

Prototype reimplementa-tion of L^AT_EX 2_ε’s block environments using templates

L^AT_EX Project*

v0.9m 2026-01-26

Abstract

Contents

1	Introduction	4
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’	11
2.2.6	The captionedtext template ‘thmlike’	12
2.2.7	The captionedtext template ‘proof’	12
2.2.8	The thmstyle template ‘std’	13

*Initial reimplementa-tion 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	14
3.1	The <code>display</code> and <code>displayflattened</code> environments	15
3.1.1	Their <code>blockenv</code> instances	15
3.1.2	Their <code>block</code> instances	16
3.2	The <code>center</code> , <code>flushleft</code> , and <code>flushright</code> environments	16
3.2.1	Their <code>blockenv</code> instances	17
3.2.2	Their <code>block</code> instances	18
3.2.3	Their <code>para</code> instances	18
3.3	The <code>quote</code> and <code>quotation</code> environments	18
3.3.1	Their <code>blockenv</code> instances	18
3.3.2	Their <code>block</code> instances	19
3.4	The <code>verse</code> environment	19
3.4.1	Their <code>blockenv</code> instances	19
3.5	The <code>verbatim</code> , <code>verbatim*</code> and <code>alltt</code> environments	20
3.5.1	Their <code>blockenv</code> instances	20
3.5.2	Their <code>block</code> instances	22
3.6	The <code>trivlist</code> environment	22
3.7	The standard lists: <code>itemize</code> , <code>enumerate</code> , and <code>description</code>	22
3.7.1	Their <code>blockenv</code> instances	23
3.7.2	Their <code>block</code> instances	24
3.7.3	Their <code>list</code> instances	24
3.7.4	Their <code>item</code> instances	25
3.8	The legacy <code>list</code> and <code>trivlist</code> environments	25
3.8.1	Its <code>blockenv</code> instance	26
3.8.2	Its <code>list</code> instance	26
3.9	Theorem-like environments declared through <code>\newtheorem</code>	27
3.9.1	The <code>blockenv</code> instances they use	27
3.9.2	The <code>captionedtext</code> instances they use	28
3.9.3	The <code>thmstyle</code> instances they use	28
3.9.4	The <code>block</code> instances they use	29
3.10	The <code>proof</code> environment (from <code>amsthm</code>)	30
3.10.1	Block instances for the proofs	31
4	Declaring <code>para</code> instances	31
5	Advice on adjusting the layout of standard block environments	33
6	Tagging support	33
6.1	Paragraph tags	33
6.1.1	Tagging recipes	35
7	Tracing and debugging	36
8	New and redefined kernel command	37

9	The Implementation	38
9.1	Candidates for kernel changes	39
9.1.1	Augmented <code>\SetKnownTemplateKeys</code>	39
9.1.2	Tracing templates and instances	40
9.1.3	Handling <code>\par</code> after the end of the list	40
9.1.4	Other useful <code>expl3</code> commands	42
9.2	Tracing and debugging interfaces	42
9.3	Template types and template interfaces	44
9.4	Implementation of templates	46
9.4.1	Some notes on the $\text{\LaTeX} 2_{\varepsilon}$ legacy switches	46
9.4.1.1	Original usage:	47
9.4.1.2	Repurpose:	47
9.4.2	Implementation of <code>blockenv</code> templates	48
9.4.3	Implementation of <code>para</code> templates	53
9.4.4	Implementation of <code>block</code> templates	55
9.4.5	Implementation of <code>list</code> templates	59
9.4.6	Implementation of <code>item</code> templates	62
9.4.7	Implementation of <code>captionedtext</code> and <code>thmstyle</code> templates	69
9.5	Tagging support commands	75
9.5.1	List tags	79
9.5.2	Tagging recipes	81
10	Support code for document-level block environments	83
10.1	Verbatim-like environments	83
10.1.1	Helper commands for <code>verbatim</code> and <code>verbatim*</code>	83
10.1.2	Helper commands for <code>alltt</code> and <code>alltt*</code>	84
10.1.3	Helper command for legacy <code>list</code> environment	85
10.2	Theorem-like environments	86
10.2.1	Declarations for theorem-like environments	86
10.2.2	Supporting QED in proofs	92
11	Support for other packages and classes	94
11.1	Replacement for <code>alltt</code>	94
11.2	Replacement for <code>amsthm</code>	94
11.3	Support for <code>amsart</code> and <code>amsbook</code> classes	94
11.4	Support for the <code>enumitem</code> interfaces	96
11.5	Support for the <code>doc</code> package	96
	Index	97

1 Introduction

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

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

The approach taking here is therefore to offer separate 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 item parameters

Semantics:

Sets up paragraph-specific parameters for H&J, e.g., to implement justification variations, the behavior of `\l` 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 item 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 item 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 item 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`, and `list` are supported. Default: `standard`

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

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.
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 $\text{\LaTeX} 2_{\epsilon}$ 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 `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 `blockenv`s that can be nested into each other is restricted by the \LaTeX counter `maxblocklevels` with a default value of 6. If this value is increased then it is necessary to provide additional instances, e.g., `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 new functionality.

2.2.3 The para template ‘std’

Attributes:

para-indent (<i>length</i>)	Default: <code>\parindent</code>
begin-hspace (<i>skip</i>) Horizontal skip added just in front of the indentation box if non-zero	Default: <code>0pt</code>
left-hspace (<i>skip</i>)	Default: <code>0pt</code>
right-hspace (<i>skip</i>)	Default: <code>0pt</code>
end-hspace (<i>skip</i>)	Default: <code>\@flushglue</code>
fixed-word-spaces (<i>boolean</i>)	Default: <code>false</code>
final-hyphen-demerits (<i>integer</i>)	Default: <code>5000</code>
newline-cmd (<i>function(0)</i>) This defines the meaning of <code>\\</code>	Default: <code>\@normalcr</code>
para-attr-class (<i>tokenlist</i>)	Default: <code>justify</code>

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!`

TODO: to be further documented

2.2.4 The list template ‘std’

Attributes:

- counter** (*tokenlist*) Counter name to be used in a numbered list or empty, if the list is unnumbered. Default: `<empty>`
- item-label** (*tokenlist*) Label “string” for a fixed label or as generated from the current counter value. Default: `<empty>`
- start** (*integer*) Start value for the counter if the list is numbered, otherwise irrelevant. Default: 1
- resume** (*boolean*) Should a numbered list be resumed from the last instance?. Default: `false`
- item-instance** (*instance*) Instance of type `item` to be used to format the label string. Default: `basic`
- item-vspace** (*skip*) The space in front of an item in the list. If not specified the value specified in the block template instance is used.
- item-penalty** (*integer*) Penalty for breaking before an item (except the first). If not specified the value specified in the block template instance is used.
- item-indent** (*length*) Horizontal displacement of the item. Default: `0pt`
- label-width** (*length*) Width reserved for the formatted item label. Default: `\labelwidth`
- label-sep** (*length*) Horizontal separation between label and following text. Default: `\labelsep`
- legacy-support** (*boolean*) Is formatting the label via `\makelabel` supported? Default: `false`

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** (*function1*) *unused*. Default: `\arabic{#1}`
- counter-ref** (*function1*) *unused*. Default: value from `counter-label`
- label-ref** (*function1*) *unused*. Default: `#1`
- label-autoref** (*function1*) *unused*. Default: `item #1`
- label-format** (*function1*) Formatting of the label, questionable the way it is used. Default: `#1`
- label-strut** (*boolean*) Add a `\strut` to the label? Default: `false`

label-align (<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>

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 (<i>tokenlist</i>)	Heading for the environment unless overwritten on document level.	Default: <code>Proof</code>
punct (<i>tokenlist</i>)	Punctuation following the heading.	Default: <code>.</code>
caption-placement (<i>choice</i>)	Supported values <code>chained</code> , <code>unchained</code> , and <code>standalone</code>	Default: <code>unchained</code>
before-hspace (<i>skip</i>)	Horizontal displacement of the heading.	Default: <code>0pt</code>

after-hspace (<i>skip</i>)	Space following the heading, only relevant if text follows on the same line.	Default: 5pt
caption-decls (<i>tokenlist</i>)	Declarations that are applied to the whole caption, e.g., some font settings.	Default: <i><empty></i>
title-format (<i>function1</i>)	Formatting applied to the title value.	Default: #1
punct-format (<i>function1</i>)	Formatting applied to the punct value.	Default: #1
body-decls (<i>tokenlist</i>)	Declarations that are applied to body of the environment, e.g., font settings.	Default: <i><empty></i>

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 (<i>boolean</i>)	Is this kind of environment numbered?	Default: true
space (<i>tokenlist</i>)	Space to be applied between elements of the heading	Default: _
punct (<i>tokenlist</i>)	Punctuation following the heading.	Default: .
caption-placement (<i>choice</i>)	Supported values chained , unchained , and standalone	Default: unchained
before-hspace (<i>skip</i>)	Horizontal displacement of the heading.	Default: 0pt
after-hspace (<i>skip</i>)	Space following the heading, only relevant if text follows on the same line.	Default: 5pt
order (<i>commalist</i>)	Order of elements in the environment caption/heading. Supported values are title , number , punct , space , and note .	Default: title,space,number,space,note
caption-decls (<i>tokenlist</i>)	Declarations that are applied to the whole caption, e.g., some font settings.	Default: <i><empty></i>
title-format (<i>function1</i>)	Formatting applied to the title value.	Default: #1
number-format (<i>function1</i>)	Formatting applied to the number value.	Default: #1
punct-format (<i>tokenlist</i>)	Formatting applied to the punct value.	Default: #1
note-format (<i>function1</i>)	Formatting applied to the note value.	Default: (#1)
body-decls (<i>tokenlist</i>)	Declarations that are applied to body of the environment, e.g., font settings.	Default: <i><empty></i>

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 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 spaces are ignored.

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

Alternatively, one can omit using `space` 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`)
`displayblockflattened (env.)` which are similar to L^AT_EX 2_ε'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 L^AT_EX 2_ε's `trivlist`, i.e., it produces a vertical block with default setting, but doesn't put a list inside but uses a `<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  % ,legacy-code       =
14  % ,block-instance    = std-display
15  % ,para-instance     =
16  % ,tagging-suppress-paras = false
17  % ,inner-instance    =
18  % ,inner-instance-type =          % not relevant as there is no inner instance
19  % ,inner-level-counter =          % not relevant as there is no inner instance
20  % ,max-inner-levels   = 4          % not relevant as there is no inner instance
21  % ,final-code        = \ignorespaces
22 }
```

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.

```

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

```

3.1.2 Their block instances

We provide 6 nesting levels (as in L^AT_EX 2_ε). If you want to provide more you need to change the `maxblocklevels` counter, offer further `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).

```

30 \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-3 (inst.) 31 \DeclareInstance{block}{std-display-1}{std}
block std-display-4 (inst.) 32 {
block std-display-5 (inst.) 33 % ,begin-vspace          = \topsep
block std-display-6 (inst.) 34 % ,begin-extra-vspace    = \partopsep
35 % ,para-vspace          = \parskip
36 % ,end-vspace           = \KeyValue{begin-vspace}
37 % ,end-extra-vspace     = \KeyValue{begin-extra-vspace}
38 % ,item-vspace          = \itemsep
39 % ,begin-penalty        = \UserName{@beginparpenalty}
40 % ,end-penalty          = \UserName{@endparpenalty}
41 % ,left-margin          = 0pt
42 % ,right-margin         = \rightmargin
43 % ,para-indent          = 0pt
44 }
45 \DeclareInstanceCopy{block}{std-display-2}{std-display-1}
46 \DeclareInstanceCopy{block}{std-display-3}{std-display-1}
47 \DeclareInstanceCopy{block}{std-display-4}{std-display-1}
48 \DeclareInstanceCopy{block}{std-display-5}{std-display-1}
49 \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.

```

50 \AddToHook{begindocument/before}{./legacy-core}{
51     \RenewDocumentEnvironment{center} { !0{} }
52     { \SimpleBlockEnv{center}{#1} } { \BlockEnvEnd }

```



```

53 \RenewDocumentEnvironment{flushright} { !0{} }
54 { \SimpleBlockEnv{flushright}{#1} } { \BlockEnvEnd }

55 \RenewDocumentEnvironment{flushleft} { !0{} }
56 { \SimpleBlockEnv{flushleft}{#1} } { \BlockEnvEnd }
57 }

```

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.

```

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

```

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

```

72 %\DeclareInstance{blockenv}{flushleft}{std}
73 %{
74 %    name                = flushleft
75 %    ,tag-name            =
76 %    ,tag-attr-class      =
77 %    ,tagging-recipe      = basic
78 %    ,tagging-suppress-paras = true
79 %    ,inner-level-counter =
80 %    ,transparent-level   = true
81 %    ,legacy-code         =
82 %    ,block-instance      = std-display
83 %    ,para-instance       = raggedright
84 %    ,inner-instance      =
85 %}

```

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

```

86 \DeclareInstanceCopy{blockenv}{flushleft}{center}
87 \EditInstance{blockenv}{flushleft}{
88     name                = flushleft
89     ,para-instance      = raggedright }

```

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

```

90 \DeclareInstanceCopy{blockenv}{flushright}{center}

```

```

91 \EditInstance{blockenv}{flushright}{
92   name           = flushright
93   ,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_εX 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.

```

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

```

3.3.1 Their blockenv instances

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

```

100 \DeclareInstance{blockenv}{quotation}{std}
101 {
102   name           = quotation
103   ,tag-name       = \UseStructureName{block/quotation}
104   ,tag-attr-class =
105   ,tagging-recipe = standard
106   ,inner-level-counter =
107   ,transparent-level = false
108   ,legacy-code    =
109   ,block-instance = quotation
110   ,inner-instance =
111 }

```

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

```

112 \DeclareInstance{blockenv}{quote}{std}
113 {
114   name           = quote

```

```

115 ,tag-name           = \UseStructureName{block/quote}
116 ,tag-attr-class     =
117 ,tagging-recipe     = standard
118 ,inner-level-counter =
119 ,transparent-level   = false
120 ,legacy-code        =
121 ,block-instance     = quote
122 ,inner-instance     =
123 }

```

3.3.2 Their block instances

`block quote-1` (*inst.*) Default layout is to indent equally from both sides.

```

block quote-2 (inst.) 124 \DeclareInstance{block}{quote-1}{std}
block quote-3 (inst.) 125 { right-margin = \KeyValue{left-margin} }
block quote-4 (inst.) 126 \DeclareInstanceCopy{block}{quote-2}{quote-1}
block quote-5 (inst.) 127 \DeclareInstanceCopy{block}{quote-3}{quote-1}
block quote-6 (inst.) 128 \DeclareInstanceCopy{block}{quote-4}{quote-1}
129 \DeclareInstanceCopy{block}{quote-5}{quote-1}
130 \DeclareInstanceCopy{block}{quote-6}{quote-1}

```

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

```

block quotation-2 (inst.) 131 \DeclareInstance{block}{quotation-1}{std}
block quotation-3 (inst.) 132 { para-indent = 1.5em , right-margin = \KeyValue{left-margin} }
block quotation-4 (inst.) 133 \DeclareInstanceCopy{block}{quotation-2}{quotation-1}
block quotation-5 (inst.) 134 \DeclareInstanceCopy{block}{quotation-3}{quotation-1}
block quotation-6 (inst.) 135 \DeclareInstanceCopy{block}{quotation-4}{quotation-1}
136 \DeclareInstanceCopy{block}{quotation-5}{quotation-1}
137 \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.

```

138 \AddToHook{begindocument/before}[./legacy]{
139   \DeclareDocumentEnvironment{verse}{!0}{}
140   { \SimpleBlockEnv{verse} {#1} } { \BlockEnvEnd }
141 }

```

3.4.1 Their blockenv instances

`blockenv verse` (*inst.*)

```

142 \DeclareInstance{blockenv}{verse}{std}
143 {
144   name           = verse
145   ,tag-name       = \UseStructureName{block/verse}
146   ,tag-attr-class =
147   ,tagging-recipe = standard
148   ,inner-level-counter =
149   ,transparent-level = false

```

```

150 ,legacy-code      =
151 ,block-instance   = quote      % reuse?
152 ,para-instance    = verse
153 ,inner-instance    =
154 }

```

The special indentation on continuation lines (the way \LaTeX 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 than
`verbatim*` (*env.*) 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.

```

155 \AddToHook{begindocument/before}[./legacy-verbatim]{
156   \RenewDocumentEnvironment{verbatim}{={legacy-code} !o !o }
157   { \SimpleBlockEnv{verbatim} {#1} \obeyedline } { \BlockEnvEnd }

158   \RenewDocumentEnvironment{verbatim*}{={legacy-code} !o !o }
159   { \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.

```

160 \NewDocumentEnvironment{alltt}{={legacy-code} !o }
161 { \SimpleBlockEnv{alltt} {#1} } { \BlockEnvEnd }
162 \NewDocumentEnvironment{alltt*}{={legacy-code} !o }
163 { \SimpleBlockEnv{alltt*} {#1} } { \BlockEnvEnd }
164 }

```

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.

```

165 \DeclareInstance{blockenv}{verbatim}{std}
166 {
167   name                = verbatim
168   ,tag-name            = \UseStructureName{block/verbatim}
169   ,tag-attr-class      =
170   ,tagging-recipe      = standard
171   ,tagging-suppress-paras = true
172   ,inner-level-counter =
173   ,transparent-level   = true
174   ,legacy-code         =
175   ,block-instance      = verbatim
176   ,inner-instance      =

```

```
177 ,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.

```
178 ,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

```
179                         \@sxverbatim
180 }
```

The role-mapping is `<verbatim>` to `<Code>` and `<codeline>` to `<Sub>` (which is role mapped to `` 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 `<Lbl>` or `<Artifact><Lbl>`.

`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.

```
181 \DeclareInstance{blockenv}{verbatim*}{std}
182 {
183   name                = verbatim
184   ,tag-name           = \UseStructureName{block/verbatim}
185   ,tag-attr-class     =
186   ,tagging-recipe     = standard
187   ,tagging-suppress-paras = true
188   ,inner-level-counter =
189   ,transparent-level  = true
190   ,legacy-code       =
191   ,block-instance    = verbatim
192   ,inner-instance    =
193   ,para-instance     = justify
194   ,final-code        = \legacyverbatimsetup{visible}
195                       \@sxverbatim
196 }
```

`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.

```
197 \DeclareInstance{blockenv}{alltt}{std}
198 {
199   name                = alltt
200   ,tag-name           = \UseStructureName{block/verbatim} % private tag instead?
201   ,tag-attr-class     =
202   ,tagging-recipe     = standard
203   ,tagging-suppress-paras = true
204   ,inner-level-counter =
205   ,transparent-level  = true
206   ,legacy-code       =
207   ,block-instance    = verbatim
208   ,inner-instance    =
209   ,para-instance     = justify
```

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

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.

```
210 ,final-code          = \legacyallttsetup {invisible}
211 }
```

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

```
212 \DeclareInstance{blockenv}{alltt*}{std}
213 {
214     name                = alltt*
215     ,tag-name            = \UseStructureName{block/verbatim} % private tag instead?
216     ,tag-attr-class      =
217     ,tagging-recipe       = standard
218     ,tagging-suppress-paras = true
219     ,inner-level-counter  =
220     ,transparent-level    = true
221     ,legacy-code         =
222     ,block-instance      = verbatim
223     ,inner-instance      =
224     ,para-instance       = justify
225     ,final-code          = \legacyallttsetup {visible}
226 }
```

3.5.2 Their block instances

`block verbatim-1` (*inst.*) Verbatim instances have their own levels so that one can specify specific indentations
`block verbatim-2` (*inst.*) or vertical separations between lines.

```
block verbatim-3 (inst.) 227 \DeclareInstance{block}{verbatim-1}{std}
block verbatim-4 (inst.) 228 {
block verbatim-5 (inst.) 229     ,left-margin      = Opt
block verbatim-6 (inst.) 230     ,para-vspace     = Opt
231 }
232 \DeclareInstanceCopy{block}{verbatim-2}{verbatim-1}
233 \DeclareInstanceCopy{block}{verbatim-3}{verbatim-1}
234 \DeclareInstanceCopy{block}{verbatim-4}{verbatim-1}
235 \DeclareInstanceCopy{block}{verbatim-5}{verbatim-1}
236 \DeclareInstanceCopy{block}{verbatim-6}{verbatim-1}
```

3.6 The `trivlist` environment

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.

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

For the standard lists everything is managed by the `blockenv` instances.

```
237 \AddToHook{begindocument/before}[./legacy-lists]{
238 \RenewDocumentEnvironment{itemize}{!0{}}
239 { \SimpleBlockEnv{itemize} {#1} } { \BlockEnvEnd }
```

maybe we should simply implement it as a `displayblock` instance (at least when doing tagging) `itemize` (*env.*)
`enumerate` (*env.*)
`description` (*env.*)

```

240 \RenewDocumentEnvironment{enumerate}{!0{}}
241   { \SimpleBlockEnv{enumerate} {#1} } { \BlockEnvEnd }
242 \DeclareDocumentEnvironment{description}{!0{}}
243   { \SimpleBlockEnv{description} {#1} } { \BlockEnvEnd }
244 }

```

3.7.1 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.

