ,final-hyphen-demerits = \finalhyphendemerits
1081   ,newline-cmd          = \\
1082   ,para-attr-class      = \l__tag_para_attr_class_tl
1083 }
1084 {
1085   \template_debug_typeout:n{~\space template:~ 'std';~
1086                               argument:~ \exp_not:n{|#1|}}
1087   \SetTemplateKeys{para}{std}{#1}

1088   \skip_set:Nn \@rightskip \rightskip
1089 }
```

`\__para_handle_indent:` We insert `\l_para_begin_skip` directly in front of the indentation box. This way it is hidden from any special setting of `\everypar` (whether that is used to remove the indentation box or whether it attempts to do something with the first token(s) of the paragraph). However, we only insert it if it differs from 0.0pt to avoid adding `\penalty 10000 \glue 0.0` all over the place.

```
1090 \tl_const:Nc \c_zero_skip_tl { \skip_use:N \z@skip }
1091 \tl_new:N \l__para_begin_skip_tl

1092 \cs_set:Npn \__para_handle_indent: {
1093   \tl_set:Nc \l__para_begin_skip_tl { \skip_use:N \l_para_begin_skip }
1094   \if_meaning:w \l__para_begin_skip_tl
1095     \c_zero_skip_tl
1096   \else:
1097     \nobreak
1098     \tex_hskip:D \l_para_begin_skip
1099   \fi:
1100   \box_use_drop:N \g_para_indent_box
1101 }
```

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

`\para_raw_noindent:` `\para_raw_noindent:` doesn't call `\__para_handle_indent:` so we have to manually do the `\l_para_begin_skip` handling.

```
1102 \cs_set:Npn \para_raw_noindent: {
1103   \mode_if_vertical:TF
1104   {
```

```

1105     \tex_everypar:D {
1106         \tex_everypar:D { \g__para_standard_everypar_tl }
1107         \tl_set:Nc \l__para_begin_skip_tl { \skip_use:N \l_para_begin_skip }
1108         \if_meaning:w \l__para_begin_skip_tl
1109             \c__zero_skip_tl
1110         \else:
1111             \nobreak
1112             \tex_hskip:D \l_para_begin_skip
1113         \fi:
1114         \the\everypar }
1115     }
1116     { \msg_error:nn { latex2e }{ raw-para } }
1117 \tex_noindent:D
1118 }

```

(End of definition for `\para_raw_noindent::`. This function is documented on page ??.)

9.3.4 Implementation of block templates

`block std (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.

```

1119 \DeclareTemplateCode{block}{std}{1}
1120 {
1121     ,begin-vspace           = \topsep
1122     ,begin-extra-vspace     = \partopsep
1123     ,begin-unchained-vspace = \l__block_unchained_skip
1124     ,para-vspace           = \parsep

```

fix

The bottom skips aren't used yet, even if set instead as before `\topsep` is applied there.

```

1125     ,end-vspace             = \l__block_botsep_skip
1126     ,end-extra-vspace       = \l__block_parbotsep_skip
1127     ,item-vspace           = \itemsep
1128     ,begin-penalty          = \@beginparpenalty
1129     ,end-penalty            = \@endparpenalty
1130     ,item-penalty           = \@itempenalty
1131     ,right-margin           = \rightmargin
1132     ,left-margin            = \leftmargin
1133     ,para-indent            = \l__block_parindent_dim
1134 }
1135 {
1136     \template_debug_typeout:n{~\space template:~ 'std';~
1137         argument:~ \exp_not:o{\exp_after:wN |#1|}}
1138     \SetKnownTemplateKeys{block}{std}{#1}

```

One aspect that needs a different handling is the management of `\par` after a block environment. If there is no empty line the following text is considered to be a continuation of the current paragraph. To manage this both `\par` and `\everypar` are redefined with the latter removing the paragraph indentation from the next paragraph. But if an empty line was seen after the block then the redefined `\par` would cancel the redefinition of `\everypar` so that the next paragraph again had its indentation. The other case in which the following indentation should not get suppressed is when two block environments directly follow each other. In the original L^AT_EX 2_ε implementation that simply worked because the `\item` command canceled any `\everypar` and all block environments were

implemented as “lists”, i.e., contained an `\item`, even if it was only there to get the vertical spacing right. With the new block implementation this is no longer the case, so we have handle the cancellation ourselves.

We can’t simply use `\mode_if_vertical:T { \par }` because that would also end the semantic paragraph. Instead we check if `@endpe` handling is requested and if so reset `\everypar`.

```
1139 \legacy_if:nT { @endpe } { \everypar{} }
```

The remaining code largely follows the logic of $\text{\LaTeX} 2_{\epsilon}$ ’s `trivlist` implementation as far as it applicable for the “display block” but coded using the L3 programming layer. However, we keep most of the legacy variables (e.g., `@noskipsec`) if there is some chance that they are set/used in classes or packages.

```
1140 \legacy_if:nTF { @noskipsec }
```

A `@noskipsec` heading is a heading that is placed in the same line as the following text (using `\everypar`) but not if that text starts with a display block, so we ensure that the heading gets typeset now.

```
1141 { \mode_leave_vertical: }
```

If no such heading is waiting we might have a block caption waiting to be typeset and this might be requested to be set “unchained”. In that case we also have to ensure that this gets typeset now.

The situation is slightly different though, because we want to end in vertical mode in that case also add some special vertical space and have to properly deal with avoiding page breaks.

```
1142 {
1143   \bool_if:NT \g__block_label_unchained_bool
1144   {
1145     \__block_debug_typeout:n{Set~ captioned~ block~ everypar \on@line }
1146     \cs_set_eq:NN \__block_everypar: \__block_captioned_everypar_std:
1147     \legacy_if:nT { @inlabel }
1148     {
1149       \hbox_unpack_drop:N \g__block_labels_box
1150       \legacy_if_gset_false:n { @inlabel }
1151       \par
1152       \nobreak
1153       \skip_vertical:n { \l__block_unchained_skip }
1154       \legacy_if_gset_true:n { @nobreak }
1155     }
1156   }
1157 }
```

The variable `\l__block_effective_top_skip` is used for the vertical skip at the start. It is initialized with `begin-vspace` and below we add `begin-extra-vspace` if the block starts in vmode and is not part of an ongoing semantic paragraph.

```
1158 \skip_set:Nn \l__block_effective_top_skip { \topsep }
```

For the vertical space at the end we do something similar.

```
1159 \skip_set:Nn \l__block_effective_bot_skip { \l__block_botsep_skip }
1160 \mode_if_vertical:TF
1161 {
```

This is similar to the standalone case for block captions so perhaps that can be combined, check

If `@endpe` is `true` then we are in the situation that a display environment wants to continue the current semantic paragraph, so we do not add `\partopsep` or `\l__block_parbotsep_skip` in that case.

```

1162     \legacy_if:nF { @endpe }
1163     {
1164         \skip_add:Nn \l__block_effective_top_skip { \partopsep }

1165         \skip_add:Nn \l__block_effective_bot_skip { \l__block_parbotsep_skip }
1166     }

```

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.

```

1167     \__kernel_displayblock_beginpar_vmode:
1168 }
1169 {

```

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`.

```

1170     \__block_skip_remove_last: \__block_skip_remove_last:
1171     \__kernel_displayblock_beginpar_hmode:w \par
1172 }

```

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

```

1173     \skip_zero:N \leftskip
1174     \skip_set_eq:NN \rightskip \@rightskip
1175     \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.

```

1176     \int_zero:N \par@deathcycles
1177     \@setpar
1178     {
1179         \legacy_if:nTF { @newlist }
1180         {
1181             \int_incr:N \par@deathcycles
1182             \int_compare:nNnTF \par@deathcycles > { 1000 }
1183             { \@noitemerr
1184               { \para_end: }
1185             }
1186         }
1187         {
1188             { \para_end: }
1189         }
1190     }

```

```

1191 \dim_set_eq:NN \parindent \l__block_parindent_dim
1192 \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
1193 \dim_add:Nn \@totalleftmargin { \leftmargin }
1194 \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.

```

1195 \__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 `@inlabel`, `minipage`, and `@nobreak`.

Now we are back to legacy list implementation ...

```

1196 \skip_set_eq:NN \@outerparskip \parskip
1197 \skip_set_eq:NN \parskip \parsep
1198 %
1199 \legacy_if:nTF { @inlabel }
1200 {
1201   \legacy_if_set_true:n { @nparlist }
1202   \hbox_gset:Nn \g__block_labels_box
1203   {
1204     \skip_horizontal:n { - \leftmargin }
1205     \hbox_unpack_drop:N \g__block_labels_box
1206     \skip_horizontal:n { \leftmargin }
1207   }
1208   \legacy_if:nF { @minipage } % Why this chunk of code?
1209   {
1210     \__block_skip_set_to_last:N \l__block_tmpa_skip
1211     \skip_vertical:n { - \l__block_tmpa_skip }
1212     \skip_vertical:n { \l__block_tmpa_skip +
1213                       \@outerparskip - \parsep }
1214   }
1215 }
1216 {
1217   \legacy_if_set_false:n { @nparlist }
1218   \legacy_if:nT { @newlist } { \@noitemerr }
1219   \legacy_if:nTF { @nobreak }
1220   {

```

We are not resetting `@nobreak` here as it should also apply to the upcoming item.

```

1221   \addpenalty{ 10000 }
1222   \addvspace{ \skip_eval:n{\@outerparskip-\parsep} }
1223 }
1224 {
1225   \addpenalty \@beginparpenalty
1226   \addvspace { \skip_eval:n { \l__block_effective_top_skip +
1227                               \@outerparskip } }
1228   \addvspace { - \parsep }
1229 }
1230 }
1231 }

```

document 2e logic used
here

`\_block_captioned_everypar_std:` The captioned text is typeset at the start of a paragraph using code triggered in `\everypar` (by setting `\_block_everypar` to this code here).

```
1232 \cs_new_protected:Npn \_block_captioned_everypar_std: {
1233     \_block_debug_typeout:n{...~ in~ captioned~ block~ everypar \on@line }
```

First set some control switches to false:

```
1234     \legacy_if_set_false:n { @minipage }
1235     \legacy_if_gset_false:n { @newlist }
```

The `@inlabel` is normally true at this point, but if we also have `@nobreak` then the same routine is called again at the next paragraph to reset `\clubpenalty` and at that point the `\g_block_labels_box` has been typeset and `@inlabel` is false.

```
1236     \legacy_if:nT { @inlabel }
1237     {
```

Typeset the saved label (aka captioned text):

```
1238         \legacy_if_gset_false:n { @inlabel }
1239         \para_omit_indent:
1240         \bool_if:NTF \l_block_caption_unbreakable_bool
1241             \box_use_drop:N
1242             \hbox_unpack_drop:N
1243         \g_block_labels_box
1244         \_kernel_list_label_after:n { \PARALABEL }      % <- change
1245                                                         % this name
1246         \penalty \c_zero_int
1247     }
```

If `@nobreak` is true we prevent a break after the first line by setting `\clubpenalty`.

```
1248     \legacy_if:nTF { @nobreak }
1249     {
1250         \legacy_if_gset_false:n { @nobreak }
1251         \int_set:Nn \clubpenalty { 10000 }
1252     }
1253     {
```

Otherwise we reset `\clubpenalty` and disable `\_block_everypar`.

```
1254         \int_set_eq:NN \clubpenalty \@clubpenalty
1255         \_block_debug_typeout:n{Set~ noop~ block~ everypar \on@line }
1256         \cs_set_eq:NN \_block_everypar: \prg_do_nothing:
1257     }
1258 }
```

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

`\_kernel_displayblock_begin:`
`\_kernel_displayblock_beginpar_hmode:w`
`\_kernel_displayblock_beginpar_vmode:`

The internal kernel hooks for tagging.

```
1259 \cs_new_protected:Npn \_kernel_displayblock_begin: {
1260     \_block_debug_typeout:n
1261     {\detokenize{\_kernel_displayblock_begin:}}
1262 }

1263 \cs_new_protected:Npn \_kernel_displayblock_beginpar_hmode:w {
1264     \_block_debug_typeout:n
1265     {\detokenize{\_kernel_displayblock_beginpar_hmode:w}}
1266 }
```

```

1267 \cs_new_protected:Npn \__kernel_displayblock_beginpar_vmode: {
1268   \__block_debug_typeout:n
1269   {\detokenize{\__kernel_displayblock_beginpar_vmode:}}
1270 }

```

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

9.3.5 Implementation of list templates

This list is one of the template types that can be used as an inner-type in a `blockenv`; the other one currently implemented is `captionedtext`.

`\@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`.

```

1271 \tl_new:N \@itemlabel      % should have a top-level definition
1272 \tl_new:N \@listctr       % should have a top-level definition

```

(End of definition for `\@itemlabel` and `\@listctr`. These functions are documented on page 40.)

`\__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 `\SetKnownTemplateKeys` 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.

```

1273 \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.

```

1274 %\cs_new:Npn \__block_save_user_keys:n #1 {
1275 %  \tl_if_empty:nTF {#1}
1276 %    { \cs_set_eq:NN \__block_evaluate_saved_user_keys:nn \use_none:nn }
1277 %    { \cs_set:Npe \__block_evaluate_saved_user_keys:nn ##1##2
1278 %      { \SetKnownTemplateKeys{##1}{##2}{\exp_not:n{#1}} } }
1279 %}

```

(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.

```

1280 \DeclareTemplateCode{list}{std}{4}
1281 {
1282   ,counter      = \l__block_counter_tl
1283   ,item-label   = \l__block_item_label_tl
1284   ,start       = \l__block_counter_start_int
1285   ,resume      = \l__block_resume_bool
1286   ,item-instance = \__block_item_instance:n

```

```

1287 ,item-vspace      = \itemsep
1288 % ,item-para-vspace = \parsep
1289 ,item-penalty      = \@itempenalty
1290 ,item-indent       = \itemindent
1291 ,label-width       = \labelwidth
1292 ,label-sep         = \labelsep
1293 ,legacy-support    = \l__block_legacy_support_bool % FMi questionable
1294 }
1295 {
1296   \template_debug_typeout:n{~\space template:~ 'std';~
1297                                     arguments:~ \exp_not:o{\exp_after:wN |#1|#2|#3|#4|}}

```

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`.

```

1298   \tl_if_empty:oTF {#1}
1299   { \cs_set_eq:NN \__block_evaluate_saved_user_keys:nn \use_none:nn }
1300   {
1301     \SetKnownTemplateKeys{list}{std}{#1}

```

The setup for `\__block_evaluate_saved_user_keys:nn` is a bit tricky and has to be done with `\cs_set:Npe` even though we don't want to expand anything and therefore use `\exp_not:n` inside. All this does is that any # passed in via #1 is doubled (e.g., from `label-format=\fbox{#1}` which is represented as `...\fbox{##1}`). Otherwise, we would end up with a replacement text like

```
\SetTemplateKeys {#1}{#2}{label-format=\fbox {#1}}
```

instead of

```
\SetTemplateKeys {#1}{#2}{label-format=\fbox {##1}}
```

resulting in very odd and puzzling behavior later on.

The definition of `\__block_evaluate_saved_user_keys:nn` made here is later used when an `\item` is processed and passes remaining keys to the item instance. After that nothing should remain, so we test that and issue an error if not.

```

1302   \cs_set:Npe \__block_evaluate_saved_user_keys:nn ##1##2
1303   { \SetKnownTemplateKeys{##1}{##2}{
1304     \exp_not:o { \UnusedTemplateKeys }
1305   }
1306   \exp_not:n {
1307     \tl_if_empty:NF \UnusedTemplateKeys
1308     {
1309       \msg_error:nneo { block } { unknown-keys }
1310       { \l__block_env_name_tl \space environment}
1311       \UnusedTemplateKeys
1312     }
1313   }
1314 }
1315 }

```

Has this list a counter name defined in the instance?

```

1316 \tl_if_empty:NTF \l__block_counter_tl
1317 {

```

If no counter name has been specified as part of the instance setup the list might still be numbered if it is a legacy list that uses `\usecounter` in the second argument of the legacy `list` environment. However, in that case we don't have to do much because `\usecounter` sets up `\@listctr` and sets it to zero so that the first item is numbered 1.

So all we do is to check if there was a `start` value given that differs from 1 and if so we change the counter value to match that. This makes it possible to define a legacy `list` in which the counter doesn't start with 1 by explicitly setting the counter value in the second argument of the `list` environment but also overwriting that through a `start` key setting on invocation.

```

1318 \int_compare:nNf \l__block_counter_start_int = 1
1319 {
1320     \int_gset:cn{ c@ \@listctr }
1321     { \l__block_counter_start_int - 1 }
1322 }
1323 }

```

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).

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.

```

1324 {
1325     \@nmbrlisttrue
1326     \tl_set_eq:NN \@listctr \l__block_counter_tl
1327     \bool_if:NF \l__block_resume_bool
1328     {
1329         \int_gset:cn{ c@ \@listctr }
1330         { \l__block_counter_start_int - 1 }
1331     }
1332 }

```

Does the current instance have 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.

```

1333 \tl_if_empty:NF \l__block_item_label_tl
1334 {
1335     \tl_set_eq:NN \@itemlabel \l__block_item_label_tl
1336 }

```

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).

