Prototype reimplementation of $\LaTeX\,2_\varepsilon$'s block environments using templates

LaTeX Project*

v0.9m 2026-01-16

Abstract

Contents

1 Introduction 3

2 Template types and templates for blocks and lists 3

2.1 Template types . . . 3
2.1.1 The template type 'blockenv' . . . 3
2.1.2 The template type 'block' . . . 4
2.1.3 The template type 'para' . . . 4
2.1.4 The template type 'list' . . . 4
2.1.5 The template type 'captionedtext' . . . 5
2.1.6 The template type 'item' . . . 5
2.1.7 The template type 'thmstyle' . . . 5
2.2 Templates . . . 6
2.2.1 The `blockenv` template 'std' . . . 6
2.2.2 The `block` template 'std' . . . 8
2.2.3 The `para` template 'std' . . . 9
2.2.4 The `list` template 'std' . . . 10
2.2.5 The `item` template 'std' . . . 10
2.2.6 The `captionedtext` template 'thmlike' . . . 11
2.2.7 The `captionedtext` template 'proof' . . . 11
2.2.8 The `thmstyle` template 'std' . . . 12

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

3 Declaring standard display block environments and their instances 13
3.1 The `display` and `displayflattened` environments . . . 14
3.1.1 Their `blockenv` instances . . . 14
3.1.2 Their `block` instances . . . 15
3.2 The `center`, `flushleft`, and `flushright` environments . . . 15
3.2.1 Their `blockenv` instances . . . 16
3.2.2 Their `block` instances . . . 17
3.2.3 Their `para` instances . . . 17
3.3 The `quote` and `quotation` environments . . . 17
3.3.1 Their `blockenv` instances . . . 17
3.3.2 Their `block` instances . . . 18
3.4 The `verse` environment . . . 18
3.4.1 Their `blockenv` instances . . . 18
3.5 The `verbatim`, `verbatim*` and `alltt` environments . . . 19
3.5.1 Their `blockenv` instances . . . 19
3.5.2 Their `block` instances . . . 21
3.6 The `trivlist` environment . . . 21
3.7 The standard lists: `itemize`, `enumerate`, and `description` . . . 21
3.7.1 Their `blockenv` instances . . . 22
3.7.2 Their `block` instances . . . 23
3.7.3 Their `list` instances . . . 23
3.7.4 Their `item` instances . . . 24
3.8 The legacy `list` and `trivlist` environments . . . 24
3.8.1 Its `blockenv` instance . . . 25
3.8.2 Its `list` instance . . . 25
3.9 Theorem-like environments declared through `\newtheorem` . . . 26
3.9.1 The `blockenv` instances they use . . . 26
3.9.2 The `captionedtext` instances they use . . . 27
3.9.3 The `thmstyle` instances they use . . . 27
3.9.4 The `block` instances they use . . . 28
3.10 The `proof` environment (from amsthm) . . . 29
3.10.1 Block instances for the proofs . . . 30

4 Declaring `para` instances 30

5 Advice on adjusting the layout of standard block environments 32

6 Tagging support 32
6.1 Paragraph tags . . . 32
6.1.1 Tagging recipes . . . 34

7 Tracing and debugging 35

8 New and redefined kernel command 36

Index 38

1 Introduction

The list implementation in LaTeX 2ε 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 `\\` 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 $\LaTeX\,2_\varepsilon$ 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 used 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 LaTeX 2ε 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 indention 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 LaTeX 2$_\varepsilon$ always chained the list items, so the ability to prohibit this is new functionality.

2.2.3 The para template 'std'

Attributes:

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

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

`left-hspace` (*skip*) Default: `0pt`

`right-hspace` (*skip*) Default: `0pt`

`end-hspace` (*skip*) Default: `\@flushglue`

`fixed-word-spaces` (*boolean*) Default: `false`

`final-hyphen-demerits` (*integer*) Default: `5000`

`newline-cmd` (*function(0)*) This defines the meaning of `\\` Default: `\@normalcr`

`para-attr-class` (*tokenlist*) Default: `justify`

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

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` (*choice*) Supported values `left`,`center`, `right`, and `parleft`. *Only partly implemented.* Default: `right`

`label-placement` (*choice*) Placement of the label in relation to a directly following label (of a following inner list). Supported values are `chained`, `unchained`, and `standalone`. Default: `chained`

`label-boxed` (*boolean*) Should the label be boxed? Default: `true`

`next-line` (*boolean*) Default: `false`

`text-font` (*tokenlist*) *unused.*

`compatibility` (*boolean*) Default: `true`

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

fix

2.2.6 The `captionedtext` template 'thmlike'

Attributes:

`counter` (*tokenlist*) Counter name to be used if the caption is numbered, otherwise empty. Default: ⟨*empty*⟩

`title` (*tokenlist*) Fixed part of the caption, e.g., a theorem-like environment may want to specify "Lemma" here. Default: ⟨*empty*⟩

`style` (*instance*) Instance of type `thmstyle` that actually implements the theorem-like environment. Default: `plain`

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

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

2.2.7 The `captionedtext` template 'proof'

Attributes:

`title` (*tokenlist*) Heading for the environment unless overwritten on document level. Default: `Proof`

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

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

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

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

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

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

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

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

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

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

2.2.8 The `thmstyle` template 'std'

Attributes:

`numbered` (*boolean*) Is this kind of environment numbered? Default: `true`

`space` (*tokenlist*) Space to be applied between elements of the heading Default: `\␣`

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

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

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

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

`order` (*commalist*) Order of elements in the environment caption/heading. Supported values are `title`, `number`, `punct`, `space`, and `note`. Default: `title,space,number,space,note`

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

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

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

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

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

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

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

The caption of the environment can consist of a title, a number, a punctuation, some 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 they way `\newtheoremstyle` worked and so we mimic that.