```

245 \DeclareInstance{blockenv}{itemize}{std}
246 {
247   name                = itemize
248   ,tag-name           = \UseStructureName{block/itemize}
249   ,tag-attr-class     = itemize
250   ,tagging-recipe     = list
251   ,inner-level-counter = \@itemdepth
252   ,transparent-level  = false
253   ,max-inner-levels   = 4
254   ,legacy-code        =
255   ,block-instance     = std-list
256   ,inner-instance-type = list
257   ,inner-instance     = itemize
258   ,para-instance      =
259 }

```

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`.

```

260 \DeclareInstance{blockenv}{enumerate}{std}
261 {
262   name                = enumerate
263   ,tag-name           = \UseStructureName{block/enumerate}
264   ,tag-attr-class     = enumerate
265   ,tagging-recipe     = list
266   ,transparent-level  = false
267   ,max-inner-levels   = 4
268   ,legacy-code        =
269   ,block-instance     = std-list
270   ,inner-level-counter = \@enumdepth
271   ,inner-instance-type = list
272   ,inner-instance     = enumerate
273 }

```

²In the $\text{\LaTeX} 2_{\epsilon}$ 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`.

`blockenv description (inst.)` The `description` environment uses the `blockenv` instance `description`, the `block` instance `list-(level)`, and the inner instance `description` (no dependency on the nesting level), i.e., the environment has the same appearance on all nesting levels.

```

274 \DeclareInstance{blockenv}{description}{std}
275 {
276     name                = description
277     ,tag-name            = \UseStructureName{block/description}
278     ,tag-attr-class      = description
279     ,tagging-recipe      = list
280     ,inner-level-counter =
281     ,transparent-level   = false
282     ,legacy-code        =
283     ,block-instance      = std-list
284     ,inner-instance-type = list
285     ,inner-instance      = description
286 }

```

3.7.2 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 done in the legacy `\list<romannumeral>`) unless a different layout is wanted.

```

block std-list-4 (inst.)
block std-list-5 (inst.)
block std-list-6 (inst.)
287 \DeclareInstance{block}{std-list-1}{std}{
288 %     begin-vspace        = \topsep
289 %     ,begin-extra-vspace = \partopsep

```

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

```

290 ,para-vspace        = \parsep
291 % ,end-vspace        = \KeyValue{begin-vspace}
292 % ,end-extra-vspace  = \KeyValue{begin-extra-vspace}
293 % ,item-vspace       = \itemsep
294 % ,begin-penalty     = \UseName{@beginparpenalty}
295 % ,end-penalty       = \UseName{@endparpenalty}
296 % ,left-margin       = \leftmargin
297 % ,right-margin      = \rightmargin
298 % ,para-indent       = 0pt
299 }

300 \DeclareInstanceCopy{block}{std-list-2}{std-list-1}
301 \DeclareInstanceCopy{block}{std-list-3}{std-list-2}
302 \DeclareInstanceCopy{block}{std-list-4}{std-list-3}
303 \DeclareInstanceCopy{block}{std-list-5}{std-list-4}
304 \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.

3.7.3 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 definitions rather than defining the `item-label` code in the instance to ensure that old documents still work.

```
list itemize-4 (inst.)
305 \DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi }
306 \DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii }
307 \DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
308 \DeclareInstance{list}{itemize-4}{std}{ item-label = \labelitemiv }
```

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

```
list enumerate-3 (inst.)
list enumerate-4 (inst.)
309 \DeclareInstance{list}{enumerate-1}{std}
310 { item-label = \labelenumi , counter = enumi }
311 \DeclareInstance{list}{enumerate-2}{std}
312 { item-label = \labelenumii , counter = enumii }
313 \DeclareInstance{list}{enumerate-3}{std}
314 { item-label = \labelenumiii , counter = enumiii }
315 \DeclareInstance{list}{enumerate-4}{std}
316 { item-label = \labelenumiv , counter = enumiv }
```

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

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

Of course, if handling of `description` lists should differ in nested lists all one has to do is to provide an `inner-level-counter` and then define `description-1`, `description-2`, etc.

3.7.4 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).

```
318 \DeclareInstance{item}{basic}{std}
319 { label-align = right }
320 \DeclareInstance{item}{description}{std}
321 {
322   ,label-format = \normalfont\bfseries #1
323   ,label-align = left
324 }
```

3.8 The legacy list and trivlist environments

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

```
325 \AddToHook{begindocument/before}[./legacy]{
326   \RenewDocumentEnvironment{list}{0}{ m m }
327   {
```

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

```
328   \tl_set:Nn \l_@@_legacy_env_params_tl
329   {
330     \tl_set:Nn \@itemlabel {#2}
```

```

331         #3
332     }

```

The L^AT_EX 2_ε lists don't support captions so we use `\SimpleBlockEnv`.

```

333     \SimpleBlockEnv{list} {#1}
334 }
335 { \BlockEnvEnd }
336 }

```

`trivlist (env.)` L^AT_EX 2_ε defined `trivlist` as an implementation of `list` (or rather the other way around).

Replace with code
not using `\list`

```

337 \AddToHook{begindocument/before}[./legacy]{
338   \RenewDocumentEnvironment{trivlist}{!O{}}{
339     { \list[#1]}{
340       {
341         \dim_zero:N \leftmargin
342         \dim_zero:N \labelwidth
343         \cs_set_eq:NN \makelabel \use:n
344       }
345     }
346   { \BlockEnvEnd }
347 }

```

3.8.1 Its `blockenv` instance

`blockenv list (inst.)` The generic `list` environment of L^AT_EX 2_ε 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 L^AT_EX 2_ε) the `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.

```

348 \DeclareInstance{blockenv}{list}{std}
349 {
350   name = list
351   ,tag-name = \UseStructureName{block/list}
352   ,tag-attr-class =
353   ,tagging-recipe = list
354   ,transparent-level = false
355   ,legacy-code = \legacylistsetup
356   ,block-instance = std-list
357   ,inner-level-counter =
358   ,inner-instance-type = list
359   ,inner-instance = legacy
360 }

```

3.8.2 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.

```

361 \DeclareInstance{list}{legacy}{std} {
362   ,item-instance = basic
363   ,legacy-support = true
364 }

```

3.9 Theorem-like environments declared through `\newtheorem`

In standard \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`.

3.9.1 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
  ,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>`.

3.9.2 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.

3.9.3 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.

```
365 \DeclareInstance{thmstyle}{plain}{std}
366 {
367   ,caption-placement = unchained
368   ,numbered          = true
369   ,space              = \
370   ,punct              = .
371   ,before-hspace      = 0pt
372   ,after-hspace       = 5pt plus 1pt minus 1pt
373   ,order              = {title, space, number, punct, space, note}
374   ,caption-decls      = \bfseries
375   ,title-format       = #1
376   ,number-format      = #1
377   ,punct-format       = #1
```

```

378     ,note-format      = (#1)
379     ,body-decls       = \itshape
380 }

```

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

```

381 \DeclareInstanceCopy{thmstyle}{remark}{plain}
382 \EditInstance{thmstyle}{remark}
383 {
384     ,caption-decls = \itshape
385     ,body-decls    = \normalfont
386 }

```

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

```

387 \DeclareInstanceCopy{thmstyle}{definition}{plain}
388 \EditInstance{thmstyle}{definition}
389 {
390     ,body-decls = \normalfont
391 }

```

`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.

```

392 \DeclareInstanceCopy{thmstyle}{legacy2e}{plain}
393 \EditInstance{thmstyle}{legacy2e}{ punct = }

```

3.9.4 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.

```

394 \DeclareInstance{block}{thm-plain-1}{std}
395 {
396     ,begin-extra-vspace = 0pt
397     ,left-margin        = 0pt
398     ,para-indent        = \parindent
399     ,para-vspace        = \parskip
400 }
401 \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.

```

402 \DeclareInstance{block}{thm-remark-1}{std}
403 {
404     ,begin-vspace      = 0.5\topsep

```

```

405 ,begin-extra-vspace = 0pt
406 ,left-margin        = 0pt
407 ,para-indent        = \parindent
408 ,para-vspace        = \parskip
409 }
410 \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.

```

block thm-legacy2e-2 (inst.) 411 \DeclareInstance{block}{thm-legacy2e-1}{std}
412 {
413   ,left-margin          = 0pt
414   ,para-indent          = \parindent
415   ,para-vspace          = \parskip
416 }
417 \DeclareInstanceCopy{block}{thm-legacy2e-2}{thm-legacy2e-1}

```

3.10 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.

```

418 \NewDocumentEnvironment{proof}{={note}o }
419 { \pushQED{\qed}%
420   \ParseLaTeXeTheoremlike {proof} \BooleanTrue {#1} }
421 { \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.

```

422 \DeclareInstance{blockenv}{proof}{std}
423 {
424   name                = proof
425   ,tag-name            = \UseStructureName{block/proof}
426   ,tag-attr-class     =
427   ,tagging-recipe     = standalone
428   ,inner-level-counter =
429   ,transparent-level  = true
430   ,legacy-code       =
431   ,block-instance     = proof
432   ,inner-instance-type = captionedtext
433   ,inner-instance     = proof
434   ,para-instance     = justify
435 }

```

`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.

```

436 \DeclareInstance{captionedtext}{proof}{proof}
437 {
438   ,title          = Proof
439   ,punct          = .
440   ,before-hspace  = Opt
441   ,after-hspace   = 5pt plus 1pt minus 1pt
442   ,caption-decls  = \itshape
443   ,title-format   = #1
444   ,punct-format   = #1
445   ,body-decls     = \normalfont
446 }

```

3.10.1 Block instances for the proofs

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

```

447 \DeclareInstance{block}{proof-1}{std}
448 {
449   ,begin-vspace    = 6pt plus 6pt
450   ,left-margin     = Opt
451   ,para-indent     = \parindent
452   ,para-vspace     = \parskip
453 }
454 \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 an 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.

```

455 \DeclareInstance{para}{justify}{std}
456 {
457   % ,para-attr-class      = justify
458   % ,para-indent         = \parindent
459   % ,begin-hspace        = Opt
460   % ,left-hspace         = \z@skip
461   % ,right-hspace        = \z@skip
462   % ,end-hspace          = \@flushglue
463   % ,final-hyphen-demerits = 5000
464   % ,newline-cmd         = \@normalcr
465 }

```

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

```

466 \DeclareInstance{para}{center}{std}
467 {
468   ,para-attr-class      = center
469   ,para-indent          = Opt

```

```

470 % ,begin-hspace          = 0pt
471 ,left-hspace             = \@flushglue
472 ,right-hspace            = \@flushglue
473 ,end-hspace              = \z@skip
474 ,final-hyphen-demerits = 0
475 ,newline-cmd             = \@centercr
476 }

```

`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`.

```

477 \DeclareInstance{para}{raggedright}{std}
478 {
479   ,para-attr-class      = raggedright
480   ,para-indent          = 0pt
481   % ,begin-hspace       = 0pt
482   ,left-hspace          = \z@skip
483   ,right-hspace         = \@flushglue
484   ,end-hspace           = \z@skip
485   ,final-hyphen-demerits = 0
486   ,newline-cmd          = \@centercr
487 }

```

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

```

488 \DeclareInstance{para}{raggedleft}{std}
489 {
490   ,para-attr-class      = raggedleft
491   ,para-indent          = 0pt
492   % ,begin-hspace       = 0pt
493   ,left-hspace          = \@flushglue
494   ,right-hspace         = \z@skip
495   ,end-hspace           = \z@skip
496   ,final-hyphen-demerits = 0
497   ,newline-cmd          = \@centercr
498 }

```

this should be
moved elsewhere

Here are the attribute definitions used in the `para-attr-class` in the above instances:

```

499 \tagpdfsetup
500 {
501   ,role/new-attribute = {justify}    {/O /Layout /TextAlign/Justify}
502   ,role/new-attribute = {center}     {/O /Layout /TextAlign/Center}
503   ,role/new-attribute = {raggedright}{/O /Layout /TextAlign/Start}
504   ,role/new-attribute = {raggedleft} {/O /Layout /TextAlign/End}
505 }

```

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

```

506 \DeclareRobustCommand\centering {\UseInstance{para}{center}}{}
507 \DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}}{}
508 \DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}}{}
509 \DeclareRobustCommand\justifying {\UseInstance{para}{justify}}{}

```

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

```

510 \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.

```
511 \DeclareInstance{para}{verse}{std}
512 {
513   para-attr-class      = justify ,
514   para-indent          = 0pt ,
515   begin-hspace         = -1.5em ,
516   left-hspace          = 1.5em ,
517   right-hspace         = 0pt ,
518   end-hspace           = \@flushglue ,
519   final-hyphen-demerits = 0 ,
520   newline-cmd          = \@centercr ,
521 }
522 \</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 `<text-unit>` and use `<text>` (role-mapped to `<P>`) only for (portions of) the actual paragraph text in a way that the `<text>`s are not nested. As a result we have for a simple paragraph the structures

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

The `<text-unit>` 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 `<text-unit>` tags.

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

```

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

```

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

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

```

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

```

The `<itemize>` is roll-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 `<text-unit>` structure but their body uses simple `<text>` blocks and not `<text-unit><text>` inside, e.g., the input

```

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

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

```

will be tagged as follows:

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

The text-unit structures are added by using the tagging sockets `para/semantic/begin` and `para/semantic/end` declared in `ltagging.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:

standalone This recipe does the following:

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

basic This recipe does the following:

- Ensure that the `blockenv` is inside a `<text-unit>` structure, if necessary, start one.
- If inside a `<text-unit><text>`, then close the `</text>` but leave the `<text-unit>` open.
- Text inside the body of the environment start with `<text-unit><text>` if `tagging-suppress-paras` is set to `false`, otherwise just with `<text>`.
- At the end of the environment close `</text>` and possibly an inner `</text-unit>` if open.

- Then look if the environment is followed by an empty line (`\par`). If so, close the outer `</text-unit>` and start any following text with `<text-unit><text>`. Otherwise, don't and following text restarts with a just a `<text>` (and no paragraph indentation)

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

- it starts an inner tagging structure (i.e., which is therefore a child of the outer `<text-unit>`).
- By default this structure is a `<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 `<text-unit>` level from the outside.
- Then the lookahead for an empty line is done as described previously.

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

- the inner structure is a list (`<L>`).
- Furthermore everything is set up so that we have list items (``) with suitable substructures (`<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 rolemap.
- 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>`, ``, 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

cleanup

```
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 text-unit 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.
--	---

<hr/> <code>\newtheorem</code> <code>\newtheoremstyle</code> <hr/>	Reimplemented to fit the template approach. <code>\newtheoremstyle</code> was defined by <code>amsthm</code> .
<hr/> <code>\@nthm</code> <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> <hr/>	These are no longer used (to be removed).
<hr/> <code>\item</code> <code>\@itemlabel</code> <hr/>	The <code>\item</code> is redefined.
<hr/> <code>\c@maxblocklevels</code> <hr/>	A counter to increase or decrease the number of supported level. If increased, one needs to supply additional level instances.
<hr/> <code>\begin</code> <hr/>	The <code>\begin</code> is slightly redefined to handle <code>\@doendpe</code> better. TODO: move to kernel
<hr/> <code>\@doendpe</code> <hr/>	The original $\text{\LaTeX 2}_{\epsilon}$ command is augmented to allow for tagging.
<hr/> <code>\para_end:</code> <hr/>	TODO: consider name, document
<hr/> <code>para/begin</code> <hr/>	The <code>para/begin</code> hook is enhanced to support list ends

9 The Implementation

```

523 <*package-start>
524 <@@=block>
525 \ProvidesPackage {latex-lab-testphase-block}
526 [\ltlabblockdate\space v\ltlabblockversion\space
527 blockenv implementation]
```

9.1 Candidates for kernel changes

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

```
528 \RequirePackage{latex-lab-kernel-changes}
```

For testing we temporarily load it here (it has to come before the definition of `\DebugBlocksOff` below:

```
529 \RequirePackage{latex-lab-testphase-context}
```

```
530 \ExplSyntaxOn
```

9.1.1 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 `={legacy-code} !o` so that people can write `\begin{verbatim}[\small]` a syntax promoted by the TUGboat class.

```
531 \cs_set_protected:Npn \SetKnownTemplateKeys #1#2#3
```

```
532 {
```

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

```
533   \tl_if_empty:oTF {#3}
```

```
534   {
```

```
535     \tl_set_eq:NN \UnusedTemplateKeys \c_empty_tl
```

```
536   }
```

```
537   {
```

```
538     \tl_if_novalue:nTF {#3}
```

```
539     {
```

```
540       \tl_set_eq:NN \UnusedTemplateKeys \c_empty_tl
```

```
541     }
```

```
542     {
```

```
543       \keys_set_known:noN { template / #1 / #2 } {#3}
```

```
544       \UnusedTemplateKeys
```

```
545     }
```

```
546   }
```

```
547 }
```

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

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

```
548 \cs_set_protected:Npn \SetTemplateKeys #1#2#3
```

```
549 {
```

```
550   \tl_if_empty:oF {#3}
```

```
551   {
```

```
552     \tl_if_novalue:nF {#3}
```

```
553     {
```

```
554       \keys_set:no { template / #1 / #2 } {#3}
```

```
555     }
```

```
556   }
```

```
557 }
```

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

9.1.2 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).

```
558 \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 ??.)
```

9.1.3 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: use sockets for this and move to the kernel eventually.

```
559 \def\@doendpe{\@endpetrue
560 \def\par
561 {
```

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.

```
562 \if@domathendpe
563 \skip_vertical:n { \tex_parskip:D }
564 \@domathendpefalse
565 \fi
566 \@restorepar
567 \clubpenalty\@clubpenalty
```

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

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

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

```
569 \@endpefalse
570 \everypar{}
571 \par
572 }
573 \everypar{{\setbox\z@\lastbox}
574 \everypar{}
575 \@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:

```

576 %           \if@domathendpe
577             \@domathendpefalse
578 %           \fi
579   }
580 }

```

(End of definition for `\@doendpe`. This function is documented on page 38.)

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

```

581 \socket_if_exist:nF{ tagsupport/@doendpe }
582 {
583   \NewTaggingSocket {@doendpe}{0}
584 }

```

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

```

585 \NewTaggingSocketPlug {@doendpe}{default}
586 {
587   \bool_if:NT \l__tag_para_bool
588   {

```

Given that restoring `\par` through the legacy $\text{\LaTeX} 2_{\varepsilon}$ 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 `<text-unit>` structure). So we need to check for this.

```

589     \legacy_if:nT { @endpe }
590     {

```

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

```

591     \__block_debug_typeout:n
592     { flattened= \bool_if:NTF
593                 \l__tag_para_flattened_bool
594                 {true}{false}
595                 \on@line }
596     \bool_if:NF \l__tag_para_flattened_bool
597     {
598         \UseTaggingSocket{para/semantic/end}
599         {
600             \__block_debug_typeout:n{Structure-end~
601               \l__tag_para_main_tag_tl\space
602               after~ displayblock \on@line      }
603         }
604     }
605 }
606 }
607 }
608 \AssignTaggingSocketPlug{@doendpe}{default}

```

```

\if@domathendpe
\@domathendpefalse
\@domathendpetrue

```

Signal that special paragraph handling after a math display is required.

```

609 \newif\if@domathendpe
610 \def\@domathendpefalse{\global\let\if@domathendpe\iffalse}
611 \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):

```

612 \AddToHook{build/column/before}{%
613   \if@endpe \@endpefalse \aftergroup\@endpetrue \fi
614 }

```

9.1.4 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.

```

615 \cs_new_protected:Npn __block_skip_set_to_last:N #1 {
616   \skip_set:Nn #1 { \tex_lastskip:D }
617 }

```

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

```

618 \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:.)

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

```

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

```

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

9.2 Tracing and debugging interfaces

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

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

```

620 \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`.

```

621 \cs_new_eq:NN __block_debug:n \use_none:n
622 \cs_new_eq:NN __block_debug_typeof:n \use_none:n

```

(End of definition for __block_debug:n and __block_debug_typeof:n.)

`\block_debug_on:` Changing the debugging status.

`\block_debug_off:`

`__block_debug_gset:`

```

623 \cs_new_protected:Npn \block_debug_on:
624 {
625   \bool_gset_true:N \g__block_debug_bool
626   \__block_debug_gset:
627 }
628 \cs_new_protected:Npn \block_debug_off:
629 {
630   \bool_gset_false:N \g__block_debug_bool
631   \__block_debug_gset:
632 }
633 \cs_new_protected:Npn \__block_debug_gset:
634 {
635   \cs_gset_protected:Npx \__block_debug:n ##1
636   { \bool_if:NT \g__block_debug_bool {##1} }
637   \cs_gset_protected:Npx \__block_debug_typeout:n ##1
638   { \bool_if:NT \g__block_debug_bool
639     { \iow_term:x { ^J [Blocks]~ ==>~ ##1} } }
640 }

```

(End of definition for `\block_debug_on:`, `\block_debug_off:`, and `__block_debug_gset:`. These functions are documented on page 36.)

`\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).

```

641 \cs_new_protected:Npn \DebugBlocksOn { \block_debug_on: \template_debug_on: }
642 \cs_new_protected:Npn \DebugBlocksOff { \block_debug_off: \template_debug_off: }
643 \DebugBlocksOff

```

(End of definition for `\DebugBlocksOn` and `\DebugBlocksOff`. These functions are documented on page 36.)

`\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.