```

1337 \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 set up `\__block_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).

Think about a better implementation at some point.

```
1338   \__block_debug_typeout:n{Set~ first~ block~ everypar \on@line }
1339   \cs_set_eq:NN \__block_everypar: \__block_item_everypar_first:

1340   \__block_debug_typeout:n{template:list:std-end}
1341 }
```

The message that is used above when we are left with keys that are unknown:

```
1352 \msg_new:nnnn { block } { unknown-keys }
1353 { Some~ keys~ specified~ on~ the~ #1~ are~ unknown. }
1354 {
1355   The~ following~ keys~ are~ unknown~ and~ their~
1356   values~ are~ ignored:\\
1357   \space\space \exp_not:n {#2}\\
1358   Perhaps~ a~ misspelling~ or~ the~ current~ template~
1359   instance~ uses~ special~ keys.
1360 }
```

We need a special variant to expand `\UnusedTemplateKeys` once when passed as the last argument as the key values can contain stuff that doesn't react nicely under expansion.

```
1351 \cs_generate_variant:Nn \msg_error:nnnn {nneo}
```

9.3.6 Implementation of item templates

`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.

```
1352 \keys_define:nn { template/item/std }
1353   { label .tl_set:N = \l__block_label_given_tl }

1354 \DeclareTemplateCode{item}{std}{1}
1355 {
1356   ,counter-label   = \__block_counter_label:n
1357   ,counter-ref     = \__block_counter_ref:n

1358   ,label-ref       = \__block_label_ref:n
1359   ,label-autoref   = \__block_label_autoref:n
1360   ,label-format    = \__block_label_format:n
1361   ,label-strut     = \l__block_label_strut_bool
1362   ,label-boxed     = \l__block_label_boxed_bool
1363   ,next-line       = \l__block_next_line_bool
1364   ,text-font       = \l__block_text_font_tl
1365   ,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)

```
1366   ,label-align     = {
1367     left   = \tl_set:Nn \l__block_item_align_tl { \relax \hss } ,
1368     center = \tl_set:Nn \l__block_item_align_tl { \hss \hss } ,
1369     right  = \tl_set:Nn \l__block_item_align_tl { \hss \relax } ,
```

```

1370     parleft = \NOT_IMPLEMENTED ,
1371 }

```

The keylabel-placement is implemented using two booleans (at the moment).

```

1372     ,label-placement = {
1373         chained      = \bool_gset_false:N \g__block_label_standalone_bool
1374                       \bool_gset_false:N \g__block_label_unchained_bool ,
1375         unchained    = \bool_gset_false:N \g__block_label_standalone_bool
1376                       \bool_gset_true:N  \g__block_label_unchained_bool ,
1377         standalone   = \bool_gset_true:N  \g__block_label_standalone_bool
1378                       \bool_gset_false:N \g__block_label_unchained_bool ,
1379     }
1380 }

```

Then typeset the label at its natural width by applying `\__block_make_label_box:n` to the label given or to a label constructed from the counter. If it is boxed and reasonably short, add padding to make it at least of size `\labelwidth`, then add another layer of box. This way, when we unpack it in `\g__block_labels_box` it correctly remains boxed in those cases. Afterwards, in the `nextline` case add `\newline` if the label did not fit in the allotted space.

```

1381 {
1382     \template_debug_typeout:n{~\space template:~ 'std';~
1383                               argument:~ \exp_not:n{|#1|}}

```

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.

```

1384     \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`. This generates an error if not all key can be used, but it should really do so only on the first item. TODO: improve error handling!

Then we do the same with all keys specified on this `\item` command (which may overwrite one or the other setting just made).

```

1385     \__block_evaluate_saved_user_keys:nn {item}{std}

```

We don't care whether all of the user keys from the list level have been applied, but those explicitly set on the `\item` command should be applicable, so we generate an error if that isn't the case:

```

1386     \SetKnownTemplateKeys{item}{std}{#1}
1387     \tl_if_empty:NF \UnusedTemplateKeys
1388     {
1389         \msg_error:nneo { block } { unknown-keys }
1390         { \noexpand\item command }
1391         \UnusedTemplateKeys
1392     }

```

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[]`.

```

1393     \tl_if_novalue:oTF \l__block_label_given_tl
1394     {

```

next line needs checking
after novalue implementa-
tion was changed

fix

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

```

1395 \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
1396 \bool_if:NTF \l__block_item_compatibility_bool % not sure that
1397 % conditional
1398 % makes sense
1399 { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}%
1400 \itemlabel } } % TODO ?
1401 { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}%
1402 \__block_counter_label:n { \@listctr } } }
1403 }
1404 {
1405 \__block_debug_typeout:n{item~ with~ optional}
1406 \__block_make_label_box:n {
1407 \MakeLinkTarget [\l__block_env_name_tl]{%
1408 \l__block_label_given_tl
1409 }
1410 }
1411 \bool_if:nT
1412 {
1413 \l__block_label_boxed_bool
1414 % TODO: is \linewidth correct?
1415 && \dim_compare_p:n
1416 { \box_wd:N \l__block_one_label_box <= \linewidth }
1417 }
1418 {
1419 \dim_compare:nNnT
1420 { \box_wd:N \l__block_one_label_box } < \labelwidth
1421 {
1422 \hbox_set_to_wd:Nnn \l__block_one_label_box { \labelwidth }
1423 {
1424 \exp_after:wN \use_i:nn \l__block_item_align_tl

```

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

```

1425 % TODO: customize?
1426 % \hbox_unpack_drop:N \l__block_one_label_box
1427 \box_use_drop:N \l__block_one_label_box
1428 \exp_after:wN \use_ii:nn \l__block_item_align_tl
1429 }
1430 }

```

Add another box level to the label box:

```

1431 \hbox_set:Nn \l__block_one_label_box
1432 { \box_use_drop:N \l__block_one_label_box }
1433 }
1434 \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \labelwidth
1435 { \bool_set_true:N \l__block_long_label_bool }
1436 { \bool_set_false:N \l__block_long_label_bool }
1437 \hbox_gset:Nn \g_block_labels_box

```

```

1438 {
1439   \hbox_unpack_drop:N \g__block_labels_box
1440   \skip_horizontal:n { \itemindent - \labelsep - \labelwidth }
1441   \hbox_unpack_drop:N \l__block_one_label_box
1442   \skip_horizontal:n { \labelsep }
1443   \bool_if:NT \l__block_next_line_bool
1444     { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
1445   % version of \newline inside an hbox that will be unpacked
1446 }
1447 % TODO??? FMi what's that?
1448 % \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 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.

```

1449 \dim_compare:nNnF \listparindent = {0pt}
1450 { \dim_set_eq:NN \parindent \listparindent }

```

Placing the list label(s) is done when the paragraph for the `\item` is started, which executes `\__block_everypar`: inside `para/begin`. By default this command does nothing, now we change it to attach the pending label or labels.

```

1451 \__block_debug_typeout:n{Set~ item~ block~ everypar \on@line }
1452 \cs_set_eq:NN \__block_everypar: \__block_item_everypar_std:
1453 }

```

```

g__block_label_standalone_bool
g__block_label_unchained_bool

```

The two booleans for implementing label-placement and below caption-placement.

```

1454 \bool_new:N \g__block_label_standalone_bool % tmp until replaced
1455 \bool_new:N \g__block_label_unchained_bool % tmp until replaced

```

(End of definition for `g__block_label_standalone_bool` and `g__block_label_unchained_bool`.)

```
\l__block_item_align_tl
```

```
1456 \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.

```

1457 \box_new:N \l__block_one_label_box
1458 \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`.

```
1459 \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).

```
1460 \cs_new_protected:Npn \_block_make_label_box:n #1
```

```
1461 {
1462   \hbox_set:Nn \l_block_one_label_box
1463   {
```

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

```
1464     \tag_socket_use:nnn {block/list/label}{ }
1465     {
1466       \_block_label_format:n
1467       {
1468         \bool_if:NT \l_block_label_strut_bool { \strut }
1469         \bool_if:NTF \l_block_legacy_support_bool
1470           \makelabel
1471           \use:n
1472           {#1}
1473       }
1474     }
```

And what gets opened also needs closing:

```
1474   }
1475 }
1476 }
```

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

`block/list/label (socket)` A tagging socket to tag the label. It takes two arguments so that it can transparently pass the label content. Declaration is in `ltagging.dtx`.
 I think this socket should just be called `list/label`
 default (plug)

```
1477 \NewTaggingSocketPlug{block/list/label}{default}
1478 {
1479   %
1480   % FMI: this needs a different logic to decide when to make the label
1481   %   an artifact (after cleaning up the \item code ), therefore
1482   %   disabled for now
1483   % \tl_if_empty:oTF \@itemlabel
1484   % {
1485   %   \tag_mc_begin:n {artifact}
1486   % }
1487   % {
1488   \tagstructbegin{tag=\UseStructureName{block/list/label}}
1489   \tagmcbegin{tag=\UseStructureName{block/list/label}}
1490   % }
1491   #2
1492   \tagmcend % end mc-\UseStructureName{block/list/label} or artifact
1493   % FMI: unconditionally for now
1494   % \tl_if_empty:oF \@itemlabel
```

```

1495 \tagstructend % end label
1496 \tagstructbegin{tag=\UseStructureName{block/list/body}}
1497 }
1498 \AssignTaggingSocketPlug{block/list/label}{default}

```

`\_block\_everypar:` The `\_block\_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).

```

1499 \cs_new_eq:NN \_block\_everypar: \prg_do_nothing:
1500 \AddToHook{para/begin}[items]{\_block\_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.

```

1501 \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\_everypar:` is set to this by the item template so that the next paragraph start runs the code below.

```

1502 \cs_new_protected:Npn \_block\_item\_everypar\_std: {
1503   \_block\_debug\_typeout:n{...~ in~ item~ block~ everypar \on@line }
1504   \legacy\_if\_set\_false:n { @minipage }
1505   \legacy\_if\_gset\_false:n { @newlist }
1506   \legacy\_if:nT { @inlabel }
1507   {
1508     \legacy\_if\_gset\_false:n { @inlabel }
1509     \box\_if\_empty:NT \g\_para\_indent\_box { \kern - \itemindent }
1510     \para\_omit\_indent:

```

TODO: that boxing/unboxing need to be reevaluated at some point

```

1511   \bool\_if:NTF \l\_block\_next\_line\_bool
1512   {
1513     \hbox\_unpack\_drop:N \g\_block\_labels\_box
1514   }
1515   {
1516     \box\_use\_drop:N \g\_block\_labels\_box
1517   }

```

After the labels are placed we start a paragraph structure (if appropriate). This is handled in the following kernel hook:

```

1518   \_kernel\_list\_label\_after:n {LI-}
1519   \penalty \c\_zero\_int
1520 }
1521 \legacy\_if:nTF { @nobreak }
1522 {
1523   \legacy\_if\_gset\_false:n { @nobreak }
1524   \int\_set:Nn \clubpenalty { 10000 }
1525 }
1526 {
1527   \int\_set\_eq:NN \clubpenalty \@clubpenalty

```

Once the label(s) are typeset and we are past any special @nobreak handling we reset `\__block_everypar:` to do nothing.

```

1528         \__block_debug_typeout:n{Set~ noop~ block~ everypar \on@line }
1529         \cs_set_eq:NN \__block_everypar: \prg_do_nothing:
1530     }
1531 }

```

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

```

1532 \cs_new_protected:Npn \__block_item_everypar_first: {
1533     \__block_debug_typeout:n{...~ in~ first~ block~ everypar \on@line }
1534     \legacy_if:nT { @newlist } { \@noitemerr }
1535 }

```

(End of definition for `\__block_everypar:`, `\__block_item_everypar_std:`, and `\__block_item_everypar_first:`.)

`\l_block_tmpa_skip`

```

1536 \skip_new:N \l_block_tmpa_skip

```

(End of definition for `\l_block_tmpa_skip`.)

`\l_block_effective_top_skip`
`\l_block_effective_bot_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.

```

1537 \skip_new:N \l_block_effective_top_skip
1538 \skip_new:N \l_block_effective_bot_skip

```

(End of definition for `\l_block_effective_top_skip` and `\l_block_effective_bot_skip`.)

item/before (hook) We provide hooks at the begin and the end of an item. The before hook is just after the tagging code opens the LI structure (if tagging is active). The after hook is just before the tagging code closes the LI structure. The hooks are executed in vmode. In the item/before hook no typesetting should be done. It is possible to add content in the item/after but for correct tagging it must be wrapped in an LBody structure (e.g., with `tag=\UseStructureName{block/list/body}`).

```

1539 \NewHook{item/before}
1540 \NewHook{item/after}

```

\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}{\langle instance \rangle}`) to prepare the upcoming item: it will be actually inserted only once some later material triggers `\everypar`.

```

1541 \AddToHook{begindocument/before}[./legacy-lists]{
1542     \RenewDocumentCommand{\item}{*={label}o }
1543     {
1544         \@inmatherr \item

```

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

```

1545         \cs_if_free:NTF \__block_item_instance:n
1546         {
1547             \latex@error{Lonely~\string\item--perhaps~a~missing~
1548                 list~environment}\@ehc
1549         }
1550         {

```

If the previous `\item` ended in an environment requesting `@endpe` handling we have to ensure that a `\par` is seen so that the semantic paragraph surrounding that environment ends (as it can't extend to the next item).

```

1551         \if@endpe \par \fi
1552         \legacy_if:nTF { @newlist }
1553         {
1554             \__kernel_list_item_begin:
1555             \hook_use:n{item/before}

```

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

```

1556             \legacy_if_gset_false:n { @newlist }
1557         }
1558         { \__block_inter_item: }

```

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

```

1559         \tl_if_novalue:nTF {#1}          % avoids reparsing label={}
1560         { \__block_item_instance:n { } }
1561         { \__block_item_instance:n {#1} }

```

The `item` instance puts the item label into `\g__block_label_standalone_bool` ready to be placed later. To make that happen we need to signal that by setting the legacy switch `@inlabel` to true. However, if this is a label that should be always placed “standalone” we instead typeset it immediately and ensure that there is no page break after it.

support extra vertical space as well?

```

1562         \bool_if:nTF \g__block_label_standalone_bool
1563         {
1564             \bool_gset_false:N \g__block_label_standalone_bool
1565             \leavevmode
1566             \box_use_drop:N \g__block_labels_box
1567             \par
1568             \legacy_if_gset_true:n { @nobreak } % do not break after
1569                                                     % a standalone item
1570         }
1571         {
1572             \legacy_if_gset_true:n { @inlabel }
1573         }
1574         \ignorespaces
1575     }
1576 }
1577 }

```

(End of definition for `\item`. This function is documented on page 40.)

`\__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.

```

1578 \cs_new_protected:Npn \__block_inter_item: {
1579     \legacy_if:nT { @inlabel }
1580     { \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 not treat the next line as an else case to the above conditional (for now).

```

1581     \mode_if_horizontal:T { \__block_skip_remove_last:
1582                             \__block_skip_remove_last: \par }

```

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

```

1583 \hook_use:n{item/after}
1584 \__kernel_list_item_end:
1585 \__kernel_list_item_begin:
1586 \hook_use:n{item/before}

1587 \addpenalty \@itempenalty
1588 \addvspace \itemsep
1589 }

```

(End of definition for __block_inter_item:.)

```

\__kernel_list_item_begin:
\__kernel_list_item_end:
1590 \cs_new_eq:NN \__kernel_list_item_begin: \prg_do_nothing:
1591 \cs_new_eq:NN \__kernel_list_item_end: \prg_do_nothing:

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

```

9.3.7 Implementation of captionedtext and thmstyle templates

`captionedtext` `thmlike` (*templ.*) The template for typical theorem-like environments is rather trivial, just setting keys and then passing used keys and the arguments to a `thmstyle` instance to do the real work.

```

1592 \DeclareTemplateCode{captionedtext}{thmlike}{4}
1593 {
1594   ,counter      = \l__block_counter_tl
1595   ,title        = \l__block_title_tl
1596   ,style        = \l__block_style:nnnn
1597 }
1598 {

```

Some debugging info as usual (showing the arguments that are passed):

```

1599 \template_debug_typeout:n{~\space template:~ 'thmlike';~
1600                      arguments:~ \exp_not:o{\exp_after:wN |#1|#2|#3|#4|}}

```

Then we check if there are any keys passed to the instance from the outside.

```

1601 \SetKnownTemplateKeys{captionedtext}{thmlike}{#1}

```

Finally, we apply the style which is just an instance of type `thmstyle`:

```

1602 \l__block_style:nnnn \UnusedTemplateKeys {#2} {#3} {#4}
1603 }

```

```

\theoremstyle All that the \theoremstyle declaration does is saving its argument so that it can be
\l__block_thmstyle_tl used in \newtheorem.
1604 \cs_new_protected:Npn \theoremstyle #1{ \tl_set:Nn \l__block_thmstyle_tl {#1} }
1605 \tl_new:N \l__block_thmstyle_tl

```

And the default is `plain`:

```

1606 \theoremstyle{plain}

```

(End of definition for `\theoremstyle` and `\l__block_thmstyle_tl`. This function is documented on page ??.)

`thmstyle std (templ.)` The `thmstyle` implements the theorem-like environment and assumes that the fixed part of the caption is already stored in `\l__block_title_tl`. The reason for this separation into two templates is that typically the same design is used for different theorem-like environments only differing in this fixed string.