3 Declaring standard display block environments and their instances

Historically the LaTeX 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} { !O{} }
   { \SimpleBlockEnv{center}{#1} } { \BlockEnvEnd }
```

instead of the more verbose

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

We use `!O{}` 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` `\BlockEndEnd`

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.

```
⟨*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 my 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.*)
`displayblockflattened` (*env.*)

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

```
\NewDocumentEnvironment{displayblock}{ !O{} }
  { \SimpleBlockEnv{displayblock} {#1} } { \BlockEnvEnd }

\NewDocumentEnvironment{displayblockflattened}{ !O{} }
  { \SimpleBlockEnv{displayblockflattened} {#1} } { \BlockEnvEnd }
```

3.1.1 Their blockenv instances

`blockenv displayblock` (*inst.*)

This is like LaTeX 2$_\varepsilon$'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.

```
\DeclareInstance{blockenv}{displayblock}{std}
{
   name                = displayblock
%  ,tagging-recipe      = standard
%  ,tag-name            =
%  ,tag-attr-class      =
  ,transparent-level    = true
%  ,legacy-code         =
%  ,block-instance      = std-display
%  ,para-instance       =
%  ,tagging-suppress-paras = false
%  ,inner-instance      =
%  ,inner-instance-type =         % not relevant as there is no inner instance
%  ,inner-level-counter =         % not relevant as there is no inner instance
%  ,max-inner-levels    = 4        % not relevant as there is no inner instance
%  ,final-code          = \ignorespaces
}
```

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

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

```
\DeclareInstance{blockenv}{displayblockflattened}{std}
{
   name                   = displayblockflattened
  ,tagging-recipe         = basic
  ,tagging-suppress-paras = true
  ,transparent-level      = true
}
```

3.1.2 Their `block` instances

We provide 6 nesting levels (as in LaTeX 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).

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

block std-display-1 (*inst.*)
block std-display-2 (*inst.*)
block std-display-3 (*inst.*)
block std-display-4 (*inst.*)
block std-display-5 (*inst.*)
block std-display-6 (*inst.*)

We show all keys here for reference, with those using their default values commented out:

```
\DeclareInstance{block}{std-display-1}{std}
  {
%    ,begin-vspace       = \topsep
%    ,begin-extra-vspace = \partopsep
%    ,para-vspace        = \parskip
%    ,end-vspace         = \KeyValue{begin-vspace}
%    ,end-extra-vspace   = \KeyValue{begin-extra-vspace}
%    ,item-vspace        = \itemsep
%    ,begin-penalty      = \UseName{@beginparpenalty}
%    ,end-penalty        = \UseName{@endparpenalty}
    ,left-margin        = 0pt
%    ,right-margin       = \rightmargin
%    ,para-indent        = 0pt
  }
```

```
\DeclareInstanceCopy{block}{std-display-2}{std-display-1}
\DeclareInstanceCopy{block}{std-display-3}{std-display-1}
\DeclareInstanceCopy{block}{std-display-4}{std-display-1}
\DeclareInstanceCopy{block}{std-display-5}{std-display-1}
\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.*)
flushleft (*env.*)
flushright (*env.*)

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

```
\AddToHook{begindocument/before}[./legacy-core]{
```

```
  \RenewDocumentEnvironment{center} { !O{} }
  { \SimpleBlockEnv{center}{#1} } { \BlockEnvEnd }
  \RenewDocumentEnvironment{flushright} { !O{} }
  { \SimpleBlockEnv{flushright}{#1} } { \BlockEnvEnd }

  \RenewDocumentEnvironment{flushleft} { !O{} }
  { \SimpleBlockEnv{flushleft}{#1} } { \BlockEnvEnd }
}
```

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.

```
\DeclareInstance{blockenv}{center}{std}
{
   name                   = center
  ,tag-name               =
  ,tag-attr-class         =
  ,tagging-recipe         = basic
  ,tagging-suppress-paras = true
  ,inner-level-counter    =
  ,transparent-level      = true
  ,legacy-code            =
  ,block-instance         = std-display
  ,para-instance          = center
  ,inner-instance         =
}
```

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

```
%\DeclareInstance{blockenv}{flushleft}{std}
%{
%   name                   = flushleft
%  ,tag-name               =
%  ,tag-attr-class         =
%  ,tagging-recipe         = basic
%  ,tagging-suppress-paras = true
%  ,inner-level-counter    =
%  ,transparent-level      = true
%  ,legacy-code            =
%  ,block-instance         = std-display
%  ,para-instance          = raggedright
%  ,inner-instance         =
%}
```

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

```
 \DeclareInstanceCopy{blockenv}{flushleft}{center}
 \EditInstance{blockenv}{flushleft}{
     name          = flushleft
    ,para-instance = raggedright }
```

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

```
\DeclareInstanceCopy{blockenv}{flushright}{center}
```

```
\EditInstance{blockenv}{flushright}{
   name          = flushright
  ,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