```

644 \cs_new_protected:Npn \DebugSwitchesOn {
645   \AddToHookWithArguments{cmd/legacy_if_gset_false:n/before}[debug]
646   {\typeout{[Switch]~==>~ ##1~==false~(global)}}
647   \AddToHookWithArguments{cmd/legacy_if_gset_true:n/before}[debug]
648   {\typeout{[Switch]~==>~ ##1~==true~(global)}}
649   \AddToHookWithArguments{cmd/legacy_if_set_false:n/before}[debug]
650   {\typeout{[Switch]~==>~ ##1~==false}}
651   \AddToHookWithArguments{cmd/legacy_if_set_true:n/before}[debug]
652   {\typeout{[Switch]~==>~ ##1~==true}}
653 }
654 \cs_new_protected:Npn \DebugSwitchesOff {
655   \RemoveFromHook{cmd/legacy_if_gset_false:n/before}[debug]
656   \RemoveFromHook{cmd/legacy_if_gset_true:n/before}[debug]
657   \RemoveFromHook{cmd/legacy_if_set_false:n/before}[debug]
658   \RemoveFromHook{cmd/legacy_if_set_true:n/before}[debug]
659 }
660 %\DebugSwitchesOn
661 %\DebugSwitchesOff

```

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

9.3 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)
662 \NewTemplateType{blockenv}{4}
663 \NewTemplateType{list}{4}
664 \NewTemplateType{captionedtext}{4}
665 \NewTemplateType{thmstyle}{4}
666 \NewTemplateType{block}{1}
667 \NewTemplateType{item}{1}
668 \NewTemplateType{para}{1}

```

`blockenv std` (*templ.*)

```

669 \DeclareTemplateInterface{blockenv}{std}{4}
670 {
671   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.

```

672 ,tag-name           : tokenlist =
673 ,tag-attr-class      : tokenlist =
674 ,tagging-recipe      : tokenlist = standard
675 ,transparent-level   : boolean   = false
676 ,legacy-code        : tokenlist =
677 ,block-instance     : tokenlist = std-display

```

Paragraph instance is normally inherited so no default.

```

678 ,para-instance      : tokenlist
679 ,inner-level-counter : tokenlist
680 ,max-inner-levels   : tokenlist = 4
681 ,inner-instance-type : tokenlist =
682 ,inner-instance     : tokenlist =
683 ,tagging-suppress-paras : boolean = false
684 ,final-code         : tokenlist = \ignorespaces
685 }

```

`block std` (*templ.*)

```

686 \DeclareTemplateInterface{block}{std}{1}
687 {
688 ,begin-vspace       : skip = \topsep
689 ,begin-extra-vspace : skip = \partopsep
690 ,begin-unchained-vspace : skip = .5\topsep
691 ,para-vspace        : skip = \parskip
692 ,end-vspace         : skip = \KeyValue{begin-vspace}
693 ,end-extra-vspace   : skip = \KeyValue{begin-extra-vspace}
694 ,item-vspace        : skip = \itemsep
695 ,begin-penalty      : integer = \UseName{@beginparpenalty}
696 ,end-penalty        : integer = \UseName{@endparpenalty}
697 ,item-penalty       : integer = \UseName{@itempenalty}

```

```

698 ,left-margin          : length = \leftmargin
699 ,right-margin         : length = \rightmargin
700 ,para-indent          : length = Opt
701 }

```

para std (*templ.*)

```

702 \DeclareTemplateInterface{para}{std}{1}
703 {
704   ,para-attr-class      : tokenlist = justify
705   ,para-indent          : length = \parindent
706   ,begin-hspace         : skip = Opt
707   ,left-hspace          : skip = Opt
708   ,right-hspace         : skip = Opt
709   ,end-hspace           : skip = \@flushglue
710   ,fixed-word-spaces    : boolean = false
711   ,final-hyphen-demerits : integer = 5000
712   ,newline-cmd          : function(0) = \@normalcr
713 }

```

list std (*templ.*)

```

714 \DeclareTemplateInterface{list}{std}{4}
715 {
716   ,counter              : tokenlist =
717   ,item-label           : tokenlist =
718   ,start                : integer = 1
719   ,resume               : boolean = false
720   ,item-instance        : instance{item} = basic
721   ,item-vspace          : skip = \itemsep
722   ,item-penalty         : integer = \UseName{@itempenalty}
723   ,item-indent          : length = \itemindent
724   ,label-width          : length = \labelwidth
725   ,label-sep            : length = \labelsep
726   ,legacy-support       : boolean = false
727 }

```

item std (*templ.*)

```

728 \DeclareTemplateInterface{item}{std}{1}
729 {
730   ,counter-label        : function{1} = \arabic{#1}
731   ,counter-ref          : function{1} = \KeyValue{counter-label}
732   ,label-ref            : function{1} = #1
733   ,label-autoref        : function{1} = item~#1
734   ,label-format         : function{1} = #1
735   ,label-strut          : boolean = false
736   ,label-align          : choice {left,center,right,parleft} = right
737   ,label-boxed          : boolean = true
738   ,next-line            : boolean = false      % <- review viz standalone below
739   ,text-font            : tokenlist
740   ,compatibility        : boolean = true
741   ,label-placement      : choice {chained,unchained,standalone} = chained ,
742 }

```

`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`.

```

743 \DeclareTemplateInterface{captionedtext}{thmlike}{4}
744 {
745   ,counter      : tokenlist =
746   ,title        : tokenlist =      % <- bad name?
747   ,style        : instance{thmstyle} = plain
748 }

```

`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`).

```

749 \DeclareTemplateInterface{captionedtext}{proof}{4}
750 {
751   ,title          : tokenlist = Proof
752   ,punct          : tokenlist = .
753   ,caption-placement : choice {chained,unchained,standalone} = unchained
754   ,before-hspace   : skip = 0pt
755   ,after-hspace    : skip = 5pt
756   ,caption-decls   : tokenlist =
757   ,title-format    : function{1} = #1
758   ,punct-format    : function{1} = #1
759   ,body-decls      : tokenlist =
760 }

```

`thmstyle std (templ.)`

```

761 \DeclareTemplateInterface{thmstyle}{std}{4}
762 {
763   ,numbered       : boolean = true
764   ,space          : tokenlist = \      % <- bad name?
765   ,punct          : tokenlist = .
766   ,caption-placement : choice {chained,unchained,standalone} = unchained
767   ,before-hspace   : skip = 0pt
768   ,after-hspace    : skip = 5pt
769   ,order          : commalist = { title, space, number, punct, space, note }
770   ,caption-decls   : tokenlist =
771   ,title-format    : function{1} = #1
772   ,number-format   : function{1} = #1
773   ,punct-format    : function{1} = #1
774   ,note-format     : function{1} = (#1)
775   ,body-decls      : tokenlist =
776 }

```

9.4 Implementation of templates

9.4.1 Some notes on the $\text{\LaTeX 2}_{\epsilon}$ legacy switches

$\text{\LaTeX 2}_{\epsilon}$ used a number of switches to manage its list environments and everything that was based on them.

For the reimplementaion 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.

9.4.1.1 Original usage:

```

777 %
778 % @newlist (global): signal that we are at the start of a list
779 %
780 % -> true at the start of a list before the first item when control
781 % is returned to document
782 % -> false in everypar setting the first item
783 % -> false at end of list if still true (after generating an error)
784 %
785 % -> tests: at list start setting @noparitemtrue and @noparlisttrue
786 %
787 %
788 % @inlabel (global): signaling that some item label waits to be typeset
789 %
790 % -> true in \@item
791 % -> false at list end
792 % -> false in everypar after label has been typeset
793 % -> false in \newpage after \leavemode to typeset item label
794 % (probably not needed)
795 %
796 % -> tests: at list start setting @noparitemtrue and @noparlisttrue
797 % -> tests: at list end to ensure that dangling label is typeset
798 % -> tests: in \@item to output a dangling item label by switching to hmode
799 % -> tests: in \everypar to output a dangling item label
800 % -> tests: in \newpage to output a dangling item label before the page is ended
801 % -> tests: in tagging hook {para/begin}{kernel}
802 %
803 %
804 % @noparlist (local):
805 %
806 % -> true at start of list if already @inlabel=true
807 % -> false at start of list otherwise
808 %
809 % -> tests: in \endtrivlist. If true suppress vertical spacing after the list
810 %
811 %
812 % @noparitem (local):
813 %
814 % -> true at start of list if already @inlabel=true
815 % -> false
816 %

```

9.4.1.2 Repurpose:

```

817 %
818 % Interpret legacy switches as follows (keeping the names for now)
819 %
820 % @newlist -> signals that we are at the start of a new block with a caption or
821 % at the start of a list block expecting an item next
822 %

```

```

823 %           In other words this is now really start of a block
824 %           with inner structure.
825 %
826 % @noparlist -> signals that we are on a new block with @inlabel already true, i.e.,
827 %               and this placement should happen horizontally
828 %
829 % @inlabel    -> Signals that we have at least one item or caption waiting to be typeset
830 %               inside the label box
831 %
832 % @noparitem -> dropped (handled directly)
833 %

```

9.4.2 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.)`

```

834 \DeclareTemplateCode{blockenv}{std}{4}
835 {
836   name                = \l__block_env_name_tl
837   ,tag-name           = \l__block_tag_name_tl
838   ,tag-attr-class     = \l__block_tag_class_tl
839   ,tagging-recipe     = \l__block_tagging_recipe_tl
840   ,transparent-level  = \l__block_transparent_level_bool
841   ,legacy-code       = \l__block_legacy_code_tl
842   ,block-instance     = \l__block_block_instance_tl
843   ,para-instance     = \l__block_para_instance_tl
844   ,tagging-suppress-paras = \l__tag_para_flattened_bool
845   ,inner-level-counter = \l__block_inner_level_counter_tl
846   ,max-inner-levels   = \l__block_max_inner_levels_tl
847   ,inner-instance-type = \l__block_inner_instance_type_tl
848   ,inner-instance     = \l__block_inner_instance_tl
849   ,final-code        = \l__block_final_code_tl
850 }
851 {
852   \template_debug_typeout:n{~\space template:~ 'std';~
853     arguments:~ \exp_not:n{|#1|#2|#3|#4|}}
854   \UseHook{blockenv}

```

We first evaluate the key list passed from the document (if any). All known keys are used, the remainder is stored in `\UnusedTemplateKeys` to be passed to any inner instances below.

```

855   \SetKnownTemplateKeys{blockenv}{std}{#1}

```

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

```

856   \int_compare:nNnTF \l__tag_block_flattened_level_int > 0

```



```

857     {
858       \int_incr:N \l__tag_block_flattened_level_int
859     }
860     {
861       \bool_if:NT \l__tag_para_flattened_bool
862       {
863         \int_incr:N \l__tag_block_flattened_level_int
864       }
865     }
866   \tl_if_empty:NF \l__block_inner_level_counter_tl
867   {
868     \int_compare:nNnTF \l__block_inner_level_counter_tl >
869       { \l__block_max_inner_levels_tl - 1 }
870     { \@toodeep }
871     { \int_incr:N \l__block_inner_level_counter_tl } % not clean "o"?
872   }

```

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.

```

873   \int_compare:nNnTF \g_block_nesting_depth_int >
874     { \c@maxblocklevels - 1 }
875   { \@toodeep }
876   {
877     \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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\varepsilon}$ lists worked as well.

```

878   \bool_if:NF \l__block_transparent_level_bool
879   {
880     \use:c { @list
881       \int_to_roman:n { \g_block_nesting_depth_int } }
882   }
883 }

```

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

```

884   \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_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.

```

885   \tl_clear:N \@itemlabel
886   \tl_clear:N \@listctr
887   \legacy_if_set_false:n { @nmbrrlist }

```

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

```

888   \l__block_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`).

```

889 \exp_args:Nee \UseInstance{block}
890 { \l__block_block_instance_tl - \int_use:N
891   \g_block_nesting_depth_int }
892 \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.

```

893 \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).

```

894 \tl_if_empty:NF \l__block_para_instance_tl
895 {

```

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

```

896   \exp_args:Ne \UseInstance{para}{ \l__block_para_instance_tl } {}
897 }

```

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.

```

898 \tl_if_empty:NF \l__block_inner_instance_tl
899 {

```

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.

```

900   \exp_args:Nee
901   \UseInstance{ \l__block_inner_instance_type_tl }
902   { \l__block_inner_instance_tl
903     \tl_if_empty:NF \l__block_inner_level_counter_tl
904       % not clean use "o"?
905       { - \int_use:N \l__block_inner_level_counter_tl }
906   }
907   \l__block_unused_blockenv_keys_tl
908   #2      % <-- \BooleanTrue or False
909   { #3 }  % <-- \NoValue or content
910   { #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.

```

911   \tl_set_eq:NN \l__block_unused_blockenv_keys_tl \UnusedTemplateKeys
912 }

```

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.

```

913 % \tl_if_empty:NF \l__block_unused_blockenv_keys_tl
914 % {
915 %   % check if only item template keys remain
916 % }

```

fix

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.

```
917 \bool_if:NT \l__block_transparent_level_bool
918 { \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.

```
919 \l__block_final_code_tl
920 }
```

decide

Might want a hook or a socket for legacy support.

```
921 \NewHook{blockenv}
```

`\BlockEnv`
`\SimpleBlockEnv`

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.

```
922 \cs_new_protected:Npn \BlockEnv          % #1#2#3#4 implicit
923 { \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.

```
924 \cs_new_protected:Npn \SimpleBlockEnv #1#2
925 { \UseInstance{blockenv}{#1}{#2} \BooleanFalse \NoValue \NoValue }
```

(End of definition for `\BlockEnv` and `\SimpleBlockEnv`. These functions are documented on page 37.)

`\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`:

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

(End of definition for `\g_block_nesting_depth_int`. This function is documented on page 37.)

`\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.

```
928 \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

```
929 \int_if_exist:NF \l_tag_block_flattened_level_int
930 {
931   \int_new:N \l_tag_block_flattened_level_int
932 }
```

(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.

```
933 \newcounter{maxblocklevels}
934 \setcounter{maxblocklevels}{6}
```

(End of definition for `\c@maxblocklevels`. This function is documented on page 38.)

`\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.

```
935 \cs_new_protected:Npn \BlockEnvEnd {
936   \__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:

```
937   \bool_if:NF \l__block_transparent_level_bool
938     { \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.

```
939   \legacy_if:nT { @inlabel }
940   {
941     \mode_leave_vertical:
942     \legacy_if_gset_false:n { @inlabel }
943   }
```

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.

```
944   \__block_if_list:T { \legacy_if:nT { @newlist } { \noitemerr } }
945   \mode_if_horizontal:TF
946     { \__block_skip_remove_last: \__block_skip_remove_last: \par }
947     { \@inmatherr{\end{\@currenvir}} }
```

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

```
948   \__kernel_displayblock_end:
```

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

```
949   \__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.

```
950   \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.

```
951 %   \__block_debug_typeout:n{@nparlist =
952 %                                   \legacy_if:nTF { @nparlist }{true}{false}}
953 \legacy_if:nF { @nparlist }
954 {
955   \__block_skip_set_to_last:N \l_tmpa_skip
956   \dim_compare:nNnT \l_tmpa_skip > \c_zero_dim
957   {
958     \skip_vertical:n { - \l_tmpa_skip }
959     \skip_vertical:n { \l_tmpa_skip + \parskip - \@outerparskip }
960   }
961   \addpenalty \@endparpenalty
962   \addvspace \l__block_topsepadd_skip
```

some redesign/ex-
tensions here?

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

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

decide

```
963 % \legacy_if_gset_true:n { @endpe }
964 }
```

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

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

```
965 \socket_use:n {block/endpe}
966 }
```

(End of definition for `\BlockEnvEnd`. This function is documented on page 37.)

`_block_if_list:T`
revisit and correct

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.

```
967 \cs_new:Npn \_block_if_list:T
968 { \tl_if_eq:NnT \l_block_block_instance_tl {std-list} }
```

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

`_kernel_displayblock_end:`

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

```
969 \cs_new_protected:Npn \_kernel_displayblock_end: {
970 \_block_debug_typeout:n{\detokenize{\_kernel_displayblock_end:}}
971 }
```

(End of definition for `_kernel_displayblock_end:`.)

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

```
972 \socket_new:nn {block/endpe} {0}
```

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

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

```
973 \socket_new_plug:nnn{block/endpe} {on} { \@endpetrue }
974 \socket_new_plug:nnn{block/endpe} {off} { \@endpefalse }
975 \socket_assign_plug:nn{block/endpe}{on}
```

9.4.3 Implementation of para templates

`para std (templ.)`

```
976 \DeclareTemplateCode{para}{std}{1}
977 {
978 ,para-indent = \parindent
```

integrate/fix

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

```
979 ,begin-hspace      = \l_para_begin_skip
980 ,left-hspace       = \leftskip
981 ,right-hspace      = \rightskip
982 ,end-hspace        = \parfillskip
```

fix

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

```
983 ,fixed-word-spaces = \l_par_fixed_word_spaces_bool % name??
984 ,final-hyphen-demerits = \finalhyphendemerits
985 ,newline-cmd       = \\
986 ,para-attr-class    = \l__tag_para_attr_class_tl
987 }
988 {
989   \template_debug_typeout:n{~\space template:~ 'std';~
990                                   argument:~ \exp_not:n{||#1|}}
991   \SetTemplateKeys{para}{std}{#1}
992   \skip_set:Nn \@rightskip \rightskip
993 }
```

`__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.

```
994 \tl_const:Ne \c_zero_skip_tl { \skip_use:N \z@skip }
995 \tl_new:N \l__para_begin_skip_tl
996 \cs_set:Npn \__para_handle_indent: {
997   \tl_set:Ne \l__para_begin_skip_tl { \skip_use:N \l_para_begin_skip }
998   \if_meaning:w \l__para_begin_skip_tl
999     \c_zero_skip_tl
1000   \else:
1001     \nobreak
1002     \tex_hskip:D \l_para_begin_skip
1003   \fi:
1004   \box_use_drop:N \g_para_indent_box
1005 }
```

(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.

```
1006 \cs_set:Npn \para_raw_noindent: {
1007   \mode_if_vertical:TF
1008   {
1009     \tex_everypar:D {
1010       \tex_everypar:D { \g__para_standard_everypar_tl }
1011       \tl_set:Ne \l__para_begin_skip_tl { \skip_use:N \l_para_begin_skip }
1012       \if_meaning:w \l__para_begin_skip_tl
1013         \c_zero_skip_tl
1014     \else:
1015       \nobreak
1016       \tex_hskip:D \l_para_begin_skip
1017     \fi:
1018   }
1019 }
```

```

1017         \fi:
1018         \the\everypar }
1019     }
1020     { \msg_error:nn { latex2e }{ raw-para } }
1021 \tex_noindent:D
1022 }

```

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

9.4.4 Implementation of block templates

`block std (templ.)` In contrast to the $\text{\LaTeX} 2_{\epsilon}$ 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.

```

1023 \DeclareTemplateCode{block}{std}{1}
1024 {
1025     ,begin-vspace           = \topsep
1026     ,begin-extra-vspace     = \partopsep
1027     ,begin-unchained-vspace = \l_block_unchained_skip
1028     ,para-vspace           = \parsep

```

fix

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

```

1029     ,end-vspace             = \l_block_botsep_skip
1030     ,end-extra-vspace      = \l_block_parbotsep_skip
1031     ,item-vspace           = \itemsep
1032     ,begin-penalty         = \@beginparpenalty
1033     ,end-penalty           = \@endparpenalty
1034     ,item-penalty          = \@itempenalty
1035     ,right-margin          = \rightmargin
1036     ,left-margin           = \leftmargin
1037     ,para-indent           = \l_block_parindent_dim
1038 }
1039 {
1040     \template_debug_typeout:n{~\space template:~ 'std';~
1041         argument:~ \exp_not:o{\exp_after:wN |#1|}}
1042     \SetKnownTemplateKeys{block}{std}{#1}

```

The code largely follows the logic of $\text{\LaTeX} 2_{\epsilon}$'s `trivlist` implementation as far as it is 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.

```

1043     \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.

```

1044         { \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.

```

1045     {

```

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

```

1046     \bool_if:NT \g__block_label_unchained_bool
1047     {
1048         \__block_debug_typeout:n{Set~ captioned~ block~ everypar \on@line }
1049         \cs_set_eq:NN \__block_everypar: \__block_captioned_everypar_std:
1050         \legacy_if:nT { @inlabel }
1051         {
1052             \hbox_unpack_drop:N \g__block_labels_box
1053             \legacy_if_gset_false:n { @inlabel }
1054             \par
1055             \nobreak
1056             \skip_vertical:n { \l__block_unchained_skip }
1057             \legacy_if_gset_true:n { @nobreak }
1058         }
1059     }
1060 }
1061 \skip_set:Nn \l__block_topsepadd_skip { \topsep }
1062 \mode_if_vertical:TF
1063 {
1064     \skip_add:Nn \l__block_topsepadd_skip { \partopsep }

```

At this point it is safe to add tagging structure(s) so we have a kernel-owned hook here for tagging. This is used to possibly start a paragraph structure (to surround the block, for example, in case of lists) and possibly do some other preparation for tagging the block.

```

1065     \__kernel_displayblock_beginpar_vmode:
1066 }
1067 {

```

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`.

```

1068     \__block_skip_remove_last: \__block_skip_remove_last:
1069     \__kernel_displayblock_beginpar_hmode:w \par
1070 }

```

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

```

1071     \skip_zero:N \leftskip
1072     \skip_set_eq:NN \rightskip \@rightskip
1073     \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.