```

1607 \DeclareTemplateCode{thmstyle}{std}{4}
1608 {
1609   ,numbered      = \l__block_numbered_bool
1610   ,separator     = \l__block_separator_tl
1611   ,punct         = \l__block_punct_tl
1612   ,caption-placement = {
1613     chained      = \bool_gset_false:N \g__block_label_standalone_bool
1614                  \bool_gset_false:N \g__block_label_unchained_bool
1615     ,unchained    = \bool_gset_false:N \g__block_label_standalone_bool
1616                  \bool_gset_true:N  \g__block_label_unchained_bool
1617     ,standalone   = \bool_gset_true:N  \g__block_label_standalone_bool
1618                  \bool_gset_false:N \g__block_label_unchained_bool
1619   }
1620   ,caption-unbreakable = \l__block_caption_unbreakable_bool
1621   ,before-hspace      = \l__block_caption_before_skip
1622   ,after-hspace       = \l__block_caption_after_skip
1623   ,order              = \l__block_order_clist
1624   ,caption-decls      = \l__block_caption_decls_tl
1625   ,title-decls        = \l__block_title_decls_tl
1626   ,number-decls       = \l__block_number_decls_tl
1627   ,punct-decls        = \l__block_punct_decls_tl
1628   ,note-decls         = \l__block_note_decls_tl
1629   ,title-format       = \__block_title_format:n
1630   ,number-format      = \__block_number_format:n
1631   ,punct-format       = \__block_punct_format:n
1632   ,note-format        = \__block_note_format:n
1633   ,body-decls         = \l__block_body_decls_tl
1634 }
1635 {

```

Some tracing:

```

1636 \template_debug_typeout:n{~\space template:~ 'std';~
1637                  arguments:~ \exp_not:o{\exp_after:wN |#1|#2|#3|#4|}}

```

Applying any user keys:

```

1638 \SetKnownTemplateKeys{thmstyle}{std}{#1}

```

Since this is the last template that gets applied for theorem-like environments, all keys should make sense, so if something is left over we better generate an error:

```

1639 \tl_if_empty:NF \UnusedTemplateKeys
1640 {
1641   \msg_error:nneo { block } { unknown-keys }
1642   { \l__block_env_name_tl \space environment}
1643   \UnusedTemplateKeys
1644 }

```

In case there is a dangling `\item` we can either join that with the caption or we can output the item first. For now we provide no customization for this, but it could be made customizable.

```

1645 % \legacy_if:nT { @inlabel } { \indent \par }

```

Determine if we do numbering:

```

1646 \bool_lazy_or:nnT
1647 { \tl_if_empty_p:N \l__block_counter_tl }
1648 { #2 }
1649 { \bool_set_false:N \l__block_numbered_bool }

```

Save any note for later use (\#3 might contain \NoValue):

```

1650 \tl_set:Nn \l__block_note_tl {#3}

```

If we use numbering then we need a link target and increment the counter.

```

1651 \bool_if:NTF \l__block_numbered_bool
1652 {
1653   \@kernel@refstepcounter{ \l__block_counter_tl }
1654   \MakeLinkTarget{ \l__block_counter_tl }
1655 }
1656 {
1657   \MakeLinkTarget[theorem-like]{}
1658 }

```

Add the caption into \g__block_labels_box:

```

1659 \hbox_gset:Nn \g__block_labels_box
1660 {
1661   \box_use_drop:N \g__block_labels_box % <- does nothing if
1662                                           % there is no dangling label

```

Now apply the declarations that are for the whole caption.

```

1663 \l__block_caption_decls_tl

```

Then we apply the tagging socket for the caption to the complete content:

```

1664 \tag_socket_use:nnn {captionedtext/caption} {}
1665 {
1666   \skip_horizontal:n { \l__block_caption_before_skip }

```

For flexibility, the inner structure is given as a clist stored in \l__block_order_clist. We loop through it and call a processing function for each item in this clist. Everything happens in a group.

```

1667 \clist_map_inline:Nn \l__block_order_clist
1668 { \group_begin:
1669   \cs_if_exist:cTF { __block_do_##1: }
1670   { \use:c { __block_do_##1: } }
1671   { \msg_error:nnnn { block } { unknown-key-value }
1672     { order } { ##1 }
1673   }
1674   \group_end:
1675 }
1676 \skip_horizontal:n { \l__block_caption_after_skip }
1677 }
1678 }

```

If the title should be standalone we immediately push it out:

```

1679 \bool_if:NTF \g__block_label_standalone_bool
1680 {
1681   \bool_gset_true:N \g__block_label_standalone_bool
1682   \para_omit_indent:

```

Do we need a `\nobreak` here or is this covered? check

check if `@newlist` could be replaced by something more general

```
1683     \box_use_drop:N \g__block_labels_box
1684     \par
1685 }
```

Otherwise we signal that we are at the start and have a label dangling. The name `@newlist` is a bit unfortunate, but for now we keep this name.

```
1686 {
1687     \legacy_if_gset_true:n { @newlist }
1688     \legacy_if_gset_true:n { @inlabel }
1689 }
```

Do not break after the first line:

```
1690     \legacy_if_gset_true:n { @nobreak }
```

Then set up a special `everypar` to handle the dangling caption:

```
1691     \__block_debug_typeout:n{Set~ captioned~ block~ everypar \on@line }
1692     \cs_set_eq:NN \__block_everypar: \__block_captioned_everypar_std:
```

Finally, set up any declarations for the body of the environment:

```
1693     \l__block_body_decls_tl
1694     \__block_debug_typeout:n{template:thmstyle:std~end}
1695 }
```

Not really sure it is good idea to error; other parts of instance declarations also fail on incorrect data without any checks so why this one? TODO decide.

```
1696 \msg_new:nnnn { block } { unknown-key-value }
1697 { The~ value~ '#2'~ in~ key~ '#1'~ is~ not~ recognized. }
1698 {
1699     Perhaps~ a~ misspelling~ or~ the~ current~ template~
1700     instance~ uses~ special~ values.
1701 }
```

`/captionedtext/caption (socket)`

```
1702 \NewTaggingSocket{captionedtext/caption}{2}
```

why kernel?

```
1703 \NewTaggingSocketPlug{captionedtext/caption}{kernel}
1704 {
1705     \tag_struct_begin:n{tag=\UseStructureName{block/theorem-like/caption}}
1706     #2
1707     \tag_struct_end:
1708 }
1709 \AssignTaggingSocketPlug{captionedtext/caption}{kernel}
```

Here are the functions that are called when the corresponding name appears in the caption clist.

`\__block_do_title:` Handle the title:

```
1710 \cs_new_protected:Npn \__block_do_title: {
```

Check if there is a title.

```
1711     \tl_if_empty:NTF \l__block_title_tl
```

If the title is empty we drop accumulated but not yet typeset separators:

```
1712     { \__block_drop_separators: }
```

Otherwise we typeset the title, first inserting separator (or separators) that have been waiting.

```
1713     { \tag_socket_use:nnn {mc} {}{
1714       \__block_insert_separators:
```

The title may have its own formatting:

```
1715         \l__block_title_decls_tl
1716         \__block_title_format:n \l__block_title_tl }
1717     }
1718 }
```

(End of definition for __block_do_title:.)

`\__block_do_note:` Formatting of a note (if present) uses the same structure.

```
1719 \cs_new_protected:Npn \__block_do_note: {
1720   \tl_if_novalue:oTF \l__block_note_tl
1721     { \__block_drop_separators: }
1722     { \tag_socket_use:nnn {mc} {}{
1723       \__block_insert_separators:
1724       \l__block_note_decls_tl
1725       \__block_note_format:n \l__block_note_tl }
1726     }
1727 }
```

(End of definition for __block_do_note:.)

`\__block_do_number:` The number (if present) has a similar formatting but it uses an Lbl structure:

```
1728 \cs_new_protected:Npn \__block_do_number: {
1729   \bool_if:NTF \l__block_numbered_bool
1730     { \tag_socket_use:nnn {struct-mc} {tag=\UseStructureName{block/theorem-like/label}}
1731       { \__block_insert_separators:
1732         \l__block_number_decls_tl
1733         \__block_number_format:n {
1734           \use:c{ the \l__block_counter_tl } }
1735       }
1736     }
1737     { \__block_drop_separators: }
1738 }
```

(End of definition for __block_do_number:.)

`\__block_do_punct:` The punctuation is handled slightly differently. It unconditionally drops any dangling separators whether or not it is empty:

```
1739 \cs_new_protected:Npn \__block_do_punct: {
1740   \__block_drop_separators:
1741   \tl_if_empty:NF \l__block_punct_tl
1742     { \tag_socket_use:nnn {mc} {}{
1743       \l__block_punct_decls_tl
1744       \__block_punct_format:n \l__block_punct_tl }
1745     }
1746 }
```

(End of definition for _block_do_punct:.)

_block_do_separator: What's still missing is what separator should do. It simply adds a \l_block_separator_tl to the \g_block_collected_separators_tl tokenlist. This way the clist can contain separator,separator,... to indicate multiple separators (mainly useful if they are some amount of space). The storage tokenlist is global as the functions are executed inside their own group, but the collected separator is used outside of that group.

```

1747 \cs_new_protected:Npn \_block\_do\_separator: {
1748   \tl_gput_right:Nn \g\_block\_collected\_separators\_tl \l\_block\_separator\_tl
1749 }

```

So _block_insert_separators: is trivial, all we have to do is to insert the collected separators and then clear the tokenlist

```

1750 \cs_new_protected:Npn \_block\_insert\_separators: {
1751   \g\_block\_collected\_separators\_tl
1752   \tl_gclear:N \g\_block\_collected\_separators\_tl
1753 }

```

Even simpler is _block_drop_separators:

```

1754 \cs_new_protected:Npn \_block\_drop\_separators: {
1755   \tl_gclear:N \g\_block\_collected\_separators\_tl
1756 }

```

What remains is to declare the tokenlist.

```

1757 \tl_new:N \g\_block\_collected\_separators\_tl

```

(End of definition for _block_do_separator: and others.)

captionedtext proof (templ.) In case of the templates for proofs we do everything in a single template.

```

1758 \DeclareTemplateCode{captionedtext}{proof}{4}
1759 {
1760   ,title           = \l\_block\_title\_tl
1761   ,punct           = \l\_block\_punct\_tl
1762   ,caption-placement = {
1763     chained        = \bool_gset_false:N \g\_block\_label\_standalone\_bool
1764                     \bool_gset_false:N \g\_block\_label\_unchained\_bool
1765     ,unchained      = \bool_gset_false:N \g\_block\_label\_standalone\_bool
1766                     \bool_gset_true:N \g\_block\_label\_unchained\_bool
1767     ,standalone     = \bool_gset_true:N \g\_block\_label\_standalone\_bool
1768                     \bool_gset_false:N \g\_block\_label\_unchained\_bool
1769   }
1770   ,caption-unbreakable = \l\_block\_caption\_unbreakable\_bool
1771   ,before-hspace      = \l\_block\_caption\_before\_skip
1772   ,after-hspace       = \l\_block\_caption\_after\_skip
1773   ,caption-decls      = \l\_block\_caption\_decls\_tl
1774   ,title-decls        = \l\_block\_title\_decls\_tl
1775   ,punct-decls        = \l\_block\_punct\_decls\_tl
1776   ,title-format       = \\_block\_title\_format:n
1777   ,punct-format       = \\_block\_punct\_format:n
1778   ,body-decls         = \l\_block\_body\_decls\_tl
1779 }
1780 {

```

Display the template's arguments when tracing:

```
1781 \template_debug_typeout:n{~\space template:~ 'proof';~
1782 arguments:~ \exp_not:o{\exp_after:wN |#1|#2|#3|#4|}}
```

Evaluate document-level key settings. As all given keys should be handled we use `\SetTemplateKeys` to raise an error if one or more are not recognized:

```
1783 \SetTemplateKeys{captionedtext}{proof}{#1}
```

By default the title is defined by the proof instance, but if the user provides an optional argument that optional argument overwrites the title (in contrast to theorem-like environments that use the optional argument to provide an additional note):

```
1784 \IfNoValueF {#3} { \tl_set:Nn \l__block_title_tl {#3} }
```

Now we prepare typesetting the title by placing it in the `\g__block_labels_box`:

```
1785 \hbox_gset:Nn \g__block_labels_box
1786 {
1787   \box_use_drop:N \g__block_labels_box % <- does nothing if there
1788                                     % is no dangling label
1789   \l__block_caption_decls_tl
1790   \tag_socket_use:nnn {captionedtext/caption} {}
1791   {
1792     \skip_horizontal:n { \l__block_caption_before_skip }
```

`\__block_do_title:` and `\__block_do_punct:` unnecessarily call `\__block_drop_separators:` but otherwise they do well, so ...

```
1793   \group_begin: \__block_do_title: \group_end:
1794   \group_begin: \__block_do_punct: \group_end:
1795   \skip_horizontal:n { \l__block_caption_after_skip }
1796 }
1797 }
```

The remaining code is identical to the one in `thmstyle std`; for documentation see there:

```
1798 \bool_if:NTF \g__block_label_standalone_bool
1799 {
1800   \bool_gset_true:N \g__block_label_standalone_bool
1801   \para_omit_indent:
1802   \bool_if:NTF \l__block_caption_unbreakable_bool
1803               \box_use_drop:N
1804               \hbox_unpack_drop:N
1805   \g__block_labels_box
1806   \par
1807 }
1808 {
1809   \legacy_if_gset_true:n { @newlist }
1810   \legacy_if_gset_true:n { @inlabel }
1811 }
1812 \legacy_if_gset_true:n { @nobreak }
1813 \__block_debug_typeout:n{Set~ captioned~ block~ everypar \on@line }
1814 \cs_set_eq:NN \__block_everypar: \__block_captioned_everypar_std:
1815 \l__block_body_decls_tl
1816 \__block_debug_typeout:n{template:captionedtext:proof~end}
1817 }
```

9.4 Tagging support commands

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

`\__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 `<semantic-para>`, i.e., it is already open. The command is mapped to `\__kernel_displayblock_beginpar_vmode:` in various tagging recipes. It is also used in the math code!

```

1818 \cs_set:Npn \__block_beginpar_vmode: {
1819   \__block_debug_typeout:n
1820   { @endpe = \legacy_if:nTF { @endpe }{true}{false} \on@line }
1821   \legacy_if:nTF { @endpe }
1822   {
1823     \legacy_if_gset_false:n { @endpe }
1824   }

```

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

```

1825   {
1826     \int_compare:nNtT \l__tag_block_flattened_level_int < 2
1827     {
1828       % \typeout{===> __block_beginpar_vmode:}
1829       % \ShowTagging{struct-stack}
1830       \UseTaggingSocket{para/semantic/begin}
1831       { \__tag_para_main_store_struct: }
1832       % \ShowTagging{struct-stack}
1833     }
1834   }
1835 }

```

(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).

The command is mapped to `\__kernel_displayblock_beginpar_hmode:w` in various tagging recipes.

```

1836 \cs_set:Npn \__block_beginpar_hmode:N #1
1837   {
1838     \UseTaggingSocket{para/textblock/end}
1839     \tagpdfparaOff \par \tagpdfparaOn
1840   }

```

(End of definition for __block_beginpar_hmode:N.)

Paragraph tagging is mainly done using the paragraph hooks. The code is in `ltagging.dtx`.

`\block/startpara/direct (socket)` A tagging socket to start a paragraph structure. It takes an argument (which is only used in debugging) that should be gobbled if tagging is not active. Not yet in `ltagging` (name and function should be reviewed).

This is a similar code to the one used in the `para/begin` hook but without testing `@endpe`. This is not needed in the standalone case and wrong inside lists.

This code is used in various places and should be a dummy if tagging is not active.

```
1841 \socket_if_exist:nF {tagsupport/block/startpara/direct}
1842 {
1843   \NewTaggingSocket {block/startpara/direct}{1}
1844 }
```

`default (plug)`

```
1845 \NewTaggingSocketPlug{block/startpara/direct}{default}
1846 {
1847   \bool_if:NT \l__tag_para_bool
1848   {
1849     \bool_if:NF \l__tag_para_flattened_bool
1850     {
1851       %          \typeout{==> block/startpara/direct}
1852       %          \ShowTagging{struct-stack}
1853       \UseTaggingSocket{para/semantic/begin}
1854               { \__tag_para_main_store_struct: }
1855     }
1856     \UseTaggingSocket{para/textblock/begin} { #1 }
1857   }
1858 }
1859 \AssignTaggingSocketPlug{block/startpara/direct}{default}
```

The `para/end` hook code is in `ltagging`. Currently we still need to remove the `tagpdf` chunk to avoid that the socket is added twice. We add empty chunks to avoid warning messages from code parts trying to remove the chunks.