LaTeX 2ε 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.*) `quotation` (*env.*) We can't use `\RenewDocumentEnvironment` for `quote` and other environments that 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.

```
\AddToHook{begindocument/before}[./legacy-quotes]{
  \DeclareDocumentEnvironment{quote}{ !O{} }
    { \SimpleBlockEnv{quote} {#1} } { \BlockEnvEnd }

  \DeclareDocumentEnvironment{quotation}{ !O{} }
    { \SimpleBlockEnv{quotation} {#1} } { \BlockEnvEnd }
}
```

3.3.1 Their `blockenv` instances

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

```
\DeclareInstance{blockenv}{quotation}{std}
{
   name                = quotation
  ,tag-name            = \UseStructureName{block/quotation}
  ,tag-attr-class      =
  ,tagging-recipe      = standard
  ,inner-level-counter =
  ,transparent-level   = false
  ,legacy-code         =
  ,block-instance      = quotation
  ,inner-instance      =
}
```

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

```
\DeclareInstance{blockenv}{quote}{std}
{
   name                = quote
```

```
  ,tag-name            = \UseStructureName{block/quote}
  ,tag-attr-class      =
  ,tagging-recipe      = standard
  ,inner-level-counter =
  ,transparent-level   = false
  ,legacy-code         =
  ,block-instance      = quote
  ,inner-instance      =
}
```

3.3.2 Their block instances

block quote-1 (*inst.*)
block quote-2 (*inst.*)
block quote-3 (*inst.*)
block quote-4 (*inst.*)
block quote-5 (*inst.*)
block quote-6 (*inst.*)

Default layout is to indent equally from both sides.

```
\DeclareInstance{block}{quote-1}{std}
  { right-margin = \KeyValue{left-margin} }
\DeclareInstanceCopy{block}{quote-2}{quote-1}
\DeclareInstanceCopy{block}{quote-3}{quote-1}
\DeclareInstanceCopy{block}{quote-4}{quote-1}
\DeclareInstanceCopy{block}{quote-5}{quote-1}
\DeclareInstanceCopy{block}{quote-6}{quote-1}
```

block quotation-1 (*inst.*)
block quotation-2 (*inst.*)
block quotation-3 (*inst.*)
block quotation-4 (*inst.*)
block quotation-5 (*inst.*)
block quotation-6 (*inst.*)

Quotation additionally changes the `para-indent`.

```
\DeclareInstance{block}{quotation-1}{std}
   { para-indent = 1.5em , right-margin = \KeyValue{left-margin} }
\DeclareInstanceCopy{block}{quotation-2}{quotation-1}
\DeclareInstanceCopy{block}{quotation-3}{quotation-1}
\DeclareInstanceCopy{block}{quotation-4}{quotation-1}
\DeclareInstanceCopy{block}{quotation-5}{quotation-1}
\DeclareInstanceCopy{block}{quotation-6}{quotation-1}
```

3.4 The verse environment

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

verse (*env.*)

Implementation is like `quote` etc.

```
\AddToHook{begindocument/before}[./legacy]{
  \DeclareDocumentEnvironment{verse}{!O{}}
    { \SimpleBlockEnv{verse} {#1} } { \BlockEnvEnd }
}
```

3.4.1 Their blockenv instances

blockenv verse (*inst.*)

```
\DeclareInstance{blockenv}{verse}{std}
{
   name                = verse
  ,tag-name            = \UseStructureName{block/verse}
  ,tag-attr-class      =
  ,tagging-recipe      = standard
  ,inner-level-counter =
  ,transparent-level   = false
  ,legacy-code        =
  ,block-instance     = quote       % reuse?
  ,para-instance      = verse
  ,inner-instance     =
}
```

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.*)
verbatim* (*env.*)

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

fix

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

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

alltt (*env.*)
alltt* (*env.*)

The alltt package implements a variation on verbatim handling where backslash and 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.

The parsing here should be adjusted as well, eventually.

```
  \NewDocumentEnvironment{alltt}{ ={legacy-code} !o }
    { \SimpleBlockEnv{alltt} {#1} } { \BlockEnvEnd }
  \NewDocumentEnvironment{alltt*}{ ={legacy-code} !o }
    { \SimpleBlockEnv{alltt*} {#1} } { \BlockEnvEnd }
}
```

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.

```
\DeclareInstance{blockenv}{verbatim}{std}
{
   name                    = verbatim
  ,tag-name                = \UseStructureName{block/verbatim}
  ,tag-attr-class          =
  ,tagging-recipe          = standard
  ,tagging-suppress-paras  = true
  ,inner-level-counter     =
  ,transparent-level       = true
  ,legacy-code             =
  ,block-instance          = verbatim
  ,inner-instance          =
   ,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 `\pdffakespace` in the pdfTEX engine) and in `verbatim*` we set it up to generate visible space chars.