```

1074     \int_zero:N \par@deathcycles
1075     \@setpar
1076     {
1077         \legacy_if:nTF { @newlist }
1078         {
1079             \int_incr:N \par@deathcycles
1080             \int_compare:nNnTF \par@deathcycles > { 1000 }
1081             { \noitemerr
1082               { \para_end: }

```



```

1083         }
1084     }
1085     {
1086         { \para_end: }
1087     }
1088 }

1089 \dim_set_eq:NN \parindent \l_block_parindent_dim
1090 \dim_add:Nn \linewidth { - \rightmargin - \leftmargin }
1091 \dim_add:Nn \@totalleftmargin { \leftmargin }
1092 \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.

```

1093 \__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 ...

```

1094 \skip_set_eq:NN \@outerparskip \parskip
1095 \skip_set_eq:NN \parskip \parsep
1096 %
1097 \legacy_if:nTF { @inlabel }
1098 {
1099     \legacy_if_set_true:n { @nparlist }
1100     \hbox_gset:Nn \g_block_labels_box
1101     {
1102         \skip_horizontal:n { - \leftmargin }
1103         \hbox_unpack_drop:N \g_block_labels_box
1104         \skip_horizontal:n { \leftmargin }
1105     }
1106     \legacy_if:nF { @minipage } % Why this chunk of code?
1107     {
1108         \__block_skip_set_to_last:N \l_block_tmpa_skip
1109         \skip_vertical:n { - \l_block_tmpa_skip }
1110         \skip_vertical:n { \l_block_tmpa_skip +
1111             \@outerparskip - \parsep }
1112     }
1113 }
1114 {
1115     \legacy_if_set_false:n { @nparlist }
1116     \legacy_if:nT { @newlist } { \noitemerr }
1117     \legacy_if:nTF { @nobreak }
1118     {

```

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

```

1119         \addpenalty{ 10000 }
1120         \addvspace{ \skip_eval:n{\@outerparskip-\parsep} }
1121     }
1122     {
1123         \addpenalty \@beginparpenalty

```

document 2e
logic used here

```

1124         \addvspace { \skip_eval:n { \l__block_topsepadd_skip +
1125                               \@outerparskip } }
1126     \addvspace { - \parsep }
1127 }
1128 }
1129 }

```

`__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).

```

1130 \cs_new_protected:Npn \__block_captioned_everypar_std: {
1131     \__block_debug_typeout:n{...~ in~ captioned~ block~ everypar \on@line }

```

First set some control switches to false:

```

1132     \legacy_if_set_false:n { @minipage }
1133     \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.

```

1134     \legacy_if:nT { @inlabel }
1135     {

```

Typeset the saved label (aka captioned text):

```

1136         \legacy_if_gset_false:n { @inlabel }
1137         \para_omit_indent:
1138         \box_use_drop:N \g__block_labels_box
1139         \__kernel_list_label_after:n { \PARALABEL }      % <- change
1140                                                         % this name
1141         \penalty \c_zero_int
1142     }

```

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

```

1143     \legacy_if:nTF { @nobreak }
1144     {
1145         \legacy_if_gset_false:n { @nobreak }
1146         \int_set:Nn \clubpenalty { 10000 }
1147     }
1148     {

```

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

```

1149         \int_set_eq:NN \clubpenalty \@clubpenalty
1150         \__block_debug_typeout:n{Set~ noop~ block~ everypar \on@line }
1151         \cs_set_eq:NN \__block_everypar: \prg_do_nothing:
1152     }
1153 }

```

(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.

```

1154 \cs_new_protected:Npn \__kernel_displayblock_begin: {
1155     \__block_debug_typeout:n
1156     {\detokenize{\__kernel_displayblock_begin:}}
1157 }
1158 \cs_new_protected:Npn \__kernel_displayblock_beginpar_hmode:w {
1159     \__block_debug_typeout:n
1160     {\detokenize{\__kernel_displayblock_beginpar_hmode:w}}
1161 }

```

```

1162 \cs_new_protected:Npn \__kernel_displayblock_beginpar_vmode: {
1163   \__block_debug_typeout:n
1164   {\detokenize{\__kernel_displayblock_beginpar_vmode:}}
1165 }

```

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

9.4.5 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`.

\@listctr

```

1166 \tl_new:N \@itemlabel      % should have a top-level definition
1167 \tl_new:N \@listctr        % should have a top-level definition

```

(End of definition for \@itemlabel and \@listctr. These functions are documented on page 38.)

__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.

```

1168 \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.

```

1169 %\cs_new:Npn \__block_save_user_keys:n #1 {
1170 %  \tl_if_empty:nTF {#1}
1171 %    { \cs_set_eq:NN \__block_evaluate_saved_user_keys:nn \use_none:nn }
1172 %    { \cs_set:Npe \__block_evaluate_saved_user_keys:nn ##1##2
1173 %      { \SetKnownTemplateKeys{##1}{##2}{\exp_not:n{#1}} } }
1174 %}

```

(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.

```

1175 \DeclareTemplateCode{list}{std}{4}
1176 {
1177   ,counter      = \l__block_counter_tl
1178   ,item-label    = \l__block_item_label_tl
1179   ,start        = \l__block_counter_start_int
1180   ,resume       = \l__block_resume_bool
1181   ,item-instance = \__block_item_instance:n
1182   ,item-vspace   = \itemsep

```

```

1183 % ,item-para-vspace = \parsep
1184 ,item-penalty = \@itempenalty
1185 ,item-indent = \itemindent
1186 ,label-width = \labelwidth
1187 ,label-sep = \labelsep
1188 ,legacy-support = \l_block_legacy_support_bool % FMi questionable
1189 }
1190 {
1191   \template_debug_typeout:n{~\space template:~ 'std';~
1192     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`.

```

1193   \tl_if_empty:oTF {#1}
1194     { \cs_set_eq:NN \_block_evaluate_saved_user_keys:nn \use_none:nn }
1195     {
1196       \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.

```

1197   \cs_set:Npe \_block_evaluate_saved_user_keys:nn ##1##2
1198     { \SetKnownTemplateKeys{##1}{##2}{
1199       \exp_not:o { \UnusedTemplateKeys }
1200     }
1201     \exp_not:n {
1202       \tl_if_empty:NF \UnusedTemplateKeys
1203         {
1204           \msg_error:nnee { block } { unknown-keys }
1205           { \l_block_env_name_tl \space environment}
1206           \UnusedTemplateKeys
1207         }
1208       }
1209     }
1210 }

```

Has this list a counter name defined in the instance?

```

1211   \tl_if_empty:NTF \l_block_counter_tl
1212   {

```

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.

```

1213     \int_compare:nNf \l__block_counter_start_int = 1
1214     {
1215         \int_gset:cn{ c@ \@listctr }
1216         { \l__block_counter_start_int - 1 }
1217     }
1218 }

```

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.

```

1219     {
1220         \@nbrlisttrue
1221         \tl_set_eq:NN \@listctr \l__block_counter_tl
1222         \bool_if:Nf \l__block_resume_bool
1223         {
1224             \int_gset:cn{ c@ \@listctr }
1225             { \l__block_counter_start_int - 1 }
1226         }
1227     }

```

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.

```

1228     \tl_if_empty:Nf \l__block_item_label_tl
1229     {
1230         \tl_set_eq:NN \@itemlabel \l__block_item_label_tl
1231     }

```

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).

```

1232     \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).

```

1233     \__block_debug_typeout:n{Set~ first~ block~ everypar \on@line }
1234     \cs_set_eq:NN \__block_everypar: \__block_item_everypar_first:

```

Think about a better implementation at some point.

```

1235 \__block_debug_typeout:n{template:list:std~end}
1236 }

```

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

```

1237 \msg_new:nnnn { block } { unknown-keys }
1238 { Some~ keys~ specified~ on~ the~ #1~ are~ unknown. }
1239 {
1240   The~ following~ keys~ are~ unknown~ and~ their~
1241   values~ are~ ignored:\\
1242   \space\space #2\\
1243   Perhaps~ a~ misspelling~ or~ the~ current~ template~
1244   instance~ uses~ special~ keys.
1245 }

```

9.4.6 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.

```

1246 \keys_define:nn { template/item/std }
1247   { label .tl_set:N = \l__block_label_given_tl }
1248 \DeclareTemplateCode{item}{std}{1}
1249 {
1250   ,counter-label    = \__block_counter_label:n
1251   ,counter-ref      = \__block_counter_ref:n
1252   ,label-ref        = \__block_label_ref:n
1253   ,label-autoref    = \__block_label_autoref:n
1254   ,label-format     = \__block_label_format:n
1255   ,label-strut      = \l__block_label_strut_bool
1256   ,label-boxed      = \l__block_label_boxed_bool
1257   ,next-line        = \l__block_next_line_bool
1258   ,text-font        = \l__block_text_font_tl
1259   ,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)

```

1260 ,label-align      = {
1261   left    = \tl_set:Nn \l__block_item_align_tl { \relax \hss } ,
1262   center  = \tl_set:Nn \l__block_item_align_tl { \hss \hss } ,
1263   right   = \tl_set:Nn \l__block_item_align_tl { \hss \relax } ,
1264   parleft = \NOT_IMPLEMENTED ,
1265 }

```

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

```

1266 ,label-placement = {
1267   chained    = \bool_gset_false:N \g__block_label_standalone_bool
1268               \bool_gset_false:N \g__block_label_unchained_bool ,
1269   unchained  = \bool_gset_false:N \g__block_label_standalone_bool
1270               \bool_gset_true:N  \g__block_label_unchained_bool ,
1271   standalone = \bool_gset_true:N  \g__block_label_standalone_bool
1272               \bool_gset_false:N \g__block_label_unchained_bool ,
1273 }
1274 }

```

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.

```
1275 {
1276   \template_debug_typeout:n{~\space template:~ 'std';~
1277   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.

```
1278   \tl_set_eq:NN \l__block_label_given_tl \c_novalue_tl
```

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

```
1279   \__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:

```
1280   \SetKnownTemplateKeys{item}{std}{#1}
1281   \tl_if_empty:NF \UnusedTemplateKeys
1282   {
1283     \msg_error:nnee { block } { unknown-keys }
1284     { \noexpand\item command }
1285     \UnusedTemplateKeys
1286   }
```

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[]`.

```
1287   \tl_if_novalue:oTF \l__block_label_given_tl
1288   {
```

next line needs checking after novalue implementation was changed

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

fix

```
1289   \tl_if_blank:oF \@listctr { \@kernel@refstepcounter \@listctr }
1290   \bool_if:NTF \l__block_item_compatibility_bool % not sure that
1291   % conditional
1292   % makes sense
1293   { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}%
1294   \@itemlabel } } % TODO ?
1295   { \__block_make_label_box:n { \MakeLinkTarget[\@listctr]{}%
1296   \__block_counter_label:n { \@listctr } } }
1297   }
1298   {
1299   \__block_debug_typeout:n{item~ with~ optional}
1300   \__block_make_label_box:n {
1301     \MakeLinkTarget [\l__block_env_name_tl]{}
1302     \l__block_label_given_tl
1303   }
```

```

1304     }
1305     \bool_if:nT
1306     {
1307         \l__block_label_boxed_bool
1308     %   TODO: is \linewidth correct?
1309         && \dim_compare_p:n
1310             { \box_wd:N \l__block_one_label_box <= \linewidth }
1311     }
1312     {
1313         \dim_compare:nNnT
1314             { \box_wd:N \l__block_one_label_box } < \labelwidth
1315         {
1316             \hbox_set_to_wd:Nnn \l__block_one_label_box { \labelwidth }
1317             {
1318                 \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.

```

1319 %   TODO: customize?
1320 %   \hbox_unpack_drop:N \l__block_one_label_box
1321     \box_use_drop:N \l__block_one_label_box
1322     \exp_after:wN \use_ii:nn \l__block_item_align_tl
1323   }
1324 }

```

Add another box level to the label box:

```

1325     \hbox_set:Nn \l__block_one_label_box
1326         { \box_use_drop:N \l__block_one_label_box }
1327   }
1328   \dim_compare:nNnTF { \box_wd:N \l__block_one_label_box } > \labelwidth
1329   { \bool_set_true:N \l__block_long_label_bool }
1330   { \bool_set_false:N \l__block_long_label_bool }
1331   \hbox_gset:Nn \g__block_labels_box
1332   {
1333       \hbox_unpack_drop:N \g__block_labels_box
1334       \skip_horizontal:n { \itemindent - \labelsep - \labelwidth }
1335       \hbox_unpack_drop:N \l__block_one_label_box
1336       \skip_horizontal:n { \labelsep }
1337       \bool_if:NT \l__block_next_line_bool
1338       { \bool_if:NT \l__block_long_label_bool { \nobreak \hfil \break } }
1339       % version of \newline inside an hbox that will be unpacked
1340   }
1341   % TODO??? FMi what's that?
1342   % \skip_set_eq:NN \parsep \l__block_item_parsep_skip

```

The next setting is for compatibility: The list template sets `\listparindent` to zero and otherwise doesn't use it any more. However, in the second argument of a legacy `list` environment the user may have set it explicitly to some other value and whatever value it had was then used for `\parindent` within the list. Now we use its value only if it differs from zero but otherwise use whatever the template instances specify. This gives 99.9% compatibility for legacy documents. 100% for definitions using the `list` environment and a setting inside, but if the user used `\listparindent` within the document, e.g.,

inside a `verse` environment there there is one case in which the setting is ignored, i.e., when it was set back to zero. That's a rather unlikely scenario, but it is not impossible. However, I couldn't think of an approach that circumvents such boundary cases.

```
1343 \dim_compare:nNnF \listparindent = {0pt}
1344 { \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.

```
1345 \__block_debug_typeout:n{Set~ item~ block~ everypar \on@line }
1346 \cs_set_eq:NN \__block_everypar: \__block_item_everypar_std:
1347 }
```

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

```
1348 \bool_new:N \g__block_label_standalone_bool % tmp until replaced
1349 \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`

```
1350 \tl_new:N \l__block_item_align_tl
(End of definition for \l__block_item_align_tl.)
```

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

```
1351 \box_new:N \l__block_one_label_box
1352 \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`.

```
1353 \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).

```
1354 \cs_new_protected:Npn \__block_make_label_box:n #1
1355 {
1356 \hbox_set:Nn \l__block_one_label_box
1357 {
```

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

```
1358 \tag_socket_use:nnn {block/list/label}{ }
1359 {
```

```

1360         \_block_label_format:n
1361         {
1362             \bool_if:NT \l_block_label_strut_bool { \strut }
1363             \bool_if:NTF \l_block_legacy_support_bool
1364                 \makelabel
1365                 \use:n
1366                 {#1}
1367         }

```

And what gets opened also needs closing:

```

1368     }
1369 }
1370 }

```

(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)

```

1371 \NewTaggingSocketPlug{block/list/label}{default}
1372 {
1373     %
1374     % FMi: this needs a different logic to decide when to make the label
1375     %     an artifact (after cleaning up the \item code ), therefore
1376     %     disabled for now
1377     % \tl_if_empty:oTF \@itemlabel
1378     % {
1379     %     \tag_mc_begin:n {artifact}
1380     % }
1381     % {
1382     \tagstructbegin{tag=\UseStructureName{block/list/label}}
1383     \tagmcbegin{tag=\UseStructureName{block/list/label}}
1384     % }
1385     #2
1386     \tagmcend % end mc-\UseStructureName{block/list/label} or artifact
1387     % FMi: unconditionally for now
1388     % \tl_if_empty:oF \@itemlabel
1389     \tagstructend % end label
1390     \tagstructbegin{tag=\UseStructureName{block/list/body}}
1391 }
1392 \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).

`_block_item_everypar_std:`

`_block_item_everypar_first:`

```

1393 \cs_new_eq:NN \_block_everypar: \prg_do_nothing:
1394 \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.

```

1395 \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.

```

1396 \cs_new_protected:Npn \__block_item_everypar_std: {
1397   \__block_debug_typeout:n{...~ in~ item~ block~ everypar \on@line }
1398   \legacy_if_set_false:n { @minipage }
1399   \legacy_if_gset_false:n { @newlist }
1400   \legacy_if:nT { @inlabel }
1401   {
1402     \legacy_if_gset_false:n { @inlabel }
1403     \box_if_empty:NT \g_para_indent_box { \kern - \itemindent }
1404     \para_omit_indent:
1405     \box_use_drop:N \g__block_labels_box

```

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

```

1406   \__kernel_list_label_after:n {LI-}
1407   \penalty \c_zero_int
1408   }
1409   \legacy_if:nTF { @nobreak }
1410   {
1411     \legacy_if_gset_false:n { @nobreak }
1412     \int_set:Nn \clubpenalty { 10000 }
1413   }
1414   {
1415     \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.

```

1416   \__block_debug_typeout:n{Set~ noop~ block~ everypar \on@line }
1417   \cs_set_eq:NN \__block_everypar: \prg_do_nothing:
1418   }
1419 }

```

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

```

1420 \cs_new_protected:Npn \__block_item_everypar_first: {
1421   \__block_debug_typeout:n{...~ in~ first~ block~ everypar \on@line }
1422   \legacy_if:nT { @newlist } { \@noitemerr }
1423 }

```

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

`\l__block_tmpa_skip`

```

1424 \skip_new:N \l__block_tmpa_skip
(End of definition for \l__block_tmpa_skip.)

```

`\l__block_topsepadd_skip`
`\l__block_effective_top_skip`

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

```

1425 \skip_new:N \l__block_topsepadd_skip
1426 \skip_new:N \l__block_effective_top_skip
(End of definition for \l__block_topsepadd_skip and \l__block_effective_top_skip.)

```

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

```
1427 \AddToHook{begindocument/before}[./legacy-lists]{
1428   \RenewDocumentCommand{\item}{={label}o }
1429   {
1430     \@inmatherr \item
```

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

```
1431     \cs_if_free:NTF \__block_item_instance:n
1432     {
1433       \@latex@error{Lonely~\string\item--perhaps-a-missing-
1434         list~environment}\@ehc
1435     }
1436     {
1437       \legacy_if:NTF { @newlist }
1438       {
1439         \__kernel_list_item_begin:
```

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

```
1440         \legacy_if_gset_false:n { @newlist }
1441       }
1442       { \__block_inter_item: }
```

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

```
1443     \tl_if_novalue:NTF {#1}          % avoids reparsing label={}
1444     { \__block_item_instance:n { } }
1445     { \__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.

```
1446     \bool_if:NTF \g__block_label_standalone_bool
1447     {
1448       \bool_gset_false:N \g__block_label_standalone_bool
1449       \leavevmode
1450       \box_use_drop:N \g__block_labels_box
1451       \par
1452       \legacy_if_gset_true:n { @nobreak } % do not break after
1453                                           % a standalone item
1454     }
1455     {
1456       \legacy_if_gset_true:n { @inlabel }
1457     }
1458     \ignorespaces
1459   }
1460 }
1461 }
```

(End of definition for `\item`. This function is documented on page 38.)

support extra vertical space as well?

`__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.

```
1462 \cs_new_protected:Npn \__block_inter_item: {
1463   \legacy_if:nT { @inlabel }
1464     { \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).

```
1465   \mode_if_horizontal:T { \__block_skip_remove_last:
1466     \__block_skip_remove_last: \par }
```

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

```
1467   \__kernel_list_item_end:
1468   \__kernel_list_item_begin:
1469   \addpenalty \@itempenalty
1470   \addvspace \itemsep
1471 }
```

(End of definition for `__block_inter_item:`)

```
\__kernel_list_item_begin:
\__kernel_list_item_end:
```

```
1472 \cs_new_eq:NN \__kernel_list_item_begin: \prg_do_nothing:
1473 \cs_new_eq:NN \__kernel_list_item_end: \prg_do_nothing:
```

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

9.4.7 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.

```
1474 \DeclareTemplateCode{captionedtext}{thmlike}{4}
1475 {
1476   ,counter          = \l__block_counter_tl
1477   ,title            = \l__block_title_tl
1478   ,style            = \l__block_style:nnnn
1479 }
1480 {
```

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

```
1481   \template_debug_typeout:n{~\space template:~ 'thmlike';~
1482     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.

```
1483   \SetKnownTemplateKeys{captionedtext}{thmlike}{#1}
```

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

```
1484   \l__block_style:nnnn \UnusedTemplateKeys {#2} {#3} {#4}
1485 }
```

`\theoremstyle` All that the `\theoremstyle` declaration does is saving its argument so that it can be used in `\newtheorem`.