```
1860 \AddToHook{para/end}[tagpdf]{}
1861 \RemoveFromHook{para/end}[tagpdf]
1862 \AddToHook{para/end}{}

1863 \def\PARALABEL{NP-}
```

`\port/kernel/endpe/vmode (socket)` A tagging socket which ends a structure. Used in `\begin` and `\para_end:`. Not yet in `ltagging` (name and function should be reviewed).

```
1864 \socket_if_exist:nF {tagsupport/kernel/endpe/vmode}
1865 {
1866   \NewTaggingSocket {kernel/endpe/vmode}{0}
1867 }
```

`default (plug)`

```
1868 \NewTaggingSocketPlug{kernel/endpe/vmode}{default}
1869 {
1870   \if@endpe \ifvmode
1871     \bool_if:NT \l__tag_para_bool
```

```

1872     {
1873       \bool_if:NF \l__tag_para_flattened_bool
1874       {
1875         \UseTaggingSocket{para/semantic/end}{}
1876       }
1877     }
1878   }
1879   \fi \fi
1880 }
1881 \AssignTaggingSocketPlug{kernel/endpe/vmode}{default}

```

\@endpefalse is needed by \para_end:, see test tagging-0097.

\para_end: If we see a \par in vmode and a <semantic-para> 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).

```

1882 \cs_set_protected:Npn \para_end: {
1883   \scan_stop:
1884   \mode_if_horizontal:TF {
1885     \mode_if_inner:F {
1886       \tex_unskip:D
1887       \hook_use:n{para/end}
1888       \@kernel@after@para@end
1889       \mode_if_horizontal:TF {
1890         \if_int_compare:w 11 = \tex_lastnodetype:D
1891           \tex_hskip:D \c_zero_dim
1892         \fi:
1893         \tex_par:D
1894         \hook_use:n{para/after}
1895         \@kernel@after@para@after
1896       }
1897       { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
1898     }
1899   }
1900   {

```

TODO 2025-07-01. This is not exactly as before, this doesn't insert an \@endpefalse when tagging is active. Check if this a problem.

```

1901     \UseTaggingSocket{kernel/endpe/vmode}%
1902     \tex_par:D
1903   }
1904 }

```

Now reset L^AT_EX 2_ε functions to use the changed \para_end: [TODO: Need to check if \@@par is ever used in a way that the vmodetagging hook is needed.]

```

1905 \cs_set_eq:NN \par \para_end:
1906 \cs_set_eq:NN \@@par \para_end:
1907 \cs_set_eq:NN \endgraf \para_end:

```

(End of definition for \para_end:. This function is documented on page 40.)

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

```

1908 \protected\def\begin#1{%
1909   \UseHook{env/#1/before}%
1910   \@ifundefined{#1}%
1911     {\def\reserved@a{\@latex@error{Environment~#1~undefined}\@eha}}%
1912     {\def\reserved@a{\def\@currenvir{#1}%
1913       \edef\@currencline{\on@line}%
1914       \@execute@begin@hook{#1}%
1915       \csname #1\endcsname}}%
1916   \@ignorefalse
1917   \begingroup

```

The original L^AT_EX 2_ε version did set `\@endpfalse` unconditionally at this point, but this is conceptually wrong because it prevents the upcoming environment from seeing if `@endpe` handling was requested, which matters if tagging is done. Instead, resetting `@endpe` should only happen at the start of block environments after they have evaluate the current status of this flag. For the same reason it is not correct to unconditionally end a semantic paragraph here, because `@endpe` is `true` we are in a semantic paragraph that should continue.

```

1918 %   \@endpfalse
1919   \reserved@a}

```

(End of definition for `\begin`. This function is documented on page 40.)

`\_kernel\_list\_label\_after:n` If starting the semantic-para and text-block tags got delayed because of a pending label we have to do it after the label got typeset. TODO: it should do nothing without tagging that's why there is a test, this should be better hidden in a tagging socket, but it is not quite clear how to do this.

internally this is now a tagging socket, why the outer test then?

```

1920 \cs_new_protected:Npn \_kernel\_list\_label\_after:n #1 {
1921   \tag_socket_use:nn {block/startpara/direct} { #1 }
1922 }

```

(End of definition for `\_kernel\_list\_label\_after:n`.)

`\_block\_inner\_begin:` Start a block that has an inner structure if it isn't also a list. This command is tagging specific, it is mapped to `\_kernel\_displayblock\_begin:` in some tagging recipes.

```

1923 \cs_new_protected:Npn \_block\_inner\_begin: {
1924   \tagstructbegin{tag=\_block\_tag\_inner\_tag\_tl}
1925 }

```

(End of definition for `\_block\_inner\_begin:`.)

`\_block\_inner\_end:` End a block (which isn't also a list). This command is tagging specific, it is mapped to `\_kernel\_displayblock\_end:` in some tagging recipes.

```

1926 \cs_new_protected:Npn \_block\_inner\_end: {
1927   \_block\_debug\_typeout:n{block-end \on@line}

```

If there is an open request for `@endpe` handling still open (from material in the environment body) then a semantic para is still open and we have to close it before closing the inner structure.

```

1928 \legacy_if:nT { @endpe }
1929 {
1930   \UseTaggingSocket{para/semantic/end}
1931   { \_block\_debug\_typeout:n{close~ /semantic-para \on@line}}

```

```

1932     }
1933     \tagstructend          % end inner structure
1934 }

```

(End of definition for _block_inner_end:.)

9.4.1 List tags

```

1935 \tl_new:N \l__tag_L_tag_tl
1936 \tl_set:Nn \l__tag_L_tag_tl {\UseStructureName{block/list}}

```

`\l__tag_L_attr_class_tl` The variable should be set to one of the list attributes `list`, `itemize` `enumerate` or `description` declared in `latex-lab-namespace`.

```

1937 \tl_new:N \l__tag_L_attr_class_tl
1938 \tl_set:Nn \l__tag_L_attr_class_tl {list}

```

(End of definition for \l__tag_L_attr_class_tl.)

`\_block\_list\_begin:` Start a list ...This command is tagging specific, it is mapped to `\_kernel\_displayblock\_begin:` in a tagging recipe.

```

1939 \cs_set:Npn \_block\_list\_begin: {
1940   \tagstructbegin
1941   {
1942     tag=\l__tag_L_tag_tl
1943     ,attribute-class=\l__tag_L_attr_class_tl
1944   }
1945 }

```

(End of definition for _block_list_begin:.)

`\_block\_list\_item\_begin:` Start tagging a list item. This command is tagging specific, it is mapped to `\_kernel\_list\_item\_begin:` in a tagging recipe.

```

1946 \cs_set:Npn \_block\_list\_item\_begin: {
1947   \tagstructbegin{tag=\UseStructureName{block/list/item}}
1948 }

```

(End of definition for _block_list_item_begin:.)

`\_block\_list\_item\_end:` When a list item ends we have to close `<itembody>` and `<LI>` but also a `<text-block>` in the special case that the item material ends in a list (identifiable via `@endpe`). This command is tagging specific. This command is copied to `\_kernel\_list\_item\_end:` in the list recipe.

```

1949 \cs_set:Npn \_block\_list\_item\_end: {
1950   \legacy_if:nT { @endpe }
1951   {
1952     \UseTaggingSocket{para/semantic/end}
1953     { \_block\_debug\_typeout:n{Structure-end~ P~ at~ item-end \on@line } }
1954   }
1955   \tagstructend \tagstructend % end \UseStructureName{block/list/body}, LI
1956 }

```

(End of definition for _block_list_item_end:.)

`\_block\_list\_end`: Finally, at the list end we have to close the open `<itembody>`, `<LI>`, `<L>`, and possibly a `<text-block>` if the last item ends with a list. However, if the user forgot to add an `\item` then there will be no `<LI>` and `<itembody>` 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. This command is tagging specific. This command is copied to `\_kernel\_displayblock\_end`: in the list recipe.

```
1957 \cs_new_protected:Npn \_block\_list\_end: {
```

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

```
1958   \legacy_if:nF { @newlist }
1959   {
1960     \legacy_if:nT { @endpe }
1961     {
1962       \UseTaggingSocket{para/semantic/end}
1963       {\_block\_debug\_typeout:n{Structure-end~ semantic-para~ at~ list-end \on@line }}
1964     }
1965     \tagstructend % end \UseStructureName{block/list/body},
1966     \hook\_use:n{item/after}
1967     \tagstructend % LI
1968   }
1969   \tagstructend % end L
1970 }
```

(End of definition for `\_block\_list\_end`.)

End of tagging related declarations.

9.4.2 Tagging recipes

`tagsupport/block/recipe` (*socket*) A tagging socket to call the tagging recipe. Declared in `ltagging`.

```
1971 \socket_if_exist:nF {tagsupport/block/recipe}
1972 {
1973   \NewTaggingSocket{block/recipe}{1}
1974 }
```

`default` (*plug*)

```
1975 \NewTaggingSocketPlug{block/recipe}{default}
1976 {
1977   \use:c { \_block\_recipe\_#1: }
1978 }
1979 \AssignTaggingSocketPlug{block/recipe}{default}
```

`\__block_recipe_noop:` The **noop** recipe does nothing and the tagging of the block is handled as if the content where inserted directly surrounded by `\par`. The recipe does not set the endpe switch. If the block ends, e.g., with a list its setting is passed forward and the following text is not indented unless there is an empty line. If the block ends with normal text, text after the block starts a new paragraph and so is indented. The recipe is meant for environments like **adjustwidth** which only change the layout and which can contain, e.g., sectioning commands.

UFi: This recipe needs more tests!

```

1980 \cs_new_protected:Npn \__block_recipe_noop:
1981 {
1982   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
1983                                     \prg_do_nothing:
1984   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
1985                                     \prg_do_nothing:
1986   \let \__kernel_displayblock_begin: \prg_do_nothing:
1987   \let \__kernel_displayblock_end:   \prg_do_nothing:
1988 }

```

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

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

```

1989 \cs_new_protected:Npn \__block_recipe_basic: {
1990   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
1991                                     \__block_beginpar_hmode:N
1992   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
1993                                     \__block_beginpar_vmode:
1994   \let \__kernel_displayblock_begin: \prg_do_nothing:
1995   \let \__kernel_displayblock_end:   \prg_do_nothing:
1996 }

```

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

`\__block_recipe_standalone:` The **standalone** recipe produces a block that ensures that a previous `<semantic-para>` ends and that after the block a new `<semantic-para>` starts.

```

1997 \cs_new_protected:Npn \__block_recipe_standalone: {
1998   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
1999                                     \prg_do_nothing:
2000   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
2001                                     \prg_do_nothing:
2002   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
2003   \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:
2004   \tl_if_empty:NTF \l__block_tag_name_tl
2005     { \tl_set:Nn \l__block_tag_inner_tag_tl {Sect} }
2006     { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
2007 }

```

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

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

- surround the block with a `<semantic-para>` structure if not already in a `<semantic-para>`. In the latter case end the MC and the `<text-block>` but leave the `<semantic-para>` open.

If we are producing flattened paragraphs, just close any `<text-block>` but do not open a `<semantic-para>`.

- Then open an new (inner) structure (by default `<Div>` but typically the one specified on the instance).
- At the end of the block close the inner structure (`<Div>` or explicit one) but leave the `<semantic-para>` open to be either continued or closed due to a following `\par`.

```

2008 \cs_new_protected:Npn \__block_recipe_standard:
2009 {
2010   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
2011                                     \__block_beginpar_hmode:N
2012   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
2013                                     \__block_beginpar_vmode:
2014   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
2015   \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:

2016   \tl_if_empty:NTF \l__block_tag_name_tl
2017     { \tl_set:Nn \l__block_tag_inner_tag_tl {\UseStructureName{block/inner}} }
2018     { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
2019 }

```

(End of definition for __block_recipe_standard:.)

`\l__block_tag_inner_tag_tl` The tag name that is used if the block has an inner structure.

```

2020 \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 `<semantic-para>`-structure or keeps the current one open (only closing the MC).
- It then starts a new structure role-mapped to L-structure and arranges for handling list items, e.g., Li, itemlabel and itembody structures.
- At the end it closes open list structures as needed but keeps the `<semantic-para>`-structure open to continue the paragraph after the list, if necessary.

```

2021 \cs_new_protected:Npn \__block_recipe_list:
2022 {
2023   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
2024                                     \__block_beginpar_hmode:N
2025   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
2026                                     \__block_beginpar_vmode:
2027   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_list_begin:
2028   \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)).

```
2029 \cs_set_eq:NN \__kernel_list_item_begin: \__block_list_item_begin:
2030 \cs_set_eq:NN \__kernel_list_item_end: \__block_list_item_end:
```

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

```
2031 \tl_if_empty:NTF \l__block_tag_name_tl
2032 { \tl_set:Nx \l__tag_L_tag_tl {\UseStructureName{block/list}} }
2033 { \tl_set_eq:NN \l__tag_L_tag_tl \l__block_tag_name_tl }
2034 \tl_if_empty:NTF \l__block_tag_class_tl
2035 { \tl_set:Nx \l__tag_L_attr_class_tl {} }
2036 { \tl_set_eq:NN \l__tag_L_attr_class_tl \l__block_tag_class_tl }
2037 }
```

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

```
2038 </package-start>
```

10 Support code for document-level block environments

10.1 Verbatim-like environments

10.1.1 Helper commands for `verbatim` and `verbatim*`

`\legacyverbatimsetup` This code is called as part of the `final-code` of the `blockenv` instance and sets up the special conventions needed for `verbatim` environments. We pass one argument to differentiate between visible and invisible spaces.

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.

```
2039 <*package-finish>
2040 <@@=
2041 \def\legacyverbatimsetup #1 {%
2042   \language\l@nohyphenation
2043   \@tempswafalse
2044   \def\par{%
2045     \if@tempswa
2046       \leavevmode \null {\@@par}\penalty\interlinepenalty
2047     \else
2048       \@tempswatrue
2049       \ifhmode{\@@par}\penalty\interlinepenalty\fi
2050     \fi
```

Do something at the very beginning of each verbatim line:

```
2051 \UseSocket{verbatim/startline}%
```

might also need a hook
not just a socket

```

2052 }%
2053 \let\do\@makeother \dospecials
2054 \obeylines \verbatim@font \@noligs
2055 \everypar \expandafter{\the\everypar \unpenalty}%
2056 \frenchspacing
2057 \AssignStructureRole {para/textblock}%
2058 {\UseStructureName{block/verbatim/codeline}}%

```

Should next line be hidden in a tagging socket?

If the argument is neither `visible` nor `invisible` nothing will happen—tough.

```

2059 \use:c { @setupverb #1 space }
2060 \@vobeyspaces
2061 }

```

(End of definition for `\legacyverbatimsetup`. This function is documented on page 39.)

`verbatim/startline (socket)` A socket that is executed at the start of each verbatim line, to be used, for example, to check for `%_` when the `doc` package is active.

We might also need a hook for line numbers.

```

2062 \NewSocket{verbatim/startline}{0}

```

`\@setupverbinvisiblespace` In the pdfTeX engine we need to use `\pdffakespace` chars for the invisible spaces. In luatex we do not want this as it would lead to doubling the number of real space chars. In dvi-mode we do not want that either: with pdftex it would error, with xetex it does nothing.

```

2063 <@@=block>
2064 \newcommand\@setupverbinvisiblespace{}
2065 \bool_lazy_or:nnF
2066 { \sys_if_engine_luatex_p: }
2067 { \sys_if_output_dvi_p: }
2068 {
2069   \renewcommand\@setupverbinvisiblespace
2070     {\def\xobeysp{\nobreakspace\pdffakespace}}
2071 }

```

(End of definition for `\@setupverbinvisiblespace`. This function is documented on page 39.)

The command `\@setupverbvisiblespace` is already defined in the kernel.

10.1.2 Helper commands for `alltt` and `alltt*`

`\legacyallttsetup` The `alltt` environment also needs some special setup. We can reuse `\legacyverbatimsetup` but we have to take out `\`, `{`, and `}` from `\dospecials` as they should remain available with their normal catcodes and adjust `'` inside math. This is lifted straight from the original package code.

```

2072 <@@= >
2073 \ExplSyntaxOff
2074 \def\legacyallttsetup #1{%
2075   \let\org@prime~%
2076   \everymath\expandafter{\the\everymath
2077     \catcode`\'=12 \let~\org@prime}%
2078   \everydisplay\expandafter{\the\everydisplay
2079     \catcode`\'=12 \let~\org@prime}%

```

This alters `\dospecials`:

```
2080 \let\org@dosppecials\dosppecials
2081 \g@remfrom@specials{\}%
2082 \g@remfrom@specials{\}%
2083 \g@remfrom@specials{\}%
```

Then call `\legacyverbatimsetup`:

```
2084 \legacyverbatimsetup {#1}%
```

And afterwards restore `\dospecials`:

```
2085 \let\dosppecials\org@dosppecials
2086 }
```

Copied from `alltt`.