```
   ,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]

```
                             \@xverbatim
}
```

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

```
\DeclareInstance{blockenv}{verbatim*}{std}
{
    name                   = verbatim
   ,tag-name               = \UseStructureName{block/verbatim}
   ,tag-attr-class         =
   ,tagging-recipe         = standard
   ,tagging-suppress-paras = true
   ,inner-level-counter    =
   ,transparent-level      = true
   ,legacy-code            =
   ,block-instance         = verbatim
   ,inner-instance         =
   ,para-instance          = justify
   ,final-code             = \legacyverbatimsetup{visible}
                             \@sxverbatim
}
```

`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.

```
\DeclareInstance{blockenv}{alltt}{std}
{
    name                   = alltt
   ,tag-name               = \UseStructureName{block/verbatim}   % private tag instead?
   ,tag-attr-class         =
   ,tagging-recipe         = standard
   ,tagging-suppress-paras = true
   ,inner-level-counter    =
   ,transparent-level      = true
   ,legacy-code            =
   ,block-instance         = verbatim
   ,inner-instance         =
   ,para-instance          = justify
```

[1]Perhaps 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.

```
  ,final-code             = \legacyalltsetup {invisible}
}
```

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

```
\DeclareInstance{blockenv}{alltt*}{std}
{
   name                   = alltt*
  ,tag-name               = \UseStructureName{block/verbatim}   % private tag instead?
  ,tag-attr-class         =
  ,tagging-recipe         = standard
  ,tagging-suppress-paras = true
  ,inner-level-counter    =
  ,transparent-level      = true
  ,legacy-code            =
  ,block-instance         = verbatim
  ,inner-instance         =
  ,para-instance          = justify
  ,final-code             = \legacyalltsetup {visible}
}
```

3.5.2 Their block instances

`block verbatim-1` (*inst.*)
`block verbatim-2` (*inst.*)
`block verbatim-3` (*inst.*)
`block verbatim-4` (*inst.*)
`block verbatim-5` (*inst.*)
`block verbatim-6` (*inst.*)

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

```
\DeclareInstance{block}{verbatim-1}{std}
  {
    ,left-margin      = 0pt
    ,para-vspace      = 0pt
  }
```

```
\DeclareInstanceCopy{block}{verbatim-2}{verbatim-1}
\DeclareInstanceCopy{block}{verbatim-3}{verbatim-1}
\DeclareInstanceCopy{block}{verbatim-4}{verbatim-1}
\DeclareInstanceCopy{block}{verbatim-5}{verbatim-1}
\DeclareInstanceCopy{block}{verbatim-6}{verbatim-1}
```

3.6 The trivlist environment

In LaTeX 2_ε `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.

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

3.7 The standard lists: itemize, enumerate, and description

`itemize` (*env.*)
`enumerate` (*env.*)
`description` (*env.*)

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

```
\AddToHook{begindocument/before}[./legacy-lists]{
  \RenewDocumentEnvironment{itemize}{!O{}}
    { \SimpleBlockEnv{itemize} {#1} } { \BlockEnvEnd }
  \RenewDocumentEnvironment{enumerate}{!O{}}
    { \SimpleBlockEnv{enumerate} {#1} } { \BlockEnvEnd }
  \DeclareDocumentEnvironment{description}{!O{}}
    { \SimpleBlockEnv{description} {#1} } { \BlockEnvEnd }
}
```

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.

```
\DeclareInstance{blockenv}{itemize}{std}
{
   name                 = itemize
  ,tag-name             = \UseStructureName{block/itemize}
  ,tag-attr-class       = itemize
  ,tagging-recipe       = list
  ,inner-level-counter = \@itemdepth
  ,transparent-level   = false
  ,max-inner-levels    = 4
  ,legacy-code         =
  ,block-instance      = std-list
  ,inner-instance-type = list
  ,inner-instance      = itemize
  ,para-instance       =
}
```

`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`.

```
\DeclareInstance{blockenv}{enumerate}{std}
{
   name                 = enumerate
  ,tag-name             = \UseStructureName{block/enumerate}
  ,tag-attr-class       = enumerate
  ,tagging-recipe       = list
  ,transparent-level   = false
  ,max-inner-levels    = 4
  ,legacy-code         =
  ,block-instance      = std-list
  ,inner-level-counter = \@enumdepth
  ,inner-instance-type = list
  ,inner-instance      = enumerate
}
```

[2] In the LaTeX 2_ε implementation justified paragraphs where 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.

```
\DeclareInstance{blockenv}{description}{std}
{
   name                 = description
  ,tag-name             = \UseStructureName{block/description}
  ,tag-attr-class       = description
  ,tagging-recipe       = list
  ,inner-level-counter =
  ,transparent-level    = false
  ,legacy-code          =
  ,block-instance       = std-list
  ,inner-instance-type = list
  ,inner-instance       = description
}
```

3.7.2 Their `block` instances

block std-list-1 (*inst.*)
block std-list-2 (*inst.*)
block std-list-3 (*inst.*)
block std-list-4 (*inst.*)
block std-list-5 (*inst.*)
block std-list-6 (*inst.*)

The block instances for the various list environments use the same underlying instance (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.

```
\DeclareInstance{block}{std-list-1}{std}{
%    begin-vspace       = \topsep
%   ,begin-extra-vspace = \partopsep
```

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

```
   ,para-vspace        = \parsep
%   ,end-vspace          = \KeyValue{begin-vspace}
%   ,end-extra-vspace   = \KeyValue{begin-extra-vspace}
%   ,item-vspace         = \itemsep
%   ,begin-penalty       = \UseName{@beginparpenalty}
%   ,end-penalty         = \UseName{@endparpenalty}
%   ,left-margin         = \leftmargin
%   ,right-margin        = \rightmargin
%   ,para-indent         = 0pt
}
```

```
\DeclareInstanceCopy{block}{std-list-2}{std-list-1}
\DeclareInstanceCopy{block}{std-list-3}{std-list-2}
\DeclareInstanceCopy{block}{std-list-4}{std-list-3}
\DeclareInstanceCopy{block}{std-list-5}{std-list-4}
\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.*) `list itemize-2` (*inst.*) `list itemize-3` (*inst.*) `list itemize-4` (*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.