```
1486 \cs_new_protected:Npn \theoremstyle #1{ \tl_set:Nn \l__block_thmstyle_tl {#1} }
1487 \tl_new:N \l__block_thmstyle_tl
```

And the default is plain:

```
1488 \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.

```
1489 \DeclareTemplateCode{thmstyle}{std}{4}
1490 {
1491   ,numbered      = \l_block_numbered_bool
1492   ,space         = \l_block_space_tl
1493   ,punct         = \l_block_punct_tl
1494   ,caption-placement = {
1495     chained      = \bool_gset_false:N \g_block_label_standalone_bool
1496                  \bool_gset_false:N \g_block_label_unchained_bool
1497     ,unchained    = \bool_gset_false:N \g_block_label_standalone_bool
1498                  \bool_gset_true:N  \g_block_label_unchained_bool
1499     ,standalone   = \bool_gset_true:N  \g_block_label_standalone_bool
1500                  \bool_gset_false:N \g_block_label_unchained_bool
1501   }
1502   ,before-hspace = \l_block_caption_before_skip
1503   ,after-hspace  = \l_block_caption_after_skip
1504   ,order         = \l_block_order_clist
1505   ,caption-decls = \l_block_caption_decls_tl
1506   ,title-format  = \__block_title_format:n
1507   ,number-format = \__block_number_format:n
1508   ,punct-format  = \__block_punct_format:n
1509   ,note-format   = \__block_note_format:n
1510   ,body-decls    = \l_block_body_decls_tl
1511 }
1512 {
```

Some tracing:

```
1513 \template_debug_typeout:n{~\space template:~ 'std';~
1514                      arguments:~ \exp_not:o{\exp_after:wN |#1|#2|#3|#4|}}}
```

Applying any user keys:

```
1515 \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:

```
1516 \tl_if_empty:NF \UnusedTemplateKeys
1517 {
1518   \msg_error:nnee { block } { unknown-keys }
1519   { \l_block_env_name_tl \space environment}
1520   \UnusedTemplateKeys
1521 }
```

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.

```
1522 % \legacy_if:nT { @inlabel } { \indent \par }
```

Determine if we do numbering:

```

1523     \bool_lazy_or:nnT
1524     { \tl_if_empty_p:N \l__block_counter_tl }
1525     { #2 }
1526     { \bool_set_false:N \l__block_numbered_bool }

```

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

```

1527     \tl_set:Nn \l__block_note_tl {#3}

```

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

```

1528     \bool_if:NTF \l__block_numbered_bool
1529     {
1530         \@kernel@refstepcounter{ \l__block_counter_tl }
1531         \MakeLinkTarget{ \l__block_counter_tl }
1532     }
1533     {
1534         \MakeLinkTarget[theorem-like]{}
1535     }

```

Add the caption into \g__block_labels_box:

```

1536     \hbox_gset:Nn \g__block_labels_box
1537     {
1538         \box_use_drop:N \g__block_labels_box % <- does nothing if
1539                                         % there is no dangling label

```

Now apply the declarations that are for the whole caption.

```

1540     \l__block_caption_decls_tl

```

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

```

1541     \tag_socket_use:nnn {captionedtext/caption} {}
1542     {
1543         \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

```

1544         \clist_map_inline:Nn \l__block_order_clist
1545         { \group_begin:
1546             \use:c { __block_do_##1: }
1547             \group_end:
1548         }
1549         \skip_horizontal:n { \l__block_caption_after_skip }
1550     }
1551 }

```

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

```

1552     \bool_if:NTF \g__block_label_standalone_bool
1553     {
1554         \bool_gset_true:N \g__block_label_standalone_bool
1555         \para_omit_indent:
1556         \box_use_drop:N \g__block_labels_box
1557         \par
1558     }

```

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

check if @newlist could be replaced by something more general

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.

```

1559     {
1560         \legacy_if_gset_true:n { @newlist }
1561         \legacy_if_gset_true:n { @inlabel }
1562     }

```

Do not break after the first line:

```

1563     \legacy_if_gset_true:n { @nobreak }

```

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

```

1564     \__block_debug_typeout:n{Set~ captioned~ block~ everypar \on@line }
1565     \cs_set_eq:NN \__block_everypar: \__block_captioned_everypar_std:

```

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

```

1566     \l__block_body_decls_tl
1567     \__block_debug_typeout:n{template:thmstyle:std~end}
1568 }

```

t/captionedtext/caption (socket)

```

1569 \NewTaggingSocket{captionedtext/caption}{2}

```

why kernel?

```

1570 \NewTaggingSocketPlug{captionedtext/caption}{kernel}
1571 {
1572     \tag_struct_begin:n{tag=\UseStructureName{block/theorem-like/caption}}
1573     #2
1574     \tag_struct_end:
1575 }
1576 \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:

```

1577 \cs_new_protected:Npn \__block_do_title: {

```

Check if there is a title.

```

1578     \tl_if_empty:NTF \l__block_title_tl

```

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

```

1579     { \__block_drop_spaces: }

```

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

```

1580     { \tag_socket_use:nnn {mc} {}{
1581         \__block_insert_spaces:

```

The title may have its own formatting:

```

1582         \__block_title_format:n \l__block_title_tl }
1583     }
1584 }

```

(End of definition for __block_do_title:.)

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

```

1585 \cs_new_protected:Npn \__block_do_note: {
1586   \tl_if_novalue:oTF \l__block_note_tl
1587   { \__block_drop_spaces: }
1588   { \tag_socket_use:nnn {mc} {} {
1589     \__block_insert_spaces:
1590     \__block_note_format:n \l__block_note_tl }
1591   }
1592 }

```

(End of definition for __block_do_note:.)

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

```

1593 \cs_new_protected:Npn \__block_do_number: {
1594   \bool_if:NTF \l__block_numbered_bool
1595   { \tag_socket_use:nnn {struct-mc} {tag=\UseStructureName{block/theorem-like/label}}
1596     { \__block_insert_spaces:
1597       \__block_number_format:n {
1598         \use:c{ the \l__block_counter_tl } }
1599     }
1600   }
1601   { \__block_drop_spaces: }
1602 }

```

(End of definition for __block_do_number:.)

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

```

1603 \cs_new_protected:Npn \__block_do_punct: {
1604   \__block_drop_spaces:
1605   \tl_if_empty:NF \l__block_punct_tl
1606   { \tag_socket_use:nnn {mc} {} {
1607     \__block_punct_format:n \l__block_punct_tl }
1608   }
1609 }

```

(End of definition for __block_do_punct:.)

`__block_do_space:` What's still missing is what `space` should do. It simply adds a `\l__block_space_tl` to the `\g__block_collected_spaces_tl` tokenlist. This way the clist can contain `space,space,...` to indicate multiple spaces. The storage tokenlist is global as the functions are executed inside their own group, but the collected space is used outside of that group.

```

1610 \cs_new_protected:Npn \__block_do_space: {
1611   \tl_gput_right:Nn \g__block_collected_spaces_tl \l__block_space_tl
1612 }

```

So `__block_insert_spaces:` is trivial, all we have to do is to insert the collected space and then clear the tokenlist

```

1613 \cs_new_protected:Npn \__block_insert_spaces: {
1614   \g__block_collected_spaces_tl
1615   \tl_gclear:N \g__block_collected_spaces_tl
1616 }

```

Even simpler is `__block_drop_spaces`:

```

1617 \cs_new_protected:Npn \__block_drop_spaces: {
1618   \tl_gclear:N \g__block_collected_spaces_tl
1619 }

```

What remains is to declare the tokenlist.

```

1620 \tl_new:N \g__block_collected_spaces_tl
(End of definition for \__block_do_space: and others.)

```

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

```

1621 \DeclareTemplateCode{captionedtext}{proof}{4}
1622 {
1623   ,title           = \l__block_title_tl
1624   ,punct           = \l__block_punct_tl
1625   ,caption-placement = {
1626     chained        = \bool_gset_false:N \g__block_label_standalone_bool
1627                     \bool_gset_false:N \g__block_label_unchained_bool
1628     ,unchained      = \bool_gset_false:N \g__block_label_standalone_bool
1629                     \bool_gset_true:N \g__block_label_unchained_bool
1630     ,standalone     = \bool_gset_true:N \g__block_label_standalone_bool
1631                     \bool_gset_false:N \g__block_label_unchained_bool
1632   }
1633   ,before-hspace   = \l__block_caption_before_skip
1634   ,after-hspace     = \l__block_caption_after_skip
1635   ,caption-decls    = \l__block_caption_decls_tl
1636   ,title-format     = \__block_title_format:n
1637   ,punct-format     = \__block_punct_format:n
1638   ,body-decls       = \l__block_body_decls_tl
1639 }
1640 {

```

Display the template's arguments when tracing:

```

1641 \template_debug_typeout:n{~\space template:~ 'proof';~
1642                      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:

```

1643 \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):

```

1644 \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`:

```

1645 \hbox_gset:Nn \g__block_labels_box
1646 {
1647   \box_use_drop:N \g__block_labels_box % <- does nothing if there
1648                                     % is no dangling label
1649   \l__block_caption_decls_tl
1650   \tag_socket_use:nnn {captionedtext/caption} {}
1651   {
1652     \skip_horizontal:n { \l__block_caption_before_skip }

```

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

```

1653         \group_begin: \__block_do_title: \group_end:
1654         \group_begin: \__block_do_punct: \group_end:
1655         \skip_horizontal:n { \l__block_caption_after_skip }
1656     }
1657 }

```

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

```

1658 \bool_if:NTF \g__block_label_standalone_bool
1659 {
1660     \bool_gset_true:N \g__block_label_standalone_bool
1661     \para_omit_indent:
1662     \box_use_drop:N \g__block_labels_box
1663     \par
1664 }
1665 {
1666     \legacy_if_gset_true:n { @newlist }
1667     \legacy_if_gset_true:n { @inlabel }
1668 }
1669 \legacy_if_gset_true:n { @nobreak }
1670 \__block_debug_typeout:n{Set~ captioned~ block~ everypar \on@line }
1671 \cs_set_eq:NN \__block_everypar: \__block_captioned_everypar_std:
1672 \l__block_body_decls_tl
1673 \__block_debug_typeout:n{template:captionedtext:proof~end}
1674 }

```

9.5 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 `<text-unit>`, 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!

```

1675 \cs_set:Npn \__block_beginpar_vmode: {
1676     \__block_debug_typeout:n
1677     { @endpe = \legacy_if:NTF { @endpe } {true}{false} \on@line }
1678     \legacy_if:NTF { @endpe }
1679     {
1680         \legacy_if_gset_false:n { @endpe }
1681     }

```

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

```

1682     {
1683         \int_compare:nNnT \l__tag_block_flattened_level_int < 2
1684         {

```

```

1685         \UseTaggingSocket{para/semantic/begin}
1686         { \_tag\_para\_main\_store\_struct: }
1687     }
1688 }
1689 }

```

(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.

```

1690 \cs\_set:Npn \_block\_beginpar\_hmode:N #1
1691 {
1692     \tag\_mc\_end:
1693     \_tag\_gincr\_para\_end\_int:
1694     \_block\_debug\_typeout:n{increment~ /P \on@line }
1695     \bool\_if:NT \l\_tag\_para\_show\_bool
1696     { \tag\_mc\_begin:n{artifact}
1697       \rlap{\color\_select:n{red}}\tiny\ \int\_use:N\g\_tag\_para\_end\_int}
1698       \tag\_mc\_end:
1699     }
1700     \tag\_struct\_end:
1701     \tagpdfparaOff \par \tagpdfparaOn
1702 }

```

(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.

```

1703 \socket\_if\_exist:nF {taggsupport/block/startpara/direct}
1704 {
1705     \NewTaggingSocket {block/startpara/direct}{1}
1706 }

```

default (plug)

```

1707 \NewTaggingSocketPlug{block/startpara/direct}{default}
1708 {
1709     \bool\_if:NF \l\_tag\_para\_flattened\_bool
1710     {

```

```

1711         \UseTaggingSocket{para/semantic/begin}
1712         { \__tag_para_main_store_struct: }
1713     }
1714     \__tag_gincr_para_begin_int:
1715     \__block_debug_typeout:n{increment~ P \on@line }
1716     \tag_struct_begin:n
1717     {
1718         tag=\l__tag_para_tag_tl
1719         ,attribute-class=\l__tag_para_attr_class_tl
1720     }
1721     \__tag_check_para_begin_show:nn {green}{#1}
1722     \tag_mc_begin:n {}
1723 }
1724 \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.

```

1725 \AddToHook{para/end}[tagpdf]{}
1726 \RemoveFromHook{para/end}[tagpdf]
1727 \AddToHook{para/end}{}
1728 \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).

```

1729 \socket_if_exist:nF {tagsupport/kernel/endpe/vmode}
1730 {
1731     \NewTaggingSocket {kernel/endpe/vmode}{0}
1732 }

```

default (plug)

```

1733 \NewTaggingSocketPlug{kernel/endpe/vmode}{default}
1734 {
1735     \if@endpe \ifvmode
1736         \bool_if:NT \l__tag_para_bool
1737         {
1738             \bool_if:NF \l__tag_para_flattened_bool
1739             {
1740                 \UseTaggingSocket{para/semantic/end}{}
1741             }

```

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

```

1742         \@endpefalse
1743     }
1744     \fi \fi
1745 }
1746 \AssignTaggingSocketPlug{kernel/endpe/vmode}{default}

```

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

```

1747 \cs_set_protected:Npn \para_end: {

```

```

1748 \scan_stop:
1749 \mode_if_horizontal:TF {
1750   \mode_if_inner:F {
1751     \tex_unskip:D
1752     \hook_use:n{para/end}
1753     \@kernel@after@para@end
1754     \mode_if_horizontal:TF {
1755       \if_int_compare:w 11 = \tex_lastnodetype:D
1756         \tex_hskip:D \c_zero_dim
1757       \fi:
1758       \tex_par:D
1759       \hook_use:n{para/after}
1760       \@kernel@after@para@after
1761     }
1762     { \msg_error:nnnn { hooks }{ para-mode }{end}{horizontal} }
1763   }
1764 }
1765 {

```

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

```

1766   \UseTaggingSocket{kernel/endpe/vmode}%
1767   \tex_par:D
1768 }
1769 }

```

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 `vmode` tagging hook is needed.]

```

1770 \cs_set_eq:NN \par \para_end:
1771 \cs_set_eq:NN \@@par \para_end:
1772 \cs_set_eq:NN \endgraf \para_end:

```

(End of definition for `\para_end`:. This function is documented on page 38.)

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

```

1773 \protected\def\begin#1{%
1774   \UseHook{env/#1/before}%
1775   \@ifundefined{#1}%
1776     {\def\reserved@a{\@latex@error{Environment~#1~undefined}\@eha}}%
1777     {\def\reserved@a{\def\@currentvir{#1}%
1778       \edef\@currentvline{\on@line}%
1779       \@execute@begin@hook{#1}%
1780       \csname #1\endcsname}}%
1781   \@ignorefalse
1782   \begingroup
1783     \UseTaggingSocket{kernel/endpe/vmode}%
1784     \reserved@a}

```

(End of definition for `\begin`. This function is documented on page 38.)

`_kernel_list_label_after:n` If starting the text-unit/text 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.

```

1785 \cs_new_protected:Npn \_kernel_list_label_after:n #1 {
1786   \bool_lazy_and:nnT { \tag_if_active_p: } {\l__tag_para_bool }

```

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

```

1787     {
1788       \tag_socket_use:nm {block/startpara/direct} { #1 }
1789     }
1790 }

```

(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.

```

1794 \cs_new_protected:Npn \__block_inner_begin: {
1795   \tagstructbegin{tag=\l__block_tag_inner_tag_tl}
1796 }

```

(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.

```

1794 \cs_new_protected:Npn \__block_inner_end: {
1795   \__block_debug_typeout:n{block-end \on@line}
1796   \legacy_if:nT { @endpe }
1797   {
1798     \UseTaggingSocket{para/semantic/end}
1799     { \__block_debug_typeout:n{close~ /text-unit \on@line}}
1800   }
1801   \tagstructend          % end inner structure
1802 }

```

(End of definition for __block_inner_end:.)

9.5.1 List tags

```

1803 \tl_new:N \l__tag_L_tag_tl
1804 \tl_set:Nn \l__tag_L_tag_tl {L}
1805
1806 \tl_new:N\l__tag_L_attr_class_tl
1807 \tl_set:Nn \l__tag_L_attr_class_tl {list}
1808
1809 \tagpdfsetup
1810 {
1811   ,role/new-attribute = {itemize}
1812                       {/0 /List /ListNumbering/Unordered}
1813   ,role/new-attribute = {enumerate}
1814                       {/0 /List /ListNumbering/Ordered}
1815   ,role/new-attribute = {description}
1816                       {/0 /List /ListNumbering/Description}
1817 }

```

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

```

1816   ,role/new-attribute = {list}{/0 /List /ListNumbering/Unordered}
1817 }

```

__block_list_begin: Start a list ...This command is tagging specific, it is mapped to **__kernel_displayblock_-begin:** in a tagging recipe.

```

1818 \cs_set:Npn \__block_list_begin: {
1819   \tagstructbegin
1820   {
1821     tag=\l__tag_L_tag_tl

```

```

1822         ,attribute-class=\l__tag_L_attr_class_tl
1823     }
1824 }

```

(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.

```

1825 \cs_set:Npn \__block_list_item_begin: {
1826     \tagstructbegin{tag=\UseStructureName{block/list/item}}
1827 }

```

(End of definition for `__block_list_item_begin:`)

`__block_list_item_end:` When a list item ends we have to close `<itembody>` and `` but also a `<text>` 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.

```

1828 \cs_set:Npn \__block_list_item_end: {
1829     \legacy_if:nT { @endpe }
1830     {
1831         \UseTaggingSocket{para/semantic/end}{
1832             % \__block_debug_typeout:n{Structure-end~ P~ at~ item-end \on@line }
1833         }
1834         \tagstructend \tagstructend % end \UseStructureName{block/list/body}, LI
1835     }

```

(End of definition for `__block_list_item_end:`)

`__block_list_end:` Finally, at the list end we have to close the open `<itembody>`, ``, `<L>`, and possibly a `<text>` if the last item ends with a list. However, if the user forgot to add an `\item` then there will be no `` and `<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.

```

1836 \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.

```

1837     \legacy_if:nF { @newlist }
1838     {
1839         \legacy_if:nT { @endpe }
1840         {
1841             \UseTaggingSocket{para/semantic/end}{
1842                 {\__block_debug_typeout:n{Structure-end~ text-unit~ at~ list-end \on@line }}
1843             }
1844             \tagstructend\tagstructend % end \UseStructureName{block/list/body}, LI
1845         }
1846         \tagstructend % end L
1847     }

```


(End of definition for `__block_list_end:`.)

End of tagging related declarations.

9.5.2 Tagging recipes

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

```

1848 \socket_if_exist:nF {tagsupport/block/recipe}
1849 {
1850   \NewTaggingSocket{block/recipe}{1}
1851 }
```

`default (plug)`

```

1852 \NewTaggingSocketPlug{block/recipe}{default}
1853 {
1854   \use:c { __block_recipe_#1: }
1855 }
1856 \AssignTaggingSocketPlug{block/recipe}{default}
```

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

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

```

1857 \cs_new_protected:Npn __block_recipe_basic: {
1858   \cs_set_eq:NN __kernel_displayblock_beginpar_hmode:w
1859                                     __block_beginpar_hmode:N
1860   \cs_set_eq:NN __kernel_displayblock_beginpar_vmode:
1861                                     __block_beginpar_vmode:
1862   \let __kernel_displayblock_begin: \prg_do_nothing:
1863   \let __kernel_displayblock_end:   \prg_do_nothing:
```

End environment `\par` handling:

```

1864   \socket_assign_plug:nn{block/endpe}{on}
1865 }
```

(End of definition for `__block_recipe_basic:`.)

`__block_recipe_standalone:`

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

```

1866 \cs_new_protected:Npn __block_recipe_standalone: {
1867   \cs_set_eq:NN __kernel_displayblock_beginpar_hmode:w
1868                                     \prg_do_nothing:
1869   \cs_set_eq:NN __kernel_displayblock_beginpar_vmode:
1870                                     \prg_do_nothing:
1871   \cs_set_eq:NN __kernel_displayblock_begin: __block_inner_begin:
1872   \cs_set_eq:NN __kernel_displayblock_end:   __block_inner_end:
```

End environment `\par` handling:

```

1873   \socket_assign_plug:nn{block/endpe}{off}
```

```

1874 \tl_if_empty:NTF \l__block_tag_name_tl
1875 { \tl_set:Nn \l__block_tag_inner_tag_tl {Sect} }
1876 { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
1877 }

```

(End of definition for `__block_recipe_standalone:`)

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

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

```

1878 \cs_new_protected:Npn \__block_recipe_standard:
1879 {
1880   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
1881                                     \__block_beginpar_hmode:N
1882   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
1883                                     \__block_beginpar_vmode:
1884   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_inner_begin:
1885   \cs_set_eq:NN \__kernel_displayblock_end:   \__block_inner_end:

```

End environment `\par` handling:

```

1886   \socket_assign_plug:nn{block/endpe}{on}

1887   \tl_if_empty:NTF \l__block_tag_name_tl
1888   { \tl_set:Nn \l__block_tag_inner_tag_tl {Div} }
1889   { \tl_set_eq:NN \l__block_tag_inner_tag_tl \l__block_tag_name_tl }
1890 }

```

(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.

```

1891 \tl_new:N \l__block_tag_inner_tag_tl

```

(End of definition for `\l__block_tag_inner_tag_tl`.)

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

- It opens a `<text-unit>`-structure or keeps the current one open (only closing the MC).
- It then starts a new structure 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 `<text-unit>`-structure open to continue the paragraph after the list, if necessary.