```
\g@remfrom@specials 2087 \def\g@remfrom@specials#1{%
2088 \def\@new@specials{}%
2089 \def\@remove##1{%
2090 \ifx##1#1\else
2091 \g@addto@macro\@new@specials{\do ##1}\fi}%
2092 \let\do\@remove\dosppecials
2093 \let\dosppecials\@new@specials
2094 }

2095 \ExplSyntaxOn
2096 <@@=block>
```

(End of definition for `\legacyallttsetup` and `\g@remfrom@specials`. These functions are documented on page 39.)

10.1.3 Helper command for legacy list environment

`\legacylistsetup` And here is the extra code for use in the list instance setup in the key `early-legacy-code`:

```
2097 \cs_new_protected:Npn \legacylistsetup {
```

Reset values to defaults:

```
2098 \dim_zero:N \listparindent
2099 \dim_zero:N \rightmargin
2100 \dim_zero:N \itemindent
```

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

```
2101 % \tl_set:Nn \@listctr {}
2102 % \legacy_if_set_false:n { @nmbrlist } % needed if lists are nested
```

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

```
2103 \let\makelabel\@mklab % TODO: customize
```

Now we use the argument with parameter settings to update some or all of the above defaults (this holds whatever was put into the second argument to the `list` environment):

```
2104 \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` role-mapped 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.

```

2105 \legacy_if:nTF { @nmbrlist }
2106 { \tl_set:Nn \l__tag_L_attr_class_tl {enumerate} } % numbered list
2107 { \tl_if_empty:NTF \@itemlabel
2108   { \tl_set:Nn \l__tag_L_attr_class_tl {list} } % no label
2109   { \tl_set:Nn \l__tag_L_attr_class_tl {itemize} } % unnumbered,
2110                                     % unordered
2111 }
2112 }

```

(End of definition for `\legacylistsetup`. This function is documented on page 39.)

`\l_block_legacy_env_params_tl` The token list in which the declarations from the second argument of `list` are temporarily stored. This is then used in `\legacylistsetup`.

```

2113 \tl_new:N\l_block_legacy_env_params_tl

```

(End of definition for `\l_block_legacy_env_params_tl`.)

10.2 Theorem-like environments

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

10.2.1 Declarations for theorem-like environments

`\newtheorem` We reimplement the extended `amsthm` version of the declaration which also supports a star form indicating that this theorem-like environment should not be numbered.

```

2114 \RenewDocumentCommand \newtheorem { s m o m o } {

```

Is the environment definable at all? If not there is not much point in continuing.

```

2115 \expandafter\@ifdefinable\csname #2\endcsname
2116 {

```

If a star was given then there is no need to set up a counter for this environment. Otherwise we do what $\text{\LaTeX}2\epsilon$ did, except that we do all the variations in one go, rather than using `\@ynthm`, `\@xnthm`, and `\@othm`.

```

2117 \IfBooleanF #1
2118 {
2119   \IfNoValueTF {#3}

```

If there was no counter to use (`#2`) then we set up a counter with the same name as the environment (`#2`).

```

2120   {
2121     \@definecounter {#2}

```

undefined the old internal commands?

If there was no “counter within” the counter representation is simple, otherwise we build it up from the two counters:

```

2122         \IfNoValueTF {#5}
2123         { % @ynthm
2124           \tl_gset:ce { the #2 }
2125           {
2126             \@thmcounter{#2}
2127           }
2128         }
2129         { % @xnthm
2130           \@newctr{#2}[#5]
2131           \tl_gset:ce { the #2 }
2132           {
2133             \expandafter\noexpand\csname the#5\endcsname
2134             \@thmcountersep
2135             \@thmcounter{#2}
2136           }
2137         }
2138       }
2139     { % @othm

```

If we should reuse an existing counter (#3 was given) we check that this counter actually exists and if so use it:

```

2140         \@ifundefined{c@#3}
2141         { \nocounterr{#3} }
2142         {
2143           \newcounteralias{#2}{#3}
2144         }
2145       }
2146     }

```

With the counter defined we are ready to declare the environments. There is a slight complication though: the “theorem-like” environments have an optional argument which contains a possible note, but now we also want to use the first optional argument to hold a key/value list with parameter settings. We therefore define this argument via `={note}o` so that a simple note, if given is assigned to a `note` key. Further processing is then delegated to the command `\ParseLaTeXeTheoremlike` which, after sorting out the argument situation, eventually calls `\BlockEnv`.

```

2147   \IfBooleanTF #1

```

The starred form of the environment suppresses the number so we pass it `\BooleanTrue`, otherwise it is identical to the normal definition.

```

2148     {
2149       \NewDocumentEnvironment{#2}{={note}o }
2150       { \ParseLaTeXeTheoremlike {#2} \BooleanTrue {##1} }
2151       { \BlockEnvEnd }
2152     }
2153     {
2154       \NewDocumentEnvironment{#2}{={note}o }
2155       { \ParseLaTeXeTheoremlike {#2} \BooleanFalse {##1} }
2156       { \BlockEnvEnd }
2157     }

```

Now it is about time to provide all necessary template instances. They depend on the `\theoremstyle` specified by the user and possibly by a `\swapnumbers` declaration. We start by checking the requested `\theoremstyle` (if none was given then `plain` is the default and if it is an unknown name we also revert to `plain` after issuing a warning).

```

2158 \IfInstanceExistsF{thmstyle}{\l__block_thmstyle_tl}
2159 { \latex@warning{Unknown~ theoremstyle~
2160 '\l__block_thmstyle_tl'~ using~ 'plain'}
2161 \theoremstyle {plain}
2162 }

```

So now we know that `\l__block_thmstyle_tl` holds a valid style. What we don't know is whether or not there are special block instances that go with that style (it might be a style that reuses the `thm-(style)block-...` instances).

```

2163 \IfInstanceExistsTF{block} { thm- \l__block_thmstyle_tl -1 }
2164 { \__block_debug_typeout:n{...~ style~ \l__block_thmstyle_tl\space exists } }
2165 { \__block_debug_typeout:n{...~ style~ \l__block_thmstyle_tl\space
2166 does~ not ~exist;~ 'plain'~ used } }

```

So here is the `blockenv` instance for the new “theorem-like” environment. It uses a `captionedtext` instance for the inner-instance which we also have to declare.

```

2167 \DeclareInstance{blockenv}{#2}{std}
2168 {
2169     name = theorem-like
2170     ,tag-name = \UseStructureName{block/theorem-like}
2171     ,tag-attr-class =
2172     ,tagging-recipe = standalone
2173     ,standalone = true
2174     ,inner-level-counter =
2175     ,transparent-level = true
2176     ,early-legacy-code =
2177     ,late-legacy-code =

```

What block instance to use is determined by checking if a special one exists or whether we should use `plain`:

```

2178     ,block-instance:e = thm-
2179                       \IfInstanceExistsTF{block}
2180                       {thm- \l__block_thmstyle_tl -1}
2181                       { \l__block_thmstyle_tl } { plain }
2182     ,inner-instance-type = captionedtext
2183     ,inner-instance = #2

```

By default the body text is justified, but perhaps we should not set anything here and use whatever is current.

```

2184     ,para-instance = justify
2185 }

```

The `captionedtext` instance is simple: the counter, if present, is either argument `#2` or `#3`; the title receives argument `#4`, and the style to use is stored in `\l__block_thmstyle_tl`. If `\swapnumbers` was requested we use a style variant with the suffix `-swap` appended.

```

2186 \DeclareInstance{captionedtext}{#2}{thmlike}
2187 {

```

The counter to use is either none or #2 based on the arguments given:

```

2188         ,counter:e = \IfBooleanF #1 { #2 }
2189         ,title      = #4
2190         ,style:e     = \l__block_thmstyle_tl
2191                     \bool_if:NT \l__block_swap_number_bool {-swap}
2192     }

```

We already know that the style `\l__block_thmstyle_tl` exists, since we have tested that earlier, but we don't know if that is also true for the `-swap` variant. So we have to check that and declare it if necessary.

```

2193     \bool_if:NT \l__block_swap_number_bool {
2194         \IfInstanceExistsF{thmstyle}{\l__block_thmstyle_tl -swap}
2195     }

```

If it doesn't exist we first make a copy of the base instance.

```

2196         \DeclareInstanceCopy{thmstyle}
2197                                 {\l__block_thmstyle_tl -swap}
2198                                 {\l__block_thmstyle_tl}

```

Then we retrieve the value of the `order` key from that instance which is a clist.

```

2199         \clist_set:Nx \l__block_order_clist
2200             { \InstanceValue { thmstyle }
2201               { \l__block_thmstyle_tl }
2202               { order }
2203             }

```

Then we step through this clist and build a new one in `\l__block_tmp_clist` with the `title` and `number` swapped. That is done under the assumption that both actually exist in the clist which would be the case if the instance was declared with `\newtheoremstyle`, i.e., for legacy setups.

```

2204         \clist_clear:N \l__block_tmp_clist
2205         \clist_map_inline:Nn \l__block_order_clist
2206             {
2207                 \clist_put_right:Nx \l__block_tmp_clist {
2208                     \str_case:nnF {##1}
2209                     { {title} {number}
2210                     {number} {title} }
2211                     {##1}
2212                 }
2213             }

```

Once that is done we put the new value for `order` in the new instance.

```

2214         \EditInstance {thmstyle}{\l__block_thmstyle_tl -swap}
2215             { order:e = \l__block_tmp_clist }
2216     }
2217 }
2218 }
2219 }

```

(End of definition for \newtheorem.)

`\ParseLaTeXeTheoremlike` The arguments to `\ParseLaTeXeTheoremlike` are as follows:

#1: instance name to use (of type “blockenv”)

- #2: unnumbered? boolean normally provided by using the star form of the environment
- #3: key/val for layout adjustments settings provided in the optional argument of the “theorem-like” environment. If a note to the theorem was given in that argument, then it has been turned into `note...=`

To be able to pick up the `note`, if provided, we make the following declaration:

```

2220 \keys_define:nn {blockenv} {
2221   , note      .tl_set:N = \l__block_note_tl
2222   , note      .groups:n = { interface }
2223 }

2224 \cs_new_protected:Npn \ParseLaTeXeTheoremlike #1 #2 #3 {
2225   %
2226   % \__block_debug_typeout:n{Parse~#1.~ Arguments~ found:~ \IfBooleanT{#2}{*}
2227   % \IfValueT{#3}{[\exp_not:n{#3}]}
2228   %

```

Normally, no note is provided, so that’s the starting point:

```

2229   \tl_set:Nn \l__block_note_tl { \NoValue }

```

Then we check #3 if it contains a key/val list containing `note`. This then sets `\l__block_note_tl` and puts all other key/vals in `\l__block_instance_keys_tl`. Otherwise the complete key/val list ends up there.

```

2230   \IfNoValueTF { #3 }
2231   { \tl_clear:N \l__block_instance_keys_tl }
2232   { \keys_set_groups:nnnN {blockenv} {interface} { #3 }
2233     \l__block_instance_keys_tl }

```

We are now ready to invoke the `blockenv` instance for #1 but we need to expand some of the values first, hence the `\use:e`.

```

2234   \use:e {
2235     \exp_not:N \BlockEnv { #1 }
2236     { \exp_not:o \l__block_instance_keys_tl }
2237     { #2 }
2238     { \exp_not:o \l__block_note_tl }
2239   }

```

We don’t have any use for the last argument (the sub-caption) in the standard setup, so we pass `\NoValue`.

```

2240     \NoValue
2241   }

```

(End of definition for \ParseLaTeXeTheoremlike. This function is documented on page ??.)

`\l__block_instance_keys_tl` Variable used above.

```

2242 \tl_new:N \l__block_instance_keys_tl

```

(End of definition for \l__block_instance_keys_tl.)

`\swapnumbers` Beside declaring theorem-like environments with `\newtheorem` and `\theoremstyle`, the `amsthm` package also introduced `\swapnumbers` to swap title and number in all environments (because that is a common requirement). The new implementation supports this approach as well.

It is implemented with a boolean `\l_block_swap_number_bool` which is toggled by `\swapnumbers`.

```
2243 \bool_new:N \l_block_swap_number_bool
2244 \cs_new_protected:Npn \swapnumbers { \bool_set_inverse:N \l_block_swap_number_bool }
```

(End of definition for `\swapnumbers`. This function is documented on page ??.)

`\newtheoremstyle` The `\newtheoremstyle` declaration was originally provided in the `amsthm` package. It has 9 mandatory arguments that sets some aspects of theorem styles. We map those to the template mechanism and generate `thmstyle` instances from them. Some of its default differ between the `pkgamsthm` package version and the version in the document classes. To account for this we set them up in macros that can be overwritten. We use a 2e name for that.

```
2245 \tl_new:N \newtheoremstyle@vspace@default
2246 \tl_set:Nn \newtheoremstyle@vspace@default { \topsep }
```

`\newtheoremstyle` has a bunch of argument conventions that haven't been fully implemented yet, e.g., #8 can be a blank (meaning normal word space or `\newline`) or a skip.

Those should eventually also be covered.

```
2247 \cs_set_protected:Npn \newtheoremstyle #1#2#3#4#5#6#7#8#9 {
```

First we build a `thmstyle` instance:

```
2248 \DeclareInstance{thmstyle}{#1}{std}{
2249   ,caption-decls = {#6}
2250   ,before-hspace:e = \tl_if_empty:nTF{#5}{0pt}{#5}
2251   ,body-decls = {#4}
2252   ,punct = {#7}
2253   ,order = {title, separator, number, separator, note, punct}
2254   ,number-decls = \upshape
2255   ,note-decls = \upshape\mdseries
```

This setting doesn't cover all syntax possibilities so far.

```
2256   ,after-hspace:e = \tl_if_empty:nTF{#8}{0pt}
2257                   {\tl_if_blank:nTF{#8}{3.3pt}{#8}}
2258 }
```

If #2 or #3 are not empty we also have to set up a `block` instance to account for the fact that special vertical spacing is requested. We base this on `thm-legacy2e-1` so that most parameters already have correct values. (Starting from scratch is difficult because we don't know if the current class defines defaults in `\@listi` and friends, so one would need to account for that.) Fix end-vspace setting (tagging/1362)

```
2259 \tl_if_empty:nF { #2#3 }
2260 {
2261   \DeclareInstanceCopy{block}{thm-#1-1}{thm-legacy2e-1}
2262   \EditInstance{block}{thm-#1-1}{
2263     ,begin-vspace:e = \tl_if_empty:nTF{#2}
2264                     { \newtheoremstyle@vspace@default }{#2}
2265     ,end-vspace:e = \tl_if_empty:nTF{#3}
2266                   { \newtheoremstyle@vspace@default }{#3}
2267   }
```

As elsewhere we provide two levels.

```
2268   \DeclareInstanceCopy{block}{thm-#1-2}{thm-#1-1}
2269 }
```

extend

More complicated is argument #9. If not empty it can contain `\thmname`, `\thmnumber`, and/or `\thmnote` to define the layout of the theorem caption. All the spacing has to be given inside the arguments of these commands which means that this doesn't work together with `\swapnumbers`, but this is the way `amsthm` was defined. If the instances are manually defined then it is easy to make them work with and without a `\swapnumbers` declaration. So basically, this here is good for a subset of cases and for backwards compatibility with `amsthm`.

The approach used doesn't cover all circumstances, e.g., if the argument contains low-level programming on top of the interface commands that the translation below will fail, but most existing code should work and the rest would need a replacement using instances that are directly set up.

```
2270 \tl_if_empty:oF { \exp_not:n{#9} }
2271 {
```

Give special definitions for the commands and then expand #9 and use the result to edit the instance we defined earlier.

```
2272 \cs_set:Npn \thmname ##1 {title-format={\exp_not:n{##1}}},}
2273 \cs_set:Npn \thmnumber ##1 {number-format={\exp_not:n{##1}}},}
2274 \cs_set:Npn \thmnote ##1 {note-format={\exp_not:n{##1}}},}
2275 \cs_set:Npn \__block_tmp:w ##1##2##3 {
2276 \exp_args:Nne \EditInstance{thmstyle}{#1}{#9}}
2277 \__block_tmp:w {##1} {##1} {##1}
```

We then also use the commands to deduce a suitable `order` and put that into the instance as well.

```
2278 \cs_set:Npn \thmname ##1 {title,}
2279 \cs_set:Npn \thmnumber ##1 {number,}
2280 \cs_set:Npn \thmnote ##1 {note,}
2281 \cs_set:Npn \__block_tmp:w##1##2##3 {
2282 \exp_args:Nne \EditInstance{thmstyle}{#1}{order={#9 punct}}}
2283 \__block_tmp:w {##1} {##1} {##1}
2284 }
```

If block tracing is turned on we show the final result:

```
2285 \__block_debug:n { \ShowInstanceValues{thmstyle}{#1} }
2286 }
```

(End of definition for `\newtheoremstyle`. This function is documented on page 39.)