```
\DeclareInstance{list}{itemize-1}{std}{ item-label = \labelitemi   }
\DeclareInstance{list}{itemize-2}{std}{ item-label = \labelitemii  }
\DeclareInstance{list}{itemize-3}{std}{ item-label = \labelitemiii }
\DeclareInstance{list}{itemize-4}{std}{ item-label = \labelitemiv  }
```

`list enumerate-1` (*inst.*) `list enumerate-2` (*inst.*) `list enumerate-3` (*inst.*) `list enumerate-4` (*inst.*)

`enumerate` environments are similar, except that we also have to say which counter to use on each level.

```
\DeclareInstance{list}{enumerate-1}{std}
  { item-label = \labelenumi ,   counter = enumi   }
\DeclareInstance{list}{enumerate-2}{std}
  { item-label = \labelenumii ,  counter = enumii  }
\DeclareInstance{list}{enumerate-3}{std}
  { item-label = \labelenumiii , counter = enumiii }
\DeclareInstance{list}{enumerate-4}{std}
  { 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:

```
\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.*) `item description` (*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).

```
\DeclareInstance{item}{basic}{std}
                { label-align = right }
\DeclareInstance{item}{description}{std}
  {
    ,label-format = \normalfont\bfseries #1
    ,label-align  = left
  }
```

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.

```
\AddToHook{begindocument/before}[./legacy]{
  \RenewDocumentEnvironment{list}{O{} m m }
    {
```

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.

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

```
          #3
        }
```

The LaTeX 2ε lists don't support captions so we use `\SimpleBlockEnv`.

```
      \SimpleBlockEnv{list} {#1}
    }
    { \BlockEnvEnd }
}
```

`trivlist` (*env.*) LaTeX 2ε defined `trivlist` as an implementation of `list` (or rather the other way around).

Replace with code not using `\list`

```
\AddToHook{begindocument/before}[./legacy]{
  \RenewDocumentEnvironment{trivlist}{ !O{} }
                            { \list[#1]{}
                              {
                                \dim_zero:N \leftmargin
                                \dim_zero:N \labelwidth
                                \cs_set_eq:NN \makelabel \use:n
                              }
                            }
    { \BlockEnvEnd }
}
```

3.8.1 Its blockenv instance

`blockenv list` (*inst.*) The generic `list` environment of LaTeX 2ε 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 LaTeX 2ε) 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.

```
\DeclareInstance{blockenv}{list}{std}
{
   name                  = list
  ,tag-name              = \UseStructureName{block/list}
  ,tag-attr-class        =
  ,tagging-recipe        = list
  ,transparent-level     = false
  ,legacy-code           = \legacylistsetup
  ,block-instance        = std-list
  ,inner-level-counter   =
  ,inner-instance-type   = list
  ,inner-instance        = legacy
}
```

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.

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

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 it 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.

```
\DeclareInstance{thmstyle}{plain}{std}
  {
    ,caption-placement = unchained
    ,numbered          = true
    ,space             = \
    ,punct             = .
    ,before-hspace     = 0pt
    ,after-hspace      = 5pt plus 1pt minus 1pt
    ,order             = {title, space, number, punct, space, note}
    ,caption-decls     = \bfseries
    ,title-format      = #1
    ,number-format     = #1
    ,punct-format      = #1
    ,note-format       = (#1)
    ,body-decls        = \itshape
  }
```

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

```
\DeclareInstanceCopy{thmstyle}{remark}{plain}
\EditInstance{thmstyle}{remark}
{
  ,caption-decls = \itshape
  ,body-decls    = \normalfont
}
```

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

```
\DeclareInstanceCopy{thmstyle}{definition}{plain}
\EditInstance{thmstyle}{definition}
{
  ,body-decls = \normalfont
}
```

`thmstyle legacy2e` (*inst.*) Vanilla LaTeX 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.

```
\DeclareInstanceCopy{thmstyle}{legacy2e}{plain}
\EditInstance{thmstyle}{legacy2e}{ punct = }
```

3.9.4 The `block` instances they use

`block thm-plain-1` (*inst.*)
`block thm-plain-2` (*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 LaTeX default reused the general value of `\parindent` and `\parskip` and, of course, they start at the outer margin.

```
\DeclareInstance{block}{thm-plain-1}{std}
{
  ,begin-extra-vspace = 0pt
  ,left-margin        = 0pt
  ,para-indent        = \parindent
  ,para-vspace        = \parskip
}
```

```
\DeclareInstanceCopy{block}{thm-plain-2}{thm-plain-1}
```

`block thm-remark-1` (*inst.*)
`block thm-remark-2` (*inst.*)
The `\thmstyle` for "remarks" is defined by amsthm to use less vertical spacing. It therefore needs its own `block` instance.

```
\DeclareInstance{block}{thm-remark-1}{std}
{
  ,begin-vspace       = 0.5\topsep
  ,begin-extra-vspace = 0pt
  ,left-margin        = 0pt
  ,para-indent        = \parindent
  ,para-vspace        = \parskip
}
```

```
\DeclareInstanceCopy{block}{thm-remark-2}{thm-remark-1}
```

`block thm-legacy2e-1` (*inst.*)
`block thm-legacy2e-2` (*inst.*)

These are like the `plain` ones but without resetting `begin-extra-vspace` to zero.