```

1892 \cs_new_protected:Npn \__block_recipe_list:
1893 {
1894   \cs_set_eq:NN \__kernel_displayblock_beginpar_hmode:w
1895                                     \__block_beginpar_hmode:N
1896   \cs_set_eq:NN \__kernel_displayblock_beginpar_vmode:
1897                                     \__block_beginpar_vmode:
1898   \cs_set_eq:NN \__kernel_displayblock_begin: \__block_list_begin:
1899   \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)).

```

1900   \cs_set_eq:NN \__kernel_list_item_begin:   \__block_list_item_begin:
1901   \cs_set_eq:NN \__kernel_list_item_end:     \__block_list_item_end:

```

End environment `\par` handling:

```

1902   \socket_assign_plug:nn{block/endpe}{on}

```

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

```

1903   \tl_if_empty:NTF \l__block_tag_name_tl
1904     { \tl_set:Nn \l__tag_L_tag_tl {L} }
1905     { \tl_set_eq:NN \l__tag_L_tag_tl \l__block_tag_name_tl }
1906   \tl_if_empty:NTF \l__block_tag_class_tl
1907     { \tl_set:Nn \l__tag_L_attr_class_tl {} }
1908     { \tl_set_eq:NN \l__tag_L_attr_class_tl \l__block_tag_class_tl }
1909 }

```

(End of definition for `__block_recipe_list:.`)

```

1910 </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.

```

1911 <*package-finish>
1912 <@@=
1913 \def\legacyverbatimsetup #1 {%
1914   \language\l@nohyphenation
1915   \@tempwafalse

```

```

1916 \def\par{%
1917   \if@tempwa
1918     \leavevmode \null {\@@par}\penalty\interlinepenalty
1919   \else
1920     \@tempwattrue
1921     \ifhmode{\@@par}\penalty\interlinepenalty\fi
1922   \fi

```

might also need a hook not just a socket

Do something at the very beginning of each verbatim line:

```

1923   \UseSocket{verbatim/startline}%
1924 }%
1925 \let\do\@makeother \dospecials
1926 \obeylines \verbatim@font \@noligs
1927 \everypar \expandafter{\the\everypar \unpenalty}%
1928 \frenchspacing
1929 \AssignStructureRole {para/textblock}%
1930   {\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.

```

1931 \use:c { @setupverb #1 space }
1932 \@vobeyspaces
1933 }

```

(End of definition for `\legacyverbatimsetup`. This function is documented on page 37.)

`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.

```

1934 \NewSocket{verbatim/startline}{0}

```

`\@setupverbinvisiblespace`

In the pdfTeX engine we need to use `\pdfmakespace` 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.

```

1935 <@@=block>
1936 \newcommand\@setupverbinvisiblespace{}
1937 \bool_lazy_or:nnF
1938 { \sys_if_engine_luatex_p: }
1939 { \sys_if_output_dvi_p: }
1940 {
1941   \renewcommand\@setupverbinvisiblespace
1942     {\def\xobeysp{\nobreakspace\pdfmakespace}}
1943 }

```

(End of definition for `\@setupverbinvisiblespace`. This function is documented on page 37.)

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.

```

1944 <@@=
1945 \ExplSyntaxOff

```

```

1946 \def\legacyallttsetup #1{%
1947   \let\org@prime~%
1948   \everymath\expandafter{\the\everymath
1949     \catcode`\'=12 \let~\org@prime}%
1950   \everydisplay\expandafter{\the\everydisplay
1951     \catcode`\'=12 \let~\org@prime}%

```

This alters `\dospecials`:

```

1952   \let\org@dosppecials\dosppecials
1953   \g@remfrom@specials{\}%
1954   \g@remfrom@specials{\}%
1955   \g@remfrom@specials{\}%

```

Then call `\legacyverbatimsetup`:

```

1956   \legacyverbatimsetup {#1}%

```

And afterwards restore `\dospecials`:

```

1957   \let\dosppecials\org@dosppecials
1958 }

```

Copied from `alltt`.

```

\g@remfrom@specials 1959 \def\g@remfrom@specials#1{%
1960   \def\@new@specials{}%
1961   \def\@remove##1{%
1962     \ifx##1#1\else
1963       \g@addto@macro\@new@specials{\do ##1}\fi}%
1964   \let\do\@remove\dosppecials
1965   \let\dosppecials\@new@specials
1966 }

1967 \ExplSyntaxOn
1968 <@@=block>

```

(End of definition for `\legacyallttsetup` and `\g@remfrom@specials`. These functions are documented on page 37.)

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 `legacy-code`:

```

1969 \cs_new_protected:Npn \legacylistsetup {

```

Reset values to defaults:

```

1970   \dim_zero:N \listparindent
1971   \dim_zero:N \rightmargin
1972   \dim_zero:N \itemindent

```

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

```

1973 %   \tl_set:Nn \@listctr {}
1974 %   \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.

```

1975   \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):

```
1976 \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.

```
1977 \legacy_if:NTF { @nbrlist }
1978 { \tl_set:Nn \l__tag_L_attr_class_tl {enumerate} } % numbered list
1979 { \tl_if_empty:NTF \@itemlabel
1980 { \tl_set:Nn \l__tag_L_attr_class_tl {list} } % no label
1981 { \tl_set:Nn \l__tag_L_attr_class_tl {itemize} } % unordered,
1982 % unordered
1983 }
1984 }
```

(End of definition for `\legacylistsetup`. This function is documented on page 37.)

```
\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`.

```
1985 \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 \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.

```
1986 \RenewDocumentCommand \newtheorem { s m o m o } {
```

Is the environment definable at all? If not there is not much point in continuing.

```
1987 \expandafter\@ifdefinable\csname #2\endcsname
1988 {
```

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`.

```
1989 \IfBooleanF #1
1990 {
1991 \IfNoValueTF {#3}
```

If there was no counter to use (`#2`) then we set up a counter with the same name as the environment (`#2`).

```
1992 {
1993 \@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:

```

1994         \IfNoValueTF {#5}
1995         { % @ynthm
1996           \tl_gset:ce { the #2 }
1997           {
1998             \@thmcounter{#2}
1999           }
2000         }
2001         { % @xnthm
2002           \@newctr{#2}[#5]
2003           \tl_gset:ce { the #2 }
2004           {
2005             \expandafter\noexpand\csname the#5\endcsname
2006             \@thmcountersep
2007             \@thmcounter{#2}
2008           }
2009         }
2010       }
2011     { % @othm

```

If we should reuse an existing counter (#3 was given) we check that this counter actually exists and if so use it:

```

2012         \ifundefined{c@#3}
2013         { \nocounterr{#3} }
2014         {
2015           \newcounteralias{#2}{#3}
2016         }
2017       }
2018     }

```

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`.

```

2019     \NewDocumentEnvironment{#2}{={note}o }
2020     { \ParseLaTeXeTheoremlike {#2} \BooleanFalse {##1} }
2021     { \BlockEnvEnd }

```

The starred form of the environment suppresses the number so we pass it `\BooleanTrue`, otherwise it is identical to the previous definition.

```

2022     \NewDocumentEnvironment{#2*}{={note}o }
2023     { \ParseLaTeXeTheoremlike {#2} \BooleanTrue {##1} }
2024     { \BlockEnvEnd }

```

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).

```

2025     \IfInstanceExistsF{thmstyle}{\l_block_thmstyle_tl}
2026     { \@latex@warning{Unknown~ theoremstyle~

```

```

2027         '\l__block_thmstyle_tl'~ using~ 'plain'}
2028         \theoremstyle {plain}
2029     }

```

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).

```

2030     \IfInstanceExistsTF{block} { thm- \l__block_thmstyle_tl -1 }
2031     { \__block_debug_typeout:n{...~ style~ \l__block_thmstyle_tl\space exists } }
2032     { \__block_debug_typeout:n{...~ style~ \l__block_thmstyle_tl\space
2033       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.

```

2034     \DeclareInstance{blockenv}{#2}{std}
2035     {
2036         ,name                = theorem-like
2037         ,tag-name             = \UseStructureName{block/theorem-like}
2038         ,tag-attr-class       =
2039         ,tagging-recipe       = standalone
2040         ,inner-level-counter   =
2041         ,transparent-level     = true
2042         ,legacy-code          =

```

What block instance to use is determined by checking if a special one exists or whether we should use `plain`:

```

2043         ,block-instance:e     = thm-
2044                               \IfInstanceExistsTF{block}
2045                               {thm- \l__block_thmstyle_tl -1}
2046                               { \l__block_thmstyle_tl } { plain }
2047         ,inner-instance-type   = captionedtext
2048         ,inner-instance        = #2

```

By default the body text is justified, but perhaps we should not set anything here and use whatever is current.

```

2049         ,para-instance        = justify
2050     }

```

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.

```

2051     \DeclareInstance{captionedtext}{#2}{thmlike}
2052     {

```

The counter to use is either none or #2 based on the arguments given:

```

2053         ,counter:e = \IfBooleanF #1 { #2 }
2054         ,title      = #4
2055         ,style:e     = \l__block_thmstyle_tl
2056                     \bool_if:NT \l__block_swap_number_bool {-swap}
2057     }

```

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.

```

2058     \bool_if:NT \l__block_swap_number_bool {

```



```

2059     \IfInstanceExistsF{thmstyle}{\l__block_thmstyle_tl -swap}
2060     {

```

If it doesn't exist we first make a copy of the base instance.

```

2061     \DeclareInstanceCopy{thmstyle}
2062         {\l__block_thmstyle_tl -swap}
2063         {\l__block_thmstyle_tl}

```

Then we retrieve the value of the `order` key from that instance which is a clist.

```

2064     \clist_set:N \l__block_order_clist
2065     { \InstanceValue { thmstyle }
2066       { \l__block_thmstyle_tl }
2067       { order }
2068     }

```

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.

```

2069     \clist_clear:N \l__block_tmp_clist
2070     \clist_map_inline:Nn \l__block_order_clist
2071     {
2072         \clist_put_right:N \l__block_tmp_clist {
2073             \str_case:nnF {##1}
2074             { {title} {number}
2075               {number} {title} }
2076             {##1}
2077         }
2078     }

```

Once that is done we put the new value for `order` in the new instance.

```

2079     \EditInstance {thmstyle}{\l__block_thmstyle_tl -swap}
2080     { order:e = \l__block_tmp_clist }
2081   }
2082 }
2083 }
2084 }

```

(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:

```

2085 \keys_define:nn {blockenv} {
2086   , note      .tl_set:N = \l__block_note_tl
2087   , note      .groups:n = { interface }
2088 }

```

```

2089 \cs_new_protected:Npn \ParseLaTeXeTheoremlike #1 #2 #3 {
2090 %
2091 % \__block_debug_typeout:n{Parse~#1.~ Arguments~ found:~ \IfBooleanT{#2}{*}
2092 % \IfValueT{#3}{[\exp_not:n{#3}]}}
2093 %

```

Normally, no note is provided, so that's the starting point:

```

2094 \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.

```

2095 \IfNoValueTF { #3 }
2096 { \tl_clear:N \l__block_instance_keys_tl }
2097 { \keys_set_groups:nnnN {blockenv} {interface} { #3 }
2098 \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`.

```

2099 \use:e {
2100 \exp_not:N \BlockEnv { #1 }
2101 { \exp_not:o \l__block_instance_keys_tl }
2102 { #2 }
2103 { \exp_not:o \l__block_note_tl }
2104 }

```

We don't have any use for the last argument (the sub-caption) in the standard setup, so we pass `\NoValue`.

```

2105 \NoValue
2106 }

```

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

\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`.

```

2107 \bool_new:N \l__block_swap_number_bool
2108 \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.

`\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.

```

2109 \cs_set_protected:Npn \newtheoremstyle #1#2#3#4#5#6#7#8#9 {

```

extend

First we build a `thmstyle` instance:

```

2110 \DeclareInstance{thmstyle}{#1}{std}{
2111   ,caption-decls = {#4}
2112   ,before-hspace:e = \tl_if_empty:nTF{#5}{0pt}{#5}
2113   ,body-decls = {#6}
2114   ,punct = {#7}

```

This setting doesn't cover all syntax possibilities.

```

2115   ,after-hspace:e = \tl_if_empty:nTF{#8}{0pt}
2116                     {\tl_if_blank:nTF{#8}{3.3pt}{#8}}
2117 }

```

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:

```

2118 \tl_if_empty:nF { #2#3 }
2119 {
2120   \DeclareInstance{block}{thm-#1-1}{std}{
2121     ,begin-vspace:e = \tl_if_empty:nTF{#2}{0pt}{#2}
2122     ,end-vspace:e = \tl_if_empty:nTF{#3}{0pt}{#3}
2123     ,left-margin = 0pt
2124     ,para-indent = \parindent
2125     ,para-vspace = \parskip
2126   }

```

As elsewhere we provide two levels.

```

2127   \DeclareInstanceCopy{block}{thm-#1-2}{thm-#1-1}
2128 }

```

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.

```

2129 \tl_if_empty:oF { \exp_not:n{#9} }
2130 {

```

Give special definitions for the commands and then expand #9 and use the result to edit the instance we defined earlier.

```

2131   \cs_set:Npn \thmname ##1 {title-format={\exp_not:n{##1}}},}
2132   \cs_set:Npn \thmnumber ##1 {number-format={\exp_not:n{##1}}},}
2133   \cs_set:Npn \thmnote ##1 {note-format={\exp_not:n{##1}}},}
2134   \cs_set:Npn \__block_tmp:w ##1##2##3 {
2135     \exp_args:Nnne \EditInstance{thmstyle}{#1}{#9}}
2136   \__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.

```

2137   \cs_set:Npn \thmname ##1 {title,}
2138   \cs_set:Npn \thmnumber ##1 {number,}

```

```

2139     \cs_set:Npn \thmnote    ##1 {punct,note}
2140     \cs_set:Npn \__block_tmp:w##1##2##3 {
2141       \exp_args:Nne \EditInstance{thmstyle}{#1}{order={#9}}}
2142     \__block_tmp:w {##1} {##1} {##1}
2143   }

```

If block tracing is turned on we show the final result:

```

2144   \__block_debug:n { \ShowInstanceValues{thmstyle}{#1} }
2145 }

```

(End of definition for `\newtheoremstyle`. This function is documented on page 38.)

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).

```

2146 \ExplSyntaxOff
2147 \def\math@qedhere{%
2148   \ifundefined{\@currentenv @qed}{%
2149     \qed@warning\quad\hbox{\qedsymbol}%
2150   }{%
2151     \xp\aftergroup\csname\@currentenv @qed\endcsname
2152   }%
2153 }
2154 \def\displaymath@qed{%
2155   \relax
2156   \ifmmode
2157     \ifinner \aftergroup\linebox@qed
2158   \else
2159     \eqno
2160     \let\eqno\relax \let\leqno\relax \let\veqno\relax
2161     \hbox{\qedsymbol}%
2162   \fi
2163 \else
2164   \aftergroup\linebox@qed
2165 \fi
2166 }
2167 \expandafter\let\csname equation*@qed\endcsname\displaymath@qed
2168 \def\equation@qed{%
2169   \iftagsleft@
2170     \hbox{\phantom{\quad\qedsymbol}}%
2171     \gdef\alt@tag{%
2172       \rlap{\hbox to\displaywidth{\hfil\qedsymbol}}%
2173       \global\let\alt@tag\@empty
2174     }%
2175   \else
2176     \gdef\alt@tag{%
2177       \global\let\alt@tag\@empty
2178       \vtop{\ialign{\hfil###\cr
2179         \tagform@theequation\cr
2180         \qedsymbol\cr}}%
2181     \setbox\z@
2182   }%

```

```

2183     \fi
2184 }
2185 \def\qed@tag{%
2186   \global\tag@true \nonumber
2187   &\omit\setboxz@h {\strut@ \qedsymbol}\tagsleft@false
2188   \place@tag@gather
2189   \kern-\tabskip
2190   \ifst@rred \else \global\@eqnswtrue \fi \global\advance\row@\@ne \cr
2191 }
2192 \def\split@qed{%
2193   \def\endsplit{\crrc\egroup \egroup \ctagsplit@false \rendsplit@
2194   \aftergroup\align@qed
2195   }%
2196 }
2197 \def\align@qed{%
2198   \ifmeasuring@ \tag*{\qedsymbol}%
2199   \else \let\math@cr__block@\qed@tag
2200   \fi
2201 }
2202 \expandafter\let\csname align*@qed\endcsname\align@qed
2203 \expandafter\let\csname gather*@qed\endcsname\align@qed
2204 %
2205 \def\math@qedhere{\quad\hbox{\qedsymbol}}%
2206 %
2207 \DeclareRobustCommand{\qed}{%
2208   \ifmmode \mathqed
2209   \else
2210     \leavevmode\unskip\penalty9999 \hbox{}\nobreak\hfill
2211     \quad\hbox{\qedsymbol}%
2212   \fi
2213 }%
2214 \let\QED@stack\@empty
2215 \let\qed@elt\relax
2216 \newcommand{\pushQED}[1]{%
2217   \toks@{\qed@elt{#1}}\@temptokena\expandafter{\QED@stack}%
2218   \xdef\QED@stack{\the\toks@\the\@temptokena}%
2219 }%
2220 \newcommand{\popQED}{%
2221   \begingroup\let\qed@elt\popQED@elt \QED@stack\relax\relax\endgroup
2222 }%
2223 \def\popQED@elt#1#2\relax{#1\gdef\QED@stack{#2}}%
2224 \newcommand{\qedhere}{%
2225   \begingroup \let\mathqed\math@qedhere
2226   \let\qed@elt\setQED@elt \QED@stack\relax\relax \endgroup
2227 }%
2228 \def\setQED@elt#1#2\relax{%
2229   \ifmeasuring@
2230   \else \iffirstchoice@ \gdef\QED@stack{\qed@elt{#2}}\fi
2231   \fi
2232   #1%
2233 }%
2234 \def\qed@warning{%
2235   \PackageWarning{amsthm}{The \@nx\qedhere command may not work
2236     correctly here}%

```

```

2237 }%
2238 \newcommand{\mathqed}{\quad\hbox{\qedsymbol}}%
2239 \DeclareRobustCommand{\qed}{%
2240   \ifmmode \mathqed
2241   \else
2242     \leavevmode\unskip\penalty9999 \hbox{}\nobreak\hfill
2243     \quad\hbox{\qedsymbol}%
2244   \fi
2245 }
2246 \newcommand{\openbox}{\leavevmode
2247   \hbox to.77778em{%
2248     \hfil\vrule
2249     \vbox to.675em{\hrule width.6em\vfil\hrule}%
2250     \vrule\hfil}}
2251 \providecommand{\qedsymbol}{\openbox}
2252 \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*`.

```

2253 \declare@file@substitution{alltt.sty}{null.tex}

```

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.

```

2254 \declare@file@substitution{amsthm.sty}{null.tex}

```

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:

```

2255 \let \amsnewtheorem \newtheorem
2256 \let \amsnewtheoremstyle \newtheoremstyle
2257 \let \amstheoremstyle \theoremstyle
2258 \let \amsproof \proof
2259 \let \amsendproof \endproof

```

Then undefine them just before the class gets loaded (quite a handful):

```

2260 \AddToHook{class/amsart/before}[block]{
2261   \let \newtheoremstyle \relax
2262   \let \theoremstyle \relax

```

```

2263 \let \proof \relax
2264 \let \endproof \relax

2265 \let \pushQED \relax
2266 \let \popQED \relax
2267 \let \qedhere \relax
2268 \let \mathqed \relax
2269 \let \openbox \relax
2270 }

```

Same for amsbook and amsproc:

```

2271 \AddToHook{class/amsbook/before}[block]{
2272 \let \newtheoremstyle \relax
2273 \let \theoremstyle \relax
2274 \let \proof \relax
2275 \let \endproof \relax
2276 \let \pushQED \relax
2277 \let \popQED \relax
2278 \let \qedhere \relax
2279 \let \mathqed \relax
2280 \let \openbox \relax
2281 }

2282 \AddToHook{class/amsproc/before}[block]{
2283 \let \newtheoremstyle \relax
2284 \let \theoremstyle \relax
2285 \let \proof \relax
2286 \let \endproof \relax
2287 \let \pushQED \relax
2288 \let \popQED \relax
2289 \let \qedhere \relax
2290 \let \mathqed \relax
2291 \let \openbox \relax
2292 }

```

And once the class is loaded restore our versions again. Note that we don't have to restored all the QED-related commands as ours are identical to those defined by the AMS.

```

2293 \AddToHook{class/amsart/after}[block]{
2294 \let \newtheorem \amsnewtheorem
2295 \let \newtheoremstyle \amsnewtheoremstyle
2296 \let \theoremstyle \amstheoremstyle
2297 \let \proof \amsproof
2298 \let \endproof \amsendproof
2299 }

2300 \AddToHook{class/amsbook/after}[block]{
2301 \let \newtheorem \amsnewtheorem
2302 \let \newtheoremstyle \amsnewtheoremstyle
2303 \let \theoremstyle \amstheoremstyle
2304 \let \proof \amsproof
2305 \let \endproof \amsendproof
2306 }

2307 \AddToHook{class/amsproc/after}[block]{
2308 \let \newtheorem \amsnewtheorem
2309 \let \newtheoremstyle \amsnewtheoremstyle

```

```

2310 \let \theoremstyle \amstheoremstyle
2311 \let \proof \amsproof
2312 \let \endproof \amsendproof
2313 }

```

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.

```

2314 %\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`.