10.2.2 Supporting QED in proofs

The `amsthm` package contains some elaborate code to support placing a QED symbol into the proof (by default at the end, but alternatively manually placed with `\qedhere`). This code is simply lifted and not adjusted in any way for now (and therefore also not documented—see the `amsthm` package for documentation for now).

```
2287 \ExplSyntaxOff
2288 \def\math@qedhere{%
2289 \ifundefined{\@currentenv @qed}{%
2290 \qed@warning\quad\hbox{\qedsymbol}%
2291 }{%
2292 \xp\aftergroup\csname\@currentenv @qed\endcsname
2293 }%
2294 }
```

```

2295 \def\displaymath@qed{%
2296   \relax
2297   \ifmmode
2298     \ifinner \aftergroup\linebox@qed
2299   \else
2300     \eqno
2301     \let\eqno\relax \let\leqno\relax \let\veqno\relax
2302     \hbox{\qedsymbol}%
2303   \fi
2304 \else
2305   \aftergroup\linebox@qed
2306 \fi
2307 }
2308 \expandafter\let\csname equation*@qed\endcsname\displaymath@qed
2309 \def\equation@qed{%
2310   \iftagsleft@
2311     \hbox{\phantom{\quad\qedsymbol}}%
2312     \gdef\alt@tag{%
2313       \rlap{\hbox to\displaywidth{\hfil\qedsymbol}}%
2314       \global\let\alt@tag\@empty
2315     }%
2316   \else
2317     \gdef\alt@tag{%
2318       \global\let\alt@tag\@empty
2319       \vtop{\ialign{\hfil####\cr
2320         \tagform@theequation\cr
2321         \qedsymbol\cr}}%

```

The original code in amsthm only contained `\setbox\z@` at this point to remove `\hbox` produced by `\maketag@_block`. However, when the sockets for the tagging are added this doesn't work any more because they come between the `\setbox` and the `\hbox`. We therefore set `measuring@` to true so that the branch without the sockets is used.

```

2322   \measuring@true
2323   \setbox\z@
2324 }%
2325 \fi
2326 }
2327 \def\qed@tag{%
2328   \global\tag@true \nonumber
2329   &\omit\setboxz@h {\strut@ \qedsymbol}\tagsleft@false
2330   \place@tag@gather
2331   \kern-\tabskip
2332   \ifst@rred \else \global\@eqnswtrue \fi \global\advance\row@\@ne \cr
2333 }
2334 \def\split@qed{%
2335   \def\endsplit{\crr\egroup \egroup \ctagsplit@false \rendsplitt@
2336     \aftergroup\align@qed
2337   }%
2338 }
2339 \def\align@qed{%
2340   \ifmeasuring@ \tag*{\qedsymbol}%
2341   \else \let\math@cr@@@ \qed@tag
2342   \fi
2343 }

```

```

2344 \expandafter\let\csname align*@qed\endcsname\align@qed
2345 \expandafter\let\csname gather*@qed\endcsname\align@qed
2346 %
2347 \DeclareRobustCommand{\qed}{%
2348   \ifmmode \mathqed
2349   \else
2350     \leavevmode\unskip\penalty9999 \hbox{}\nobreak\hfill
2351     \quad\hbox{\qedsymbol}%
2352   \fi
2353 }%
2354 \let\QED@stack\@empty
2355 \let\qed@elt\relax
2356 \newcommand{\pushQED}[1]{%
2357   \toks@{\qed@elt{#1}}\@temptokena\expandafter{\QED@stack}%
2358   \xdef\QED@stack{\the\toks@\the\@temptokena}%
2359 }%
2360 \newcommand{\popQED}{%
2361   \begingroup\let\qed@elt\popQED@elt \QED@stack\relax\relax\endgroup
2362 }%
2363 \def\popQED@elt#1#2\relax{#1\gdef\QED@stack{#2}}%
2364 \newcommand{\qedhere}{%
2365   \begingroup \let\mathqed\math@qedhere
2366   \let\qed@elt\setQED@elt \QED@stack\relax\relax \endgroup
2367 }%
2368 \def\setQED@elt#1#2\relax{%
2369   \ifmeasuring@
2370   \else \iffirstchoice@ \gdef\QED@stack{\qed@elt{#2}}\fi
2371   \fi
2372   #1%
2373 }%
2374 \def\qed@warning{%
2375   \PackageWarning{amsthm}{The \@nx\qedhere command may not work
2376     correctly here}%
2377 }%
2378 \newcommand{\mathqed}{\quad\hbox{\qedsymbol}}%
2379 \DeclareRobustCommand{\qed}{%
2380   \ifmmode \mathqed
2381   \else
2382     \leavevmode\unskip\penalty9999 \hbox{}\nobreak\hfill
2383     \quad\hbox{\qedsymbol}%
2384   \fi
2385 }
2386 \newcommand{\openbox}{\leavevmode
2387   \hbox to.77778em{%
2388     \hfil\vrule
2389     \vbox to.675em{\hrule width.6em\vfil\hrule}%
2390     \vrule\hfil}}
2391 \providecommand{\qedsymbol}{\openbox}
2392 \ExplSyntaxOn

```

11 Support for other packages and classes

11.1 Replacement for alltt

The tools package alltt by Leslie Lamport has been completely implemented using the template approach and is therefore no longer necessary. In fact it has also been extended by providing alltt*.

```
2393 \expandafter\let\csname ver@alltt.sty\endcsname\fmtversion
```

11.2 Replacement for amsthm

The amsthm package is basically supported out of the box (though there are currently still a few limitation with \newtheoremstyle and perhaps also in other places). So this here is a bit premature, but for now we disable loading amsthm and wait to see how far this gets us. We may have to provide a bit more for better compatibility.

```
2394 \expandafter\let\csname ver@amsthm.sty\endcsname\fmtversion
```

11.3 Support for amsart and amsbook classes

Unfortunately, the amsart class contains a full implementation of amsthm inside the class (why ever) and they use \newcommand, sigh.

Thus, to make the new code work with this class we have to hide some definitions, load the class and only afterwards restore our own versions.

So first save some of the problematical definitions under some other names:

```
2395 \let \amsnewtheorem      \newtheorem
2396 \let \amsnewtheoremstyle \newtheoremstyle
2397 \let \amstheoremstyle    \theoremstyle
2398 \let \amsproof           \proof
2399 \let \amsendproof        \endproof
```

Then undefine them just before the class gets loaded (quite a handful):

```
2400 \AddToHook{class/amsart/before}[block]{
2401   \let \newtheoremstyle \relax
2402   \let \theoremstyle    \relax

2403   \let \proof           \relax
2404   \let \endproof        \relax

2405   \let \pushQED         \relax
2406   \let \popQED          \relax
2407   \let \qedhere         \relax
2408   \let \mathqed         \relax
2409   \let \openbox         \relax
2410 }
```

Same for amsbook and amsproc:

```
2411 \AddToHook{class/amsbook/before}[block]{
2412   \let \newtheoremstyle \relax
2413   \let \theoremstyle    \relax
2414   \let \proof           \relax
2415   \let \endproof        \relax
2416   \let \pushQED         \relax
```

```

2417 \let \popQED      \relax
2418 \let \qedhere     \relax
2419 \let \mathqed      \relax
2420 \let \openbox      \relax
2421 }

2422 \AddToHook{class/amsproc/before}[block]{
2423   \let \newtheoremstyle \relax
2424   \let \theoremstyle    \relax
2425   \let \proof           \relax
2426   \let \endproof        \relax
2427   \let \pushQED         \relax
2428   \let \popQED          \relax
2429   \let \qedhere         \relax
2430   \let \mathqed         \relax
2431   \let \openbox         \relax
2432 }

```

And once the class is loaded restore our versions again. Note that we don't have to restore all the QED-related commands as ours are identical to those defined by the AMS.

```

2433 \AddToHook{class/amsart/after}[block]{
2434   \let \newtheorem      \amsnewtheorem
2435   \let \newtheoremstyle \amsnewtheoremstyle
2436   \let \theoremstyle    \amstheoremstyle
2437   \let \proof           \amsproof
2438   \let \endproof        \amsendproof
2439 }

2440 \AddToHook{class/amsbook/after}[block]{
2441   \let \newtheorem      \amsnewtheorem
2442   \let \newtheoremstyle \amsnewtheoremstyle
2443   \let \theoremstyle    \amstheoremstyle
2444   \let \proof           \amsproof
2445   \let \endproof        \amsendproof
2446 }

2447 \AddToHook{class/amsproc/after}[block]{
2448   \let \newtheorem      \amsnewtheorem
2449   \let \newtheoremstyle \amsnewtheoremstyle
2450   \let \theoremstyle    \amstheoremstyle
2451   \let \proof           \amsproof
2452   \let \endproof        \amsendproof
2453 }

```

11.4 Support for the `enumitem` interfaces

The current implementation incorporates most features of `enumitem`. The plan is that the `enumitem` interfaces are either natively available or are emulated and mapped to new interfaces, so that documents using `enumitem` work seamlessly.

Most (or even all of the `enumitem` keys have gotten new names, so there the task is to map old names to new names. One question to decide here is which (if any) of the original keys should remain natively available even if `enumitem` is not loaded, and which should only be supported if the document explicitly loads `enumitem`, i.e., support them

only for compatibility with old documents. Providing the full set by default means one ends up with a fairly inconsistent interface, but not providing some of them may result in people unnecessarily loading `enumitem` in new documents just to get at, say, `nosep`.

The `enumitem` package also provides declarations to build out new lists and adjust the layout of existing list using commands like `\newlist` or `\setlist`. With the new implementation this is normally done differently, e.g., defining instances and simple document level commands via `\SimpleBlockEnv`, etc. However, we probably also want a declaration such as `\newlist` (same name?) to provide a simple way to make this happen in the document preamble in one go.

I'm less sure about `\setlist` at least as far as its optional argument is concerned (even though we have to support it in an emulation).

We put the code that emulates `enumitem` in a separate file to be loaded instead of the original package, but eventually some of the code from there has to move back to the kernel to be always present.

```
2454 %\declare@file@substitution{enumitem.sty}{latex-lab-enumitem.sty}
```

But for now we simply disable `enumitem` loading and unconditionally load our replacement into the kernel for ease of testing. In the end we have to decide which parts of the interface (if any) we provide out of the box and which parts are only available if the document requests `enumitem`.

We do pretend, however, that `enumitem` has been loaded so that code testing for that takes the right branch.

```
2455 \expandafter\let\csname ver@enumitem.sty\endcsname\fmtversion
2456 \RequirePackage{latex-lab-enumitem}
```

11.5 Support for the `doc` package

When the `doc` package is loaded it wants to remove a `%` sign from the start of each verbatim line. For this it uses `\check@percent` which we stick into the `verbatim/startline` socket.

`doc` (*plug*)

```
2457 \NewSocketPlug {verbatim/startline}{doc}{ \check@percent }
2458 \AddToHook{package/doc/after}{
2459   \AssignSocketPlug{verbatim/startline}{doc}
2460 }
2461 \</package-finish>
```

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

\' 2077, 2079
 @@ commands:
 \l_@@_legacy_env_params_tl 352
 \\ 1081, 1346, 1347, 2081
 \{ 2082
 \} 2083
 _ 394, 815

A

\addpenalty 1052, 1221, 1225, 1587
 \AddPunctuation 32, 475
 \AddToHook 51, 97, 146, 164, 250, 349, 361,
 657, 1500, 1541, 1860, 1862, 2400,
 2411, 2422, 2433, 2440, 2447, 2458
 \AddToHookWithArguments 690, 692, 694, 696
 \addvspace .. 1053, 1222, 1226, 1228, 1588
 \advance 2332
 \aftergroup 57, 611,
 616, 658, 1056, 2292, 2298, 2305, 2336
 alltt (env.) 169
 alltt* (env.) 169
 \amsendproof 2399, 2438, 2445, 2452
 \amsnewtheorem 2395, 2434, 2441, 2448
 \amsnewtheoremstyle 2396, 2435, 2442, 2449
 \amsproof 2398, 2437, 2444, 2451
 \amstheoremstyle .. 2397, 2436, 2443, 2450
 \arabic 777
 \AssignSocketPlug 2459
 \AssignStructureRole 2057
 \AssignTaggingSocketPlug
 ... 653, 1498, 1709, 1859, 1881, 1979

B

\baselineskip 7
 \begin 40, 84, 1908
 \begingroup 1917, 2361, 2365
 \bfseries 346, 399
 block (type) 707
 block commands:
 \block_debug_off: .. 38, 668, 673, 687
 \block_debug_on: ... 38, 668, 668, 686
 \g_block_nesting_depth_int .. 39,
 953, 957, 961, 971, 999, 1008, 1020
 block internal commands:
 __block_beginpar_hmode:N
 1836, 1836, 1991, 2011, 2024

__block_beginpar_vmode:
 1818, 1818, 1993, 2013, 2026
 \l_block_block_instance_tl 899, 970
 \l_block_body_decls_tl
 1633, 1693, 1778, 1815
 \l_block_botsep_skip 1125, 1159
 \l_block_caption_after_skip ...
 1622, 1676, 1772, 1795
 \l_block_caption_before_skip ..
 1621, 1666, 1771, 1792
 \l_block_caption_decls_tl
 1624, 1663, 1773, 1789
 \l_block_caption_unbreakable_-
 bool 1240, 1620, 1770, 1802
 __block_captioned_everypar_std:
 1146, 1232, 1232, 1692, 1814
 \g_block_collected_separators_-
 tl 80, 1747
 __block_counter_label:n . 1356, 1402
 __block_counter_ref:n 1357
 \l_block_counter_start_int
 1284, 1318, 1321, 1330
 \l_block_counter_tl . 1282, 1316,
 1326, 1594, 1647, 1653, 1654, 1734
 __block_debug:n . 666, 666, 680, 2285
 \g_block_debug_bool
 45, 665, 670, 675, 681, 683
 __block_debug_endpe_typeout:n .
 ... 604, 608, 621, 632, 649, 927, 934
 __block_debug_gset: 668, 671, 676, 678
 __block_debug_typeout:n
 633, 642, 666,
 667, 682, 1018, 1042, 1070, 1145,
 1233, 1255, 1260, 1264, 1268, 1338,
 1340, 1405, 1451, 1503, 1528, 1533,
 1691, 1694, 1813, 1816, 1819, 1927,
 1931, 1953, 1963, 2164, 2165, 2226
 __block_do_note: 1719, 1719
 __block_do_number: 1728, 1728
 __block_do_punct: 82, 1739, 1739, 1794
 __block_do_separator: ... 1747, 1747
 __block_do_title: 82, 1710, 1710, 1793
 __block_drop_separators: ... 81,
 82, 1712, 1721, 1737, 1740, 1747, 1754
 \l_block_early_legacy_code_tl .
 53, 897, 968
 \l_block_effective_bot_skip ...
 1053, 1159, 1165, 1537

\l_block_effective_top_skip ...	1562, 1564, 1613, 1615, 1617, 1679,
..... 57, 61, 1158, 1164, 1226, 1537	1681, 1763, 1765, 1767, 1798, 1800
\l_block_env_name_tl	g_block_label_standalone_bool 1454
.... 892, 913, 1032, 1310, 1407, 1642	\l_block_label_strut_bool 1361, 1468
_block_evaluate_saved_user_-	\g_block_label_unchained_bool .
keys:nn	 1143, 1374, 1376, 1378, 1455,
1273, 1276, 1277, 1299, 1302, 1385	1614, 1616, 1618, 1764, 1766, 1768
_block_everypar	g_block_label_unchained_bool . 1454
63, 64	\g_block_labels_box
_block_everypar:	63, 68,
67, 71–73, 1146, 1256, 1339, 1452,	71, 78, 82, 1149, 1202, 1205, 1243,
1499, 1499, 1500, 1529, 1692, 1814	1437, 1439, 1457, 1513, 1516, 1566,
\l_block_final_code_tl 55, 906, 1000	1659, 1661, 1683, 1785, 1787, 1805
\l_block_flattened_bool	\l_block_late_legacy_code_tl ..
..... 928, 931, 1002	 898, 978
_block_if_list:TF	\l_block_legacy_env_params_tl .
.. 1026, 1040, 1064, 1064, 1066, 1069	 93, 2104, 2113
_block_inner_begin:	\l_block_legacy_support_bool ..
..... 1923, 1923, 2002, 2014	 1293, 1469
_block_inner_end:	_block_list_begin: 1939, 1939, 2027
..... 1926, 1926, 2003, 2015	_block_list_end: . 1957, 1957, 2028
\l_block_inner_instance_tl	_block_list_item_begin:
..... 905, 979, 983	 1946, 1946, 2029
\l_block_inner_instance_type_tl	_block_list_item_end:
..... 904, 982, 1065, 1067	 1949, 1949, 2030
\l_block_inner_level_counter_tl	\l_block_long_label_bool
..... 902, 946, 948, 951, 984, 986	 1435, 1436, 1444, 1459
_block_insert_separators:	_block_make_label_box:n
.... 80, 1714, 1723, 1731, 1747, 1750	 68, 1399, 1401, 1406, 1460, 1460
\l_block_instance_keys_tl	\l_block_max_inner_levels_tl ..
..... 97, 2231, 2233, 2236, 2242	 903, 949
_block_inter_item:	\l_block_next_line_bool
..... 74, 1558, 1578, 1578	 1363, 1443, 1511
\l_block_item_align_tl	\l_block_note_decls_tl .. 1628, 1724
.. 1367, 1368, 1369, 1424, 1428, 1456	_block_note_format:n ... 1632, 1725
\l_block_item_compatibility_-	\l_block_note_tl
bool	97, 1650, 1720, 1725, 2221, 2229, 2238
_block_item_everypar_first: ..	\l_block_number_decls_tl 1626, 1732
..... 1339, 1499, 1532	_block_number_format:n . 1630, 1733
_block_item_everypar_std:	\l_block_numbered_bool
..... 1452, 1499, 1502	 1609, 1649, 1651, 1729
_block_item_instance:n	\l_block_one_label_box
..... 74, 1286, 1545, 1560, 1561	. 71, 1416, 1420, 1422, 1426, 1427,
\l_block_item_label_tl	1431, 1432, 1434, 1441, 1457, 1462
..... 1283, 1333, 1335	\l_block_order_clist
\l_block_item_parsep_skip ... 1448	 78, 1623, 1667, 2199, 2205
_block_label_autoref:n 1359	\l_block_para_instance_tl
\l_block_label_boxed_bool 1362, 1413	 900, 974, 976
_block_label_format:n	\l_block_parbotsep_skip
..... 71, 1360, 1460, 1466	 61, 1126, 1165
\l_block_label_given_tl	\l_block_parindent_dim .. 1133, 1191
..... 69, 1353, 1384, 1393, 1408	\l_block_punct_decls_tl
_block_label_ref:n	 1627, 1743, 1775
\g_block_label_standalone_bool	_block_punct_format:n
..... 75, 1373, 1375, 1377, 1454,	 1631, 1744, 1777