```
\DeclareInstance{block}{thm-legacy2e-1}{std}
{
  ,left-margin        = 0pt
  ,para-indent        = \parindent
  ,para-vspace        = \parskip
}
```

```
\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.

```
\NewDocumentEnvironment{proof}{ ={note}o }
   { \pushQED{\qed}%
     \ParseLaTeXeTheoremlike {proof} \BooleanTrue {#1} }
   { \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.

```
 \DeclareInstance{blockenv}{proof}{std}
{
   name                 = proof
  ,tag-name             = \UseStructureName{block/proof}
  ,tag-attr-class       =
  ,tagging-recipe       = standalone
  ,inner-level-counter  =
  ,transparent-level    = true
  ,legacy-code          =
  ,block-instance       = proof
  ,inner-instance-type  = captionedtext
  ,inner-instance       = proof
  ,para-instance        = justify
}
```

`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.

```
\DeclareInstance{captionedtext}{proof}{proof}
  {
    ,title         = Proof
    ,punct         = .
    ,before-hspace = 0pt
    ,after-hspace  = 5pt plus 1pt minus 1pt
    ,caption-decls = \itshape
    ,title-format  = #1
    ,punct-format  = #1
    ,body-decls    = \normalfont
  }
```

3.10.1 Block instances for the proofs

`block proof-1` (*inst.*)
`block proof-2` (*inst.*) Blocks for proofs are pretty normal (the values are taken from the amsthm implementation):

```
\DeclareInstance{block}{proof-1}{std}
{
  ,begin-vspace       = 6pt plus 6pt
  ,left-margin        = 0pt
  ,para-indent        = \parindent
  ,para-vspace        = \parskip
}
\DeclareInstanceCopy{block}{proof-2}{proof-1}
```

4 Declaring para instances

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

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

```
\DeclareInstance{para}{justify}{std}
{
%  ,para-attr-class       = justify
%  ,para-indent           = \parindent
%  ,begin-hspace          = 0pt
%  ,left-hspace           = \z@skip
%  ,right-hspace          = \z@skip
%  ,end-hspace            = \@flushglue
%  ,final-hyphen-demerits =  5000
%  ,newline-cmd           = \@normalcr
}
```

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

```
\DeclareInstance{para}{center}{std}
{
  ,para-attr-class        = center
  ,para-indent           = 0pt
% ,begin-hspace          = 0pt
  ,left-hspace           = \@flushglue
  ,right-hspace          = \@flushglue
  ,end-hspace            = \z@skip
  ,final-hyphen-demerits = 0
  ,newline-cmd           = \@centercr
}
```

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

```
\DeclareInstance{para}{raggedright}{std}
{
  ,para-attr-class       = raggedright
  ,para-indent           = 0pt
% ,begin-hspace          = 0pt
  ,left-hspace           = \z@skip
  ,right-hspace          = \@flushglue
  ,end-hspace            = \z@skip
  ,final-hyphen-demerits = 0
  ,newline-cmd           = \@centercr
}
```

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

```
\DeclareInstance{para}{raggedleft}{std}
{
  ,para-attr-class       = raggedleft
  ,para-indent           = 0pt
% ,begin-hspace          = 0pt
  ,left-hspace           = \@flushglue
  ,right-hspace          = \z@skip
  ,end-hspace            = \z@skip
  ,final-hyphen-demerits = 0
  ,newline-cmd           = \@centercr
}
```

this should be moved elsewhere

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

```
\tagpdfsetup
  {
    ,role/new-attribute = {justify}    {/O /Layout /TextAlign/Justify}
    ,role/new-attribute = {center}     {/O /Layout /TextAlign/Center}
    ,role/new-attribute = {raggedright}{/O /Layout /TextAlign/Start}
    ,role/new-attribute = {raggedleft} {/O /Layout /TextAlign/End}
  }
```

`\centering`
`\raggedleft`
`\raggedright`
`\justifying`

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

```
\DeclareRobustCommand\centering  {\UseInstance{para}{center}{}}
\DeclareRobustCommand\raggedleft {\UseInstance{para}{raggedleft}{}}
\DeclareRobustCommand\raggedright{\UseInstance{para}{raggedright}{}}
\DeclareRobustCommand\justifying {\UseInstance{para}{justify}{}}
```

LATEX's default is to typeset paragraphs justified.