```

2315 \declare@file@substitution{enumitem.sty}{null}
2316 \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)`

```

2317 \NewSocketPlug {verbatim/startline}{doc}{ \check@percent }
2318 \AddToHook{package/doc/after}{
2319   \AssignSocketPlug{verbatim/startline}{doc}
2320 }
2321 </package-finish>

```



```

2322 <*latex-lab>
2323 \ProvidesFile{block-latex-lab-testphase.ltx}
2324     [\ltxlabblockdate\space v\ltxlabblockversion\space
2325         blockenv implementation]
2326 \RequirePackage{latex-lab-testphase-block}
2327 </latex-lab>

```

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

\'*1949, 1951*
 @@ commands:
 \l_@@_legacy_env_params_tl *328*
 \ \ *985, 1241, 1242, 1953*
 \{ *1954*
 \} *1955*
 _ *369, 764, 1697*

A

\addpenalty *961, 1119, 1123, 1469*
 \AddToHook *50, 94, 138, 155, 237, 325, 337,*
 612, 1394, 1427, 1725, 1727, 2260,
 2271, 2282, 2293, 2300, 2307, 2318
 \AddToHookWithArguments *645, 647, 649, 651*
 \advspace *962, 1120, 1124, 1126, 1470*
 \advance *2190*
 \aftergroup *613, 2151, 2157, 2164, 2194*
 alltt (env.) *160*
 alltt* (env.) *160*
 \amsendproof *2259, 2298, 2305, 2312*
 \amsnewtheorem *2255, 2294, 2301, 2308*
 \amsnewtheoremstyle *2256, 2295, 2302, 2309*
 \amsproof *2258, 2297, 2304, 2311*
 \amstheoremstyle *2257, 2296, 2303, 2310*
 \arabic *730*
 \AssignSocketPlug *2319*
 \AssignStructureRole *1929*
 \AssignTaggingSocketPlug *608, 1392, 1576, 1724, 1746, 1856*

B

\begin *38, 77, 1773*
 \begingroup *1782, 2221, 2225*
 \bfseries *322, 374*
 block (type) *662*
 block commands:
 \block_debug_off: *36, 623, 628, 642*
 \block_debug_on: *36, 623, 623, 641*

\g_block_nesting_depth_int *37, 873, 877, 881, 891, 918, 926, 938*

block internal commands:

_block_beginpar_hmode:N *1690, 1690, 1859, 1881, 1895*
 _block_beginpar_vmode: *1675, 1675, 1861, 1883, 1897*
 \l_block_block_instance_tl *842, 890, 968*
 \l_block_body_decls_tl *1510, 1566, 1638, 1672*
 \l_block_botsep_skip *1029*
 \l_block_caption_after_skip *1503, 1549, 1634, 1655*
 \l_block_caption_before_skip *1502, 1543, 1633, 1652*
 \l_block_caption_decls_tl *1505, 1540, 1635, 1649*
 _block_captioned_everypar_std: *1049, 1130, 1130, 1565, 1671*
 \g_block_collected_spaces_tl *73, 1610*
 _block_counter_label:n *1250, 1296*
 _block_counter_ref:n *1251*
 \l_block_counter_start_int *1179, 1213, 1216, 1225*
 \l_block_counter_tl *1177, 1211, 1221, 1476, 1524, 1530, 1531, 1598*
 _block_debug:n *621, 621, 635, 2144*
 \g_block_debug_bool *42, 620, 625, 630, 636, 638*
 _block_debug_gset: *623, 626, 631, 633*
 _block_debug_typeout:n *591, 600, 621, 622, 637, 936, 951, 970, 1048, 1131, 1150, 1155, 1159, 1163, 1233, 1235, 1299, 1345, 1397, 1416, 1421, 1564, 1567, 1670, 1673, 1676, 1694, 1715, 1795, 1799, 1832, 1842, 2031, 2032, 2091*
 _block_do_note: *1585, 1585*

_block_do_number:	1593, 1593	\g_block_label_standalone_bool	
_block_do_punct: 75, 1603, 1603, 1654		68, 1267, 1269, 1271, 1348,
_block_do_space:	1610, 1610		1446, 1448, 1495, 1497, 1499, 1552,
_block_do_title: 75, 1577, 1577, 1653			1554, 1626, 1628, 1630, 1658, 1660
_block_drop_spaces:	74,	g_block_label_standalone_bool	1348
75, 1579, 1587, 1601, 1604, 1610, 1617		\l_block_label_strut_bool	1255, 1362
\l_block_effective_top_skip . .	1425	\g_block_label_unchained_bool .	
\l_block_env_name_tl	1046, 1268, 1270, 1272, 1349,
.	836, 1205, 1301, 1519		1496, 1498, 1500, 1627, 1629, 1631
_block_evaluate_saved_user_-		g_block_label_unchained_bool .	1348
keys:nn	59, 60, 63, 1168,	\g_block_labels_box	58,
1168, 1171, 1172, 1194, 1197, 1279		63, 65, 71, 74, 1052, 1100, 1103,	
_block_everypar	58	1138, 1331, 1333, 1351, 1405, 1450,	
_block_everypar:		1536, 1538, 1556, 1645, 1647, 1662	
61, 65-67, 1049, 1151, 1234, 1346,		\l_block_legacy_code_tl	49, 841, 888
1393, 1393, 1394, 1417, 1565, 1671		\l_block_legacy_env_params_tl .	
\l_block_final_code_tl .	51, 849, 919	85, 1976, 1985
_block_if_list:TF	944, 949, 967, 967	\l_block_legacy_support_bool . .	
_block_inner_begin:	1188, 1363
.	1791, 1791, 1871, 1884	_block_list_begin: 1818, 1818, 1898	
_block_inner_end:		_block_list_end: .	1836, 1836, 1899
.	1794, 1794, 1872, 1885	_block_list_item_begin:	
\l_block_inner_instance_tl	1825, 1825, 1900
.	848, 898, 902	_block_list_item_end:	
\l_block_inner_instance_type_tl		1828, 1828, 1901
.	847, 901	\l_block_long_label_bool	
\l_block_inner_level_counter_tl		1329, 1330, 1338, 1353
.	845, 866, 868, 871, 903, 905	_block_make_label_box:n	
_block_insert_spaces:	63, 1293, 1295, 1300, 1354, 1354
.	73, 1581, 1589, 1596, 1610, 1613	\l_block_max_inner_levels_tl . .	
\l_block_instance_keys_tl	846, 869
.	90, 2096, 2098, 2101	\l_block_next_line_bool .	1257, 1337
_block_inter_item:		_block_note_format:n . . .	1509, 1590
.	68, 1442, 1462, 1462	\l_block_note_tl	
\l_block_item_align_tl	90, 1527, 1586, 1590, 2086, 2094, 2103
. .	1261, 1262, 1263, 1318, 1322, 1350	_block_number_format:n .	1507, 1597
\l_block_item_compatibility_-		\l_block_numbered_bool	
bool	1259, 1290	1491, 1526, 1528, 1594
_block_item_everypar_first: . .		\l_block_one_label_box	
.	1234, 1393, 1420	65, 1310, 1314, 1316, 1320, 1321,
_block_item_everypar_std:		1325, 1326, 1328, 1335, 1351, 1356	
.	1346, 1393, 1396	\l_block_order_clist	
_block_item_instance:n	71, 1504, 1544, 2064, 2070
.	68, 1181, 1431, 1444, 1445	\l_block_para_instance_tl	
\l_block_item_label_tl	843, 894, 896
.	1178, 1228, 1230	\l_block_parbotsep_skip	1030
\l_block_item_parsep_skip . . .	1342	\l_block_parindent_dim . .	1037, 1089
_block_label_autoref:n	1253	_block_punct_format:n	
\l_block_label_boxed_bool	1256, 1307	1508, 1607, 1637
_block_label_format:n		\l_block_punct_tl	
.	65, 1254, 1354, 1360	1493, 1605, 1607, 1624
\l_block_label_given_tl		_block_recipe_basic: . . .	1857, 1857
.	63, 1247, 1278, 1287, 1302	_block_recipe_list:	1892, 1892
_block_label_ref:n	1252		

_block_recipe_standalone:	1866, 1866	block std-display-1 (instance)	31
_block_recipe_standard:	1878, 1878	block std-display-2 (instance)	31
\l_block_resume_bool	1180, 1222	block std-display-3 (instance)	31
_block_save_user_keys:n	1169	block std-display-4 (instance)	31
_block_skip_remove_last:	615, 618, 946, 1068, 1465, 1466	block std-display-5 (instance)	31
_block_skip_set_to_last:N	615, 615, 955, 1108	block std-display-6 (instance)	31
\l_block_space_tl	73, 1492, 1611	block std-list-1 (instance)	287
\l_block_swap_number_bool	90, 2056, 2058, 2107, 2108	block std-list-2 (instance)	287
\l_block_tag_class_tl 838, 1906, 1908		block std-list-3 (instance)	287
\l_block_tag_inner_tag_tl	1792, 1875, 1876, 1888, 1889, 1891	block std-list-4 (instance)	287
\l_block_tag_name_tl	837, 1874, 1876, 1887, 1889, 1903, 1905	block std-list-5 (instance)	287
\l_block_tagging_recipe_tl 839, 884		block std-list-6 (instance)	287
\l_block_text_font_tl	1258	block thm-legacy2e-1 (instance)	411
\l_block_thmstyle_tl	88, 1486, 2025, 2027, 2030, 2031, 2032, 2045, 2046, 2055, 2059, 2062, 2063, 2066, 2079	block thm-legacy2e-2 (instance)	411
_block_title_format:n	1506, 1582, 1636	block thm-plain-1 (instance)	394
\l_block_title_tl	70, 1477, 1578, 1582, 1623, 1644	block thm-plain-2 (instance)	394
_block_tmp:w	2134, 2136, 2140, 2142	block thm-remark-1 (instance)	402
\l_block_tmp_clist	89, 2069, 2072, 2080	block thm-remark-2 (instance)	402
\l_block_tmpa_skip	1108, 1109, 1110, 1424	block verbatim-1 (instance)	227
\l_block_topsepadd_skip	53, 962, 1061, 1064, 1124, 1425	block verbatim-2 (instance)	227
\l_block_transparent_level_bool	840, 878, 917, 937	block verbatim-3 (instance)	227
\l_block_unchained_skip	1027, 1056	block verbatim-4 (instance)	227
\l_block_unused_blockenv_keys_-tl	50, 893, 907, 911, 913, 928	block verbatim-5 (instance)	227
block proof-1 (instance)	447	block verbatim-6 (instance)	227
block proof-2 (instance)	447	block/endpe (socket)	972
block quotation-1 (instance)	131	block/list/label (socket)	1371
block quotation-2 (instance)	131	\BlockEnv	15
block quotation-3 (instance)	131	\BlockEnv	15, 37, 87, 922, 2100
block quotation-4 (instance)	131	blockenv (hook)	921
block quotation-5 (instance)	131	blockenv (type)	662
block quotation-6 (instance)	131	blockenv alltt (instance)	197
block quote-1 (instance)	124	blockenv alltt* (instance)	212
block quote-2 (instance)	124	blockenv center (instance)	58
block quote-3 (instance)	124	blockenv description (instance)	274
block quote-4 (instance)	124	blockenv displayblock (instance)	6
block quote-5 (instance)	124	blockenv displayblockflattened (instance)	23
block quote-6 (instance)	124	blockenv enumerate (instance)	260
block std (template)	686, 1023	blockenv flushleft (instance)	72
		blockenv flushright (instance)	90
		blockenv itemize (instance)	245
		blockenv list (instance)	348
		blockenv proof (instance)	422
		blockenv quotation (instance)	100
		blockenv quote (instance)	112
		blockenv std (template)	669, 834
		blockenv verbatim (instance)	165
		blockenv verbatim* (instance)	181
		blockenv verse (instance)	142
		\BlockEnvEnd	15
		\BlockEnvEnd	15, 37, 3, 5, 52, 54, 56, 96, 98, 140, 157, 159, 161, 163, 239, 241, 243, 335, 346, 421, 935, 2021, 2024
		\BockEnvEnd	51

bool commands:	
\bool_gset_false:N . . . 630, 1267, 1268, 1269, 1272, 1448, 1495, 1496, 1497, 1500, 1626, 1627, 1628, 1631	
\bool_gset_true:N 625, 1270, 1271, 1498, 1499, 1554, 1629, 1630, 1660	
\bool_if:NTF 587, 592, 596, 636, 638, 861, 878, 917, 937, 1046, 1222, 1290, 1337, 1338, 1362, 1363, 1446, 1528, 1552, 1594, 1658, 1695, 1709, 1736, 1738, 2056, 2058	
\bool_if:nTF 1305	
\bool_lazy_and:nnTF 1786	
\bool_lazy_or:nnTF 1523, 1937	
\bool_new:N 620, 1348, 1349, 1353, 2107	
\bool_set_false:N 1330, 1526	
\bool_set_inverse:N 2108	
\bool_set_true:N 1329	
\BooleanFalse 14, 925, 2020	
\BooleanTrue 14, 87, 420, 908, 2023	
box commands:	
\box_if_empty:NTF 1403	
\box_new:N 1351, 1352	
\box_use_drop:N 1004, 1138, 1321, 1326, 1405, 1450, 1538, 1556, 1647, 1662	
\box_wd:N 1310, 1314, 1328	
\break 1338	
C	
captionedtext (type) 662	
captionedtext proof (instance) 436	
captionedtext proof (template) . 749, 1621	
captionedtext thmlike (template) 743, 1474	
\catcode 1949, 1951	
center (env.) 50	
\centering 506	
clist commands:	
\clist_clear:N 2069	
\clist_map_inline:Nn 1544, 2070	
\clist_put_right:Nn 2072	
\clist_set:Nn 2064	
\clubpenalty 58, 567, 1146, 1149, 1412, 1415	
color commands:	
\color_select:n 1697	
\cr 2178, 2179, 2180, 2190	
\crr 2193	
cs commands:	
\cs_generate_variant:Nn 619	
\cs_gset_protected:Npx 635, 637	
\cs_if_free:NTF 1431	
\cs_new:Npn 967, 1169	
\cs_new_eq:NN 618, 621, 622, 1168, 1393, 1472, 1473	
\cs_new_protected:Npn 558, 615, 623, 628, 633, 641, 642, 644, 654, 922, 924, 926, 935, 969, 1130, 1154, 1158, 1162, 1354, 1396, 1420, 1462, 1486, 1577, 1585, 1593, 1603, 1610, 1613, 1617, 1785, 1791, 1794, 1836, 1857, 1866, 1878, 1892, 1969, 2089, 2108	
\cs_set:Npe 60, 1172, 1197	
\cs_set:Npn 996, 1006, 1675, 1690, 1818, 1825, 1828, 2131, 2132, 2133, 2134, 2137, 2138, 2139, 2140	
\cs_set_eq:NN 343, 1049, 1151, 1171, 1194, 1234, 1346, 1417, 1565, 1671, 1770, 1771, 1772, 1858, 1860, 1867, 1869, 1871, 1872, 1880, 1882, 1884, 1885, 1894, 1896, 1898, 1899, 1900, 1901	
\cs_set_protected:Npn 531, 548, 1747, 2109	
\csname 1780, 1987, 2005, 2151, 2167, 2202, 2203	
D	
\DebugBlocksOff 36, 39, 641	
\DebugBlocksOn 36, 641	
\DebugLegacySwitchesOn 43	
\DebugSwitchesOff 644	
\DebugSwitchesOn 644	
\DebugTemplatesOff 36	
\DebugTemplatesOn 36	
\DeclareDocumentEnvironment 18, 95, 97, 139, 242	
\DeclareHookRule 1395	
\DeclareInstance 6, 23, 31, 58, 72, 100, 112, 124, 131, 142, 165, 181, 197, 212, 227, 245, 260, 274, 287, 305, 306, 307, 308, 309, 311, 313, 315, 317, 318, 320, 348, 361, 365, 394, 402, 411, 422, 436, 447, 455, 466, 477, 488, 511, 2034, 2051, 2110, 2120	
\DeclareInstanceCopy 45, 46, 47, 48, 49, 86, 90, 126, 127, 128, 129, 130, 133, 134, 135, 136, 137, 232, 233, 234, 235, 236, 300, 301, 302, 303, 304, 381, 387, 392, 401, 410, 417, 454, 2061, 2127	
\DeclareRobustCommand 506, 507, 508, 509, 2207, 2239	
\DeclareTemplateCode 834, 976, 1023, 1175, 1248, 1474, 1489, 1621	
\DeclareTemplateInterface 669, 686, 702, 714, 728, 743, 749, 761	

<code>\def</code>	559, 560, 610, 611, 1728, 1773, 1776, 1777, 1913, 1916, 1942, 1946, 1959, 1960, 1961, 2147, 2154, 2168, 2185, 2192, 2193, 2197, 2205, 2223, 2228, 2234	<code>flushright</code>	50
<code>default (plug)</code> .	585, 1371, 1707, 1733, 1852	<code>itemize</code>	237
<code>description (env.)</code>	237	<code>list</code>	325
<code>\detokenize</code>	970, 1156, 1160, 1164	<code>proof</code>	418
<code>dim</code> commands:		<code>quotation</code>	94
<code>\dim_add:Nn</code>	1090, 1091	<code>quote</code>	94
<code>\dim_compare:nNnTF</code>	956, 1313, 1328, 1343	<code>trivlist</code>	337
<code>\dim_compare_p:n</code>	1309	<code>verbatim</code>	155
<code>\dim_set_eq:NN</code>	1089, 1344	<code>verbatim*</code>	155
<code>\dim_zero:N</code> .	341, 342, 1970, 1971, 1972	<code>verse</code>	138
<code>\c_zero_dim</code>	956, 1756	<code>\eqno</code>	2159, 2160
<code>displayblock (env.)</code>	2	<code>\everydisplay</code>	1950
<code>displayblockflattened (env.)</code>	2	<code>\everymath</code>	1948
<code>\displaywidth</code>	2172	<code>\everypar</code>	40, 54, 55, 58, 570, 573, 574, 799, 1018, 1927
<code>\do</code>	1925, 1963, 1964	<code>exp</code> commands:	
<code>doc (plug)</code>	2317	<code>\exp_after:wN</code>	1041, 1192, 1318, 1322, 1482, 1514, 1642
<code>\dospecials</code>	84, 85, 1925, 1952, 1957, 1964, 1965	<code>\exp_args:Ne</code>	896
<code>dospecials</code> commands:		<code>\exp_args:Nee</code>	889, 900
<code>\dospecials:</code>	85	<code>\exp_args:Nnne</code>	2135, 2141
E		<code>\exp_not:N</code>	2100
<code>\edef</code>	1778	<code>\exp_not:n</code>	60, 853, 990, 1041, 1173, 1192, 1199, 1201, 1277, 1482, 1514, 1642, 2092, 2101, 2103, 2129, 2131, 2132, 2133
<code>\EditInstance</code>	87, 91, 382, 388, 393, 2079, 2135, 2141	<code>\expandafter</code>	1927, 1948, 1950, 1987, 2005, 2167, 2202, 2203, 2217
<code>\egroup</code>	2193	<code>\ExplSyntaxOff</code>	1945, 2146
<code>\else</code>	1919, 1962, 2158, 2163, 2175, 2190, 2199, 2209, 2230, 2241	<code>\ExplSyntaxOn</code>	530, 1967, 2252
<code>else</code> commands:		F	
<code>\else:</code>	1000, 1014	<code>\fi</code>	565, 578, 613, 1744, 1921, 1922, 1963, 2162, 2165, 2183, 2190, 2200, 2212, 2230, 2231, 2244
<code>\end</code>	947	<code>fi</code> commands:	
<code>\endcsname</code>	1780, 1987, 2005, 2151, 2167, 2202, 2203	<code>\fi:</code>	1003, 1017, 1757
<code>\endgraf</code>	1772	<code>\finalhyphendemerits</code>	984
<code>\endgroup</code>	2221, 2226	<code>flushleft (env.)</code>	50
<code>\endproof</code>	2259, 2264, 2275, 2286, 2298, 2305, 2312	<code>flushright (env.)</code>	50
<code>\endsplit</code>	2193	<code>\frenchspacing</code>	1928
<code>\endtrivlist</code>	809	G	
<code>enumerate (env.)</code>	237	<code>\gdef</code>	2171, 2176, 2223, 2230
<code>environments:</code>		<code>\global</code> ...	610, 611, 2173, 2177, 2186, 2190
<code>alltt</code>	160	<code>group</code> commands:	
<code>alltt*</code>	160	<code>\group_begin:</code>	1545, 1653, 1654
<code>center</code>	50	<code>\group_end:</code>	1547, 1653, 1654
<code>description</code>	237	H	
<code>displayblock</code>	2	<code>\hbox</code>	2149, 2161, 2170, 2172, 2205, 2210, 2211, 2238, 2242, 2243, 2247
<code>displayblockflattened</code>	2	<code>hbox</code> commands:	
<code>enumerate</code>	237	<code>\hbox_gset:Nn</code> ..	1100, 1331, 1536, 1645
<code>flushleft</code>	50		