\l__block_punct_tl	1611, 1741, 1744, 1761	block quotation-6 (instance)	139
__block_recipe_basic: ...	1989, 1989	block quote-1 (instance)	129
__block_recipe_list:	2021, 2021	block quote-2 (instance)	129
__block_recipe_noop:	1980, 1980	block quote-3 (instance)	129
__block_recipe_standalone:	1997, 1997	block quote-4 (instance)	129
__block_recipe_standard:	2008, 2008	block quote-5 (instance)	129
\l__block_resume_bool	1285, 1327	block quote-6 (instance)	129
__block_save_user_keys:n	1274	block std (template)	733, 1119
\l__block_separator_tl 80, 1610, 1748		block std-display-1 (instance)	32
__block_skip_remove_last:	660, 663, 1037, 1170, 1581, 1582	block std-display-2 (instance)	32
__block_skip_set_to_last:N	660, 660, 1046, 1210	block std-display-3 (instance)	32
\l__block_standalone_bool	907, 924, 925, 1056	block std-display-4 (instance)	32
\l__block_swap_number_bool	98, 2191, 2193, 2243, 2244	block std-display-5 (instance)	32
\l__block_tag_class_tl 894, 2034, 2036		block std-display-6 (instance)	32
\l__block_tag_inner_tag_tl	1924, 2005, 2006, 2017, 2018, 2020	block std-list-1 (instance)	306
\l__block_tag_name_tl	893, 2004, 2006, 2016, 2018, 2031, 2033	block std-list-2 (instance)	306
\l__block_tagging_recipe_tl	895, 923, 964, 1057, 1059	block std-list-3 (instance)	306
\l__block_text_font_tl	1364	block std-list-4 (instance)	306
\l__block_thmstyle_tl	95, 96, 1604, 2158, 2160, 2163, 2164, 2165, 2180, 2181, 2190, 2194, 2197, 2198, 2201, 2214	block std-list-5 (instance)	306
\l__block_title_decls_tl	1625, 1715, 1774	block std-list-6 (instance)	306
__block_title_format:n	1629, 1716, 1776	block thm-legacy2e-1 (instance)	440
\l__block_title_tl	76, 1595, 1711, 1716, 1760, 1784	block thm-legacy2e-2 (instance)	440
__block_tmp:w . 2275, 2277, 2281, 2283		block thm-plain-1 (instance)	423
\l__block_tmp_clist	97, 2204, 2207, 2215	block thm-plain-2 (instance)	423
\l__block_tmpa_skip	1210, 1211, 1212, 1536	block thm-remark-1 (instance)	431
\l__block_transparent_level_bool	896, 958, 998, 1019	block thm-remark-2 (instance)	431
\l__block_unchained_skip . 1123, 1153		block verbatim-1 (instance)	240
\l__block_unused_blockenv_keys_-tl 54, 973, 988, 992, 994, 1010		block verbatim-2 (instance)	240
block proof-1 (instance)	478	block verbatim-3 (instance)	240
block proof-2 (instance)	478	block verbatim-4 (instance)	240
block quotation-1 (instance)	139	block verbatim-5 (instance)	240
block quotation-2 (instance)	139	block verbatim-6 (instance)	240
block quotation-3 (instance)	139	block/list/label (socket)	1477
block quotation-4 (instance)	139	\BlockEnv	15
block quotation-5 (instance)	139	\BlockEnv	15, 39, 95, 1004, 2235
		blockenv (hook)	1003
		blockenv (type)	707
		blockenv alltt (instance)	208
		blockenv alltt* (instance)	224
		blockenv center (instance)	59
		blockenv description (instance)	290
		blockenv displayblock (instance)	6
		blockenv displayblockflattened (instance)	24
		blockenv enumerate (instance)	274
		blockenv flushleft (instance)	74
		blockenv flushright (instance)	93
		blockenv itemize (instance)	258
		blockenv list (instance)	372
		blockenv proof (instance)	451
		blockenv quotation (instance)	103
		blockenv quote (instance)	116
		blockenv std (template)	714, 890
		blockenv verbatim (instance)	174

blockenv verbatim* (instance)	191	\cr	2319, 2320, 2321, 2332
blockenv verse (instance)	150	\crrc	2335
\BlockEnvEnd	15	cs commands:	
\BlockEnvEnd	15, 39, 3, 5, 53, 55, 57, 99, 101, 148, 166, 168, 170, 172, 252, 254, 256, 359, 370, 450, 1017, 2151, 2156	\cs_generate_variant:Nn . . .	664, 1351
\BockEnvEnd	55	\cs_gset_protected:Npe	680, 682
bool commands:		\cs_if_exist:NTF	1669
\bool_gset_false:N . . .	675, 1373, 1374, 1375, 1378, 1564, 1613, 1614, 1615, 1618, 1763, 1764, 1765, 1768	\cs_if_free:NTF	1545
\bool_gset_true:N . . .	670, 1376, 1377, 1616, 1617, 1681, 1766, 1767, 1800	\cs_new:Npn	1064, 1066, 1274
\bool_if:NTF	628, 634, 638, 681, 683, 925, 941, 958, 998, 1019, 1056, 1143, 1240, 1327, 1396, 1443, 1444, 1468, 1469, 1511, 1562, 1651, 1679, 1729, 1798, 1802, 1847, 1849, 1871, 1873, 2191, 2193	\cs_new_eq:NN	621, 663, 666, 667, 1273, 1499, 1590, 1591
\bool_if:nTF	1411	\cs_new_protected:Npn	580, 660, 668, 673, 678, 686, 687, 689, 699, 1004, 1006, 1008, 1017, 1068, 1232, 1259, 1263, 1267, 1460, 1502, 1532, 1578, 1604, 1710, 1719, 1728, 1739, 1747, 1750, 1754, 1920, 1923, 1926, 1957, 1980, 1989, 1997, 2008, 2021, 2097, 2224, 2244
\bool_lazy_or:nnTF	1646, 2065	\cs_set:Npe	65, 1277, 1302
\bool_new:N	665, 1002, 1454, 1455, 1459, 2243	\cs_set:Npn	1092, 1102, 1818, 1836, 1939, 1946, 1949, 2272, 2273, 2274, 2275, 2278, 2279, 2280, 2281
\bool_set_eq:NN	928, 931	\cs_set_eq:NN	367, 1146, 1256, 1276, 1299, 1339, 1452, 1529, 1692, 1814, 1905, 1906, 1907, 1982, 1984, 1990, 1992, 1998, 2000, 2002, 2003, 2010, 2012, 2014, 2015, 2023, 2025, 2027, 2028, 2029, 2030
\bool_set_false:N . . .	929, 1436, 1649	\cs_set_protected:Npn	553, 570, 1882, 2247
\bool_set_inverse:N	2244	\csname	1915, 2115, 2133, 2292, 2308, 2344, 2345, 2393, 2394, 2455
\bool_set_true:N	924, 1435	\currentgrouptype	610, 613, 615
\BooleanFalse	15, 1007, 2155	D	
\BooleanTrue	14, 95, 449, 989, 2150	\DebugBlocksOff	38, 686
box commands:		\DebugBlocksOn	38, 686
\box_if_empty:NTF	1509	\DebugLegacySwitchesOn	46
\box_new:N	1457, 1458	\DebugSwitchesOff	689
\box_use_drop:N	1100, 1241, 1427, 1432, 1516, 1566, 1661, 1683, 1787, 1803	\DebugSwitchesOn	689
\box_wd:N	1416, 1420, 1434	\DebugTemplatesOff	38, 687
\break	1444	\DebugTemplatesOn	38, 686
C		\DeclareDocumentEnvironment	19, 98, 100, 147, 255
captionedtext (type)	707	\DeclareHookRule	1501
captionedtext proof (instance)	467	\DeclareInstance . . .	6, 24, 32, 59, 74, 103, 116, 129, 139, 150, 174, 191, 208, 224, 240, 258, 274, 290, 306, 324, 325, 326, 327, 328, 330, 332, 334, 336, 342, 344, 372, 386, 390, 423, 431, 440, 451, 467, 478, 486, 497, 508, 519, 535, 2167, 2186, 2248
captionedtext proof (template) .	796, 1758	\DeclareInstanceCopy	46, 47, 48, 49, 50, 89, 93,
captionedtext thmlike (template)	790, 1592		
\catcode	21, 2077, 2079		
center (env.)	51		
\centering	530		
clist commands:			
\clist_clear:N	2204		
\clist_map_inline:Nn	1667, 2205		
\clist_put_right:Nn	2207		
\clist_set:Nn	2199		
\clubpenalty	63, 64, 585, 1251, 1254, 1524, 1527		

<code>\frenchspacing</code>	2056	<code>\ignorespaces</code>	22, 730, 1574
<code>\indent</code>	1580, 1645	instances:	
G		<code>block proof-1</code>	478
<code>\gdef</code>	2312, 2317, 2363, 2370	<code>block proof-2</code>	478
<code>\global</code>	605,	<code>block quotation-1</code>	139
	609, 655, 656, 2314, 2318, 2328, 2332	<code>block quotation-2</code>	139
group commands:		<code>block quotation-3</code>	139
<code>\group_begin:</code>	1668, 1793, 1794	<code>block quotation-4</code>	139
<code>\group_end:</code>	1674, 1793, 1794	<code>block quotation-5</code>	139
H		<code>block quotation-6</code>	139
<code>\hbox</code>	101, 2290, 2302, 2311, 2313,	<code>block quote-1</code>	129
	2350, 2351, 2378, 2382, 2383, 2387	<code>block quote-2</code>	129
hbox commands:		<code>block quote-3</code>	129
<code>\hbox_gset:Nn</code> ..	1202, 1437, 1659, 1785	<code>block quote-4</code>	129
<code>\hbox_set:Nn</code>	1431, 1462	<code>block quote-5</code>	129
<code>\hbox_set_to_wd:Nnn</code>	1422	<code>block quote-6</code>	129
<code>\hbox_unpack_drop:N</code> ..	1149, 1205,	<code>block std-display-1</code>	32
	1242, 1426, 1439, 1441, 1513, 1804	<code>block std-display-2</code>	32
<code>\hfil</code>	1444, 2313, 2319, 2388, 2390	<code>block std-display-3</code>	32
<code>\hfill</code>	2350, 2382	<code>block std-display-4</code>	32
hook commands:		<code>block std-display-5</code>	32
<code>\hook_use:n</code>	1069,	<code>block std-display-6</code>	32
	1555, 1583, 1586, 1887, 1894, 1966	<code>block std-list-1</code>	306
<code>\hook_use:nnw</code>	913	<code>block std-list-2</code>	306
Hooks:		<code>block std-list-3</code>	306
<code>blockenv</code>	1003	<code>block std-list-4</code>	306
<code>item/after</code>	1539	<code>block std-list-5</code>	306
<code>item/before</code>	1539	<code>block std-list-6</code>	306
<code>para/begin</code>	71, 72	<code>block thm-legacy2e-1</code>	440
<code>\hrule</code>	2389	<code>block thm-legacy2e-2</code>	440
<code>\hss</code>	1367, 1368, 1369	<code>block thm-plain-1</code>	423
I		<code>block thm-plain-2</code>	423
<code>\ialign</code>	2319	<code>block thm-remark-1</code>	431
if commands:		<code>block thm-remark-2</code>	431
<code>\if_int_compare:w</code>	1890	<code>block verbatim-1</code>	240
<code>\if_meaning:w</code>	1094, 1108	<code>block verbatim-2</code>	240
<code>\IfBooleanF</code>	2117, 2188	<code>block verbatim-3</code>	240
<code>\IfBooleanT</code>	2226	<code>block verbatim-4</code>	240
<code>\IfBooleanTF</code>	2147	<code>block verbatim-5</code>	240
<code>\iffalse</code>	605, 655	<code>block verbatim-6</code>	240
<code>\ifhmode</code>	2049	<code>blockenv alltt</code>	208
<code>\ifinner</code>	2298	<code>blockenv alltt*</code>	224
<code>\IfInstanceExistsF</code>	2158, 2194	<code>blockenv center</code>	59
<code>\IfInstanceExistsTF</code>	2163, 2179	<code>blockenv description</code>	290
<code>\ifmmode</code>	2297, 2348, 2380	<code>blockenv displayblock</code>	6
<code>\IfNoValueF</code>	1784	<code>blockenv displayblockflattened</code> ..	24
<code>\IfNoValueTF</code>	2119, 2122, 2230	<code>blockenv enumerate</code>	274
<code>\ifnum</code>	610, 613, 615	<code>blockenv flushleft</code>	74
<code>\iftrue</code>	609, 656	<code>blockenv flushright</code>	93
<code>\IfValueT</code>	2227	<code>blockenv itemize</code>	258
<code>\ifvmode</code>	1870	<code>blockenv list</code>	372
<code>\ifx</code>	2090	<code>blockenv proof</code>	451
		<code>blockenv quotation</code>	103

blockenv quote	116	itemize (env.)	250
blockenv verbatim	174	\itemsep	39, 312, 741, 768, 1127, 1287, 1588
blockenv verbatim*	191	\itshape	408, 413, 473
blockenv verse	150		
captionedtext proof	467	J	
item basic	342	\justifying	530
item description	342		
list description	336	K	
list enumerate-1	328	\kern	1509, 2331
list enumerate-2	328	kernel (plug)	1703
list enumerate-3	328	kernel internal commands:	
list enumerate-4	328	_kernel_displayblock_begin:	..
list itemize-1	324	86, 87, 89, 1195, 1259,	
list itemize-2	324	1259, 1986, 1994, 2002, 2014, 2027	
list itemize-3	324	_kernel_displayblock_beginpar-	
list itemize-4	324	hmode:w	83, 1171, 1259,
list legacy	386	1263, 1982, 1990, 1998, 2010, 2023	
para center	497	_kernel_displayblock_beginpar-	
para justify	486	vmode:	83, 1167, 1259,
para raggedleft	519	1267, 1984, 1992, 2000, 2012, 2025	
para raggedright	508	_kernel_displayblock_end:	...
para verse	535	86, 87, 89, 1039, 1068,	
thmstyle definition	416	1068, 1987, 1995, 2003, 2015, 2028	
thmstyle legacy2e	421	_kernel_list_item_begin:	...
thmstyle plain	390	87, 1554, 1585, 1590, 1590, 2029	
thmstyle remark	410	_kernel_list_item_end:	...
\InstanceValue	2200	87, 1584, 1590, 1591, 2030	
int commands:		_kernel_list_label_after:n	...
_int_compare:nNnTF	...	1244, 1518, 1920, 1920	
936, 948, 953, 1182, 1318, 1826		keys commands:	
_int_gdecr:N	999, 1020	_keys_define:nn	68, 1352, 2220
_int_gincr:N	957	_keys_set:nn	576
_int_gset:Nn	1320, 1329	_keys_set_groups:nnnN	2232
_int_if_exist:NTF	1011	_keys_set_known:nnN	565
_int_incr:N	938, 943, 951, 1181	\KeyValue	37,
_int_new:N	1013	38, 131, 140, 310, 311, 739, 740, 778	
_int_set:Nn	1251, 1524		
_int_set_eq:NN	1254, 1527	L	
_int_to_roman:n	961	l internal commands:	
_int_use:N	970, 986	_l_block_style:nnnn	1596, 1602
_int_zero:N	1176	\labelenumi	329
_c_zero_int	1246, 1519	\labelenumii	331
\interlinepenalty	2046, 2049	\labelenumiii	333
iow commands:		\labelenumiv	335
_iow_term:n	684	\labelitemi	324
item (type)	707	\labelitemii	325
\item	10, 24, 40, 56, 60,	\labelitemiii	326
66, 69, 74, 77, 1390, 1481, 1541, 1580		\labelitemiv	327
item basic (instance)	342	\labelsep	772, 1292, 1440, 1442
item description (instance)	342	\labelwidth	...
item std (template)	775, 1352	366, 771, 1291, 1420, 1422, 1434, 1440	
item/after (hook)	1539	\language	2042
item/before (hook)	1539	\lastbox	595
\itemindent	770, 1290, 1440, 1509, 2100	\leavevmode	849, 1565, 2046, 2350, 2382, 2386