```
\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.

```
\DeclareInstance{para}{verse}{std}
{
  para-attr-class       = justify ,
  para-indent           = 0pt ,
  begin-hspace          = -1.5em ,
  left-hspace           =  1.5em ,
  right-hspace          = 0pt ,
  end-hspace            = \@flushglue ,
  final-hyphen-demerits = 0 ,
  newline-cmd           = \@centercr ,
}
```

```
⟨/class-code⟩
```

5 Advice on adjusting the layout of standard block environments

to document

6 Tagging support

6.1 Paragraph tags

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

In the PDF model and in the HTML model that is not supported — a limitation that conflicts with real 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 /O /Layout /TextAlign/Center>
    centered lines
  </text>
  <text /O /Layout /TextAlign/Center>
    with a paragraph break between them
  </text>
  <text>
    followed by some more text.
</text-unit>
```

The text-unit structures are added by using the tagging sockets `para/semantic/begin` and `para/semantic/end` declared in `lttagging.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 (`<LI>`) with suitable substructures (`<itemlabel>` for the item labels and `<itembody>` for the item bodies).
- If the key `tag-name` is specified, this is used as the tag name for the whole list instead of `<L>`. Of course, it should then have a suitable 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>`, `</LI>`, and `</L>` (or the tag name used) are closed.
- Then the lookahead for an empty line is done as described previously.

7 Tracing and debugging

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

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

cleanup

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

```
\DocumentMetadata{tagging=on, lang=en}

\documentclass{article}

\DebugBlocksOn

\begin{document}
```

```
    \begin{itemize}[item-vspace=3pt]
      \item             A normal item
      \item[\textbf{+}] A special item
    \end{itemize}
  \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

`\SimpleBlockEnv`
`\BlockEnv`
`\BlockEnvEnd`
`\g_block_nesting_depth_int`

to be documented

`\legacyverbatimsetup`
`\legacyallttsetup`
`\legacylistsetup`

to be documented

`\@setupverbinvisiblespace`

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

`\newtheorem`
`\newtheoremstyle` Reimplemented to fit the template approach. `\newtheoremstyle` was defined by amsthm.

`\@nthm`
`\@xnthm`
`\@ynthm`
`\@thm`
`\@xthm`
`\@ythm`
`\@othm`
`\@begintheorem`
`\@opargbegintheorem`
`\@endtheorem` These are no longer used (to be removed).

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

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

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

`\@doendpe` The original LaTeX 2$_\varepsilon$ command is augmented to allow for tagging.

`\para_end:` TODO: consider name, document

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

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

@@ commands:
- \l_@@_legacy_env_params_tl 328

\␣ 369

A

\AddToHook . 50, 94, 138, 155, 237, 325, 337
alltt (env.) 160
alltt* (env.) 160

B

\begin *37*
\bfseries 322, 374
block commands:
- \block_debug_off: *35*
- \block_debug_on: *35*
- \g_block_nesting_depth_int *36*

block proof-1 (instance) 447
block proof-2 (instance) 447
block quotation-1 (instance) 131
block quotation-2 (instance) 131
block quotation-3 (instance) 131
block quotation-4 (instance) 131
block quotation-5 (instance) 131
block quotation-6 (instance) 131
block quote-1 (instance) 124
block quote-2 (instance) 124
block quote-3 (instance) 124
block quote-4 (instance) 124
block quote-5 (instance) 124
block quote-6 (instance) 124
block std-display-1 (instance) 31
block std-display-2 (instance) 31
block std-display-3 (instance) 31
block std-display-4 (instance) 31
block std-display-5 (instance) 31
block std-display-6 (instance) 31
block std-list-1 (instance) 287
block std-list-2 (instance) 287
block std-list-3 (instance) 287
block std-list-4 (instance) 287
block std-list-5 (instance) 287
block std-list-6 (instance) 287
block thm-legacy2e-1 (instance) 411
block thm-legacy2e-2 (instance) 411
block thm-plain-1 (instance) 394
block thm-plain-2 (instance) 394
block thm-remark-1 (instance) 402
block thm-remark-2 (instance) 402
block verbatim-1 (instance) 227
block verbatim-2 (instance) 227
block verbatim-3 (instance) 227
block verbatim-4 (instance) 227
block verbatim-5 (instance) 227
block verbatim-6 (instance) 227
\BlockEnv *14*
\BlockEnv *14*, *36*
blockenv alltt (instance) 197
blockenv alltt* (instance) 212
blockenv center (instance) 58
blockenv description (instance) 274
blockenv displayblock (instance) 6
blockenv displayblockflattened (instance) 23
blockenv enumerate (instance) 260
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 verbatim (instance) 165
blockenv verbatim* (instance) 181
blockenv verse (instance) 142
\BlockEnvEnd *14*
\BlockEnvEnd *14*, *36*, 3, 5, 52, 54, 56, 96, 98, 140, 157, 159, 161, 163, 239, 241, 243, 335, 346, 421
\BooleanFalse *13*
\BooleanTrue *13*, 420

C

captionedtext proof (instance) 436
center (env.) 50
\centering 506
cs commands:
- \cs_set_eq:NN 343

D

\DebugBlocksOff *35*
\DebugBlocksOn *35*
\DebugTemplatesOff *35*
\DebugTemplatesOn *35*
\DeclareDocumentEnvironment *17*, 95, 97, 139, 242

\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
\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
\DeclareRobustCommand . 506, 507, 508, 509
description (env.) 237
dim commands:
  \dim_zero:N 341, 342
displayblock (env.) 2
displayblockflattened (env.) 2

E

\EditInstance 87, 91, 382, 388, 393
enumerate (env.) 237
environments:
  alltt 160
  alltt* 160
  center 50
  description 237
  displayblock 2
  displayblockflattened 2
  enumerate 237
  flushleft 50
  flushright 50
  itemize 237
  list 325
  proof 418
  quotation 94
  quote 94
  trivlist 337
  verbatim 155
  verbatim* 155
  verse 138