\hbox_set:Nn	1325, 1356	block std-display-5	31
\hbox_set_to_wd:Nnn	1316	block std-display-6	31
\hbox_unpack_drop:N		block std-list-1	287
.....	1052, 1103, 1320, 1333, 1335	block std-list-2	287
\hfil	1338, 2172, 2178, 2248, 2250	block std-list-3	287
\hfill	2210, 2242	block std-list-4	287
hook commands:		block std-list-5	287
\hook_use:n	1752, 1759	block std-list-6	287
Hooks:		block thm-legacy2e-1	411
blockenv	921	block thm-legacy2e-2	411
para/begin	65, 66	block thm-plain-1	394
\hrule	2249	block thm-plain-2	394
\hss	1261, 1262, 1263	block thm-remark-1	402
		block thm-remark-2	402
		block verbatim-1	227
		block verbatim-2	227
		block verbatim-3	227
		block verbatim-4	227
		block verbatim-5	227
		block verbatim-6	227
		blockenv alltt	197
		blockenv alltt*	212
		blockenv center	58
		blockenv description	274
		blockenv displayblock	6
		blockenv displayblockflattened ..	23
		blockenv enumerate	260
		blockenv flushleft	72
		blockenv flushright	90
		blockenv itemize	245
		blockenv list	348
		blockenv proof	422
		blockenv quotation	100
		blockenv quote	112
		blockenv verbatim	165
		blockenv verbatim*	181
		blockenv verse	142
		captionedtext proof	436
		item basic	318
		item description	318
		list description	317
		list enumerate-1	309
		list enumerate-2	309
		list enumerate-3	309
		list enumerate-4	309
		list itemize-1	305
		list itemize-2	305
		list itemize-3	305
		list itemize-4	305
		list legacy	361
		para center	466
		para justify	455
		para raggedleft	488
		para raggedright	477

I

\ialign	2178
if commands:	
\if_int_compare:w	1755
\if_meaning:w	998, 1012
\IfBooleanF	1989, 2053
\IfBooleanT	2091
\iffalse	610
\ifhmode	1921
\ifinner	2157
\IfInstanceExistsF	2025, 2059
\IfInstanceExistsTF	2030, 2044
\ifmmode	2156, 2208, 2240
\IfNoValueF	1644
\IfNoValueTF	1991, 1994, 2095
\iftrue	611
\IfValueT	2092
\ifvmode	1735
\ifx	1962
\ignorespaces	21, 684, 1458
\indent	1464, 1522
instances:	
block proof-1	447
block proof-2	447
block quotation-1	131
block quotation-2	131
block quotation-3	131
block quotation-4	131
block quotation-5	131
block quotation-6	131
block quote-1	124
block quote-2	124
block quote-3	124
block quote-4	124
block quote-5	124
block quote-6	124
block std-display-1	31
block std-display-2	31
block std-display-3	31
block std-display-4	31

par internal commands:	\ProvidesFile 2323
\l__par_fixed_word_spaces_bool . 983	\ProvidesPackage 525
para (type) 662	\pushQED . . 30, 419, 2216, 2265, 2276, 2287
para center (instance) 466	
para commands:	Q
\l_para_begin_skip	\qed 419, 2207, 2239
. 54, 979, 997, 1002, 1011, 1016	\qedhere . . 92, 2224, 2235, 2267, 2278, 2289
\para_end: . . . 38, 56, 77, 78, 1082,	\qedsymbol
1086, 1747, 1747, 1770, 1771, 1772	2149, 2161, 2170, 2172, 2180, 2187,
\g_para_indent_box 1004, 1403	2198, 2205, 2211, 2238, 2243, 2251
\para_omit_indent:	\quad . . . 2149, 2170, 2205, 2211, 2238, 2243
. 1137, 1404, 1555, 1661	quotation (env.) 94
\para_raw_noindent: . . . 54, 1006, 1006	quote (env.) 94
para internal commands:	R
\l__para_begin_skip_tl	\raggedleft 506
. 995, 997, 998, 1011, 1012	\raggedright 506
__para_handle_indent: . . 54, 994, 996	\relax 1261, 1263, 2155, 2160, 2215, 2221,
\g__para_standard_everypar_tl . 1010	2223, 2226, 2228, 2261, 2262, 2263,
para justify (instance) 455	2264, 2265, 2266, 2267, 2268, 2269,
para raggedleft (instance) 488	2272, 2273, 2274, 2275, 2276, 2277,
para raggedright (instance) 477	2278, 2279, 2280, 2283, 2284, 2285,
para std (template) 702, 976	2286, 2287, 2288, 2289, 2290, 2291
para verse (instance) 511	\RemoveFromHook . . 655, 656, 657, 658, 1726
para/begin (hook) 65, 66	\renewcommand 1941
para/begin 38	\RenewDocumentCommand 1428, 1986
\PARALABEL 1139, 1728	\RenewDocumentEnvironment 18,
\parfillskip 982, 1073	51, 53, 55, 156, 158, 238, 240, 326, 338
\parindent 398, 407, 414,	\RequirePackage 528, 529, 2316, 2326
451, 458, 705, 978, 1089, 1344, 2124	\rightmargin 42, 297, 699, 1035, 1090, 1971
\ParseLaTeXeTheoremlike	\rightskip 981, 992, 1072
. 30, 87, 89, 420, 2020, 2023, 2085	\rlap 1697, 2172
\parsep 290, 1028,	
1095, 1111, 1120, 1126, 1183, 1342	S
\parskip 9, 40, 35, 399, 408,	scan commands:
415, 452, 691, 959, 1094, 1095, 2125	\scan_stop: 1748
\partopsep 34, 289, 689, 1026, 1064	\setbox 573, 2181
\pdfspaces 1942	\setcounter 30, 934
\penalty 1141, 1407, 1918, 1921, 2210, 2242	\SetKnownTemplateKeys
\phantom 2170 39, 531, 855, 1042,
Plugs:	1173, 1196, 1198, 1280, 1483, 1515
default 585, 1371, 1707, 1733, 1852	\setlist 96
doc 2317	\SetTemplateKeys . . 39, 74, 548, 991, 1643
kernel 1570	\ShowInstanceValues 2144
off 53, 53, 973	\SimpleBlockEnv 14
on 53, 53, 973	\SimpleBlockEnv 14, 26, 37, 96,
\popQED . . . 30, 421, 2220, 2266, 2277, 2288	3, 5, 52, 54, 56, 96, 98, 140, 157,
prg commands:	159, 161, 163, 239, 241, 243, 333, 922
\prg_do_nothing: 1151, 1393, 1417,	skip commands:
1472, 1473, 1862, 1863, 1868, 1870	\skip_add:Nn 1064
proof (env.) 418	\skip_eval:n 1120, 1124
\proof 2258,	\skip_horizontal:n . . 1102, 1104,
2263, 2274, 2285, 2297, 2304, 2311	1334, 1336, 1543, 1549, 1652, 1655
\protected 1773	\skip_new:N 1424, 1425, 1426
\providecommand 2251	

\skip_set:Nn	616, 992, 1061	tag internal commands:	
\skip_set_eq:NN	1072, 1073, 1094, 1095, 1342	\l__tag_block_flattened_level_-int	48, 856, 858, 863, 929, 1683
\skip_use:N	994, 997, 1011	__tag_check_para_begin_show:nn	1721
\skip_vertical:n	563, 958, 959, 1056, 1109, 1110	__tag_gincr_para_begin_int:	1714
\skip_zero:N	1071	__tag_gincr_para_end_int:	1693
\l_tmpa_skip	955, 956, 958, 959	\l__tag_L_attr_class_tl	1806, 1807, 1822, 1907, 1908, 1978, 1980, 1981
socket commands:		\l__tag_L_tag_tl	1803, 1804, 1821, 1904, 1905
\socket_assign_plug:nn	975, 1864, 1873, 1886, 1902	\l__tag_para_attr_class_tl	986, 1719
\socket_if_exist:nTF	581, 1703, 1729, 1848	\l__tag_para_bool	587, 1736, 1786
\socket_new:nn	972	\g__tag_para_end_int	1697
\socket_new_plug:nnn	973, 974	\l__tag_para_flattened_bool	593, 596, 844, 861, 1709, 1738
\socket_use:n	965	__tag_para_main_store_struct:	1686, 1712
Sockets:		\l__tag_para_main_tag_tl	601
block/endpe	972	\l__tag_para_show_bool	1695
block/list/label	1371	\l__tag_para_tag_tl	1718
tagsupport/@doendpe	581	\tagmcbegin	1383
tagsupport/block/recipe	1848	\tagmcend	1386
tagsupport/block/startpara/direct	1703	\tagpdfparaOff	1701
tagsupport/captionedtext/caption	1569	\tagpdfparaOn	1701
tagsupport/kernel/endpe/vmode	1729	\tagpdfsetup	499, 1808
verbatim/startline	96, 1934	\tagstructbegin	1382, 1390, 1792, 1819, 1826
\space	526, 601, 852, 989, 1040, 1191, 1205, 1242, 1276, 1481, 1513, 1519, 1641, 2031, 2032, 2324	\tagstructend	1389, 1801, 1834, 1844, 1846
str commands:		tagsupport/@doendpe (socket)	581
\str_case:nnTF	2073	tagsupport/block/recipe (socket)	1848
\string	1433	tagsupport/block/startpara/direct (socket)	1703
\strut	1362	tagsupport/captionedtext/caption (socket)	1569
\swapnumbers	14, 27, 28, 87, 88, 90, 91, 2107	tagsupport/kernel/endpe/vmode (socket)	1729
sys commands:		template commands:	
\sys_if_engine luatex_p:	1938	\template_debug_off:	642
\sys_if_output_dvi_p:	1939	\template_debug_on:	641
		\template_debug_typeout:n	558, 558, 852, 989, 1040, 1191, 1276, 1481, 1513, 1641
T		template internal commands:	
\tabskip	2189	__template_debug_typeout:n	558
\tag	2198	template types:	
tag commands:		block	662
\tag_if_active_p:	1786	blockenv	662
\tag_mc_begin:n	1379, 1696, 1722	captionedtext	662
\tag_mc_end:	1692, 1698	item	662
\tag_socket_use:n	568	list	662
\tag_socket_use:nn	1788	para	662
\tag_socket_use:nnn	1358, 1541, 1580, 1588, 1595, 1606, 1650	thmstyle	662
\tag_struct_begin:n	1572, 1716	templates:	
\tag_struct_end:	1574, 1700	block std	686, 1023
		blockenv std	669, 834

captionedtext proof	749, 1621	\@mklab	1975
captionedtext thmlike	743, 1474	\@one	2190
item std	728, 1246	\@new@specials	1960, 1963, 1965
list std	714, 1175	\@newctr	2002
para std	702, 976	\@nmbrlisttrue	1220
thmstyle std	761, 1489	\@nocounterr	2013
T _E X and L ^A T _E X 2 _ε commands:			
\@@par	78, 1771, 1918, 1921	\@noitemerr	61, 80, 944, 1081, 1116, 1422
\@beginparpenalty	9, 1032, 1123	\@noligs	1926
\@begintheorem	38	\@normalcr	10, 464, 712
\@centercr	475, 486, 497, 520	\@nthm	38
\@clubpenalty	567, 1149, 1415	\@nx	2235
\@currenvir	947, 1777, 2148, 2151	\@opargbegintheorem	38
\@currvline	1778	\@othm	38, 86
\@definecounter	1993	\@outerparskip	959, 1094, 1111, 1120, 1125
\@doendpe	38, 40, 41, 559	\@remove	1961, 1964
\@domathendpefalse	564, 577, 609	\@restorepar	566
\@domathendpetrue	609	\@rightskip	992, 1072
\@eha	1776	\@setpar	1075
\@ehc	1434	\@setupverbinvisiblespace	37, 1935
\@empty	2173, 2177, 2214	\@setupverbvisiblespace	84
\@endparpenalty	9, 961, 1033	\@sxverbatim	21, 195
\@endpefalse	77, 569, 575, 613, 974, 1742	\@tempwafalse	1915
\@endpetrue	559, 613, 973	\@tempwatrue	1920
\@endtheorem	38	\@temptokena	2217, 2218
\@enumdepth	23, 270	\@thm	38
\@eqnswtrue	2190	\@thmcounter	1998, 2007
\@execute@begin@hook	1779	\@thmcountersep	2006
\@flushglue	10, 462, 471, 472, 483, 493, 518, 709, 1073	\@toodeep	870, 875
\@ifdefinable	1987	\@topsep	67
\@ifundefined	1775, 2012, 2148	\@topsepadd	67
\@ignorefalse	1781	\@totalleftmargin	83, 1091, 1092
\@inmatherr	947, 1430	\@vobeyspaces	1932
\@item	790, 798	\@xnthm	38, 86
\@itemdepth	23, 251	\@xobeysp	1942
\@itemlabel	38, 59, 61, 330, 885, 1166, 1230, 1294, 1377, 1388, 1979	\@xp	2151
\@itempenalty	9, 1034, 1184, 1469	\@xthm	38
\@kernel@after@para@after	1760	\@xverbatim	179
\@kernel@after@para@end	1753	\@ynthm	38, 86
\@kernel@refstepcounter	1289, 1530	\@ythm	38
\@labels	65	\endpefalse	78
\@latex@error	1433, 1776	\align@qed	2194, 2197, 2202, 2203
\@latex@warning	2026	\alt@tag	2171, 2173, 2176, 2177
\@list...	8	\arabic	11
\@listctr	59, 61, 886, 1166, 1215, 1221, 1224, 1289, 1293, 1295, 1296, 1973	\begin	38
\@listdepth	8, 51, 926	\bibitem	80
\@listi	7, 8	\c@maxblocklevels	38, 874, 933
\@listii	7, 8	\check@percent	96, 2317
\@listvi	8	\ctagsplit@false	2193
\@makeother	1925	\declare@file@substitution	2253, 2254, 2314, 2315
		\displaymath@qed	2154, 2167
		\end	40
		\equation@qed	2168

<code>\everypar</code>	68, 69	<code>\setboxz@h</code>	2187
<code>\g@addto@macro</code>	1963	<code>\SetKnownTemplateKeys</code>	59
<code>\g@remfrom@specials</code>	1953, 1954, 1955, 1959	<code>\setQED@elt</code>	2226, 2228
<code>\if@domathendpe</code>	562, 576, 609	<code>\spacefactor</code>	61
<code>\if@endpe</code>	613, 1735	<code>\split@qed</code>	2192
<code>\if@tempwa</code>	1917	<code>\strut</code>	11, 65
<code>\iffirstchoice@</code>	2230	<code>\strut@</code>	2187
<code>\ifmeasuring@</code>	2198, 2229	<code>\tag@true</code>	2186
<code>\ifst@rred</code>	2190	<code>\tagform@</code>	2179
<code>\iftagsleft@</code>	2169	<code>\tagsleft@false</code>	2187
<code>\ignorespaces</code>	8, 51	<code>\toks@</code>	2217, 2218
<code>\item</code>	15,	<code>\topsep</code>	9
38, 52, 53, 56, 57, 59–63, 65, 67, 80, 83		<code>\UseInstance</code>	68
<code>\itemsep</code>	9	<code>\verbatim@font</code>	1926
<code>\l@nohyphenation</code>	1914	<code>\z@</code>	573, 2181
<code>\labelsep</code>	11	<code>\z@skip</code>	460, 461, 473, 482, 484, 494, 495, 994
<code>\labelwidth</code>	11, 63, 65	tex commands:	
<code>\leftmargin</code>	9	<code>\tex_everypar:D</code>	1009, 1010
<code>\leftskip</code>	83	<code>\tex_hskip:D</code>	1002, 1016, 1756
<code>\legacylistsetup</code>	25	<code>\tex_lastnodetype:D</code>	1755
<code>\linebox@qed</code>	2157, 2164	<code>\tex_lastskip:D</code>	616
<code>\list</code>	26	<code>\tex_noindent:D</code>	1021
<code>\list<romannumeral></code>	16, 24	<code>\tex_par:D</code>	1758, 1767
<code>\listparindent</code>	55, 64	<code>\tex_parshape:D</code>	1092
<code>\makelabel</code>	11, 27, 65, 85	<code>\tex_parskip:D</code>	563
<code>\math@qedhere</code>	2147, 2205, 2225	<code>\tex_unskip:D</code>	618, 1751
<code>\newline</code>	63	<code>\the</code>	1018, 1927, 1948, 1950, 2218
<code>\newtheorem</code>	86	<code>\theequation</code>	2179
<code>\noitemerr</code>	52	<code>\theoremstyle</code>	27,
<code>\on@line</code>	595, 602, 936,	28, 69, 87, 90, 1486, 2028, 2257,	
1048, 1131, 1150, 1233, 1345, 1397,		2262, 2273, 2284, 2296, 2303, 2310	
1416, 1421, 1564, 1670, 1677, 1694,		<code>\thmname</code>	91, 2131, 2137
1715, 1778, 1795, 1799, 1832, 1842		<code>\thmnote</code>	91, 2133, 2139
<code>\org@dosppecials</code>	1952, 1957	<code>\thmnumber</code>	91, 2132, 2138
<code>\org@prime</code>	1947, 1949, 1951	thmstyle (type)	662
<code>\par</code>	36,	<code>\thmstyle</code>	29
40, 41, 53, 56, 61, 69, 76, 77, 81–83		thmstyle definition (instance)	387
<code>\par@deathcycles</code> ...	1074, 1079, 1080	thmstyle legacy2e (instance)	392
<code>\parindent</code>	10, 29, 64	thmstyle plain (instance)	365
<code>\parskip</code>	9, 29, 57	thmstyle remark (instance)	381
<code>\partopsep</code>	9	thmstyle std (template)	761, 1489
<code>\pdfmakespace</code>	21, 84	<code>\tiny</code>	1697
<code>\place@tag@gather</code>	2188	tl commands:	
<code>\popQED@elt</code>	2221, 2223	<code>\c_empty_tl</code>	535, 540
<code>\qed@elt</code> .	2215, 2217, 2221, 2226, 2230	<code>\c_novalue_tl</code>	63, 1278
<code>\QED@stack</code>	2214,	<code>\tl_clear:N</code>	885, 886, 2096
2217, 2218, 2221, 2223, 2226, 2230		<code>\tl_const:Nn</code>	994
<code>\qed@tag</code>	2185, 2199	<code>\tl_gclear:N</code>	1615, 1618
<code>\qed@warning</code>	2149, 2234	<code>\tl_gput_right:Nn</code>	1611
<code>\rendsplit@</code>	2193	<code>\tl_gset:Nn</code>	1996, 2003
<code>\reserved@a</code>	1776, 1777, 1784	<code>\tl_if_blank:nTF</code>	1289, 2116
<code>\rightmargin</code>	9	<code>\tl_if_empty:NTF</code>	866, 894, 898, 903, 913,
<code>\row@</code>	2190		

1202, 1211, 1228, 1281, 1516, 1578, 1605, 1874, 1887, 1903, 1906, 1979	\use:n 90, 343, 1365, 2099
\tl_if_empty:nTF 533, 550, 1170, 1193, 1377, 1388, 2112, 2115, 2118, 2121, 2122, 2129	\use_i:nn 1318
\tl_if_empty_p:N 1524	\use_ii:nn 1322
\tl_if_eq:NnTF 968	\use_none:n 621, 622
\tl_if_novalue:nTF 538, 552, 619, 619, 1287, 1443, 1586	\use_none:nn 1168, 1171, 1194
\tl_new:N 928, 995, 1166, 1167, 1350, 1487, 1620, 1803, 1806, 1891, 1985	\usecounter 61
\tl_set:Nn 328, 330, 997, 1011, 1261, 1262, 1263, 1486, 1527, 1644, 1804, 1807, 1875, 1888, 1904, 1907, 1973, 1978, 1980, 1981, 2094	\UseHook 854, 1774
\tl_set_eq:NN 535, 540, 893, 911, 1221, 1230, 1278, 1876, 1889, 1905, 1908	\UseInstance 50, 506, 507, 508, 509, 889, 896, 901, 923, 925
tl internal commands:	\UseName 39, 40, 294, 295, 695, 696, 697, 722
\c_zero_skip_tl 994, 999, 1013	\UseSocket 1923
\topsep 55, 33, 288, 404, 688, 690, 1025, 1061	\UseStructureName . 103, 115, 145, 168, 184, 200, 215, 248, 263, 277, 351, 425, 1382, 1383, 1386, 1390, 1572, 1595, 1826, 1834, 1844, 1930, 2037
trivlist (env.) 337	\UseTaggingSocket 598, 884, 1685, 1711, 1740, 1766, 1783, 1798, 1831, 1841
\typeout 646, 648, 650, 652	
	V
U	\vbox 2249
\unpenalty 1927	\veqno 2160
\unskip 2210, 2242	verbatim (env.) 155
\UnusedTemplateKeys 48, 50, 535, 540, 544, 892, 893, 911, 1199, 1202, 1206, 1281, 1285, 1484, 1516, 1520	verbatim* (env.) 155
use commands:	verbatim/startline (socket) 96, 1934
\use:N 880, 1546, 1598, 1854, 1931	verse (env.) 138
	\vfil 2249
	\vrule 2248, 2250
	\vtop 2178
	X
	\xdef 2218