\leftmargin	315,
365, 745, 1132, 1192, 1193, 1204, 1206	
\leftskip	1076, 1173
legacy commands:	
\legacy_if:nTF	
....	630, 1021, 1027, 1043, 1044, 1139, 1140, 1147, 1162, 1179, 1199, 1208, 1218, 1219, 1236, 1248, 1506, 1521, 1534, 1552, 1579, 1645, 1820, 1821, 1928, 1950, 1958, 1960, 2105
\legacy_if_gset_false:n	
....	1024, 1040, 1041, 1150, 1235, 1238, 1250, 1505, 1508, 1523, 1556, 1823
\legacy_if_gset_true:n	
....	1054, 1154, 1337, 1568, 1572, 1687, 1688, 1690, 1809, 1810, 1812
\legacy_if_set_false:n	
.....	967, 1217, 1234, 1504, 2102
\legacy_if_set_true:n	1201
\legacyallttsetup	.. 39, 222, 238, <u>2072</u>
\legacylistsetup	... 28, 39, 93, 379, <u>2097</u>
\legacyverbatimsetup	 39, 92, 188, 205, <u>2039</u> , 2084
\leqno	 2301
\let	605, 609, 655, 656, 1986, 1987, 1994, 1995, 2053, 2075, 2077, 2079, 2080, 2085, 2092, 2093, 2103, 2301, 2308, 2314, 2318, 2341, 2344, 2345, 2354, 2355, 2361, 2365, 2366, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2434, 2435, 2436, 2437, 2438, 2441, 2442, 2443, 2444, 2445, 2448, 2449, 2450, 2451, 2452, 2455
\linewidth	 1192, 1194, 1414, 1416
list (env.)	 <u>349</u>
list (type)	 <u>707</u>
\list	 <u>363</u>
list description (instance)	 <u>336</u>
list enumerate-1 (instance)	 <u>328</u>
list enumerate-2 (instance)	 <u>328</u>
list enumerate-3 (instance)	 <u>328</u>
list enumerate-4 (instance)	 <u>328</u>
list itemize-1 (instance)	 <u>324</u>
list itemize-2 (instance)	 <u>324</u>
list itemize-3 (instance)	 <u>324</u>
list itemize-4 (instance)	 <u>324</u>
list legacy (instance)	 <u>386</u>
list std (template)	 <u>761</u> , <u>1280</u>
\listparindent	 7, 1449, 1450, 2098
\ltxlabblockdate	 550
\ltxlabblockversion	 550
M	
\makelabel	 367, 1470, 2103
\MakeLinkTarget	1399, 1401, 1407, 1654, 1657
maketag@@ internal commands:	
\maketag@@_block	 101
math internal commands:	
_math_tag_dollardollar_- display_end:	 42
\mathqed	 2348, 2365, 2378, 2380, 2408, 2419, 2430
\mdseries	 403, 2255
mode commands:	
\mode_if_horizontal:TF	 1036, 1581, 1884, 1889
\mode_if_inner:TF	 1885
\mode_if_vertical:TF	 1103, 1160
\mode_leave_vertical:	 1023, 1141
msg commands:	
\msg_error:nn	 1116
\msg_error:nnnn	 1031, 1309, 1351, 1389, 1641, 1671, 1897
\msg_new:nnnn	 1342, 1696
N	
\newcommand	 102, 811, 2064, 2356, 2360, 2364, 2378, 2386
\newcount	 305
\newcounter	 1015
\newcounteralias	 2143
\NewDocumentEnvironment	 2, 4, 169, 171, 447, 2149, 2154
\NewHook	 1539, 1540
\NewHookWithArguments	 1003
\newif	 654
\newline	 98, 1445
\newlist	 104
\newpage	 849, 856
\NewSocket	 2062
\NewSocketPlug	 2457
\NewTaggingSocket	 624, 1702, 1843, 1866, 1973
\NewTaggingSocketPlug	 626, 1477, 1703, 1845, 1868, 1975
\NewTemplateType	 707, 708, 709, 710, 711, 712, 713
\newtheorem	 28, 29, 31, 39, 76, 98, <u>2114</u> , 2395, 2434, 2441, 2448
\newtheoremstyle	 14, 15, 28, 30, 39, 97, 98, <u>102</u> , <u>2245</u> , 2396, 2401, 2412, 2423, 2435, 2442, 2449
\nobreak	 78, 1097, 1111, 1152, 1444, 2350, 2382

S

scan commands:

- \scan_stop: 1883
- \setbox 101, 595, 2323
- \setcounter 31, 1016
- \SetKnownTemplateKeys
..... 40, 51, 52, 553, 922, 1138,
1278, 1301, 1303, 1386, 1601, 1638
- \setlist 51, 104
- \SetTemplateKeys .. 41, 81, 570, 1087, 1783
- \ShowInstanceValues 2285
- \ShowTagging 1829, 1832, 1852
- \SimpleBlockEnv 15
- \SimpleBlockEnv 15, 27, 39, 104,
3, 5, 53, 55, 57, 99, 101, 148, 166,
168, 170, 172, 252, 254, 256, 357, 1004

skip commands:

- \skip_add:Nn 1164, 1165
- \skip_eval:n 1222, 1226
- \skip_horizontal:n .. 1204, 1206,
1440, 1442, 1666, 1676, 1792, 1795
- \skip_new:N 1536, 1537, 1538
- \skip_set:Nn 661, 1088, 1158, 1159
- \skip_set_eq:NN
..... 1174, 1175, 1196, 1197, 1448
- \skip_use:N 1090, 1093, 1107
- \skip_vertical:n
42, 587, 1049, 1050, 1153, 1211, 1212
- \skip_zero:N 1173
- \l_tmpa_skip ... 1046, 1047, 1049, 1050

\small 7

socket commands:

- \socket_if_exist:nTF
..... 622, 1841, 1864, 1971

Sockets:

- block/list/label 1477
- tagssupport/@doendpe 622
- tagssupport/block/recipe 1971
- tagssupport/block/startpara/direct
..... 1841
- tagssupport/captionedtext/caption
..... 1702
- tagssupport/kernel/endpe/vmode . 1864
- verbatim/startline 104, 2062

\space 550, 643, 910, 1032,
1085, 1136, 1296, 1310, 1347, 1382,
1599, 1636, 1642, 1781, 2164, 2165

str commands:

- \str_case:nnTF 2208
- \string 1547
- \strut 1468
- \swapnumbers 15, 28, 30, 95, 96, 98, 99, 2243

sys commands:

- \sys_if_engine luatex_p: 2066

\sys_if_output_dvi_p: 2067

T

\tabskip 2331

\tag 2340

tag commands:

- \tag_mc_begin:n 1485
- \tag_socket_use:n 590
- \tag_socket_use:nn 1921
- \tag_socket_use:nnn 1464,
1664, 1713, 1722, 1730, 1742, 1790
- \tag_struct_begin:n 1705
- \tag_struct_end: 1707

tag internal commands:

- \l__tag_block_flattened_level_-
int ... 52, 936, 938, 943, 1011, 1826
- \l__tag_L_attr_class_tl ... 1937,
1943, 2035, 2036, 2106, 2108, 2109
- \l__tag_L_tag_tl
..... 1935, 1936, 1942, 2032, 2033
- \l__tag_para_attr_class_tl ... 1082
- \l__tag_para_bool ... 628, 1847, 1871
- \l__tag_para_flattened_bool . 635,
638, 901, 928, 929, 931, 941, 1849, 1873
- __tag_para_main_store_struct: .
..... 1831, 1854
- \l__tag_para_main_tag_tl 643

\tagmcbegin 1489

\tagmcend 1492

\tagpdfparaOff 1839

\tagpdfparaOn 1839

\tagstructbegin 1488, 1496, 1924, 1940, 1947

\tagstructend
.. 1495, 1933, 1955, 1965, 1967, 1969

tagssupport/@doendpe (socket) 622

tagssupport/block/recipe (socket) .. 1971

tagssupport/block/startpara/direct
(socket) 1841

tagssupport/captionedtext/caption
(socket) 1702

tagssupport/kernel/endpe/vmode (socket)
..... 1864

template commands:

- \template_debug_typeout:n
..... 580, 580, 910, 1085,
1136, 1296, 1382, 1599, 1636, 1781

template internal commands:

- __template_debug_typeout:n 580

template types:

- block 707
- blockenv 707
- captionedtext 707
- item 707
- list 707

para	707	\@listctr	
thmstyle	707	... 64 , 66 , 966 , 1271 , 1320 , 1326 ,	
templates:		1329 , 1395 , 1399 , 1401 , 1402 , 2101	
block std	733 , 1119	\@listdepth	8 , 55 , 1008
blockenv std	714 , 890	\@listi	7 , 8 , 99
captionedtext proof	796 , 1758	\@listii	7 , 8
captionedtext thmlike	790 , 1592	\@listvi	8
item std	775 , 1352	\@makeother	2053
list std	761 , 1280	\@mklab	2103
para std	749 , 1072	\@ne	615 , 2332
thmstyle std	812 , 1607	\@new@specials	2088 , 2091 , 2093
TeX and L ^A T _E X 2 _ε commands:		\@newctr	2130
\@@par	85 , 1906 , 2046 , 2049	\@nmbrlisttrue	1325
\@beginparpenalty	9 , 1128 , 1225	\@nocounterr	2141
\@begintheorem	39	\@noitemerr	
\@centercr	506 , 517 , 528 , 544	... 67 , 87 , 1027 , 1183 , 1218 , 1534	
\@clubpenalty	585 , 1254 , 1527	\@noligs	2054
\@currenvir	1038 , 1912 , 2289 , 2292	\@normalcr	10 , 495 , 759
\@currenvline	1913	\@nthm	39
\@definecounter	2121	\@nx	2375
\@descriptiondepth	296 , 305	\@opargbegintheorem	39
\@doendpe	40 , 41 , 43 , 581	\@othm	39 , 94
\@domathendpefalse	588 , 599 , 654	\@outerparskip	
\@domathendpetrue	654	... 1050 , 1196 , 1213 , 1222 , 1227	
\@eha	1911	\@remove	2089 , 2092
\@ehc	1548	\@restorepar	584
\@empty	2314 , 2318 , 2354	\@rightskip	1088 , 1174
\@endparpenalty	9 , 1052 , 1129	\@setpar	1177
\@endpefalse	84 ,	\@setupverbinvisiblespace .	39 , 2063
85 , 591 , 597 , 603 , 658 , 1060 , 1877 , 1918		\@setupverbvisiblespace	92
\@endpetrue	57 , 581 , 603 , 658 , 1061	\@sxverbatim	22 , 206
\@endtheorem	39	\@tempswafalse	2043
\@enumdepth	24 , 286	\@tempswatrue	2048
\@eqnswtrue	2332	\@temptokena	2357 , 2358
\@execute@begin@hook	1914	\@thm	39
\@flushglue	10 ,	\@thmcounter	2126 , 2135
493 , 502 , 503 , 514 , 524 , 542 , 756 , 1175		\@thmcountersep	2134
\@ifdefinable	2115	\@toodeep	950 , 955
\@ifundefined	1910 , 2140 , 2289	\@topsep	73
\@ignorefalse	1916	\@topsepadd	73
\@inmatherr	1038 , 1544	\@totalleftmargin	91 , 1193 , 1194
\@item	846 , 854	\@vobeyspaces	2060
\@itemdepth	24 , 264	\@xnthm	39 , 94
\@itemlabel	40 , 64 , 67 , 354 ,	\@xobeysp	2070
965 , 1271 , 1335 , 1400 , 1483 , 1494 , 2107		\@xp	2292
\@itempenalty	9 , 1130 , 1289 , 1587	\@xthm	39
\@kernel@after@para@after	1895	\@xverbatim	189
\@kernel@after@para@end	1888	\@ynthm	39 , 94
\@kernel@refstepcounter ..	1395 , 1653	\@ythm	39
\@labels	71	\endpefalse	85
\@latex@error	1547 , 1911	\align@qed	2336 , 2339 , 2344 , 2345
\@latex@warning	2159	\alt@tag	2312 , 2314 , 2317 , 2318
\@list...	8	\arabic	12
		\begin	40

<code>\bibitem</code>	87	<code>\partopsep</code>	9
<code>\c@maxblocklevels</code>	40, 954, 1015	<code>\pdfmakespace</code>	22, 92
<code>\check@percent</code>	104, 2457	<code>\place@tag@gather</code>	2330
<code>\ctagsplit@false</code>	2335	<code>\popQED@elt</code>	2361, 2363
<code>\declare@file@substitution</code>	2454	<code>\propagate@doendpe</code>	611, 616
<code>\displaymath@qed</code>	2295, 2308	<code>\qed@elt</code>	2355, 2357, 2361, 2366, 2370
<code>\end</code>	41	<code>\QED@stack</code>	2354, 2357, 2358, 2361, 2363, 2366, 2370
<code>\equation@qed</code>	2309	<code>\qed@tag</code>	2327, 2341
<code>\everypar</code>	74, 75	<code>\qed@warning</code>	2290, 2374
<code>\g@addto@macro</code>	2091	<code>\rendsplit@</code>	2335
<code>\g@remfrom@specials</code>	2081, 2082, 2083, 2087	<code>\reserved@a</code>	1911, 1912, 1919
<code>\if@domathendpe</code>	586, 598, 654	<code>\rightmargin</code>	9
<code>\if@endpe</code>	604, 605, 608, 609, 658, 1551, 1870	<code>\row@</code>	2332
<code>\if@tempswa</code>	2045	<code>\setboxz@h</code>	2329
<code>\iffirstchoice@</code>	2370	<code>\SetKnownTemplateKeys</code>	64
<code>\ifmeasuring@</code>	2340, 2369	<code>\setQED@elt</code>	2366, 2368
<code>\ifst@rred</code>	2332	<code>\spacefactor</code>	67
<code>\iftagsleft@</code>	2310	<code>\split@qed</code>	2334
<code>\ignorespaces</code>	8, 55	<code>\strut</code>	12, 71
<code>\item</code>	16, 40, 56, 58, 62, 64, 65, 67–69, 71, 73, 87, 90	<code>\strut@</code>	2329
<code>\itemsep</code>	9	<code>\tag@true</code>	2328
<code>\l@nohyphenation</code>	2042	<code>\tagform@</code>	2320
<code>\labelsep</code>	11	<code>\tagsleft@false</code>	2329
<code>\labelwidth</code>	11, 68, 71	<code>\toks@</code>	2357, 2358
<code>\leftmargin</code>	9	<code>\topsep</code>	9
<code>\leftskip</code>	91	<code>\UseInstance</code>	74
<code>\legacylistsetup</code>	27	<code>\verbatim@font</code>	2054
<code>\linebox@qed</code>	2298, 2305	<code>\z@</code>	595, 613, 2323
<code>\list</code>	27	<code>\z@skip</code>	491, 492, 504, 513, 525, 526, 1090
<code>\list<romannumeral></code>	17, 25, 26	tex commands:	
<code>\listparindent</code>	59, 70	<code>\tex_everypar:D</code>	1105, 1106
<code>\makelabel</code>	11, 28, 71, 93	<code>\tex_hskip:D</code>	1098, 1112, 1891
<code>\math@cr@@@</code>	2341	<code>\tex_lastnodetype:D</code>	1890
<code>\math@qedhere</code>	2288, 2365	<code>\tex_lastskip:D</code>	661
<code>\measuring@true</code>	2322	<code>\tex_noindent:D</code>	1117
<code>\newline</code>	68	<code>\tex_par:D</code>	1893, 1902
<code>\newtheorem</code>	94	<code>\tex_parshape:D</code>	1194
<code>\newtheoremstyle@vspace@default</code>	2245, 2246, 2264, 2266	<code>\tex_parskip:D</code>	587
<code>\noitemerr</code>	56	<code>\tex_unskip:D</code>	663, 1886
<code>\on@line</code>	604, 608, 637, 644, 1018, 1145, 1233, 1255, 1338, 1451, 1503, 1528, 1533, 1691, 1813, 1820, 1913, 1927, 1931, 1953, 1963	<code>\the</code>	1114, 2055, 2076, 2078, 2358
<code>\org@dosppecials</code>	2080, 2085	<code>\theequation</code>	2320
<code>\org@prime</code>	2075, 2077, 2079	<code>\theoremstyle</code>	28, 30, 76, 95, 98, 1604, 2161, 2397, 2402, 2413, 2424, 2436, 2443, 2450
<code>\par</code>	37, 41–43, 57, 61, 62, 67, 75, 83, 85, 89	<code>\thmname</code>	99, 2272, 2278
<code>\par@deathcycles</code>	1176, 1181, 1182	<code>\thmnote</code>	99, 2274, 2280
<code>\parindent</code>	10, 31, 70	<code>\thmnumber</code>	99, 2273, 2279
<code>\parskip</code>	9, 31, 62	<code>\thmstyle (type)</code>	707
		<code>\thmstyle</code>	31
		<code>\thmstyle definition (instance)</code>	416
		<code>\thmstyle legacy2e (instance)</code>	421
		<code>\thmstyle plain (instance)</code>	390
		<code>\thmstyle remark (instance)</code>	410