F

flushleft (env.) 50
flushright (env.) 50

I

\ignorespaces 21
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
  block std-display-5 31
  block std-display-6 31
  block std-list-1 287
  block std-list-2 287
  block std-list-3 287
  block std-list-4 287
  block std-list-5 287
  block std-list-6 287
  block thm-legacy2e-1 411
  block thm-legacy2e-2 411
  block thm-plain-1 394
  block thm-plain-2 394
  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
para verse 511
thmstyle definition 387
thmstyle legacy2e 392
thmstyle plain 365
thmstyle remark 381
\item *9*, *37*
item basic (instance) 318
item description (instance) 318
itemize (env.) 237
\itemsep 38, 293
\itshape 379, 384, 442

J

\justifying 506

K

\KeyValue 36, 37, 125, 132, 291, 292

L

\labelenumi 310
\labelenumii 312
\labelenumiii 314
\labelenumiv 316
\labelitemi 305
\labelitemii 306
\labelitemiii 307
\labelitemiv 308
\labelwidth 342
\leftmargin 296, 341
\legacyalltt setup *36*, 210, 225
\legacylistsetup *25*, *36*, 355
\legacyverbatimsetup *36*, 178, 194
list (env.) 325
\list 339
list description (instance) 317
list enumerate-1 (instance) 309
list enumerate-2 (instance) 309
list enumerate-3 (instance) 309
list enumerate-4 (instance) 309
list itemize-1 (instance) 305
list itemize-2 (instance) 305
list itemize-3 (instance) 305
list itemize-4 (instance) 305
list legacy (instance) 361
\listparindent *6*

M

\makelabel 343

N

\NewDocumentEnvironment 2, 4, 160, 162, 418
\newtheorem *26–28*, *37*
\newtheoremstyle *13*, *26*, *27*, *37*
\normalfont 322, 385, 390, 445
\NoValue *3–5*, *12*, *13*
\Novalue *13*

O

\obeyedline *19*, 157, 159

P

\par *16*
para center (instance) 466
para commands:
  \para_end: *37*
para justify (instance) 455
para raggedleft (instance) 488
para raggedright (instance) 477
para verse (instance) 511
para/begin *37*
\parindent 398, 407, 414, 451, 458
\ParseLaTeXeTheoremlike *29*, 420
\parsep 290
\parskip *8*, 35, 399, 408, 415, 452
\partopsep 34, 289
\popQED *29*, 421
proof (env.) 418
\pushQED *29*, 419

Q

\qed 419
quotation (env.) 94
quote (env.) 94

R

\raggedleft 506
\raggedright 506
\RenewDocumentEnvironment *17*,
  51, 53, 55, 156, 158, 238, 240, 326, 338
\rightmargin 42, 297

S

\setcounter 30
\SimpleBlockEnv *13*
\SimpleBlockEnv *13*,
  25, *36*, 3, 5, 52, 54, 56, 96, 98, 140,
  157, 159, 161, 163, 239, 241, 243, 333
\swapnumbers *13*, *26*, *27*

T

\tagpdfsetup 499
TEX and LATEX 2_ε commands:
  \@beginparpenalty *8*
  \@begintheorem *37*
  \@centercr 475, 486, 497, 520
  \@doendpe *37*
  \@endparpenalty *8*
  \@endtheorem *37*
  \@enumdepth *22*, 270
  \@flushglue . . . .
    . . . . . . *9*, 462, 471, 472, 483, 493, 518
  \@itemdepth *22*, 251
  \@itemlabel *37*, 330
  \@itempenalty *8*
  \@list... . . . . *7*
  \@listdepth *7*
  \@listi *6*, *7*
  \@listii *6*, *7*
  \@listvi *7*
  \@normalcr *9*, 464
  \@nthm *37*
  \@opargbegintheorem *37*
  \@othm *37*
  \@setupverbinvisiblespace *36*
  \@sxverbatim *20*, 195
  \@thm *37*
  \@xnthm *37*
  \@xthm *37*
  \@xverbatim 179
  \@ynthm *37*
  \@ythm *37*
  \arabic *10*
  \begin *37*
  \c@maxblocklevels *37*
  \ignorespaces *7*
  \item *14*, *37*
  \itemsep *8*
  \labelsep *10*
  \labelwidth *10*
  \leftmargin *8*
  \legacylistsetup *24*
  \list *25*
  \list⟨*romannumeral*⟩ *15*, *23*
  \makelabel *10*, *26*
  \par *35*
  \parindent *9*, *28*
  \parskip *8*, *28*
  \partopsep *8*
  \pdffakespace *20*
  \rightmargin *8*
  \strut *10*
  \topsep *8*
  \z@skip 460, 461, 473, 482, 484, 494, 495
\theoremstyle *26*, *27*
\thmstyle *28*
thmstyle definition (instance) . . . . . <u>387</u>
thmstyle legacy2e (instance) . . . . <u>392</u>
thmstyle plain (instance) . . . . <u>365</u>
thmstyle remark (instance) . . . . <u>381</u>
tl commands:
  \tl_set:Nn 328, 330
\topsep 33, 288, 404
trivlist (env.) . . . . <u>337</u>

U

use commands:
  \use:n 343
\UseInstance 506, 507, 508, 509
\UseName 39, 40, 294, 295
\UseStructureName . 103, 115, 145, 168,
    184, 200, 215, 248, 263, 277, 351, 425

V

verbatim (env.) . . . . <u>155</u>
verbatim* (env.) . . . . <u>155</u>
verse (env.) . . . . <u>138</u>